UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

José Eduardo Henriques da Silva

Programação Genética Cartesiana com Recombinação e Mutação Guiada Aplicada ao Projeto de Circuitos Lógicos Combinacionais

Juiz de Fora

2019

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

José Eduardo Henriques da Silva

Programação Genética Cartesiana com Recombinação e Mutação Guiada Aplicada ao Projeto de Circuitos Lógicos Combinacionais

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Heder Soares Bernardino

Juiz de Fora

2019

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Henriques da Silva, José Eduardo.
Programação Genética Cartesiana com Recombinação e Mutação Guiada Aplicada ao Projeto de Circuitos Lógicos Combinacionais / José Eduardo Henriques da Silva. -- 2019.
160 f.

Orientador: Heder Soares Bernardino
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, ICE/Engenharia. Programa de Pós-Graduação em Ciência da Computação, 2019.

1. Programação Genética Cartesiana. 2. Computação Evolucionista. 3. Circuitos Lógicos Combinacionais. 4. Hardware Evolutivo. I. Bernardino, Heder Soares, orient. II. Título.

José Eduardo Henriques da Silva

Programação Genética Cartesiana com Recombinação e Mutação Guiada Aplicada ao Projeto de Circuitos Lógicos Combinacionais

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 28 de Fevereiro de 2019.

BANCA EXAMINADORA

Prof. D.Sc. Heder Soares Bernardino - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Itamar Leite de Oliveira
Universidade Federal de Juiz de Fora

Prof. D.Sc. Douglas Adriano Augusto
Fiocruz

A Deus em primeiro lugar. Aos meus pais, orientador e amigos pelo apoio incondicional.

AGRADECIMENTOS

Obrigado a meu orientador Heder Soares Bernardino pelas diversas horas de conversas e eslarecimento de dúvidas. Ao apoio dos familiares e amigos e a todos os professores que transmitiram-me seus conhecimentos para chegar até aqui.

"Ninguém é tão grande que não
possa aprender, nem tão pequeno
que não possa ensinar."
Esopo

RESUMO

Diversas abordagens são encontradas na literatura no que tange ao uso de técnicas de computação evolucionista para o projeto de circuitos digitais. Entretanto, o problema da escalabilidade continua sendo um gargalo para o *hardware* evolutivo. Estas técnicas mostraram-se promissoras por encontrar projetos que fogem à concepção humana. Dentre os circuitos digitais, os lógicos combinacionais compreendem uma grande parte, incluindo circuitos aritméticos (somadores e multiplicadores), comparadores, entre outros. A Programação Genética Cartesiana (CGP) é apontada como o método evolutivo mais eficiente para o projeto de circuitos digitais. O uso da CGP no projeto de circuitos lógicos combinacionais é abordado aqui, além da proposta de métodos para auxiliá-la. As principais contribuições são: (i) um estudo sobre as dificuldades da CGP na obtenção de circuitos factíveis, (ii) um operador de recombinação, (iii) um operador de mutação que age no pior subgrafo da CGP, denominado *Guided Active Mutation*(GAM), (iv) uma abordagem que modifica a Estratégia Evolutiva (ES) comumente usada na CGP passando a operar com duas estratégias de mutação, (v) o uso de multiplexadores como elemento lógico no conjunto de funções da CGP, e (vi) um novo método de evolução de circuitos lógicos em três etapas através do acoplamento de um multiplexador de duas entradas em cada uma das saídas do circuito. Os experimentos computacionais que validaram os métodos propostos foram compostos de problemas largamente utilizados na literatura e de circuitos *benchmark*. Nos experimentos realizados, o operador de recombinação mostra-se importante para o aumento da quantidade de circuitos factíveis encontrados e, quando combinado com o operador de mutação GAM e a modificação da ES, para a diminuição no número de avaliações necessárias para obter soluções factíveis. Além disso, as abordagens que utilizam multiplexadores, além de também diminuir o número de avaliações necessárias para obter factibilidade, foram capazes de evoluir circuitos com maior quantidade de entradas do que os métodos tradicionais.

Palavras-chave: Programação Genética Cartesiana. Computação Evolucionista. Circuitos Lógicos Combinacionais. *Hardware* Evolutivo.

ABSTRACT

Several approaches are found in the literature regarding the use of evolutionary computational techniques for the digital circuits design. However, the scalability problem remains a bottleneck for evolvable hardware. These techniques have proved promising for finding designs different from those of human conception. Among the digital circuits, the combinational logic ones comprise a large part including arithmetic circuits (multipliers and adders), comparators, among others. The Cartesian Genetic Programming (CGP) is pointed out as the most efficient evolutionary method for the digital circuits design. The use of CGP in the design of combinational logic circuits is addressed here, as well as the proposal of methods to assist it. The main contributions are: (i) a study of CGP's difficulties in obtaining feasible circuits, (ii) a recombination operator, (iii) a mutation operator acting on the worst individuals' subgraph , called Guided Active Mutation (GAM), (iv) an approach that modifies the Evolutionary Strategy (ES) commonly used in CGP, operating with two mutation strategies, (v) the use of multiplexers as a logical element in the CGP function set, and (vi) a new evolving method of logic circuits in three stages by coupling a multiplexer with two inputs in each circuit output. The computational experiments that validated the proposed methods were composed of problems widely used in the literature and benchmark circuits. In the experiments using recombination, this operator shown to be important to increase the amount of feasible circuits and, when combined with GAM mutation operator and the modification of ES, to decrease the number of evaluations necessary to obtain feasible solutions. In addition, approaches that use multiplexers decreases the number of evaluations needed to find a feasible solution, and they were able to evolve circuits with more inputs than traditional methods.

Keywords: Cartesian Genetic Programming. Evolutionary Computation. Combinational Logic Circuits. Evolvable Hardware.

LISTA DE FIGURAS

2.1 Sinal analógico × sinal digital. Figura adaptada de (MANFRINI *et al.*, 2017). 27

2.2 Simbologia, tabelas verdade e expressões das portas lógicas. 30

2.3 Multiplexador de 2 entradas e um controle. Para duas entradas basta 1 *bit* para o controle (seleção). 31

2.4 Regiões de simplificação de um mapa de Veitch-Karnaugh para duas variáveis. 32

2.5 Mapa de Veitch-Karnaugh do exemplo da Tabela 2.2. 33

2.6 Mapa de Veitch-Karnaugh para o exemplo da Tabela 2.3 35

2.7 Implementação do circuito exemplo cuja função simplificada é apresentada na Equação 2.3 utilizando-se portas lógicas de duas entradas. 36

3.1 Fluxograma de funcionamento dos algoritmos evolutivos. 40

3.2 Representação de um cromossomo binário. Os alelos representam os valores que cada gene pode assumir. Cada gene ocupa uma posição denominada *locus*. 42

3.3 Os nós em cinza foram selecionados para receberem a mutação do tipo *bit flip*. 45

3.4 Geração de dois filhos a partir de dois pais pelo operador de recombinação de cruzamento em 2 pontos. 46

3.5 Árvore de sintaxe representando max(x+x, x+3y). Figura adaptada de (POLI *et al.*, 2008) 49

3.6 Representação de um indivíduo da CGP e seu respectivo genótipo. Os nós em cinza são ativos e os nós em branco inativos. As linhas tracejadas representam conexões que não contribuem para o fenótipo. No genótipo, os três primeiros inteiros representam as entradas primárias do programa e os três últimos inteiros, as saídas. Os nós 3 a 11 são os nós presentes na matriz de representação (genótipo) do indivíduo. 51

3.7 Fenótipos de todas as saídas do programa em relação ao indivíduo da figura 3.6. As lacunas com os números 0 a 2 são as entradas primárias do programa. As lacunas com os números 9, 10 e 11 são as saídas do programa. Os nós entre as entradas e as saídas são enumerados de acordo com sua posição no genótipo do indivíduo. 52

3.8 Gráficos de barra das probabilidades de transição de portas obtidas pelo BSAM. 57
3.9 Codificação matricial proposta por Loius e utilizada por Coello - figura extraída e adaptada de (LOUIS, 1993). 62

4.1 Representações de F = 0 e F = 1 no Mapa de Karnaugh. 71
4.2 Representação de F = D no Mapa de Karnaugh. 72
4.3 Representações das composições de 2, 4, 6 e 8 mintermos, não simplificantes entre si. 72
4.4 Representações das composições de 2, 4, 6 e 8 mintermos simplificantes entre si. 73
4.5 Bloco de 4 e bloco de 2 mintermos não simplificantes entre si no Mapa de Karnaugh. 73
4.6 Bloco de 4 mintermos e um mintermo isolado no Mapa de Karnaugh. 73
4.7 Mapa de Karnaugh faltando um mintermo (0100) para completar uma simplificação de 8. 73
4.8 *Boxplots* para F=0 e F=1. 75
4.9 *Boxplots* de todas as funções que representavam literais. 77
4.10 Circuito obtido que realiza a função F = $\overline{A}$ 78
4.11 *Boxplots* para todos os mintermos. 79
4.12 *Boxplots* para um bit 0 em posições variadas da tabela verdade. 81
4.13 *Boxplots* para as composições de 2 mintermos não simplificantes entre si. . . . 83
4.14 *Boxplots* das composições de 4 mintermos não simplificantes entre si. 85
4.15 *Boxplots* das composições de 6 mintermos não simplificantes entre si. 87
4.16 *Boxplots* das composições de 8 mintermos não simplificantes entre si. 89
4.17 *Boxplots* das composições de 2 mintermos simplificantes entre si. 91
4.18 *Boxplots* das composições de 4 mintermos simplificantes entre si. 92
4.19 *Boxplots* das composições de 6 mintermos simplificantes entre si. 94
4.20 *Boxplots* das composições de 8 mintermos simplificantes entre si. 95
4.21 *Boxplots* das composições de 4 e 2 mintermos não simplificantes entre si. . . . 97
4.22 *Boxplots* das composições de 4 mintermos com 1 mintermo isolado. 99
4.23 *Boxplots* das composições de 8 mintermos com 1 mintermo retirado. 99
4.24 Número de avaliações necessárias para obter a tabela verdade proposta com todas as modificações. 102

5.1 Fluxograma do X-CGP. 107
5.2 (a) e (b) são dois indivíduos da população. (c) é o novo indivíduo gerado pelo *crossover* proposto; a saída O1 é do segundo indivíduo (b) é incorporado para o genótipo do primeiro (a). 109
5.3 Procedimento de incorporação dos subgrafos considerando que O0 é melhor no indivíduo 1 (a), O1 é igual em ambos os indivíduos e O2 é melhor no indivíduo 2 (b). 110
5.4 Procedimento de liberação de nós redundantes. 111
5.5 Fluxograma do operador de mutação GAM. 113
5.6 (a) é o pai da geração 1 e seu subgrafo que compõe a saída O1 é apresentado em (b), em vermelho. (c) é o pai da geração 2 criado por uma mutação no nó 10 do pai anterior com seu respectivo subgrafo em vermelho.. Ambos subgrafos em vermelho de (c) e em azul de (d) acertam 6 de 8 bits da tabela verdade. Um deles será escolhido aleatoriamente para receber o operador GAM e gerar a descendência subsequente. 114
5.7 Representação de um nó cuja função é multiplexador. Todos os nós do genótipo possuirão 4 genes e o gene de controle só é considerado ativo quando a função executada é um multiplexador. 117
5.8 Ordem de evolução dos subcircuitos na abordagem de evolução em 3 etapas. Primeiramente, obtém-se um circuito factível levando em consideração a união das tabelas-verdade dos subcircuitos que compõem I0 e I1 para todas as saídas do circuito. Após isso, na segunda etapa, maximiza-se a similaridade entre I0 e I1. Por fim obtém-se o subcircuito de controle cuja tabela verdade torna-se mais fácil de ser obtida conforme aumenta-se as similaridades entre I0 e I1. 119
5.9 Representação do indivíduo da evolução em 3 etapas. Nós em cinza são ativos, brancos inativos, entradas inferiores nos multiplexadores são os subcircuitos de controle. As linhas contínuas definem o fenótipo. 119

6.1 Performance Profiles das três melhores variantes do X-CGP. Os valores normalizados de AUC são 1.0, 0.9977 e 0.9952 para MAM(1)-L9, MAM(1)-L3 e MAM(3)-L5, respectivamente. 128

LISTA DE TABELAS

2.1 Tabela verdade de 4 entradas e uma saída. 29
2.2 Tabela verdade de exemplo para simplificação. 33
2.3 Tabela verdade exemplo para a geração de um CLC possuindo 4 entradas e uma saída. 35

3.1 Seleção proporcional à aptidão (SPA) versus Seleção por Classificação Linear (SCL). 45
3.2 Codificação utilizada e suas respectivas funções 50
3.3 Número de mutações benéficas observada quando minimiza-se o erro em relação à tabela verdade. 57

4.1 Tabela verdade cuja obtenção do circuito apresenta dificuldades pela CGP. . . 70
4.2 Número de avaliações necessárias para obter um circuito factível quando F=0 e F=1. 74
4.3 Número de avaliações necessárias para obter um circuito factível quando F=0 e F=1 para 10, 15 e 20 entradas. 76
4.4 Número de portas necessárias para obter cada uma das funções que expressam uma entrada primária ou seu complemento. 76
4.5 Funções obtidas pela CGP para cada um dos literais desejados. 78
4.6 Número de avaliações necessárias para obter cada um dos mintermos. 79
4.7 Número de portas necessárias para obter cada um dos mintermos. 80
4.8 Número de avaliações necessárias para a investigação com *bit* 0 em posição variável. 81
4.9 Composições de 2 mintermos não simplificantes entre si utilizados. 82
4.10 Número de avaliações necessárias para composições de 2 mintermos não simplificantes entre si. 83
4.11 Composições de 4 mintermos não simplificantes entre si utilizados. 84
4.12 Número de avaliações necessárias para composições de 4 mintermos não simplificantes entre si. 84

4.13 Número de portas lógicas necessárias para composições de 4 mintermos não simplificantes. 86
4.14 Composições de 6 mintermos não simplificantes entre si utilizados. 86
4.15 Número de avaliações necessárias para composições de 6 mintermos não simplificantes. 87
4.16 Composições de 8 mintermos não simplificantes entre si utilizados. 88
4.17 Número de avaliações necessárias para composições de 8 mintermos não simplificantes. 89
4.18 Composições de 2 mintermos simplificantes entre si utilizados. 90
4.19 Número de avaliações necessárias para as composições de 2 mintermos simplificantes. 91
4.20 Composições de 4 mintermos simplificantes entre si utilizados. 92
4.21 Número de avaliações necessárias para as composições de 4 mintermos simplificantes. 92
4.22 Composições de 6 mintermos simplificantes entre si utilizados. 93
4.23 Número de avaliações necessárias para as composições de 6 mintermos simplificantes. 93
4.24 Composições de 8 mintermos simplificantes entre si utilizados. 95
4.25 Número de avaliações necessárias para as composições de 8 mintermos simplificantes. 95
4.26 Composições de 4 e 2 mintermos não simplificantes entre si utilizados. 96
4.27 Número de avaliações necessárias para as composições de 4 e 2 mintermos não simplificantes entre si. 96
4.28 Composições de 4 mintermos com 1 mintermo isolado utilizados. 98
4.29 Número de avaliações necessárias para as composições de 4 mintermos com 1 mintermo isolado. 98
4.30 Composições de 8 mintermos com um mintermo retirado. 98
4.31 Número de avaliações necessárias para as composições de 8 mintermos com 1 mintermo retirado. 100
4.32 Investigações em modificações feitas na Tabela 4.1. 101
4.33 Número de avaliações necessárias para as variações da Tabela 4.1. 101

5.1 Correspondências em relação à tabela verdade. 108

5.2 Correspondências em relação à tabela verdade por geração. 113
5.3 Tabelas verdade no início do processo evolutivo. 120
5.4 Tabelas verdade no momento de satisfação do critério. 120
5.5 Tabelas verdade no momento de satisfação do critério. 121
5.6 Tabelas verdade no fim da segunda etapa. 121
5.7 Tabelas verdade que deve ser obtida no controle. 122

6.1 Problemas utilizados nos experimentos do X-CGP e para o operador de mutação GAM (ni: número de entradas, no: número de saídas, nc: número de colunas, EI: quantidade de execuções independentes e Max. Aval: número máximo de avaliações da função objetivo). 126
6.2 Problemas utilizados para o ajuste de parâmetros (ni: número de entradas, no: número de saídas, nc: número de colunas e Max. Aval.: número máximo de avaliações permitidas). 127
6.3 Resultados dos experimentos computacionais do X-CGP em comparação à CGP padrão com SAM e ao X-CGP-OP (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 129
6.4 Resultados dos experimentos computacionais do X-CGP em comparação à CGP padrão com SAM e ao X-CGP-OP (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 130
6.5 Resultados dos testes estatísticos entre CGP(1), X-CGP(2) e X-CGP-OP(3). . 131
6.6 Resultados dos experimentos computacionais do GAM em comparação ao X-CGP e ao X-GAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 134

6.7 Resultados dos experimentos computacionais do GAM em comparação ao X-CGP e ao X-GAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 135
6.8 Resultados dos testes estatísticos entre X-CGP(1), GAM(2) e X-GAM(3). . . 136
6.9 Resultados dos experimentos computacionais do X-CGP, X-GAM e X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 138
6.10 Resultados dos experimentos computacionais do X-CGP, X-GAM e X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 139
6.11 Resultados dos testes estatísticos entre X-CGP(1), X-GAM(2) e X-SAMGAM(3).140
6.12 Resultados da otimização entre a CGP padrão e o X-SAMGAM. 141
6.13 Problemas utilizados nos experimentos que utilizam abordagens com multiplexadores. 142
6.14 Resultados dos experimentos computacionais do X-SAMGAM-MUX em comparação com o X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 144
6.15 Resultados dos experimentos computacionais do X-SAMGAM-MUX em comparação com o X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 145
6.16 Resultados dos testes estatísticos entre X-SAMGAM e X-SAMGAM-MUX. . . 146

6.17 Resultados dos experimentos computacionais do X-CGP-3E em comparação ao X-SG-MUX e ao X-SAMGAM. (P: Problema, Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso). 148

LISTA DE SÍMBOLOS

$\cdot$	Denota o operador de multiplicação em álgebra booleana.
$\oplus$	Denota o operador Ou Exclusivo em álgebra booleana.
$\odot$	Denota o operador Coinciência em álgebra booleana.
μ	Denota a quantidade de progenitores por geração.
λ	Denota a quantidade de descendência criada por geração.
Γ	Representa o conjunto de funções utilizadas na CGP.

LISTA DE ABREVIATURAS E SIGLAS

3E Evolução em 3 Etapas

AE Algoritmo Evolucionário

AG/GA Algoritmo Genético/Genetic Algorithm

AUC Area Under the Curve

BGA Binary Genetic Algorithm

BSAM Biased Single Active Mutation

CGP Cartesian Genetic Programming

CI Circuito Integrado

CLC Circuito Lógico Combinacional

DAG Directed Acyclic Graph

ECGP Embedded Cartesian Genetic Programming

EE/ES Estratégias Evolutivas/Evolutionary Strategies

GAM Guided Active Mutation

FPGA Field Programmable Gate Array

MAM Multiple Active Mutation

MUX Multiplexador

NGA N-cadinality Genetic Algorithm

NGD Neutral Genetic Drift

PE Programação Evolutiva

PG/GP Programação Genética/Genetic Programming

PLA Programmable Logic Array

PSO Particle Swarm Optimization

SA Simulated Annealing

SAM Single Active Mutation

ULA Unidade Lógica Aritmética

X-CGP Programação Genética Cartesiana com Recombinação

X-CGP-3E Programação Genética Cartesiana com Recombinação e Evolução em 3 Etapas

X-CGP-OP Programação Genética Cartesiana com Recombinação com Parâmetros Otimizados

X-GAM Programação Genética Cartesiana com Recombinação e GAM

X-SAMGAM Programação Genética Cartesiana com Recombinação com SAM e GAM

X-SAMGAM-MUX Programação Genética Cartesiana com Recombinação, SAM e GAM e Multiplexadores como Elemento Lógico

SUMÁRIO

1 INTRODUÇÃO 22

1.1 OBJETIVOS 24

1.1.1 Organização do Texto 24

2 FUNDAMENTOS DE CIRCUITOS DIGITAIS 26

2.1 SISTEMAS ANALÓGICOS E DIGITAIS 26

2.2 CIRCUITOS LÓGICOS 27

2.2.1 Tabelas Verdade 28

2.2.2 Portas Lógicas 28

2.2.3 Multiplexadores 31

2.2.4 Simplificação de Circuitos Lógicos 31

2.2.4.1 Mapa de Veitch-Karnaugh 31

2.2.4.2 Ferramentas Computacionais 33

2.3 PROJETO DE CIRCUITOS LÓGICOS COMBINACIONAIS 34

3 COMPUTAÇÃO EVOLUCIONISTA 37

3.1 INSPIRAÇÃO BIOLÓGICA 37

3.2 PROBLEMAS DE OTIMIZAÇÃO 38

3.3 ALGORITMOS EVOLUTIVOS 39

3.3.1 Componentes de Algoritmos Evolutivos 41

3.3.1.1 Representação 42

3.3.1.2 Função de Aptidão 42

3.3.1.3 População 43

3.3.1.4 Mecanismo de Seleção de Pais 43

3.3.1.5 Operadores de Variação 45

3.3.1.6 Mecanismo de Seleção de Sobrevivência (*Replacement*) 46

3.3.1.7 Inicialização e Critérios de Parada 46

3.4 ESTRATÉGIAS EVOLUTIVAS 47

3.5 PROGRAMAÇÃO GENÉTICA 48

3.6 PROGRAMAÇÃO GENÉTICA CARTESIANA 48

3.6.1 Função de Aptidão 53
3.6.2 Operadores de Mutação 53
3.6.2.1 Mutação Pontual 54
3.6.2.2 Skip 54
3.6.2.3 Accumulate 55
3.6.2.4 Single Active Mutation - SAM 55
3.6.2.5 Biased SAM - BSAM 56
3.6.3 Operadores de Recombinação 58
3.6.4 Neutralidade, Redundância e Deriva Gênica Neutra 59
3.7 PROJETO EVOLUTIVO DE CIRCUITOS DIGITAIS 61

4 INVESTIGAÇÃO DE DIFICULDADES DA CGP 70
4.1 DESCRIÇÃO DOS PROBLEMAS INVESTIGADOS 71
4.1.1 Resultados Preliminares 74
4.1.2 Investigação 1 - F = 0 ou F = 1 74
4.1.3 Investigação 2 - Funções iguais às entradas primárias ou seus complementos 75
4.1.4 Investigação 3 - Um único mintermo 77
4.1.5 Investigação 4 - Um único *bit* 0 com posição variável 80
4.1.6 Investigação 5 - Composições não simplificantes entre si 82
4.1.6.1 2 mintermos 82
4.1.6.2 4 mintermos 84
4.1.6.3 6 mintermos 86
4.1.6.4 8 mintermos 88
4.1.7 Investigação 6 - Composições simplificantes entre si 89
4.1.7.1 2 mintermos 90
4.1.7.2 4 mintermos 90
4.1.7.3 6 mintermos 91
4.1.7.4 8 mintermos 94
4.1.8 Investigação 7 - Composições Híbridas 96
4.1.8.1 Bloco de 4 mintermos e um bloco de 2 mintermos 96
4.1.8.2 Bloco de 4 mintermos e um mintermo isolado 97
4.1.8.3 Bloco de 8 mintermos com um mintermo retirado 98
4.1.9 Investigação 8 - Estudos da Tabela 4.1 100

4.2 CONCLUSÕES PRELIMINARES 101

5 MÉTODOS PROPOSTOS 105

5.1 OPERADOR DE RECOMBINAÇÃO 105

5.2 OPERADOR DE MUTAÇÃO GUIADA 111

5.2.1 Uso do GAM com Crossover 115

5.3 MODIFICAÇÃO DAS ESTRATÉGIAS EVOLUTIVAS DA CGP: SAM E GAM 115

5.4 MULTIPLEXADOR COMO ELEMENTO LÓGICO 116

5.5 EVOLUÇÃO EM 3 ETAPAS 117

5.5.1 Primeira Etapa 120

5.5.2 Segunda Etapa 121

5.5.3 Terceira Etapa 122

6 EXPERIMENTOS COMPUTACIONAIS 124

6.1 DESCRIÇÃO DOS PROBLEMAS INVESTIGADOS 125

6.2 CROSSOVER (X-CGP) 125

6.2.1 Problemas 125

6.2.2 Ajuste de Parâmetros do X-CGP 126

6.2.3 Análise Comparativa dos Resultados 128

6.3 OPERADOR DE MUTAÇÃO GUIADA (GAM) 132

6.3.1 Análise Comparativa dos Resultados 132

6.4 ESTRATÉGIA SAM + GAM 133

6.4.1 Análise Comparativa dos Resultados 136

6.5 MUX COMO ELEMENTO LÓGICO 140

6.5.1 Problemas 140

6.5.2 Análise Comparativa dos Resultados 142

6.6 EVOLUÇÃO EM 3 ETAPAS 146

6.6.1 Análise Comparativa dos Resultados 146

7 CONCLUSÕES E TRABALHOS FUTUROS 149

REFERÊNCIAS 153

1 INTRODUÇÃO

Os circuitos eletrônicos estão cada vez mais presentes no cotidiano das pessoas principalmente com o avanço e popularização dos dispositivos de microinformática como celulares e *tablets*. Na eletrônica digital o projeto e otimização de circuitos lógicos são tarefas de grande importância, seja na necessidade de implementar uma nova funcionalidade ou de atender outros critérios, tais como espaço e tamanho a serem minimizados, consumo de potência, *delay*, tolerância a falhas, entre outros.

Desde o desenvolvimento da eletrônica digital e a descoberta da possibilidade em modelar os circuitos como expressões lógicas por meio da álgebra de Boole, e ainda acelerados pela miniaturização dos dispositivos eletrônicos, os projetistas perceberam a necessidade de obter projetos inovadores para circuitos que já existiam, como os multiplicadores e os circuitos aritméticos, em geral. Este fato se torna bastante perceptível quando se deu o início da era dos circuitos integrados, que são dispositivos capazes de compactar diversas funcionalidades em um pequeno bloco de silício como, por exemplo as Unidades Lógicas Aritméticas (ULAs).

Para o projeto tradicional utiliza-se, além das regras de simplificação da álgebra de Boole, ferramentas gráficas e computacionais, tais como o mapa de Veitch-Karnaugh, o método de Quine Mc-Cluskey (MCCLUSKEY, 1956) e o ESPRESSO (BRAYTON *et al.*, 1984). Apesar de ser possível utilizar tais técnicas, o projeto é um processo de grande complexidade que demanda tempo e experiência de conjuntos de regras específicas por parte dos projetistas, principalmente quando os circuitos são grandes, isto é, possuem muitas entradas e muitas saídas, além de possuírem muitos níveis de profundidade. Além disso, algumas técnicas tais como o ESPRESSO utilizam simplificações baseadas na soma de produtos e só trabalham com as portas lógicas básicas (AND, OR e NOT) sendo que muitas simplificações podem ser realizadas com as demais portas ainda que fujam à concepção humana.

Por outro lado, a computação evolutiva mostrou-se bastante eficiente na resolução de diversas tarefas de otimização e o projeto de circuitos lógicos combinacionais. Apesar de na década de 50 Friedman (FRIEDMAN, 1956) relatar a aplicação de técnicas de computação evolutiva no projeto de circuitos eletrônicos, foi somente na década 90 que a

então área denominada de *hardware* evolutivo começou a receber mais atenção mostrando a capacidade de obter projetos inovadores a partir dos métodos de computação evolutiva. O *hardware* evolutivo é comumente entendido como uma abordagem que utiliza técnicas de algoritmos evolutivos e outros algoritmos bio-inspirados a fim de projetar, otimizar, adaptar ou reparar automaticamente diversos tipos de *hardware* (VASICEK; SEKANINA, 2014).

Apesar do sucesso obtido pelas técnicas evolucionistas no projeto de circuitos eletrônicos, estas são sujeitas ao problema da escalabilidade (VASICEK; SEKANINA, 2015). Isso ocorre pois o tempo de processamento para encontrar uma boa solução cresce muito com o número de entradas. Este custo computacional está ligado ao tempo necessário para a avaliação de um indivíduo e que cresce exponencialmente com o número de entradas quando circuitos combinacionais são considerados. Além disso, nos últimos anos, a capacidade de evolução de circuitos digitais tem sido questionada pelo fato de existir uma grande distância entre os circuitos que são evoluídos na área acadêmica e as reais necessidades da indústria (VASICEK, 2018).

Atualmente, a Programação Genética Cartesiana (CGP) (MILLER, 2011) é apontada como um dos métodos mais eficientes para a evolução e otimização de circuitos digitais (VASICEK; SEKANINA, 2015). Apesar do multiplicador 4×4-*bits* ter sido o circuito mais complexo já evoluído pela CGP tradicional (VASICEK; SEKANINA, 2015), a CGP é capaz de otimizar circuitos com centenas de entradas (VASICEK, 2015). Além disso, uma variante da CGP que utiliza diagramas de decisão binária (VASICEK; SEKANINA, 2014) foi capaz de obter circuitos *benchmark* de 28 entradas, pois utiliza uma abordagem capaz de contornar o problema da escalabilidade.

Tendo em vista essas questões relacionadas às dificuldades de evolução de circuitos digitais, uma das contribuições deste trabalho está justamente focada em ampliar a capacidade de exploração do espaço de busca da CGP a fim de facilitar a obtenção de circuitos lógicos combinacionais. Pretende-se, portanto, propor e apresentar novas técnicas que auxiliem no projeto de circuitos lógicos combinacionais, particularmente na obtenção de circuitos totalmente funcionais. Não resolve-se aqui o problema da escalabilidade mas as técnicas são adequadas para que o problema da escalabilidade seja reduzido.

1.1 OBJETIVOS

O objetivo geral é analisar por meio de experimentos computacionais a CGP e propor operadores de movimento (recombinação e mutação) que auxiliam no projeto de circuitos lógicos combinacionais.

Para alcançar esse objetivo geral, os seguintes objetivos secundários também são almejados:

- apresentar um estudo e investigar as dificuldades da CGP na obtenção de soluções factíveis (aquelas soluções cujos circuitos obtidos atendem integralmente a tabela verdade);
- propor e apresentar um operador de recombinação para o projeto de circuitos lógicos combinacionais;
- propor e apresentar um operador de mutação guiada para o projeto de circuitos lógicos combinacionais;
- modificar a Estratégia Evolutiva (ES) comumente usada pela CGP passando a operar com duas estratégias de mutação;
- incluir multiplexadores como elemento lógico no conjunto de funções da CGP;
- propor um método de evolução de circuitos lógicos em três etapas através do acoplamento de um multiplexador em cada saída do circuito;
- validar os métodos propostos com experimentos computacionais que abrangem circuitos comumente utilizados na literatura e circuitos *benchmarks*.

1.1.1 ORGANIZAÇÃO DO TEXTO

O restante deste trabalho está dividido como segue. O Capítulo 2 apresenta os fundamentos necessários para o entender o funcionamento e o projeto de circuitos lógicos combinacionais. O Capítulo 3 apresenta fundamentos de computação evolucionista incluindo algoritmos genéticos, programação genética e apresenta detalhes sobre a programação genética cartesiana além de uma revisão bibliográfica sobre o projeto evolutivo de circuitos digitais. O Capítulo 4 apresenta uma investigação sobre as dificuldades da CGP através

do levantamento de problemas fictícios e propõe métodos de mitigação. O Capítulo 5 apresenta os métodos propostos. O Capítulo 6 apresenta os experimentos computacionais e um estudo de ajuste de parâmetros do operador de recombinação proposto. Por fim, o Capítulo 7 apresenta as conclusões e os possíveis trabalhos futuros.

2 FUNDAMENTOS DE CIRCUITOS DIGITAIS

Neste capítulo são apresentados os conceitos fundamentais sobre circuitos digitais incluindo a diferenciação entre sistemas analógicos e digitais, fundamentação sobre circuitos lógicos tais como tabelas verdade, portas lógicas, métodos de simplificação de circuitos lógicos e questões sobre o projeto de circuitos lógicos combinacionais.

2.1 SISTEMAS ANALÓGICOS E DIGITAIS

Os circuitos eletrônicos podem ser divididos em duas grandes categorias: analógicos e digitais. A principal diferença entre elas consiste no fato de que na eletrônica digital as grandezas envolvidas são valores discretos enquanto na eletrônica analógica, contínuos. A Figura 2.1 apresenta uma representação dos sinais analógicos e digitais.

Um sistema digital é uma combinação de dispositivos projetados para manipular informação lógica ou quantidades físicas representadas no formato digital (TOCCI *et al.*, 2003). Já um sistema analógico contém dispositivos que manipulam quantidades físicas representadas na forma analógica que podem variar ao longo de uma faixa contínua de valores (TOCCI *et al.*, 2003).

Dentre as principais vantagens dos sistemas digitais pode-se destacar o projeto mais fácil devido à existência de apenas dois estados (*HIGH* e *LOW*), armazenam-se informações de maneira mais fácil do que os analógicos, possui facilidade para manter a precisão e exatidão em todo o sistema pois o sinal digital não se deteriora ao ser processado, e possuem menos interferência por ruídos (como flutuações espúrias de tensão). Além disso, os circuitos integrados digitais podem ser fabricados com mais dispositivos internos, pois, em muitos casos, os analógicos necessitam de capacitores de alto valor, resistores de precisão, indutores e transformadores que não permitem alto grau de integração (TOCCI *et al.*, 2003). Entretanto, as técnicas digitais possuem duas grandes limitações: o mundo real é analógico e o processamento de sinais digitais é demorado.

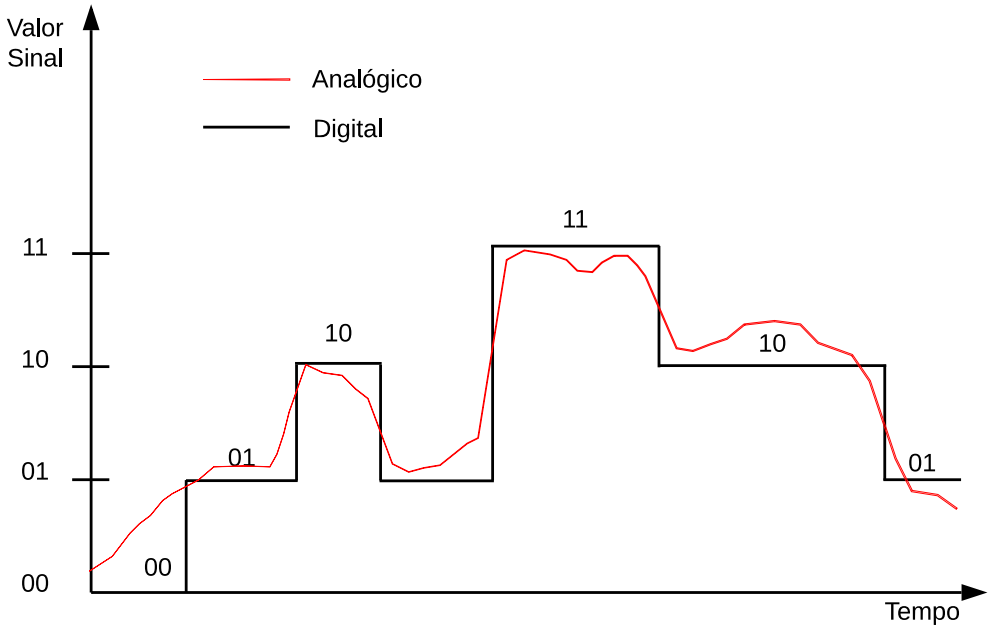


Figura 2.1: Sinal analógico × sinal digital. Figura adaptada de (MANFRINI *et al.*, 2017).

2.2 CIRCUITOS LÓGICOS

O modo como um circuito digital responde a uma entrada é denominado lógica do circuito (TOCCI *et al.*, 2003). Cada circuito digital obedece a um conjunto de regras lógicas e por isso pode ser denominado circuito lógico, que pode ser escrito na forma de uma expressão lógica cuja função é descrever o relacionamento entre suas entradas e saídas utilizando a álgebra booleana. A álgebra booleana difere da álgebra convencional no fato de que na álgebra booleana as constantes e variáveis podem assumir somente dois valores, 0 ou 1, e que representam os níveis de tensão que são aplicados. Por isso as variáveis booleanas não representam efetivamente números, mas o nível lógico no qual as variáveis se encontram.

Os circuitos lógicos podem ser classificados de duas maneiras distintas no que diz respeito ao seu comportamento em relação às suas entradas e suas saídas (EIBEN *et al.*, 2003):

- Circuitos Lógicos Combinacionais: são aqueles cujas saídas, em qualquer instante de tempo, dependem somente das combinações de suas entradas;
- Circuitos Lógicos Sequenciais: são aqueles cujas saídas, além das combinações de suas entradas, dependem do estado anterior de elementos do próprio circuito. Os circuitos lógicos sequenciais podem ser entendidos também como circuitos lógicos

combinacionais adicionados de elementos de memória, tais como *latches* e *flip-flops*.

Neste trabalho ataca-se o projeto de circuitos lógicos combinacionais no qual a CGP ainda enfrenta dificuldades devido ao problema da escalabilidade.

2.2.1 TABELAS VERDADE

Uma tabela-verdade é uma descrição das saídas de um circuito lógico levando-se em consideração todas as possíveis combinações de níveis lógicos presentes nas entradas do circuito (TOCCI *et al.*, 2003). Por sua vez, as entradas de um circuito são consideradas variáveis lógicas cujos níveis lógicos determinam, a qualquer momento, os níveis das saídas (TOCCI *et al.*, 2003). Qualquer circuito lógico pode ser expresso através das três operações lógicas básicas da álgebra booleana: OR (soma), AND (produto) e NOT (negação). Os detalhes sobre os funcionamentos das portas lógicas e as operações que realizam são apresentados na Seção 2.2.2 (página 28).

Uma técnica comumente utilizada para extrair a expressão lógica representada por uma tabela verdade é através da soma de produtos. Esta técnica multiplica as variáveis de entrada (AND entre as entradas) cuja saída está em nível lógico 1 e soma todos esses termos (OR em todos os termos). A Tabela 2.1 exemplifica uma tabela verdade de 4 entradas ($ABCD$) e uma saída (F), e sua expressão lógica correspondente é apresentada na Equação 2.1.

$$F = \overline{A}\ \overline{B}\ C\ \overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}D + ABCD \qquad (2.1)$$

2.2.2 PORTAS LÓGICAS

As portas lógicas implementam as funções da álgebra booleana. Além das funções básicas citadas anteriormente (AND, OR e NOT), pode-se combiná-las de tal maneira a obter novas funcionalidades. As funções implementadas pelas portas lógicas são apresentadas em seguida e a representação das mesmas com suas respectivas tabelas verdade e expressão booleana são apresentadas na Figura 2.2.

- **AND**: a função retorna valor 1 se, e somente se, todas as variáveis de entrada forem 1 caso contrário, retorna 0. Algebricamente denotada por: F = A · B.

Tabela 2.1: Tabela verdade de 4 entradas e uma saída.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

- **OR**: A função retorna valor 0 se, e somente se, todas as variáveis de entrada forem 0 caso contrário, retorna 1. Algebricamente denotada por: F = A + B.

- **NOT**: A função retorna o valor complementar à variável de entrada, isto é: F(0) = 1 e F(1) = 0. Algebricamente denotada por: F = $\overline{A}$.

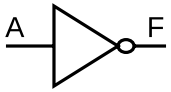
- **NAND**: É o inverso da AND, isto é, retorna 1 se, e somente se, ao menos uma das variáveis de entrada for 0 caso contrário, retorna 0. Algebricamente denotada por: F = $(\overline{A \cdot B})$.

- **NOR**: É o inverso da OR, isto é, retorna 0 se, e somente se, ao menos uma das variáveis de entrada for 1 caso contrário, retorna 1. Algebricamente denotada por: F = $(\overline{A + B})$.

- **XOR**: A função possui valor binário 1 quando as entradas possuem valores diferentes e 0 caso contrário. Algebricamente denotada por: A $\oplus$ B.

- **XNOR**: É o inverso da XOR, isto é, a função possui valor binário 1 quando as entradas possuem valores iguais e 0 caso contrário. Algebricamente denotada por: A $\odot$ B.

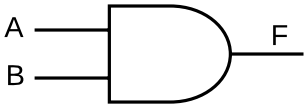
NOT



A	F
0	1
1	0

$F = \overline{A}$

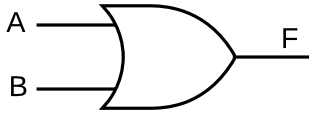
AND



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

$F = A \cdot B$

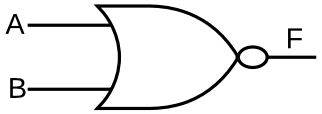
OR



A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

$F = A + B$

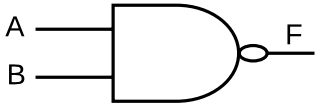
NOR



A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

$F = \overline{A + B}$

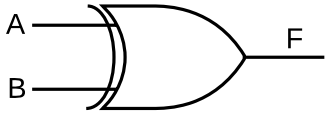
NAND



A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

$F = \overline{A \cdot B}$

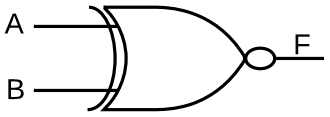
XOR



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

$F = A \oplus B$

XNOR



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

$F = A \odot B$

Figura 2.2: Simbologia, tabelas verdade e expressões das portas lógicas.

2.2.3 MULTIPLEXADORES

O multiplexador digital, ou seletor de dados, é um circuito lógico que recebe diversos dados digitais de entrada e seleciona um, em determinado instante, para transferi-lo para a saída. O envio do dado de entrada desejado para a saída é controlado pelas entradas de seleção, frequentemente denominadas endereço ou controle (TOCCI *et al.*, 2003). A Figura 2.3 apresenta um multiplexador de 2 entradas e um controle. Para um multiplexador de duas entradas existem duas situações: quando o nível lógico do controle for 0 a saída é comutada para o subcircuito I0 caso contrário, a saída é comutada para o subcircuito I1.

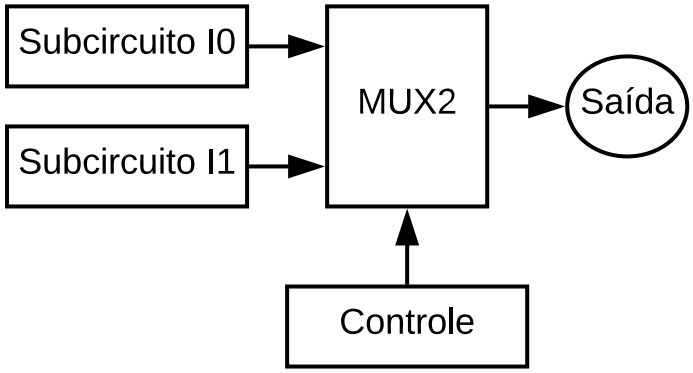


Figura 2.3: Multiplexador de 2 entradas e um controle. Para duas entradas basta 1 *bit* para o controle (seleção).

2.2.4 SIMPLIFICAÇÃO DE CIRCUITOS LÓGICOS

Existem diversas simplificações que podem ser realizadas em circuitos lógicos no que diz respeito à minimização do número de elementos lógicos necessários para implementar sua expressão. Entre elas, existem as simplificações algébricas e os teoremas da álgebra de Boole, os mapas de Karnaugh e algumas ferramentas computacionais. Nestas, o foco é dado ao ESPRESSO (BRAYTON *et al.*, 1984) pois é uma ferramenta amplamente utilizada para a otimização dos circuitos.

2.2.4.1 Mapa de Veitch-Karnaugh

Os mapas de Veitch-Karnaugh (mapas K) utilizam como princípio a identificação visual de grupos de mintermos (soma de produtos) que podem ser simplificados. O mapa K fornece a mesma informação da tabela verdade só que em um formato diferente. Cada linha na tabela verdade corresponde a um quadrado no mapa K. É um método gráfico usado

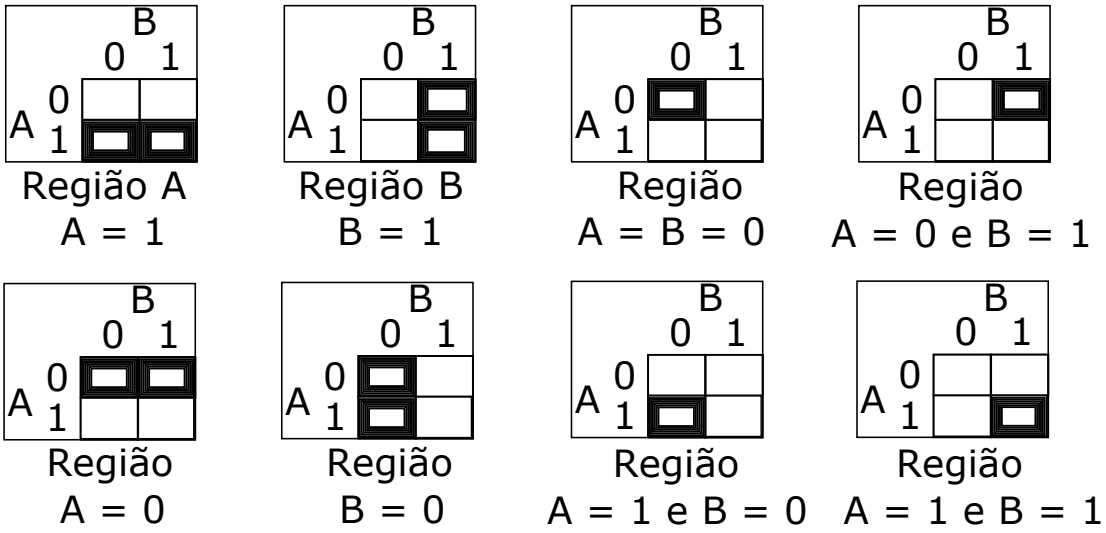


Figura 2.4: Regiões de simplificação de um mapa de Veitch-Karnaugh para duas variáveis.

para simplificar uma equação lógica ou para converter uma tabela-verdade no circuito lógico correspondente de maneira simples e metódica (TOCCI *et al.*, 2003). Os quadrados do mapa K são nomeados de modo que os quadrados adjacentes, tanto horizontalmente quanto verticalmente, difiram apenas em uma variável. Por exemplo, o quadrado superior esquerdo no mapa de duas variáveis apresentado na Figura 2.5 é $\overline{AB}$ (A = B = 0) enquanto o imediatamente à esquerda é o $\overline{A}B$ (A = 0 e B = 1). Considerando duas variáveis, existem 8 regiões de simplificação, conforme apresentado na Figura 2.4.

Um exemplo é apresentado na Tabela 2.2 e na Figura 2.5. Depois de dispor os mintermos nas posições do mapa, exatamente como é mostrado na tabela verdade, deve-se agrupar as regiões que apresentam nível lógico 1 no maior número possível de pares. Caso a combinação das entradas apresente nível lógico 0 na saída, o mesmo valor deve ser colocado no quadrado do mapa K que representa tal combinação. O mesmo é válido para o nível lógico 1 na saída. Além disso, existe a condição de irrelevância (popularmente chamada de *don't care*). Esta situação indica que não importa o nível lógico da saída para uma determinada combinação de entradas. Com isso pode-se utilizar a lacuna que representa essa combinação no mapa K para auxiliar a simplificação, comumente denotada por "X".

Um exemplo de uso do mapa K é apresentado na Figura 2.5 que representa a Tabela 2.2. A partir da Figura 2.5 nota-se que o Par 1 (vertical) alterna seu valor na variável A, entre A e seu complemento (o nível lógico oposto ao atual) e, portanto, pode ser eliminada, resultando em F = B. Já o Par 2 (horizontal) mantém o nível lógico de A e alterna seu

Tabela 2.2: Tabela verdade de exemplo para simplificação.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

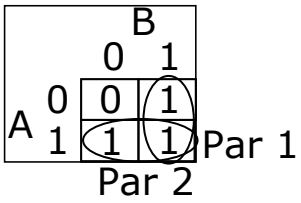


Figura 2.5: Mapa de Veitch-Karnaugh do exemplo da Tabela 2.2.

valor na variável B, entre o valor B e seu complemento, logo, F = A. Como os mapas de Veitch-Karnaugh atuam em mintermos, soma-se as simplificações dos pares 1 e 2 obtendo assim: F = A + B.

A partir de 5 variáveis no mapa o processo de simplificação fica mais complexo de ser montado. Por isso, o mapa de Veitch-Karnaugh não é muito utilizado para simplificações de circuitos com muitas entradas. Para auxiliar neste quesito, ferramentas computacionais são utilizadas.

2.2.4.2 Ferramentas Computacionais

Existem diversas ferramentas computacionais que podem ser utilizadas para a simplificação de circuitos lógicos. Dentre elas, existe o método de Quine Mc-Cluskey (MCCLUSKEY, 1956), também chamado de método tabular que busca pelos agrupamentos em diversos níveis de acordo com o número de variáveis. Entretanto, este método possui alto custo computacional que aumenta exponencialmente com o número de variáveis (MANFRINI *et al.*, 2017).

A ferramenta computacional mais utilizada é o ESPRESSO (BRAYTON *et al.*, 1984). Este algoritmo de simplificação é baseado em uma heurística de minimização lógica. Ao invés de expandir uma função lógica em seus mintermos, o ESPRESSO manipula cubos, representando os termos de produto através de suas representações internas baseadas nos conjuntos de mintermos e *don't cares* iterativamente. Apesar do resultado da minimização não garantir o mínimo global, na prática ele é uma boa aproximação enquanto a solução é

sempre livre de redundância quando são consideradas apenas as operações básicas (AND, OR e NOT). Na prática, o ESPRESSO não é capaz de lidar com simplificações entre as demais portas lógicas. Comparado a outros métodos, o ESPRESSO é essencialmente mais eficiente reduzindo o uso de memória e o tempo de computação em várias ordens de magnitude (BRAYTON *et al.*, 1984). Além disso, não existe restrição ao número de variáveis, funções de saída e termos de produto de um bloco de função combinacional.

A entrada para o ESPRESSO é uma tabela verdade da funcionalidade desejada. O resultado é uma tabela reduzida descrevendo a função de acordo com as opções selecionadas. Por padrão, os termos de produto serão compartilhados o máximo possível pelas diversas funções de saída mas o programa pode ser instruído a lidar com as expressões de cada saída separadamente. Isso permite uma implementação eficiente em *arrays* lógicos de dois níveis como PLAs (*Programmable Logic Array*).

O algoritmo do ESPRESSO provou-se tão bem sucedido que é incorporado como função de minimização lógica padrão em diversas ferramentas de síntese. Para implementar uma função de lógica multi-nível, o resultado da minimização é otimizada através de fatoração e mapeada nas células lógicas básicas disponíveis na tecnologia alvo, tal como é feito com FPGAs (*Field Programmable Gate Array*).

2.3 PROJETO DE CIRCUITOS LÓGICOS COMBINACIONAIS

Os circuitos lógicos combinacionais (CLCs) são aqueles cuja função de saída representa uma combinação dos níveis lógicos de sua entrada. Para cada uma destas combinações existe um e somente um nível lógico correspondente na saída cuja origem é aquela e somente aquela combinação das entradas. Estes circuitos são utilizados em diversas aplicações, principalmente em circuitos aritméticos tais como somadores e multiplicadores.

Os CLCs são expressos comumente em uma tabela verdade através da soma de produtos. Um exemplo é apresentado na Tabela 2.3 onde um circuito de 4 entradas (A, B, C e D) uma saída (F) é utilizado como base e sua expressão lógica é apresentada na Equação 2.2. A coluna mintermo apresenta os mintermos, que representam as entradas cuja saída apresenta nível lógico 1 e que serão utilizados para a representação sob a forma de soma de produtos.

Tabela 2.3: Tabela verdade exemplo para a geração de um CLC possuindo 4 entradas e uma saída.

A	B	C	D	F	Mintermo
0	0	0	0	1	$\overline{A}\overline{B}\overline{C}\overline{D}$
0	0	0	1	1	$\overline{A}\overline{B}\overline{C}D$
0	0	1	0	0	-
0	0	1	1	1	$\overline{A}\overline{B}CD$
0	1	0	0	1	$\overline{A}B\overline{C}\overline{D}$
0	1	0	1	1	$\overline{A}B\overline{C}D$
0	1	1	0	0	-
0	1	1	1	1	$\overline{A}BCD$
1	0	0	0	0	-
1	0	0	1	0	-
1	0	1	0	0	-
1	0	1	1	1	$A\overline{B}CD$
1	1	0	0	1	$AB\overline{C}\overline{D}$
1	1	0	1	0	-
1	1	1	0	1	$ABC\overline{D}$
1	1	1	1	0	-

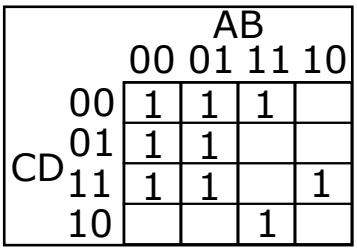


Figura 2.6: Mapa de Veitch-Karnaugh para o exemplo da Tabela 2.3

$$\begin{aligned} F = \ & \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \\ & \overline{A}BCD + A\overline{B}CD + AB\overline{C}\,\overline{D} + ABC\overline{D} \end{aligned} \tag{2.2}$$

Com a tabela que descreve o funcionamento do circuito, pode-se simplificá-lo utilizando o método mais conveniente. Utilizando um mapa de Veitch-Karnaugh obtém-se o resultado apresentado na Figura 2.6, cuja expressão lógica resultante é apresentada na Equação 2.3.

$$F = \overline{A}\,\overline{C} + \overline{A}D + AB\overline{D} + \overline{B}CD \tag{2.3}$$

Caso exista a possibilidade de simplificar ainda mais uma expressão através da álgebra de boole, este passo deve ser feito logo após a obtenção da simplificação do mapa de Veitch-

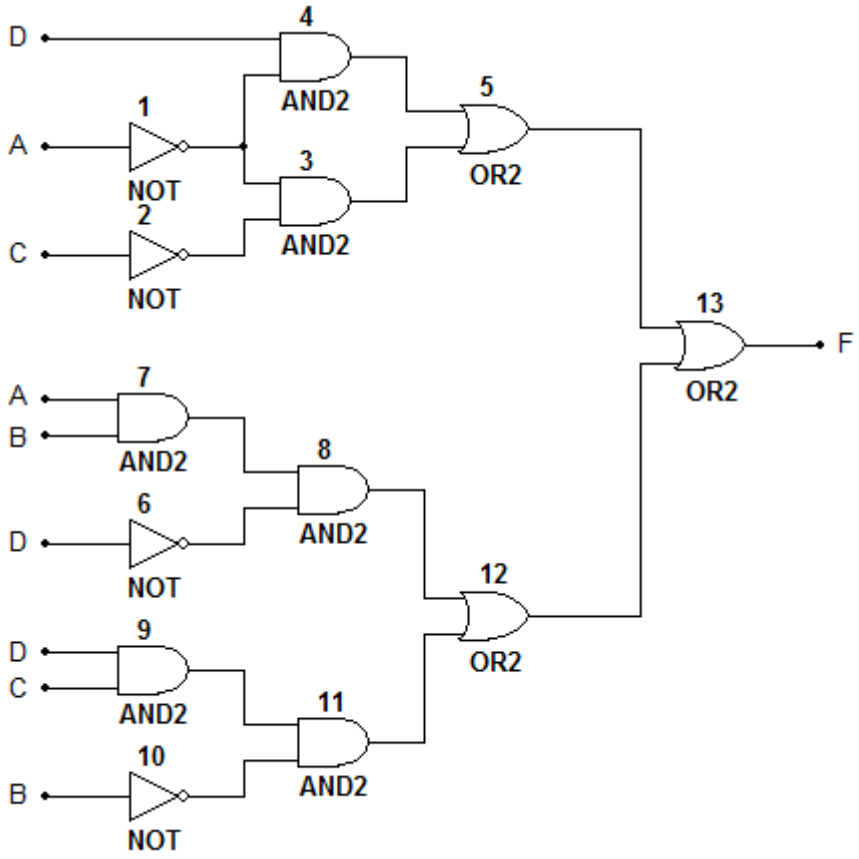


Figura 2.7: Implementação do circuito exemplo cuja função simplificada é apresentada na Equação 2.3 utilizando-se portas lógicas de duas entradas.

Karnaugh. A partir da equação final podemos de fato implementar o circuito. O circuito exemplo é apresentado na Figura 2.7, onde apenas portas lógicas com duas entradas foram utilizadas.

3 COMPUTAÇÃO EVOLUCIONISTA

Neste capítulo os conceitos da computação evolucionista são abordados com foco na programação genética cartesiana, pois é a metaheurística utilizada para a evolução de circuitos digitais neste trabalho. A inspiração biológica por trás dos algoritmos evolutivos é apresentada seguida pelo funcionamento e pelos principais componentes desses algoritmos, incluindo a representação, a função de aptidão, população, mecanismos de seleção de pais, os operadores de variação (mutação e recombinação), mecanismo de seleção de sobrevivência e os métodos de inicialização e critérios de parada dos algoritmos evolutivos. Além disso, os fundamentos sobre as estratégias evolutivas que consistem no mecanismo de busca da CGP são apresentados. Após, são apresentadas as principais características da programação genética e por fim é apresentada a programação genética cartesiana. São mostradas as principais características da CGP, incluindo seus parâmetros de representação, operadores de mutação e de recombinação e é abordada a questão da neutralidade, redundância e deriva gênica neutra que se mostraram importantes para o processo evolutivo da CGP.

3.1 INSPIRAÇÃO BIOLÓGICA

A seleção natural, como princípio central da evolução, foi formulada inicialmente por Charles Darwin (DARWIN, 1969) tempos após a descoberta dos mecanismos genéticos (MICHALEWICZ, 1996). No início, estas duas teorias eram independentes. Darwin, após concluir que os mais aptos tendem a sobreviver e que assim geram mais descendentes, elaborou a hipótese da fusão de herança, isto é, as características parentais se misturam no organismo da prole (MICHALEWICZ, 1996). A teoria de seleção natural de Darwin levantou várias objeções com o avanço das descobertas da genética, primeiro por F. Jenkins: cruzamentos em pequenos intervalos de tempo geram um nivelamento em quaisquer distinções hereditárias e não há seleção em populações homogêneas, conhecido como Pesadelos de Jenkins. Em 1865, quando G. Mendel descobriu os princípios básicos da transferência de fatores hereditários de pais para filhos, que mostrou a natureza discreta desses fatores, os Pesadelos de Jenkins puderam ser explicados, já que por causa dessa discrição não há dissolução de distinções hereditárias. As leis mendelianas tornaram-se conhecidas pela

comunidade científica após eles terem sidos redescobertos independentemente em 1900 por H. de Vries, K. Correns e K. von Tschermak (MICHALEWICZ, 1996). A genética foi totalmente desenvolvida por T. Morgan e seus colaboradores, que provaram experimentalmente que cromossomos são os principais portadores de informação hereditária e que os genes, que apresentam fatores hereditários, estão alinhados nos cromossomos. Mais tarde, fatos experimentais acumulados mostraram leis para serem válidas para todos os organismos sexualmente reprodutores.

No entanto, as leis de Mendel, mesmo depois de terem sido redescobertas, a teoria da seleção de Darwin permaneceu independente da genética teórica que estava surgindo, desvinculada de seus conceitos. Além disso, elas eram opostas entre si. Na década de 1920 foi provado que a genética de Mendel e a teoria da seleção natural de Darwin não são conflituosas e que quando combinadas, culminam na teoria evolutiva moderna (MICHALEWICZ, 1996).

3.2 PROBLEMAS DE OTIMIZAÇÃO

A otimização é um processo de melhoramento iterativo de uma solução com respeito a uma função objetivo específica (GOLDBARG *et al.*, 2017). Esta pode ser classificada como contínua, quando as variáveis assumem valores reais, combinatória, quando as variáveis assumem valores discretos ou mista, quando existem variáveis inteiras e contínuas (GOLDBERG, 1989).

Um problema de otimização é aquele onde se procura determinar os valores extremos de uma função, isto é, o maior ou o menor valor que uma função pode assumir em um dado intervalo. A forma padrão de um problema de otimização é dado por

$$\begin{aligned} &\text{Otimizar} && f(x) \\ &\text{sujeito a} && \\ & && g_i(x) \leq 0,\ i = 1, ..., m \\ & && h_j(x) = 0,\ j = 1, ..., p \end{aligned} \tag{3.1}$$

onde $f(x)$ é a função objetivo para ser maximizada ou minimizada sobre a variável x, $g_i(x) \leq 0$ é chamada de restrições de desigualdade, e $h_j(x) = 0$ é chamada de restrições de igualdade.

O problema de otimização abordado nesta dissertação é descrito na Seção 3.6.1 (página 53).

3.3 ALGORITMOS EVOLUTIVOS

Existe uma grande quantidade de algoritmos que são considerados algoritmos evolutivos. Essa classificação se dá devido ao fato de que a ideia por trás de todos eles é essencialmente a mesma (EIBEN *et al.*, 2003): dada uma população de indivíduos, a pressão do ambiente causa seleção natural (sobrevivência dos mais aptos), que causa um aumento na aptidão da população. Dada uma função de qualidade a ser maximizada, pode-se criar aleatoriamente um conjunto de soluções candidatas e aplicar a função de qualidade como uma medição abstrata de suas aptidões. Com base nesta aptidão, alguns dos melhores candidatos são escolhidos para semear a próxima geração através da aplicação de operadores específicos tais como recombinação e mutação.

A Figura 3.1 apresenta o ciclo de funcionamento de um algoritmo evolutivo genérico. O procedimento de inicialização gera a população inicial que é avaliada em seguida. Caso essa população já satisfaça os critérios estabelecidos para factibilidade o processo é finalizado. Caso contrário, seleciona-se os parentais que receberão os operadores de variação: recombinação e mutação. A recombinação é um operador aplicado a dois ou mais candidatos, denominados parentais, cujo resultado é um ou mais novos candidatos, denominados filhos. Já a mutação é aplicada a um candidato e os resultados geram um novo candidato. Após a aplicação dos operadores, os novos indivíduos são avaliados. Executando sucessivas recombinações e mutações gera-se um conjunto de novos candidatos (descendência) que, baseada em sua aptidão, competem com os demais candidatos a fim de ocupar um lugar na geração seguinte. Esses novos indivíduos são analisados a fim de verificar se o critério de parada foi atingido. Em caso afirmativo, o processo é finalizado. Caso contrário, o processo é repetido até que o critério de parada seja satisfeito ou até que o limite computacional seja alcançado.

Neste processo iterativo existem dois pontos que formam a base dos sistemas evolutivos (EIBEN *et al.*, 2003):

- operadores de variação: Recombinação e mutação criam a diversidade e, consequentemente, são responsáveis pelo aparecimento de novas características na população;

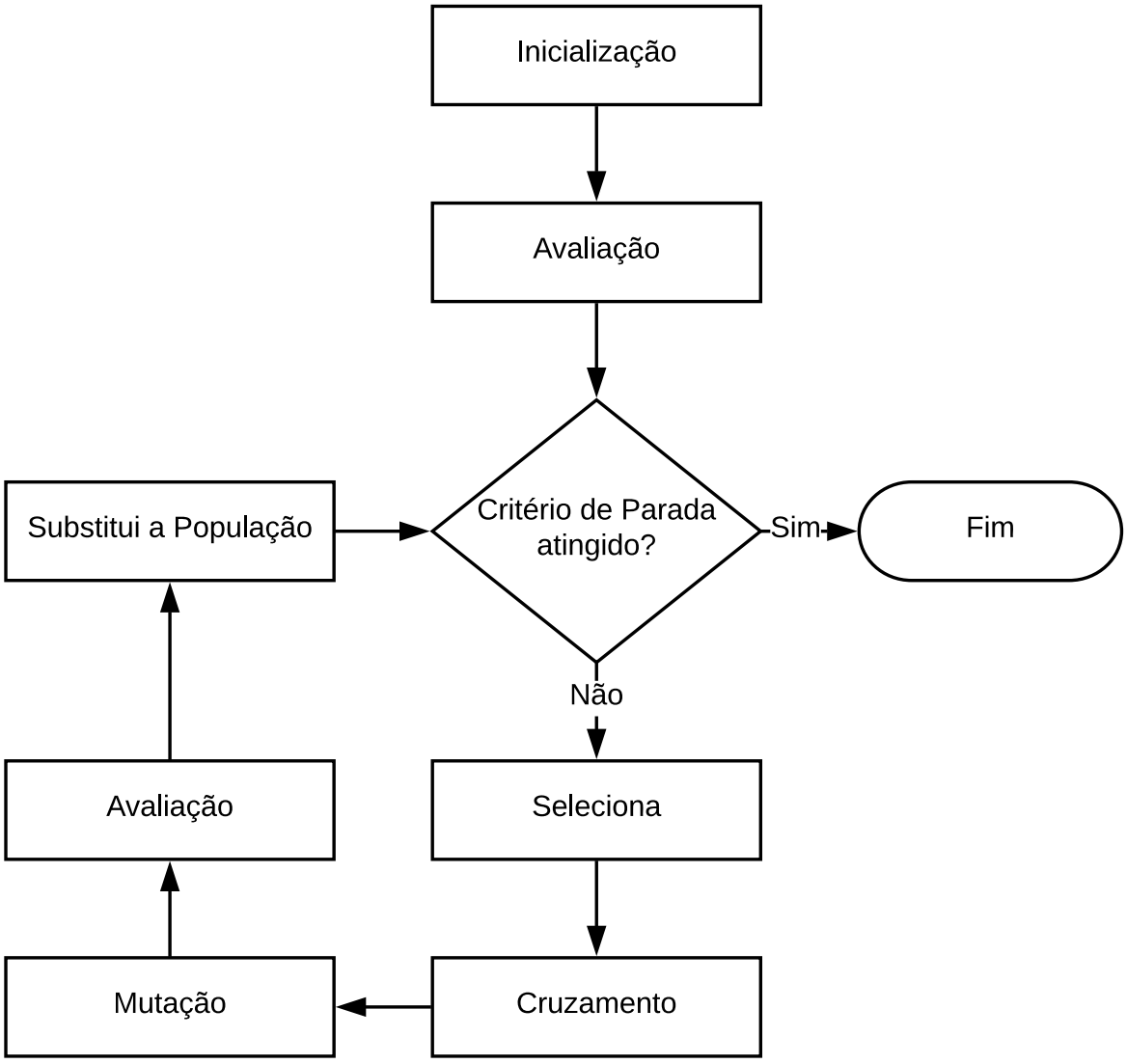


Figura 3.1: Fluxograma de funcionamento dos algoritmos evolutivos.

- a seleção age como uma forma de impulsionar o aumento da qualidade média da população.

A evolução é comumente entendida como um processo de adaptação (EIBEN *et al.*, 2003) e nesta perspectiva, a aptidão é vista como uma métrica do nível de adaptação do indivíduo no ambiente (espaço de busca).

Muitos componentes do processo evolutivo são estocásticos. Durante a seleção, os indivíduos mais aptos possuem maior chance de serem selecionados do que os menos aptos. Até mesmo o pior indivíduo pode ter a chance de se tornar progenitor ou sobreviver. Para recombinação de indivíduos, a escolha de quais partes serão recombinadas também é aleatória. Similarmente na mutação, as partes que serão modificadas de uma solução candidata e as novas componentes que as substituirão são selecionadas aleatoriamente.

O processo de busca é guiado através dos operadores de variação e seleção. Os Algoritmos Evolutivos (AEs) possuem características específicas, das quais se destacam(EIBEN *et al.*, 2003):

- AEs são baseados em população (maioria);
- AEs utilizam, em sua maioria, recombinação para combinar informações de soluções candidatas em uma nova solução candidata; e
- AEs são estocásticos.

Além disso, no que tange à representação, tipicamente os candidatos são representados por uma estrutura de dados que representa uma solução sob a forma de uma *string* com um alfabeto finito nos Algoritmos Genéticos (AGs), vetores valorados com números reais em Estratégias Evolutivas (EEs), máquinas de estado finitos na Programação Evolucionária clássica (PEs) e árvores em Programação Genética (PGs). Tecnicamente uma dada representação será preferível sobre outras se ela se encaixa melhor no problema, isto é, se faz a codificação de uma solução candidata de maneira mais fácil ou mais natural (EIBEN *et al.*, 2003).

3.3.1 COMPONENTES DE ALGORITMOS EVOLUTIVOS

AEs possuem componentes, procedimentos ou operadores que devem ser especificados a fim de definir o tipo particular de AE. Além disso, para obter um algoritmo executável,

o procedimento de inicialização e a condição de parada devem também ser definidos. Os componentes mais importantes foram apresentados no fluxograma da Figura 3.1 e são explorados a seguir. O conteúdo apresentado aqui é um resumo dos principais pontos detalhados em (EIBEN *et al.*, 2003).

3.3.1.1 Representação

O primeiro passo para definir um AE é criar uma relação entre o mundo real e o mundo do AE, isto é, criar uma ponte entre o contexto original do problema e o espaço de soluções do algoritmo de busca onde a evolução ocorrerá (MICHALEWICZ, 1996; EIBEN *et al.*, 2003). Objetos formando soluções possíveis no contexto do problema são referidos como fenótipos enquanto sua codificação, isto é, os indivíduos do AE propriamente ditos, são chamados de genótipos (EIBEN *et al.*, 2003). No lado do contexto original do problema, solução candidata, fenótipo e indivíduo são terminologias comumente utilizadas para denotar pontos do espaço de soluções possíveis. Esse espaço é denominado espaço fenotípico. No lado dos AEs, genótipo, cromossomo e indivíduo podem ser utilizados para locais no espaço onde a busca evolutiva toma lugar. Esse espaço é chamado de espaço genotípico.

O cromossomo é responsável por armazenar os genes, que podem ser interpretados como características elementares da solução. Cada gene é alocado em uma posição, denominada *locus* e podem assumir valores diferentes, denominados alelos (MANFRINI *et al.*, 2017). Inicialmente, Holland (1975) propôs uma representação binária do cromossomo, isto é, os alelos só podem assumir os valores 0 e 1. A Figura 3.2 ilustra um cromossomo binário e seus componentes.

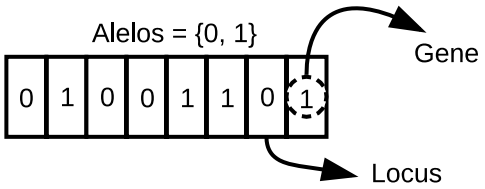


Figura 3.2: Representação de um cromossomo binário. Os alelos representam os valores que cada gene pode assumir. Cada gene ocupa uma posição denominada *locus*.

3.3.1.2 Função de Aptidão

O papel da função de aptidão é medir o quão boa é esta solução. Mais precisamente esta função representa o quão apto um indivíduo é ao ambiente no qual está submetido.

Traduz quantitativamente as características qualitativas da solução candidata. Quanto maior é o valor de aptidão atribuído a um indivíduo, mais apto ele é e, portanto, melhor. Neste trabalho a função de aptidão adotada para a CGP consiste na quantidade de acertos em relação à tabela verdade. Para cada correspondência correta soma-se um à função de aptidão cujo limite superior é dado por: $A_{max} = 2^{ni} \times no$, onde A_{max} é o valor máximo de aptidão, ni é o número de entradas e no o número de saídas.

3.3.1.3 População

O papel da população é manter uma amostra de possíveis soluções. Além disso, uma população é um multiconjunto de genótipos.

Diferentemente dos operadores de variação que agem em um ou dois pais, os operadores de seleção (seleção de pais e seleção de sobrevivência) trabalham no nível de população. Normalmente, o tamanho da população em AEs é constante.

3.3.1.4 Mecanismo de Seleção de Pais

O papel da seleção de pais é escolher indivíduos com base em sua qualidade. Mais especificamente, os mecanismos de seleção de pais tendem a permitir que os melhores indivíduos se tornem pais na geração seguinte (EIBEN *et al.*, 2003). Um indivíduo é dito pai se foi selecionado para ser operado a fim de criar descendência. Juntamente com os mecanismos de seleção de sobrevivência, a seleção de pai é responsável por promover aumentos da qualidade. A seleção de pais é tipicamente probabilística, o que significa dizer que indivíduos de alta qualidade possuem maior chance de se tornarem pais do que aqueles com baixa qualidade. Contudo, indivíduos de baixa qualidade têm chance de serem escolhidos.

O algoritmo genético clássico utiliza um esquema de seleção de indivíduos para a próxima geração chamado roleta (MICHALEWICZ, 1996). Entretanto, outros mecanismos de seleção amplamente utilizados são os de classificação e torneio. A seguir são apresentadas as principais características destes mecanismos de seleção.

Seleção por Roleta A seleção por roleta atribui a cada indivíduo da população uma probabilidade de passar para a próxima geração. Este valor é proporcional à aptidão do indivíduo em relação ao somatório das aptidões de todos os indivíduos da população. Essa

probabilidade é calculada como:

$$P_{sel} = \frac{f}{\sum_{i=1}^{npop} f_i} \tag{3.2}$$

onde, P_{sel} é a probabilidade de seleção de um determinado indivíduo, f é a aptidão do indivíduo que se deseja determinar o valor de P_{sel}, f_i é a aptidão do indivíduo i e n_{pop} é o número de indivíduos da população.

Uma característica da seleção por roleta consiste no fato de ser probabilístico. Neste caso, não existe garantia de que o melhor indivíduo da população seja selecionado para a próxima geração, apesar de ele ter uma alta probabilidade de ser selecionado.

Seleção por Classificação A seleção por classificação é um método que foi inspirado pelas desvantagens observadas do método de seleção proporcional à aptidão.

Este método de seleção preserva uma pressão de seleção constante, classificando a população com base na aptidão e, em seguida, alocando probabilidades de seleção aos indivíduos de acordo com sua classificação, ao invés de utilizar os valores de aptidão reais.

O mapeamento do número de classificação para a probabilidade de seleção é arbitrário e pode ser feito de várias maneiras, por exemplo, linear ou exponencialmente decrescente, mantendo a condição de que a soma sobre a população das probabilidades deve ser unitária.

A fórmula usual para calcular a probabilidade de seleção para esquemas de classificação linear é parametrizada por um valor s (1,0 < s ≤ 2,0) (EIBEN *et al.*, 2003). No caso de um GA geracional, onde $\mu = \lambda$, esse é o número esperado de descendentes alocados para o indivíduo mais apto. Se este indivíduo tem classificação μ e o pior tem rank 1, então a probabilidade de seleção (P) para um indivíduo de classificação i é:

$$P(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \tag{3.3}$$

A Tabela 3.1 apresenta um exemplo de como as probabilidades de seleção diferem para uma população de três indivíduos diferentes com seleção proporcional e baseada em classificação com diferentes valores de s.

Seleção por Torneio No método de seleção por torneio uma determinada quantidade de indivíduos é escolhida aleatoriamente para formar uma sub-população temporária.

Tabela 3.1: Seleção proporcional à aptidão (SPA) versus Seleção por Classificação Linear (SCL).

Individuo	Aptidão	*Rank*	P_{SPA}	P_{SCL} (s = 2)	P_{SCL} (s = 1,5)
A	1	1	0,1	0	0,167
B	5	2	0,5	0,67	0,5
C	4	2	0,4	0,33	0,33

Deste grupo, o indivíduo mais apto é selecionado para a reprodução. A quantidade de indivíduos escolhida define o tamanho do torneio e controla a pressão de seleção (MANFRINI *et al.*, 2017).

3.3.1.5 Operadores de Variação

O papel dos operadores de variação é criar novos indivíduos a partir dos existentes (EIBEN *et al.*, 2003). Gerar um filho equivale a caminhar para um novo ponto no espaço de busca. Os operadores de variação são comumente divididos em dois tipos, baseados na sua aridade [1], denominados mutação e recombinação.

Mutação Um operador de variação unário é comumente chamado de mutação. Quando aplicado gera-se um novo indivíduo denominado filho ou descendência. É importante ressaltar que um operador unário arbitrário não é necessariamente visto como mutação. Uma heurística específica de problema agindo em um indivíduo pode ser nomeada de mutação por ser unário, contudo, na mutação geral é suposto causar uma aleatoriedade e não um enviesamento. A Figura 3.3 apresenta uma mutação do tipo *bit flip*, isto é, a mutação em um gene para seu complemento.

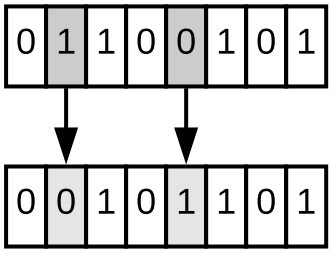


Figura 3.3: Os nós em cinza foram selecionados para receberem a mutação do tipo *bit flip*.

[1] É o número de objetos que o operador toma como entradas

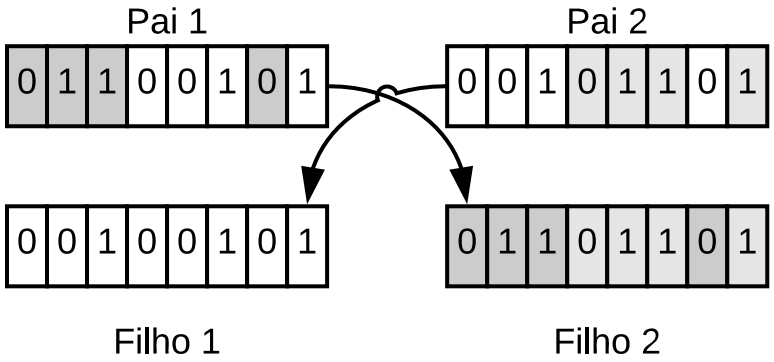


Figura 3.4: Geração de dois filhos a partir de dois pais pelo operador de recombinação de cruzamento em 2 pontos.

Recombinação Um operador de variação binário é chamado de recombinação ou *crossover*. Como o nome indica, tal operador combina informações dos genótipos de dois pais em uma ou duas descendências.

O princípio por trás da recombinação é simples: combinando dois indivíduos com características diferentes, pode-se produzir uma descendência que combina características de ambos (EIBEN *et al.*, 2003). A Figura 3.4 apresenta um cruzamento entre dois indivíduos gerando dois filhos a partir da recombinação de dois pontos.

3.3.1.6 Mecanismo de Seleção de Sobrevivência (*Replacement*)

O papel da seleção de sobrevivência é distinguir os indivíduos baseado em suas qualidades. Similar à seleção dos pais, mas é utilizado em estágios diferentes do ciclo evolutivo. O mecanismo de seleção de sobrevivência é utilizado após a criação de uma descendência. Como o tamanho da população é normalmente constante, deve-se escolher quais indivíduos serão mantidos para a geração seguinte. Essa decisão é geralmente baseada nos valores de aptidão dos indivíduos, favorecendo aqueles com maior qualidade. Oposto à seleção parental, que é tipicamente estocástica, a seleção de sobrevivência é quase sempre determinística (EIBEN *et al.*, 2003).

3.3.1.7 Inicialização e Critérios de Parada

A inicialização dos AEs é um procedimento simples. A primeira população é semeada por indivíduos gerados de maneira aleatória (EIBEN *et al.*, 2003). Entretanto, existem outras maneiras de semear a população inicial. No caso de circuitos digitais é possível iniciar a população com circuitos totalmente funcionais, por exemplo. Essa primeira população,

denominada população inicial, é avaliada e submetida ao processo evolutivo conforme o fluxograma da Figura 3.1 (página 40). O processo é repetido por diversas iterações até que o critério de parada seja atingido.

Por critério de parada entende-se a situação na qual o processo evolutivo deve ser finalizado. Pode-se distinguir dois casos de critérios de parada adequados (EIBEN *et al.*, 2003): obter uma solução com certa qualidade ou atingir uma quantidade de recursos computacionais disponibilizados. Se o problema possui um ótimo global e ele é conhecido então o critério de parada pode ser definido para quando esse valor é alcançado integralmente ou com algum determinado erro. Quando não existe ótimo global ou quando o mesmo é desconhecido, outro critério de parada deve ser definido pois o algoritmo nunca terminará.

Dentre os critérios comuns utilizados na literatura, destacam-se (EIBEN *et al.*, 2003):

- tempo máximo decorrido de processamento;
- número total de avaliações da função de aptidão;
- para um dado período de tempo (por número de gerações ou avaliações da função de aptidão), a melhora da aptidão permanece abaixo de um limiar.

Neste trabalho o critério de parada adotado consiste em atingir o número máximo de avaliações permitidas ou encontrar uma solução factível, isto é, um circuito que atenda completamente à tabela verdade desejada. Nos experimentos onde considera-se a otimização, o critério de parada é atingir o número máximo de avaliações permitidas da função objetivo.

3.4 ESTRATÉGIAS EVOLUTIVAS

As Estratégias Evolutivas (EEs) são outro membro da família dos algoritmos evolutivos. Foram desenvolvidas no início dos anos 60 por Rechenberg e Schwefel. Dentre as principais características deste AE destacam-se (EIBEN *et al.*, 2003):

- EEs são tipicamente usadas para otimização contínua;
- existe uma forte ênfase na mutação para criar descendência;

- a mutação é implementada através da adição de uma perturbação aleatória a partir de uma distribuição gaussiana; e
- parâmetros de mutação são modificados durante a execução do algoritmo.

A seleção de pais nas EEs não é enviesada pelos valores de aptidão (EIBEN *et al.*, 2003). Quando uma recombinação requer um pai, o mesmo é retirado aleatoriamente de uma distribuição uniforme da população de μ indivíduos. Depois de criar λ descendências e calcular suas aptidões, os μ melhores são escolhidos deterministicamente, em duas variantes das EEs: somente a partir da descendência (μ, λ) ou a partir da união dos pais e das descendências $(\mu + \lambda)$. A pressão de seleção nas estratégias evolutivas é muito alta porque λ normalmente é muito superior à μ. Recomenda-se uma proporção de 1/7 entre pais e descendências geradas (EIBEN *et al.*, 2003).

3.5 PROGRAMAÇÃO GENÉTICA

A Programação Genética (PG) é uma técnica de computação evolutiva que resolve problemas automaticamente, sem requerer conhecimento do usuário através da evolução de uma população de programas de computador, geração por geração, transformando as populações em outras, novas e melhores estocasticamente (EIBEN *et al.*, 2003). A PG trabalha com a aleatoriedade e nunca pode-se garantir um resultado. Entretanto essa mesma aleatoriedade da PG pode auxiliá-la a sair de armadilhas em que as técnicas determinísticas podem ficar presas.

Os programas são usualmente expressos por árvores que são modificadas ao longo das gerações por meio dos operadores de variação (recombinação e mutação) (POLI *et al.*, 2008). Um indivíduo é apresentado na Figura 3.5 que implementa a função $\max(x + x, x + 3 * y)$. As variáveis e constantes no programa (x, y e 3) são folhas da árvore e na PG são chamados de terminais. As operações que realizam são os nós internos e são denominados funções. O conjunto de funções e terminais formam o conjunto de primitivas da PG.

3.6 PROGRAMAÇÃO GENÉTICA CARTESIANA

A Programação Genética Cartesiana (*Cartesian Genetic Programming - CGP)* é uma técnica para evolução automática de programas de computador e outras estruturas com-

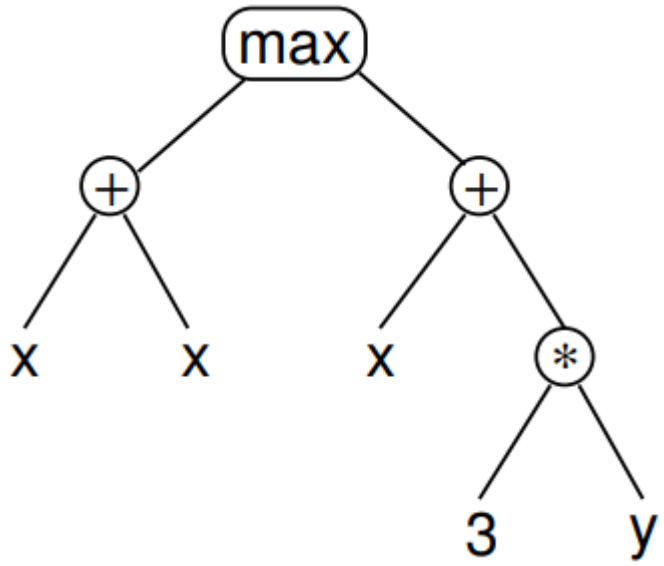


Figura 3.5: Árvore de sintaxe representando max(x+x, x+3y). Figura adaptada de (POLI *et al.*, 2008)

putacionais usando ideias inspiradas na teoria da evolução de Darwin e de seleção natural (MILLER, 2011). A CGP surgiu a partir de um método para evoluir circuitos digitais por Miller em 1997 (MILLER *et al.*, 1997). Contudo, o termo CGP só apareceu em 1999 e foi proposto como uma forma geral de programação genética em 2000 (MILLER, 2011). Atualmente a CGP é aplicada em diversas áreas, tais como controladores robóticos (GARCÍA; COELLO, 2002), redes neurais (GOLDMAN; PUNCH, 2013) e classificadores de imagem (GOLDMAN; PUNCH, 2015).

Essa técnica de programação genética representa os programas como grafos direcionados acíclicos (DAGs). Um dos benefícios de usar uma representação baseada em grafos é que, por definição, permitem implicitamente o reuso de nós, uma vez que um nó pode ser conectado à saída de qualquer outro nó anterior do grafo (MILLER, 2011).

Esses grafos são representados como uma matriz bidimensional de nós. Os genes que formam o genótipo da CGP são inteiros que representam de onde um nó obtém suas entradas (denominados genes conexão), qual operação esse nó realiza nos dados (denominados genes função) e onde a saída desses dados será obtida. Não existe nenhum valor limitante quanto à quantidade de entradas que um nó pode ter. Entretanto para este trabalho adotamos um valor máximo de entradas igual a dois.

As funções realizadas por um nó são definidas em um conjunto de funções e variam de acordo com a aplicação. O conjunto de funções é representado pela letra grega Γ.

Tabela 3.2: Codificação utilizada e suas respectivas funções

Codificação	Função Realizada
100	AND
110	OR
130	XOR
800	NOT
900	WIRE

Em equações matemáticas o conjunto de funções abrange operações aritméticas tais como soma, subtração, multiplicação e divisão. Em notação formal: $\Gamma = \{+, -, *, /\}$. Já no caso de circuitos digitais o conjunto de funções será composto por funções ou portas lógicas: $\Gamma = \{AND, OR, NOT, XOR, WIRE^2\}$. As funções são codificadas como inteiros e suas respectivas correspondências, utilizadas neste trabalho, são apresentadas na Tabela 3.2. Todas as portas lógicas consideradas possuem duas entradas e uma saída, com exceção dos operadores unários NOT e WIRE.

Quando um genótipo é decodificado, alguns nós podem ser ignorados pelo fato de não fazerem parte do caminho ativo da saída. Esses nós que não contribuem diretamente para o fenótipo são denominados inativos.

Além disso, a CGP possui três parâmetros definidos pelo usuário:

- n_c: é o número de colunas da matriz,
- n_r: é o número de linhas da matriz, e
- *levels-back*: determina a conectividade do grafo.

O parâmetro *levels-back*(lb) também é representado por um número inteiro. Esse valor representa em qual(is) coluna(s) um nó pode obter suas entradas. No caso de lb = 1, os nós só podem obter suas entradas de nós que ocupam a coluna imediatamente à esquerda ou diretamente nas entradas do programa. Caso seja definido lb = 2, os nós podem obter suas entradas de nós até duas colunas imediatamente à sua esquerda e assim sucessivamente. Além disso, independentemente do valor atribuído à *levels-back*, qualquer nó pode conectar-se diretamente às entradas do programa.

Variar os parâmetros n_r e n_c resultam em topologias diferentes do grafo. Um caso especial desses três parâmetros ocorre quando o número de linhas é um e *levels-back*

[2]Neste trabalho o termo WIRE é utilizado para representar uma extensão de um ponto do circuito. Esta função lógica apenas repete o sinal de um ponto para outro e não é considerado uma porta lógica na contabilização de portas de um circuito.

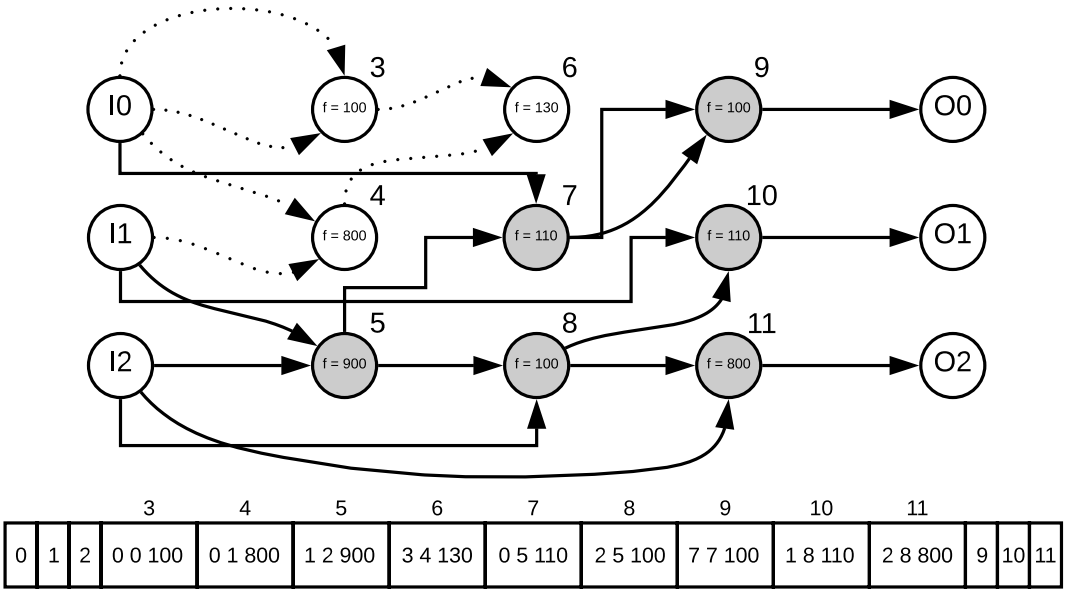


Figura 3.6: Representação de um indivíduo da CGP e seu respectivo genótipo. Os nós em cinza são ativos e os nós em branco inativos. As linhas tracejadas representam conexões que não contribuem para o fenótipo. No genótipo, os três primeiros inteiros representam as entradas primárias do programa e os três últimos inteiros, as saídas. Os nós 3 a 11 são os nós presentes na matriz de representação (genótipo) do indivíduo.

é igual ao número de colunas. Neste caso o genótipo pode representar qualquer grafo direcionado acíclico (MILLER, 2011).

É importante ressaltar que o parâmetro lb deve estar entre 1 e n_c. Isso garante que não existam ciclos no grafo. Essa restrição é muito importante principalmente quando se está evoluindo circuitos lógicos combinacionais que, por definição, não podem possuir retroalimentação.

A quantidade de nós presentes no genótipo da CGP é fixo e é determinado pelo produto do número de linhas pelo número de colunas (Ln = nc · nr). Entretanto, o tamanho do fenótipo (em termos de número de nós computacionais) pode ser qualquer coisa entre 0 nós, caso todas as saídas do programa sejam conectadas diretamente às entradas, e Ln quando todos os nós do grafo forem requeridos (MILLER, 2011).

A representação de um indivíduo da CGP com $n_c = 3$, $n_r = 3$ e lb $= n_c$ e seu respectivo genótipo são apresentados na Figura 3.6, onde I0, I1 e I2 são as entradas do programa, os nós 3 a 11 são os nós da matriz com suas respectivas funções, e O0, O1 e O2 são as saídas do programa.

Ainda na Figura 3.6, os nós em cinza são considerados ativos, pois contribuem efetivamente para o fenótipo do indivíduo, e os nós em branco são inativos. As linhas tracejadas

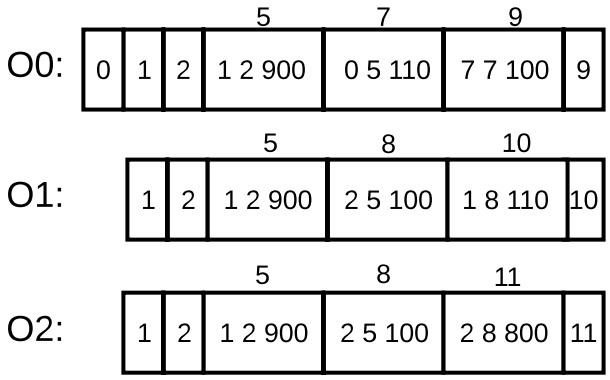


Figura 3.7: Fenótipos de todas as saídas do programa em relação ao indivíduo da figura 3.6. As lacunas com os números 0 a 2 são as entradas primárias do programa. As lacunas com os números 9, 10 e 11 são as saídas do programa. Os nós entre as entradas e as saídas são enumerados de acordo com sua posição no genótipo do indivíduo.

mostram as conexões destes nós inativos e as linhas cheias as conexões entre os nós ativos. O genótipo é composto inicialmente pelas entradas do programa (0, 1 e 2) seguido pelos genes que compõem os nós (3 a 11) e por fim as saídas do programa (9, 10 e 11). Os genes que compõem os nós, por sua vez, possuem as informações de onde o nó obtém suas entradas e a função que realiza. No caso do nó 6, por exemplo, suas entradas são obtidas a partir dos nós 3 e 4 e realizam a função XOR, codificada como 130.

A decodificação genótipo-fenótipo pode ser feita percorrendo-se todos os nós ativos a partir de uma determinada saída. O nó requerido pela saída é analisado e os nós requeridos por ele são obtidos. Esse processo é repetido até que os nós requeiram apenas entradas primárias do programa. No caso da saída O0, seu subgrafo correspondente (fenótipo) é formado pelos nós 9, 7 e 5 e as entradas I0, I1 e I2. Os fenótipos de todas as saídas da Figura 3.6 são apresentados na Figura 3.7.

A presença dos nós inativos é uma característica importante da CGP pois esses nós podem se tornar ativos a partir da aplicação de um operador genético, geralmente mutações, e passarem a contribuir efetivamente para o fenótipo do indivíduo. Além disso, mesmo que dois indivíduos da população apresentem mesmo valor de aptidão é muito provável que seus nós inativos sejam diferentes. Isso inclusive é aproveitado pela CGP, pois quando um filho possui a mesma aptidão do pai, o filho é promovido a pai (TURNER; MILLER, 2015). Essa característica auxilia o algoritmo a não ficar preso em ótimos locais e existe um grande estudo sobre a importância desses nós (TURNER; MILLER, 2015). O efeito observado pela presença dos nós inativos no genótipo da CGP é conhecido como Deriva Gênica Neutra (*NGD - Neutral Genetic Drift*).

Os valores que os genes podem assumir, denominado alelos, são restritos na CGP. Quando o genótipo é inicializado ou mutado, essas restrições devem ser obedecidas (MANFRINI *et al.*, 2017). Antes de tudo, os alelos dos genes função devem tomar funções válidas no conjunto de funções primitivas.

Uma variante simples do algoritmo evolutivo conhecido por $(1 + \lambda)$ é amplamente usada para CGP. Geralmente λ é escolhido para ser 4 (MILLER, 2011). Além disso, a CGP utiliza elitismo, mantendo sempre o melhor indivíduo da geração anterior para a geração seguinte.

3.6.1 FUNÇÃO DE APTIDÃO

Neste trabalho, a CGP é aplicada ao projeto de circuitos lógicos combinacionais. O problema consiste em obter um circuito que atenda integralmente a tabela verdade e otimizá-lo em termos da quantidade de portas lógicas, a ser minimizada. Cada linha da tabela verdade é considerada uma restrição a ser atendida. O número de restrições do problema é dado por $F_{max} = 2^{ni} \times no$, onde ni é o número de entradas e no é o número de saídas do circuito. Quando todas as restrições são atendidas o circuito é dito factível.

O tratamento de restrições adotado foi primeiramente atender as restrições (achar uma solução factível) e após isso resolver o problema de otimização (minimizar o número de portas lógicas), sem deixar que o progenitor se torne infactível. Quando os experimentos objetivam obter um circuito factível, utiliza-se o critério de atendimento da tabela verdade como objetivo. Quando, deseja-se resolver o problema de obter um circuito totalmente funcional ótimo, a estratégia de busca é dividida em duas etapas: inicialmente, obtém-se o circuito factível e, após isso, evolui-se na direção do critério de otimização (sem deixar a solução se tornar infactível).

3.6.2 OPERADORES DE MUTAÇÃO

Existem diversos operadores de mutação relatados na literatura. A mutação na CGP consiste em modificar valores de genes, de conexão ou de função, para outros valores também válidos. Caso a mutação ocorra em um gene de conexão, o novo valor assumido por este gene deverá respeitar o parâmetro *levels-back* (lb). Se a mutação ocorre em um gene de função o mesmo tem seu valor alterado para outro valor também válido no conjunto de funções da CGP. Miller ressalta que o operador de mutação mais utilizado

na CGP é o operador de mutação pontual (MILLER, 2011). Entretanto, evitar que a busca avalie um indivíduo fenotipicamente igual a seu progenitor é de suma importância quando o critério de parada é a quantidade de avaliações disponíveis no processo evolutivo. Existem abordagens visando o não desperdício de avaliações da função objetivo, isto é, evitar-se que um genótipo seja reavaliado quando as mutações ocorrem apenas em nós inativos e, portanto, não diferem fenotipicamente de seu progenitor. Neste âmbito, destaca-se o trabalho de Goldman e Punch (2013), onde 3 operadores são propostos: Skip, Accumulate e Single. Os três operadores propostos são comparados com o operador de mutação pontual. Além disso Manfrini *et al.* (2016) propuseram um operador de mutação enviesado que aproveita informações acerca das mutações benéficas ocorridas durante o processo evolutivo em prol da troca inteligente de portas lógicas para aplicações com circuitos lógicos combinacionais, denominado *Biased Single Active Mutation* (BSAM). As principais características de cada um deles são destacadas nos subtópicos a seguir.

3.6.2.1 Mutação Pontual

É o operador de mutação mais utilizado na CGP (MILLER, 2011). Nele, um alelo de um gene aleatoriamente escolhido é modificado para outro valor válido, também aleatório. O número de genes no genótipo que podem ser mutados em uma única aplicação do operador é definido pelo usuário e é normalmente uma porcentagem do número total de genes no genótipo. Essa porcentagem recebe o nome de taxa de mutação e é representado por μ_r. Para saber a quantidade aproximada de genes que serão mutados em um genótipo de tamanho L_g, basta multiplicar a taxa de mutação pela quantidade total de genes existentes no genótipo. Desta forma: $\mu_g = L_g \times \mu_r$.

3.6.2.2 Skip

Este operador de mutação tem como principal característica a não reavaliação da descendência caso os nós ativos da descendência sejam os mesmos do progenitor. Faz-se necessária uma taxa de mutação que determina quantos nós do genótipo sofrerão mutação. Em sua aplicação, verifica-se se o indivíduo gerado pela mutação e o progenitor atual são ativamente iguais (possuem os mesmos nós ativos) e, em caso positivo, atribui-se o valor de aptidão do progenitor à descendência sem a necessidade de reavaliar esse indivíduo.

3.6.2.3 Accumulate

O conceito por trás deste operador é permitir a mudança dos nós inativos por múltiplas gerações (GOLDMAN; PUNCH, 2013). Este operador também é dependente de uma taxa de mutação. Se uma descendência criada é ativamente igual ao seu progenitor então a própria descendência sofre uma sequência de mutações até que o indivíduo resultante tenha um ou mais nós ativos diferentes do pai. Se este indivíduo final, resultante da sequência de mutações, não tiver uma aptidão pior do que o indivíduo resultante da primeira mutação, a descendência toma seu lugar. Caso contrário, o indivíduo resultante da última mutação, antes da descendência ser ativamente diferente do pai, substitui o indivíduo original.

Para facilitar o entendimento, seja a primeira descendência criada a partir de F definida como F_0. Caso F_0 seja ativamente igual à seu progenitor F, uma série de mutações são aplicadas no próprio F_0 gerando uma sequência F_1, F_2, ... F_N, onde F_N é ativamente diferente do progenitor F. Ao obter o indivíduo ativamente diferente do progenitor, F_N, mantém-se F_0 de acordo com a seguinte condição:

$$F_0 = \begin{cases} F_N, & \text{se } \textit{fitness}(F_N) \geq \textit{fitness}(F_0) \\ F_{N-1}, & \text{caso contrário.} \end{cases} \tag{3.4}$$

onde N é o número da descendência gerada. Desta maneira, a descendência pode ter de zero a múltiplos nós ativos modificados quando este operador é adotado. Este operador visa explorar a deriva gênica neutra, onde o indivíduo resultante possui uma grande quantidade de nós inativos diferentes do progenitor da geração.

3.6.2.4 Single Active Mutation - SAM

O operador *Skip* possui potencial similar à mutação pontual de gerar grandes quantidades de indivíduos ativamente iguais aos seus progenitores pois, caso o indivíduo gerado seja fenotipicamente igual ao progenitor ele não é reavaliado e recebe a aptidão de seu progenitor mas é mantido na população. Por sua vez o *Accumulate* cria variações da própria descendência caso a mutação não gere um indivíduo fenotipicamente diferente de seu progenitor. Isto significa dizer que este indivíduo sofre muitas mutações e tende a se distanciar consideravelmente do local do espaço de busca que se encontra o progenitor e que está sendo explorado. A fim de evitar esse problema, Goldman & Punch propuseram

o método *Single (SAM)* que garante que ao menos um gene ativo é mutado para cada descendência. Enquanto um gene ativo não é mutado, seleciona-se outro gene aleatoriamente, e altera-se seu valor. As principais características deste método são (GOLDMAN; PUNCH, 2013):

- exatamente um gene ativo é mutado para toda a descendência;
- zero ou mais genes inativos podem ser mutados;
- um gene que é ativo será mutado com mais frequência do que um não ativo; e
- não há necessidade de taxa de mutação.

Este operador é visivelmente diferente dos anteriores, que possuem potencial de mutar múltiplos nós ativos. Entretanto, limitar a mutação à apenas um gene ativo não evita o SAM a realizar grandes modificações fenotípicas, as quais são possíveis através da conexão com nós inativos (GOLDMAN; PUNCH, 2013).

Quando não se conhece uma taxa de mutação ótima, Goldman e Punch (2013) ressaltam que o SAM é uma boa opção. Além disso, SAM requer o menor número de avaliações para encontrar uma solução factível (soluções que antedem à 100% de sua tabela verdade) e apresentou a maior redução no tempo de execução dos testes, chegando a 29%.

Um ponto importante a ser ressaltado é o fato de que o SAM não foi testado para mutar mais de um nó ativo por iteração. Além disso, nada é falado sobre a redundância da mutação, isto é, a mutação trocar o valor de um gene para o mesmo valor anteriormente presente. Ambas as características são exploradas na presente dissertação e foram abordadas em Silva e Bernardino (2018).

3.6.2.5 Biased SAM - BSAM

Manfrini *et al.* (2016) propuseram um novo operador de mutação para o projeto de CLCs. Este operador consiste em enviesar o *SAM* para que a troca de portas lógicas ocorra com uma certa probabilidade. Essa probabilidade é calculada através da quantidade de vezes em que a transição entre as portas foi benéfica. Essas transições são consideradas benéficas quando a mutação ocorre no gene responsável pela representação da porta lógica (função) e quando essa mutação gera descendência com aptidão maior que a do progenitor. Cada transição é armazenada em uma matriz cujas linhas representam a porta lógica do

progenitor e as colunas a porta lógica da descendência. Quando tal mutação benéfica ocorre, a célula que representa a transição ocorrida é incrementada em um. Ao final do processo evolutivo é possível determinar as transições que foram mais benéficas para a obtenção de circuitos factíveis. A matriz resultante é transformada em uma matriz de distribuição de probabilidades que enviesa a mutação nas portas. Uma matriz e a representação de suas respectivas probabilidades são apresentadas na Tabela 3.3 e na Figura 3.8.

Tabela 3.3: Número de mutações benéficas observada quando minimiza-se o erro em relação à tabela verdade.

	WIRE	**AND**	**OR**	**XOR**	**NOT**
WIRE	0	11	10	4	0
AND	7	0	4	3	2
OR	11	2	0	18	2
XOR	0	2	6	0	6
NOT	0	3	1	3	0

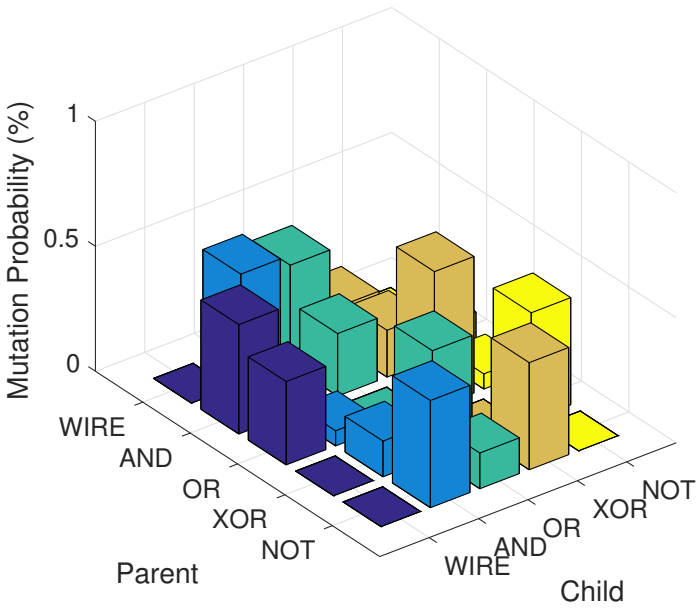


Figura 3.8: Gráficos de barra das probabilidades de transição de portas obtidas pelo BSAM.

Para criar a matriz transição de probabilidade, Manfrini utilizou 4 problemas da literatura e o conjunto de portas $\Gamma = \{$AND, OR, NOT, NAND, NOR, XOR , WIRE, C0, C1$\}$, onde C0 e C1 representam geradores de níveis lógicos constantes 0 e 1 respectivamente.

Ao fim da execução destes problemas, e com as probabilidades definidas, tal matriz foi utilizada em problemas diferentes dos utilizados para o treinamento e os resultados obtidos mostraram que o BSAM auxilia na obtenção de circuitos factíveis.

3.6.3 OPERADORES DE RECOMBINAÇÃO

Miller (2011) ressalta que os operadores de recombinação não receberam atenção na CGP pois as tentativas se mostraram prejudiciais aos subgrafos que codificam a CGP. Apesar disto, alguns pesquisadores desenvolveram alguns operadores cujas características são apresentadas em seguida.

Clegg *et al.* (2007) introduziram quatro variantes de um operador de recombinação para problemas de regressão utilizando representação inteira. Essas quatro variantes falharam em aprimorar a convergência da CGP. Após essas tentativas foi apresentado outro operador que representa o DAG como uma lista de valores reais de tamanho fixo no intervalo $[0, 1]$. A recombinação é realizada utilizando o *crossover* aritmético com um fator de peso aleatório, como utilizado nos algoritmos genéticos de representação real. Esta última proposta obteve comportamento melhor que os 4 anteriores porém seu desempenho é inconclusivo quando comparado com a CGP tradicional.

Kalkreuth *et al.* (2017) desenvolveram um operador de recombinação inspirado na recombinação de árvores da Programação Genética (KOZA, 1992). O ponto de recombinação é definido a partir da seleção de dois pontos, um em cada pai, considerando os nós ativos e o tamanho dos caminhos. Apenas o material genético do caminho ativo é recombinado e uma descendência é gerada. Os experimentos computacionais foram compostos por problemas de regressão simbólica, operadores de *design* de imagens e funções booleanas. Este método obteve resultados melhores que o *crossover* aritmético e apresentou menor número de avaliações da função objetivo para encontrar as soluções.

A CGP é dependente de posição uma vez que o efeito ou significado de um componente no programa é definido pela sua posição absoluta no genótipo. A maneira na qual os componentes são referenciados na CGP é considerado arbitrário uma vez que não existe correlação entre as coordenadas absolutas dos componentes e seu comportamento. Cai *et al.* (2006) obtiveram um efeito positivo por meio de um operador de recombinação utilizando uma representação implícita de contexto na qual a recombinação se mostrou eficiente para o problema de paridade-3.

Walker *et al.* (2006) desenvolveram um operador de recombinação para representações multi-cromossômicas. Neste caso, circuitos de múltiplas saídas são representados utilizando um grafo para cada saída e um novo indivíduo é criado a partir da recombinação dos grafos que codificam as melhores saídas. Os resultados obtidos por esta técnica foram comparados com aqueles obtidos pela abordagem de cromossomo único tradicional da CGP e uma versão da ECGP (*Embedded Cartesian Genetic Programming*) que utiliza o conceito de modularidade. O operador proposto obteve resultados melhores que as demais técnicas especialmente quando utilizado para resolver problemas com muitas saídas.

Recentemente, Husa e Kalkreuth (2018) realizaram um estudo comparativo sobre operadores de recombinação na CGP e propuseram um novo, denominado *Block Crossover*. Inspirado pelo *cone-based crossover* para ECGP de Kaufmann e Platzner (2008), o operador foi aplicado em funções booleanas e a ideia foi estender o operador para a CGP tradicional. O bloco contém um número de nós desejados e todos os nós no bloco são interligados através de suas entradas e saídas. Além disso, todos os nós no bloco são parte do caminho ativo do genótipo. Contudo, o *block crossover* não apresentou melhores resultados do que as demais abordagens citadas anteriormente.

3.6.4 NEUTRALIDADE, REDUNDÂNCIA E DERIVA GÊNICA NEUTRA

A deriva gênica neutra é um mecanismo evolutivo que pode ajudar na fuga de ótimos locais e que é fator fundamental para o sucesso da CGP, conforme relatado em diversos trabalhos (MILLER, 2011; TURNER; MILLER, 2015; VASICEK; SEKANINA, 2015; GOLDMAN; PUNCH, 2015). Tipicamente, a mutação era tida como um mecanismo de escape dos ótimos locais. Entretanto, a mutação sozinha tende a ser insuficiente e, assim sendo, torna-se importante o conceito de deriva gênica neutra (NGD), cuja origem é do campo da biologia. Uma das dificuldades em estudar a NGD é identificar quais genes são geneticamente redundantes, isto é, genes que são decodificados para o fenótipo, mas não constribuem para sua semântica (INGD - Deriva Gênica Neutra Implícita) e genes que não são decodificados para o fenótipo (ENGD - Deriva Gênica Neutra Explícita) (TURNER; MILLER, 2015).

Aumentar a quantidade de nós disponíveis aumenta a quantidade de nós inativos e, consequentemente, melhora a busca evolutiva por favorecer a ENGD. Entretanto, não se

pode aumentar indefinidamente a quantidade de nós disponíveis do genótipo. Especula-se que a razão do uso de mais nós disponíveis é benéfico pois compensa o viés do tamanho na CGP, isto é, muitos nós são disponibilizados mas poucos são ativos e contribuem efetivamente para o fenótipo dos indivíduos.

Existem muitas formas de redundância na programação genética. Os genes podem ser explicitamente ou implicitamente redundantes. É possível que muitos genótipos produzam os mesmos fenótipos. Isso cria uma redundância no mapeamento genótipo-fenótipo. Também é possível que muitos genótipos produzam fenótipos com a mesma semântica. Isso cria redundância no mapeamento genótipo-semântica. Adicionalmente, os espaços de solução por si só contém redundâncias com muitas possíveis soluções de mesma aptidão.

Os cromossomos da CGP contêm genes que podem ser ignorados na construção do fenótipo. Esses nós inativos são geneticamente redundantes. Contudo, tais genes redundantes podem se tornar ativos, através de mutação. Esse tipo de redundância genética é referida como Redundância Genética Explícita pois estes genes são explicitamente removidos durante o processo de decodificação genótipo-fenótipo. Uma redundância genética implícita também é possível na CGP, onde uma seção do fenótipo não tem influência na sua semântica.

A NGD descreve a mudança no material genético inativo durante a evolução. O neutro refere-se a não ter efeito na semântica ou no fenótipo. Uma das razões da NGD ser tomada como importante é por que ela pode guiar a uma diversidade genética da população (não fenotípica), mesmo quando presa em ótimos locais.

Outra forma de observar a influência da NGD é através do efeito dos genes redundantes presentes no espaço de busca. Como os genes redundantes por definição não afetam a aptidão, elas criam platôs de igual aptidão no espaço de busca. É possível mover-se ao longo dos platôs no espaço de busca através das mutações desses genes inativos ou redundantes. Dependendo da posição dos platôs, diferentes soluções podem ser possíveis via mutação. Um nó inativo pode produzir um comportamento (a) caso se torne ativo em uma área do espaço de busca ou comportamental, ou (b) se torne ativo em outra área do espaço de busca.

Outro comportamento da NGD é a habilidade de alterar genes inativos mesmo quando não está preso em ótimos locais. Quando genes redundantes são mutados ao mesmo tempo que genes ativos que melhoram a solução, as mudanças nos genes inativos também são

passadas aos filhos. Isso resulta em soluções que estão distantes entre si em apenas uma mutação. O mesmo não acontece caso as modificações ocorram em nós não ativos. Isso efetivamente significa que pode-se caminhar pelos platôs no espaço de busca mesmo quando não agarrados em ótimos locais.

Altos níveis de redundância genética explícita (95%) produzem o melhor espaço evolutivo para CGP e pode ser obtido através do aumento do número de nós disponíveis (MILLER, 2011; TURNER; MILLER, 2015). Desta forma a porcentagem de nós ativos diminui e a efetividade da busca evolutiva aumenta (TURNER; MILLER, 2015). Contudo, mostrou-se que os nós ativos estão concentrados próximo às entradas do programa e os nós inativos próximo das saídas do programa. Com isso o benefício gerado pela presença dos genes redundantes é prejudicado pois não encontram-se entre os ativos.

3.7 PROJETO EVOLUTIVO DE CIRCUITOS DIGITAIS

O primeiro trabalho relacionado ao projeto evolutivo de circuitos digitais refere-se à dissertação de mestrado de Friedman em 1956, conforme relatado em Simon (2013). Entretanto, somente a partir de 1993, com a tese de doutorado de (LOUIS, 1993), o projeto evolutivo de circuitos digitais começou a receber mais atenção. Pouco antes de Loius, Koza (KOZA, 1992), utilizou programação genética para o projeto de circuitos lógicos combinacionais (CLCs). Ele foi capaz de evoluir, por exemplo, um somador 2-*bit* utilizando um pequeno conjunto de portas (AND, OR e NOT). Nesta tese, Louis combina sistemas baseados no conhecimento do domínio de circuitos com algoritmos genéticos (AGs), utilizando um operador genético denominado *masked crossover* que se adapta à codificação o que o torna apto a explorar informações não utilizadas pelos operadores de recombinação tradicionais. Os resultados apresentados, apesar de encorajadores para certos exemplos, não se mostraram capazes de resolver o problema do projeto de CLC's completamente. Tanto Koza quanto Loius tinham como objetivo projetar CLC's completamente funcionais e não otimizá-los. Esses trabalhos serviram de inspiração para Coello e seus colaboradores iniciarem uma investigação mais detalhada acerca do comportamento dos algortimos genéticos quando aplicados ao projeto de circuitos lógicos combinacionais tendo em vista também a otimização dos mesmos. Essa otimização refere-se à minimização do número de portas lógicas do circuito e, consequentemente, diminuição do espaço ocupado pelo circuito. Em todos os seus trabalhos uma representação matricial para os circuitos e uma codificação

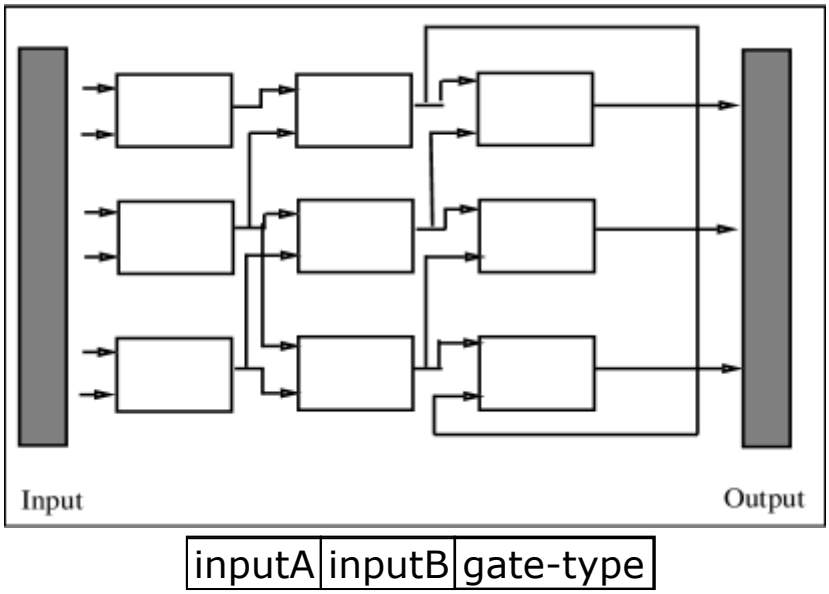


Figura 3.9: Codificação matricial proposta por Loius e utilizada por Coello - figura extraída e adaptada de (LOUIS, 1993).

padrão do genótipo foram utilizadas, conforme Figura 3.9. Esta é a mesma representação utilizada inicialmente por Loius (LOUIS, 1993), onde *inputA, inputB* e *gate-type* são, respectivamente, as entradas deste nó e a função booleana que elas implementam.

Nesta representação, cada posição da matriz simboliza uma porta lógica de um conjunto pré-determinado que continha portas AND, OR, XOR, NOT e WIRE. O WIRE é considerado uma função nula, pois representa apenas uma extensão de algum ponto do circuito. Os nós dessa matriz podiam conectar-se aos nós na coluna imediatamente anterior, à esquerda, ou diretamente nas entradas primárias do programa. Dentre os principais avanços promovidos por Coello e seus colaboradores, iniciados em 1997, destacam-se:

Em 1997, o primeiro trabalho (COELLO *et al.*, 1997) aborda os conceitos iniciais dos trabalhos de Coello com a evolução e otimização de circuitos lógicos combinacionais. Dois AGs foram utilizados: um com representação binária (denominado BGA) e um cuja representação é feita com um alfabeto de cardinalidade n (denominado NGA). A função de aptidão escolhida é dividida em dois estágios: primeiro o objetivo é acertar toda a tabela verdade do circuito, ou seja, obter um circuito totalmente funcional. Quando este critério é satisfeito, a função objetivo modifica-se para otimizar o circuito em termos de quantidade de portas lógicas existentes. Para isso, Coello bonifica as soluções que possuem a maior quantidade de WIREs pois, como não são consideradas portas lógicas, teoricamente os circuitos com mais WIREs são mais compactos. Essa hipótese foi descartada em (SILVA,

2018), uma vez que maximizar o número de WIREs não necessariamente minimiza o número de portas lógicas. O que acontece, de fato, é um aumento na quantidade de nós ativos, nós estes que implementam a função WIRE. Os problemas utilizados em (COELLO *et al.*, 1997) eram pequenos, consistindo de no máximo 4 entradas e 4 saídas. Para todas as abordagens e problemas utilizou-se uma população de 3000 cromossomos e foram permitidas no máximo 200 gerações de evolução. Os resultados obtidos são comparados entre si (BGA e NGA) e com circuitos projetados por humanos, e mostraram-se competitivos com os *designs* humanos. Além disso, obteve-se uma menor quantidade de portas lógicas via algoritmo genético. Vale ressaltar que o NGA teve comportamento superior ao BGA.

Coello e seus colaboradores publicaram em 2000 o trabalho (COELLO *et al.*, 2000), cuja ideia consiste em utilizar uma técnica baseada em população que considera cada *bit* das saídas do circuito como restrições iguais que devem ser atendidas, denominada MGA. Uma pequena subpopulação é designada para cada um dos objetivos. Quando uma das restrições é atendida, as subpopulações designadas para esta restrição são incorporadas nos demais indivíduos a fim de corrigir os demais erros em relação à tabela verdade. Quando todas as restrições são satisfeitas, todos os indivíduos cooperam para minimizar o número de portas lógicas da solução obtida. A função objetivo é a mesma do trabalho anterior. Os problemas utilizados são similares aos utilizados no primeiro trabalho, com circuitos em que as quantidades de entradas variavam entre 3 e 4 e o número de saídas variava entre 1 e 3. Os resultados obtidos mostram que os recursos computacionais necessários para o projeto de CLC's é reduzido além de mostrar-se melhor ou equivalente ao NGA em termos de otimização. Em 2002 no artigo (COELLO; AGUIRRE, 2002) os conceitos do MGA são apresentados de maneira mais detalhada, mas nenhum experimento mais complexo foi realizado.

Ainda em 2000, Coello et al. propuseram o uso do algoritmo de colônia de formigas para o *design* de CLC's (COELLO *et al.*, 2000). A colônia de formigas é baseada em um sistema multi-agente onde interações de baixo nível entre agentes únicos (formigas artificiais) resultam em um comportamento complexo de toda colônia. As formigas criam o circuito enquanto atravessam os caminhos representados pela matriz apresentada na Figura 3.9. Além do projeto, Coello et al. também consideram a otimização do circuito em termos de tamanho e, por isso, consideram a distância percorrida pelas formigas desde

as entradas até as saídas como um objetivo a ser minimizado. Nos experimentos são apresentados os três problemas iniciais abordados no artigo anterior (COELLO *et al.*, 2000) e os resultados são comparados com o BGA, projetos humanos e projetos simplificados a partir do mapa de Karnaugh. O método proposto em (COELLO *et al.*, 2000) alcançou resultados equivalentes aos resultados obtidos pelo BGA e melhores que os obtidos a partir do mapa de Karnaugh. Um fato importante a ser levantado é que o sistema estava limitado a circuitos de uma única saída. Em 2002 um outro artigo mais detalhado sobre esse sistema foi publicado (COELLO *et al.*, 2002), onde a técnica foi aprimorada a fim de suportar circuitos com mais de uma saída. Essa nova abordagem mostrou melhoras significativas em relação à anterior mas as conclusões são as mesmas.

No mesmo ano, os conceitos apresentados em Coello *et al.* (1997) foram revistos em Coello *et al.* (2000). Esse trabalho incorpora o conceito de representação de tamanho adaptativo, isto é, a matriz utilizada para codificar os circuitos possui tamanho variável. Todas as matrizes são iniciadas com tamanho 5×5 e um AG é executado. A matriz muda seu tamanho aumentando uma coluna caso soluções factíveis não sejam encontradas na quantidade de avaliações disponíveis. O GA é reexecutado e, se mesmo assim soluções factíveis não são obtidas, aumenta-se uma linha. O processo de aumentar uma coluna e uma linha é repetido até que o GA encontre uma solução factível. Nesta abordagem utiliza-se o elitismo, mantendo somente o melhor indivíduo da geração anterior e o número de indivíduos é variado de 100 a 3000. Os experimentos são realizados em 5 problemas com número máximo de entradas e saídas iguais a 4. Os resultados indicam que essa abordagem foi capaz de obter circuitos mais compactos do que as abordagens evolucionistas anteriores e do que as técnicas tradicionais de projeto.

Em 2002, Perez (PÉREZ *et al.*, 2002) mostrou como técnicas de *case-based reasoning* podem ser utilizadas para extrair e reusar soluções previamente encontradas por uma heurística (algoritmo genético) usados para resolver problemas em um domínio específico (circuitos lógicos combinacionais). Esse reuso de soluções parcialmente construídos permite melhorar o tempo de convergência da heurística. Como resultado, foram obtidas novas regras de simplificação de álgebra booleana.

Outra proposta para o projeto de CLC's é abordado em Coello *et al.* (2003). Esta proposta consiste no uso de uma versão binária da Otimização por Enxame de Partículas (PSO) no nível de porta. Apesar de a literatura reportar que o uso de PSO com repre-

sentação real é mais bem sucedida do que a representação binária, Coello defende que nos testes preliminares, o uso de um PSO binário obteve melhor comportamento para o projeto de CLCs. Os experimentos foram conduzidos em exemplos com 3 ou 4 entradas e 3 saídas, no máximo. Os resultados mostraram que o PSO foi competitivo com o NGA mas que o PSO não foi capaz de obter bons resultados quando o circuito considerado possui mais de uma saída.

Ainda em 2003, Coello *et al.* (2003) apresentaram um estudo comparativo entre diferentes heurísticas seriais e paralelas para o projeto de CLC's. As heurísticas consideradas são um AG de representação binária com elitismo e incesto, sem mutação, um híbrido denominado GASA1 que é um AG com *Recozimento Simulado (SA)* e um outro híbrido denominado GASA2 que é uma variação do GASA1 onde o SA é aplicado em uma seleção de indivíduos da população final. A função de aptidão é a mesma utilizada em todas as propostas, divida em duas etapas: primeiro obter um circuito factível e depois bonificar soluções de acordo com a quantidade de WIREs presentes. Os exemplos apresentados consistem de circuitos com 4 ou 5 entradas e de 1 a 4 saídas. As conclusões indicaram que o uso de SA traz benefícios no projeto de CLC's em termo de convergência. Além disso, como esperado, o paralelismo reduziu o tempo necessário para a execução dos algoritmos além de mostrar benefícios em termo de qualidade das soluções produzidas. Entretanto nenhuma das abordagens propostas obteve uma taxa de sucesso (porcentagem das execuções que alcançaram uma solução factível) superior a 40%.

Miller também desenvolveu, concomitantemente e de forma independente de Coello e seus colaboradores, trabalhos relacionados ao denominado *Hardware* Evolutivo. Com o passar do tempo, outros pesquisadores como Vassilev, Vasicek, Sekanina e Kalganova uniram-se a Miller a fim de aprofundar os conhecimentos no domínio. A partir de 1994, Miller e sua equipe começaram a explorar o potencial de AGs para minimização de funções *booleanas* (MILLER; THOMSON, 1995; KALGANOVA *et al.*, 1998a,b), além de trabalhar com a evolução de circuitos com funções de aptidão multiobjetivo (KALGANOVA; MILLER, 1999; KALGANOVA, 1999) e estudos sobre as *landscapes* (VASSILEV *et al.*, 1999). Porém, foi a partir de Miller *et al.* (1997) que iniciou-se a busca por técnicas evolucionistas para o projeto de CLC's. Dentre os principais avanços promovidos por Miller e seus colaboradores, destacam-se:

Em Miller (1999) a CGP é apresentada e os parâmetros de representação: número de

colunas (nc), número de linhas (nr) e *levels-back* (lb) para uma matriz de nós processantes e o funcionamento são explicados detalhadamente. Além disso, concluiu-se naquele trabalho que o uso de populações pequenas gera melhores resultados.

Em Miller *et al.* (2000a,b) são exploradas as características e os princípios da evolução de circuitos digitais no que diz respeito ao *layout*, profundidade, elementos lógicos e neutralidade. Os experimentos são realizados em circuitos aritméticos e as conclusões demonstram que sistemas complexos podem ser construídos a partir de pequenos blocos. Além disso, a evolução de multiplicadores 3-*bit* apresentou uma melhora de 20% no que diz respeito ao número de elementos lógicos quando comparados aos resultados obtidos até então na literatura.

O problema do vetor de teste é levantado em Imamura *et al.* (2000), mostrando a complexidade em utilizar a CGP quando o objetivo é encontrar um circuito factível através da evolução, isto é, o uso de pequenas amostras da tabela verdade não são capazes de guiar a busca por um circuito 100% funcional.

Yu e Miller (2001) apresentaram um estudo sobre as *landscapes* do espaço de busca de alguns problemas envolvendo circuitos digitais e explora as questões de neutralidade. As conclusões mostram que existe relação direta entre a neutralidade e o aumento da taxa de sucesso.

Ainda em 2001, Miller (2001) apresentou um estudo sobre os problemas da evolução de funções booleanas através da CGP analisando variantes da CGP totalmente conectadas sem redundância e a CGP tradicional. Os experimentos são realizados variando o tamanho do programa e comparando o tempo necessário para solucioná-los. É o princípio dos estudos sobre o problema da escalabilidade. As conclusões indicam que a presença dos nós inativos melhoram a busca da CGP.

Miller e Hartmann (2001) exploraram a evolução de portas lógicas voltadas para a geração de circuitos tolerantes à falha. Pequenos ruídos são adicionados à estas portas que são evoluídas e as conclusões mostram que a presença destas portas fazem com que o circuito em evolução aumente sua robustez e consequentemente a tolerância à falha.

Sekanina (2006) dissertou sobre os problemas e os limites da evolução de circuitos digitais da época tendo em vista o tamanho dos genótipos e a complexidade dos circuitos e levanta o problema da função de aptidão como gargalo pois a mesma mede a distância entre a solução atual e o circuito 100% factível. Além disso, apresenta um estudo compa-

rativo sobre os métodos evolucionários para evoluir circuitos digitais. Como conclusões, é possível aumentar a tolerância à falha de circuitos digitais diretamente através dos circuitos virtualmente reconfiguráveis e a presença dos componentes redundantes também auxiliam no aumento de tolerância à falha.

Em 2006 apresentou-se os primeiros estudos sobre a importância da redundância e da deriva gênica neutra na CGP (MILLER; SMITH, 2006). As conclusões mostram que o melhor desempenho foi encontrado quando emprega-se níveis extremamente altos de redundância.

No mesmo ano, Moore e Venayagamoorthy (2006) evoluíram circuitos digitais através de um algoritmo híbrido de PSO e Evolução Diferencial. Os resultados mostraram-se promissores, alcançando 100% de factibilidade nas execuções e obtendo na maioria das vezes o circuito ótimo em termos de número de portas lógicas, apesar dos circuitos de teste possuírem no máximo 4 entradas e 5 saídas.

O primeiro operador de recombinação criado para CGP é apresentado por Clegg em Clegg *et al.* (2007). Entretanto, este operador não obteve sucesso em melhorar a convergência da CGP.

Em 2008 é introduzido em Walker e Miller (2008) os princípios da utilização de módulos na CGP. Os módulos são evoluídos dentro do genótipo da CGP com operações específicas que podem ser utilizados como *building blocks* para várias saídas do programa. Os resultados indicam que a evolução de uma saída por cromossomo auxilia na obtenção de circuitos 100% funcionais.

Bremner *et al.* (2010) apresentaram um trabalho sobre a viabilidade do uso de *building-blocks* complexos, denominados moléculas, adicionados no paradigma da CGP. Aumentando a complexidade destes *building-blocks* aumenta-se também o tamanho do espaço de busca a ser explorado. Como conclusão foi observado que a funcionalidade presente nas moléculas foi explorada pela evolução de tal maneira que foge completamente às técnicas de projetos, assim como os outros casos tradicionais.

Miller (2011) lança um livro com o estado da arte da CGP, explicando todas as características da CGP tradicional, sua representação e os efeitos de seus parâmetros além de demonstrar o uso das variantes da CGP desenvolvidos até então. Diversas aplicações também são apresentadas como a evolução de circuitos, expressões matemáticas, redes neurais e criação de arte.

Vasicek e Sekanina (2014) apresentaram uma variante da CGP que utiliza diagramas de decisão binária que foi capaz de obter circuitos *benchmark* de 28 entradas através de uma abordagem que consegue contornar o problema da escalabilidade pois não é necessário criar tabelas verdade para todos os nós do fenótipo. Ao invés disso, dois diagramas de decisão binária são utilizados, um com o circuito em evolução e outro cuja funcionalidade é a desejada. Através do acoplamento de XORs entre as saídas dos dois circuitos e contando-se quantas combinações dos critérios de satisfabilidade (SAT) existem, computa-se a aptidão do indivíduo. Quando um valor 0 é obtido, o circuito é dito factível.

Goldman e Punch (2015) analisa o comportamento da evolução de circuitos digitais via CGP e propõe novos operadores de mutação, incluindo o *Single Active Mutation (SAM)*, um operador que sempre atua em um nó ativo, e em zero ou mais nós inativos, garantindo diferença fenotípica entre a descendência e o progenitor. Dessa forma evita-se o desperdício de avaliações da função objetivo. Este operador de mutação mostrou-se mais eficiente do que o operador de mutação pontual.

Turner e Miller (2015) apresentaram um estudo sobre a importância da deriva gênica neutra na CGP e explora a capacidade de gerar soluções através do uso da redundância explícita e implícita através da comparação entre a CGP tradicional e uma CGP onde só eram permitidas mutações em nós ativos para a deriva gênica neutra explícita (ENGD) e através da comparação entre a CGP tradicional e uma CGP onde só eram analisadas situações nas quais havia melhora na aptidão para a deriva gênica neutra implícita (INGD). As conclusões mostram que a ENGD é benéfica para a CGP obter soluções e que a INGD auxilia no escape de ótimos locais.

Ainda em 2015, Vasicek e Sekanina (2015) apresentaram um método para evoluir circuitos digitais aproximados através de CGP e mostra a capacidade de otimização da CGP utilizando circuitos com centenas de entradas e milhares de portas lógicas em (VASICEK, 2015).

Em 2018 foi publicado um livro em homenagem aos 60 anos de Miller (STEPNEY; ADAMATZKY, 2018) onde diversos trabalhos envolvendo a CGP foram publicados. Dentre eles, um de Vasicek (VASICEK, 2018) onde levantam-se os problemas do uso de metaheurísticas para a evolução de circuitos e sua distância dos problemas do mundo real e propõe um caminho para futuras pesquisas a fim de contornar os empecilhos já existentes.

O projeto evolutivo de circuitos digitais já foi bastante explorado pela literatura. Di-

versos tipos de algoritmos evolutivos foram utilizados e as tentativas de evoluir circuitos com grande número de entradas, principalmente nos lógicos combinacionais, continuam sendo um gargalo na área do *hardware* evolutivo. No que tange à CGP, apesar de bastante explorada e sendo indicada como o método mais eficiente para a evolução de circuitos digitais, o problema da escalabilidade impede o progresso da evolução de CLCs. Essa revisão bibliográfica traz os principais avanços promovidos por Miller e seus colaboradores para a evolução de circuitos digitais via CGP e apresenta as tentativas de criação de operadores de variação eficientes além de explorar as importantes questões da deriva gênica neutra. Apesar disso, os problemas utilizados tanto por Coello quanto por Miller ainda são, de maneira geral, pequenos no que diz respeito ao número de entradas. Somente em Vasicek e Sekanina (2014) foi possível evoluir um circuito totalmente funcional de 28 entradas. Isso reforça a necessidade de métodos de busca mais eficientes para a evolução de CLCs via CGP.

4 INVESTIGAÇÃO DE DIFICULDADES DA CGP

Neste capítulo é apresentado um estudo sobre as dificuldades da CGP. A CGP é considerada como apresentando dificuldade nas tabelas verdade em que esta técnica não obteve 100% de factibilidade. Por intermédio de diversos problemas fictícios, o comportamento da CGP é analisado nestes ambientes com o intuito de explorar e entender as maneiras pelas quais a CGP busca e encontra soluções factíveis. Esse estudo serve como base para as propostas de uso do multiplexador como elemento lógico e da evolução em 3 etapas que são detalhados nas Seções 5.4 (página 116) e 5.5 (página 117) respectivamente.

A CGP vem se mostrando uma ferramenta poderosa no projeto e otimização de circuitos digitais. Entretanto, alguns problemas aparentemente simples, que envolvem poucas entradas e poucas saídas, são difíceis para a CGP. Um destes problemas é um circuito de 4 entradas e uma saída, cuja tabela verdade e função são apresentadas na Tabela 4.1.

Tabela 4.1: Tabela verdade cuja obtenção do circuito apresenta dificuldades pela CGP.

A	**B**	**C**	**D**	**F**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Além disso, a CGP apresenta comportamentos diferentes para circuitos com a mesma quantidade de entradas, como no caso de somadores e multiplicadores de 4 *bits*, ambos com 8 entradas e 8 saídas.

Tendo em vista este comportamento diferenciado, propõe-se neste trabalho uma investigação das razões que levam isso a ocorrer, com o objetivo de definir quais são as

características dos problemas e sua relação com a facilidade ou dificuldade de resolução via CGP (em termos de número de avaliações necessários para obter as soluções factíveis), levantar questionamentos sobre o comportamento da heurística e, por fim, propor métodos que auxiliem a busca da CGP nas expressões nas quais têm dificuldade.

4.1 DESCRIÇÃO DOS PROBLEMAS INVESTIGADOS

Utiliza-se como base um circuito fictício de 4 entradas e uma saída, cuja função de saída é modificada, isto é, a função assume diversos valores que são apresentados a seguir, para em um primeiro momento analisar onde a CGP se comporta bem e onde encontra dificuldades. Como o objetivo é descobrir quais são as dificuldades da CGP, o número de saídas pôde ser reduzido para um, pois se deseja analisar o comportamento da CGP com saídas específicas. Os problemas investigados são apresentados a seguir com suas respectivas representações nos mapas de Veitch-Karnaugh:

1. F = 0 e F = 1: Quando F = 0, a tabela verdade é composta somente por *bits* "0" e quando F = 1, somente por *bits* "1". A representação destas funções é apresentada na Figura 4.1;

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

(a) F = 0

CD \ AB	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

(b) F = 1

Figura 4.1: Representações de F = 0 e F = 1 no Mapa de Karnaugh.

2. F = A, B, C, D, $\overline{A}$, $\overline{B}$, $\overline{C}$, $\overline{D}$. Neste caso a tabela verdade representa uma entrada primária ou seu complemento e é composta por uma sequência de mesma quantidade de "0's"e "1's", ou seja, são distribuições de dado totalmente balanceadas, conforme apresentado na Figura 4.2;

3. F = ABCD ou qualquer outro mintermo que utilize as quatro variáveis de entrada. Isto significa um único *bit* 1 em qualquer posição do mapa de Karnaugh;

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	0	0	0	0

Figura 4.2: Representação de F = D no Mapa de Karnaugh.

4. F = tabela verdade utilizando 15 dos 16 possíveis mintermos. Isso significa dizer que a tabela contém 15 *bits* 1 e um único *bit* 0, variando sua posição nas 16 possíveis.

5. F = composições de dois a oito mintermos não simplificantes entre si[1]. As representações no Mapa de Karnaugh são apresentadas na Figura 4.3;

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	1	0
10	0	0	0	0

(a) Dois mintermos não simplificantes entre si

CD \ AB	00	01	11	10
00	0	0	0	1
01	1	0	0	0
11	0	0	1	0
10	1	0	0	0

(b) Quatro mintermos não simplificantes entre si

CD \ AB	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	1	0	1	0

(c) Seis mintermos não simplificantes entre si

CD \ AB	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

(d) Oito mintermos não simplificantes entre si

Figura 4.3: Representações das composições de 2, 4, 6 e 8 mintermos, não simplificantes entre si.

6. F = composições de dois, quatro, seis e oito mintermos simplificantes entre si em 5 posições diferentes na tabela verdade. No caso de oito mintermos simplificantes entre si, a função booleana equivalente é a mesma da abordagem do item 2. Representações do Mapa de Karnaugh na Figura 4.4;

7. F = um bloco de 4 mintermos simplificantes entre si e um bloco de 2 mintermos também simplificantes entre si, porém os blocos de 4 e de 2 mintermos são independentes, conforme Figura 4.5;

8. F = um bloco de 4 mintermos simplificantes entre si e um mintermo isolado, em 5 posições diferentes, independente do bloco de 4 mintermos, conforme Figura 4.6;

[1]Dois mintermos não são simplificantes entre si quando não possuem variação em um único *bit*. 0000 e 0001 são simplificantes entre si pois a variável D é a única que mudou. Já 0000 e 0011 não são simplificantes entre si pois houve mudança em mais de um variável ao mesmo tempo.

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	0	0	0
10	0	0	0	0

(a) Dois mintermos simplificantes entre si

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	0	1	1
10	0	0	0	0

(b) Quatro mintermos simplificantes entre si

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

(c) Seis mintermos simplificantes entre si

CD \ AB	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

(d) Oito mintermos simplificantes entre si

Figura 4.4: Representações das composições de 2, 4, 6 e 8 mintermos simplificantes entre si.

CD \ AB	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	0	0	1	1
10	0	0	1	1

Figura 4.5: Bloco de 4 e bloco de 2 mintermos não simplificantes entre si no Mapa de Karnaugh.

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	1	1
10	0	0	1	1

Figura 4.6: Bloco de 4 mintermos e um mintermo isolado no Mapa de Karnaugh.

9. F = blocos de 8 mintermos simplificantes entre si e um desses mintermos é retirado, em 5 composições diferentes, apresentado na Figura 4.7;

CD \ AB	00	01	11	10
00	1	0	1	1
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

Figura 4.7: Mapa de Karnaugh faltando um mintermo (0100) para completar uma simplificação de 8.

10. F = Variações da Tabela 4.1.

Os resultados obtidos a partir dessas abordagens serviram como base para o levanta-

mento de investigações acerca do comportamento da CGP. Cada uma das conclusões é apresentada após os resultados obtidos e, por conseguinte, a fim de verificar a possível generalização das conclusões levantadas, novos experimentos são realizados.

4.1.1 RESULTADOS PRELIMINARES

Para todos os testes preliminares foram realizadas 30 execuções independentes com os seguintes parâmetros: $n_r = 1$, $n_c = lb = 100$, número máximo de avaliações de 100.000 e o conjunto de funções $\Gamma = \{AND, OR, XOR, NOT, WIRE\}$. A função de aptidão é a quantidade de acertos em relação a tabela verdade. Além disso, foi utilizado somente o operador de mutação SAM. As execuções foram realizadas até que se obtivesse um circuito factível ou que o número máximo de avaliações de função objetivo fosse atingido.

Para apresentar os resultados são utilizadas as seguintes métricas: o melhor (menor número de avaliações necessárias para obter um circuito factível), o primeiro quartil (Q1), a mediana (Med), o terceiro quartil (Q3), a média, o desvio padrão (DP) o pior (maior número de avaliações necessárias), o desvio absoluto da mediana (DAM), o intervalo inter-quartil (IIQ) do número de avaliações e a taxa de sucesso (TS), que representa a porcentagem das execuções independentes que foram capazes de obter soluções factíveis. O objetivo aqui é encontrar circuitos factíveis e não otimizá-los.

A seguir são apresentados os resultados para cada uma das investigações propostas e as conclusões preliminares acerca das investigações levantadas na Seção 4.1.

4.1.2 INVESTIGAÇÃO 1 - F = 0 OU F = 1

A CGP apresentou facilidade para obter circuitos tanto com F=0 quanto com F=1. O número de avaliações necessárias não foi superior a 149 e a quantidade de avaliações necessárias para obter ambas as funções pode ser vista na Tabela 4.2 e nos *boxplots* da Figura 4.8.

Tabela 4.2: Número de avaliações necessárias para obter um circuito factível quando F=0 e F=1.

F	**Melhor**	**Q1**	**Med**	**Q3**	**Média**	**DP**	**Pior**	**DAM**	**IIQ**	**TS**
0	1	6,5	16	27,75	28,8	34,27	127	10,5	21,25	100%
1	1	11,25	21	33,25	30,57	35,14	149	11	22	100%

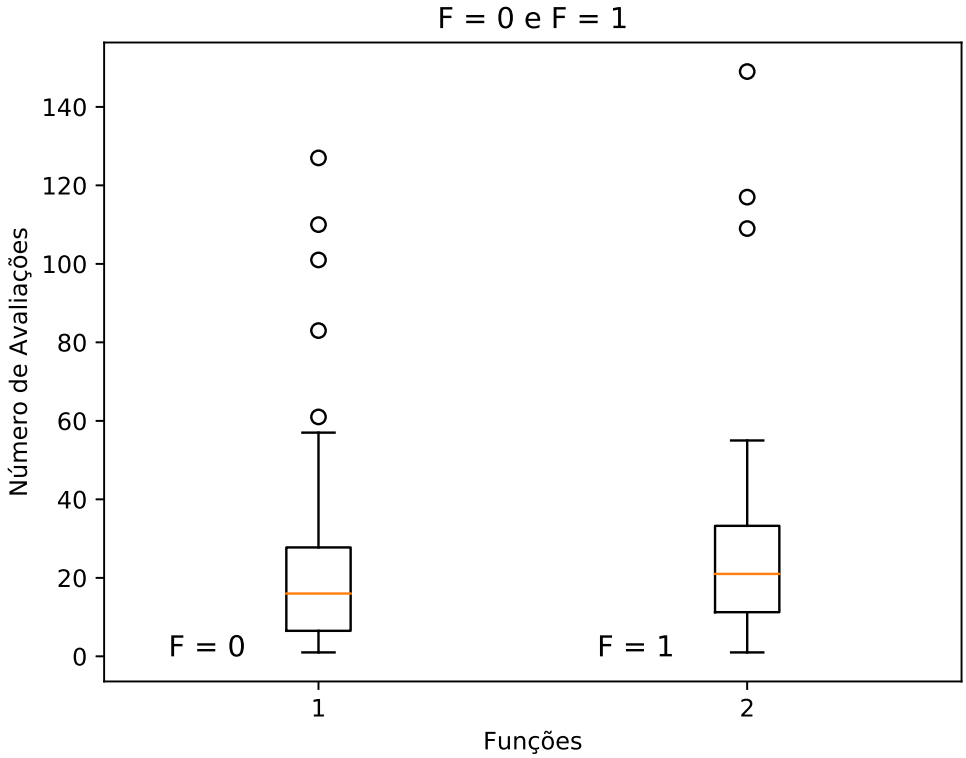


Figura 4.8: *Boxplots* para F=0 e F=1.

Para ambas as funções analisadas foram necessárias menos de 1,5% das avaliações máximas permitidas. Aparentemente, a CGP apresenta grande facilidade para obter F=0 e F=1 independentemente do número de entradas e saídas do circuito em questão. De fato, em testes adicionais, verificou-se que a CGP encontra facilidade para obter F=0 e F=1 independentemente do número de entradas. Testes foram realizados com 10, 15 e 20 entradas e os resultados podem ser vistos na Tabela 4.3. Portanto a CGP apresenta bom comportamento quando as funções são constantes. Além disso, esses resultados indicam que o balanceamento de dados não apresenta grande influência para o sucesso da CGP visto que ambas as funções exploradas nesta investigação são compostas de representantes de uma única classe.

4.1.3 INVESTIGAÇÃO 2 - FUNÇÕES IGUAIS ÀS ENTRADAS PRIMÁRIAS OU SEUS COMPLEMENTOS

Nesta investigação a tabela verdade representa uma das entradas primárias ou seus complementos, isto é, a entrada negada. Os resultados são apresentados na Tabela 4.4 e nos

Tabela 4.3: Número de avaliações necessárias para obter um circuito factível quando F=0 e F=1 para 10, 15 e 20 entradas.

F	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
10 entradas										
0	1	19,25	48,5	79,25	67,1	68,41	268	30,5	60	100%
1	7	15	40	70,25	53,97	48,4	207	28,5	55,25	100%
15 entradas										
0	6	17,25	43	81,25	52,1	41,7	158	29,5	64	100%
1	3	31,25	42,5	92,5	79,17	93,77	422	26,5	61,25	100%
20 entradas										
0	17	35	63	102	88,06	108,3	487	35	67	100%
1	42	51	56	92	190,6	292,09	712	14	41	100%

boxplots da Figura 4.9.

Tabela 4.4: Número de portas necessárias para obter cada uma das funções que expressam uma entrada primária ou seu complemento.

Função	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
A	1	14,5	22	89,75	62,43	78,29	340	18,5	75,25	100%
$\overline{A}$	1	22,5	51,5	106,25	79,27	84,13	327	45	84	100%
B	1	23	38,5	83	60,53	61,77	218	31,5	60	100%
$\overline{B}$	1	12,75	42,5	106,75	64,27	75,58	344	34,5	94	100%
C	1	17,5	40,5	73,75	64,97	86,73	379	28	56,25	100%
$\overline{C}$	7	30,75	57	103,5	85,6	91,5	430	35,5	72,75	100%
D	2	12	32	62,5	53,03	58,21	213	27	50,5	100%
$\overline{D}$	1	11	39,5	73,75	51,53	50,92	202	31,5	62,75	100%

Em muitos momentos a CGP é capaz de encontrar a função simplificada de forma direta, isto é, caso a tabela verdade represente o literal A, muitas vezes a CGP encontra o próprio A ou o A com WIREs. Em alguns outros momentos encontra simplificações simples do tipo $\overline{\overline{A}}$ ou até mesmo $\overline{\overline{\overline{A}}}$. Entretanto, em outros momentos a CGP cria funções booleanas complexas tendo em vista o que é solicitado. Isso ocorreu para todas as funções utilizadas que são todos os literais e seus complementos conforme apresentado na coluna Função da Tabela 4.4.

Além disso, os resultados indicam que a CGP apresenta facilidade para obter a função desejada, independentemente de qual literal está em questão. Uma coisa que pode ser destacada é o fato de que as funções que representam literais negados necessitam de um pouco mais de avaliações da função objetivo para serem obtidos. Em contraste à investigação anterior, nesta os dados são sempre balanceados. Os resultados indicam que quando as funções representam entradas primárias do circuito ou seus complementos a CGP é capaz de obter os circuitos equivalentes com facilidade.

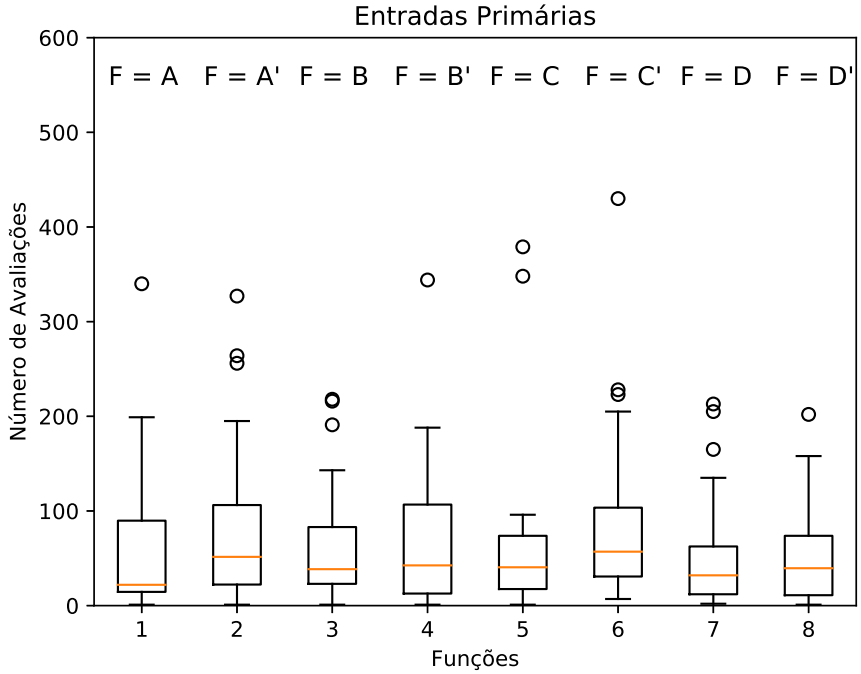


Figura 4.9: *Boxplots* de todas as funções que representavam literais.

Obviamente obter a função mais simples possível não é um objetivo primário aplicado pois o objetivo não é otimizar o circuito. Entretanto é interessante verificar que mesmo que tradicionalmente o projeto desses circuitos seja bastante simples, a CGP apresenta bastante facilidade para encontrar essas expressões booleanas de maneira pouco tradicional. Um exemplo de circuito que foge ao tradicionalismo é apresentado na Figura 4.10. Os números acima das portas lógicas são os nós que as representavam no genótipo.

A Tabela 4.5 apresenta as funções mais comuns obtidas (Recorrentes) pela CGP para cada um dos literais testados, além de algumas funções (Outras Funções) pouco intuitivas. É importante ressaltar que todas as funções apresentadas têm a mesma funcionalidade. A forma mais simplificada possível é justamente o literal apresentado na primeira coluna.

4.1.4 INVESTIGAÇÃO 3 - UM ÚNICO MINTERMO

Para um único mintermo, considerando a facilidade do problema envolvido (4 entradas e 1 saída) a CGP se comportou dentro do esperado encontrou uma solução factível com poucas avaliações de função objetivo (2.122,26 avaliações na média).

Além disso, é possível verificar que alguns mintermos são mais facilmente obtidos do que outros, conforme apresentado na Tabela 4.6 e os *boxplots* da Figura 4.11.

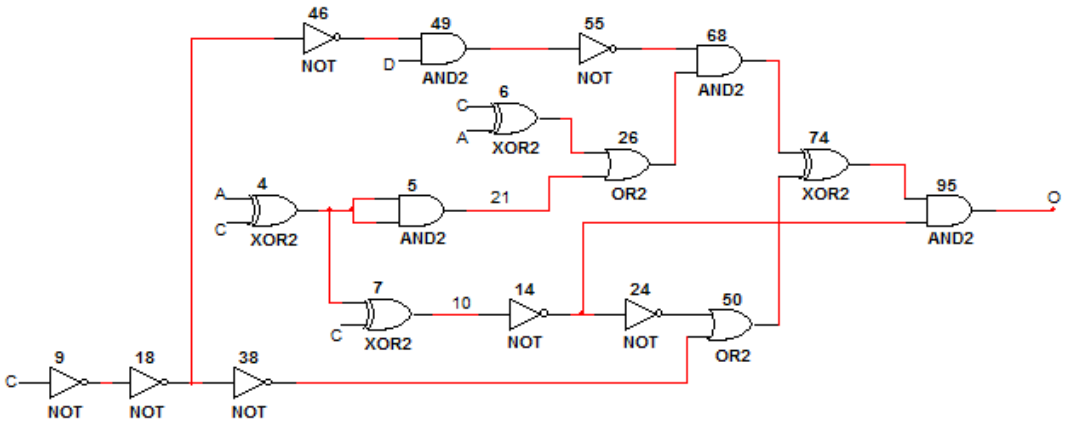


Figura 4.10: Circuito obtido que realiza a função F = $\overline{A}$

Tabela 4.5: Funções obtidas pela CGP para cada um dos literais desejados.

Desejada	**Recorrentes**	**Outras Funções**
A	A A+A A·A	(A+A) ⊕ C + $\overline{\overline{(A+A)}}$ A · [A·A + (D+D ⊕ C)] [(A·C) ⊕ A] + A
$\overline{A}$	$\overline{A}$ $\overline{(A+A)}$	[A · ($\overline{D}$ ⊕ A⊙D)] $\overline{(B \oplus B)+A)}$
B	B $\overline{\overline{B}}$ B+B	($\overline{B}+\overline{B}$) $(\overline{A}\cdot(A+A))\oplus\overline{B}$ $((D\oplus A)\cdot B)+B$
$\overline{B}$	$\overline{B}$ $\overline{(B+B)}$ $\overline{\overline{B}}$	$\overline{[(D\cdot B)+B]}$ $\overline{[(D+B)\cdot B]}$ $\{[(\overline{A}\oplus A)+(\overline{A}\oplus A)]+C\}\oplus\overline{B}$
C	C $\overline{\overline{C}}$ $(C\cdot C)\cdot(C\cdot C)$	A ⊕ (A+A ⊕ C) $[(\overline{\overline{B}}\cdot B)\oplus C]\oplus\overline{\overline{B}}$ $(B\oplus C)\oplus B$
$\overline{C}$	$\overline{C}$ $\overline{(C+C)}$ $\overline{(C*C)}$	$\overline{\overline{\overline{C}}}$ (A+C) ⊕ $[\overline{(\overline{A+B})}+C]$ $\overline{[(C\cdot C\cdot B\cdot B)+C\cdot C]}$
D	D $\overline{\overline{D}}$	$(\overline{D}\cdot D)\oplus D$ $\overline{[\overline{C}\cdot\overline{C}\oplus\overline{D}]}\cdot D$
$\overline{D}$	$\overline{D}$ $\overline{(D+D)}$ $\overline{(D\cdot D)}$	$\overline{\overline{\overline{D}}}$ $\overline{D}\cdot\overline{D}$ $\overline{[(B\oplus D)\oplus\overline{D}]}$

O mintermo mais fácil de ser obtido foi o ABCD. Pensando em termos de projeto tradicional, através da soma de produtos, este é o único mintermo que não utiliza portas NOT. Com as portas de 2 entradas utilizadas nessa abordagem de CGP, seriam necessárias 3 portas AND para atender este mintermo. De fato, este *layout* de circuito foi obtido em

Tabela 4.6: Número de avaliações necessárias para obter cada um dos mintermos.

F	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
$\overline{A}\overline{B}C\overline{D}$	83	1088,75	2796	4528,5	3390,37	2892,39	12118	1772	3439,75	100%
$\overline{A}\overline{B}\overline{C}D$	71	622,75	1279,5	3569,25	2324,4	2246,66	7764	1102,5	2946,5	100%
$\overline{A}\overline{B}\overline{C}\overline{D}$	64	728,5	1369,5	2985	2699,2	3651,92	17163	998	2256,5	100%
$\overline{A}BCD$	14	591,5	1632	3121,25	2482,53	2699,83	10036	1215	2529,75	100%
$\overline{A}BC\overline{D}$	54	891	1284	2492,25	1737,67	1493,94	6887	840,5	1601,25	100%
$\overline{A}B\overline{C}D$	22	200	793	2916,25	2150,67	2866,07	10576	744	2716,25	100%
$\overline{A}B\overline{C}\overline{D}$	128	517,5	1193	3128	2033,4	2050,93	8008	759,5	2610,5	100%
$A\overline{B}CD$	10	293,25	1070	2794,5	1786,6	1966,8	7725	848	2501,25	100%
$A\overline{B}C\overline{D}$	51	934,75	1948,5	4567,5	2884,77	2771,87	13021	1617,5	3632,75	100%
$A\overline{B}\overline{C}D$	72	607	1654	3496,75	2320,77	2629,17	13570	1266	2889,75	100%
$A\overline{B}\overline{C}\overline{D}$	53	691,5	1128,5	2699	1864,97	1642,64	5397	824	2007,5	100%
$AB\overline{C}D$	25	339,75	1556,5	3598,75	2512,43	2985,6	12174	1334,5	3259	100%
$AB\overline{C}\overline{D}$	11	829,5	1215	2314	2031,17	2842,68	15642	767,5	1484,5	100%
$AB\overline{C}D$	38	396	917,5	1614,75	1205,4	1122,22	5028	591	1218,75	100%
$ABC\overline{D}$	17	484,75	1277	2061	1573,17	1602,84	8590	844,5	1576,25	100%
ABCD	45	516,5	768	1469,75	958,67	724,07	2526	515	953,25	100%

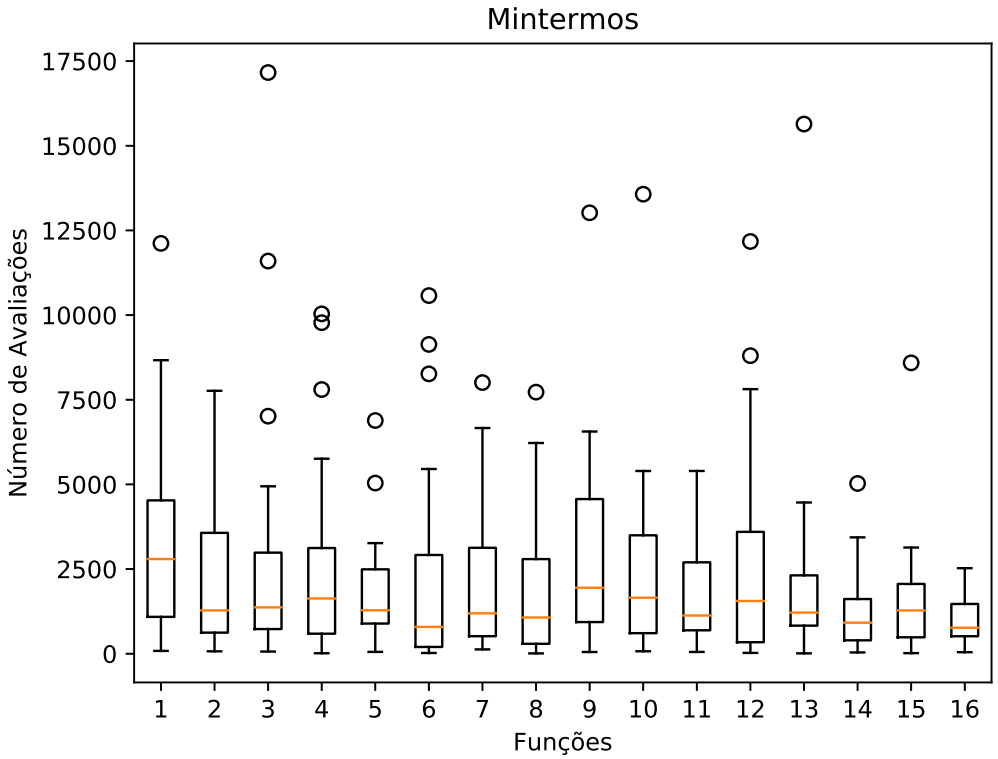


Figura 4.11: *Boxplots* para todos os mintermos.

2 das 30 execuções, o que exclui a possibilidade da CGP ter utilizado desta faceta do tradicionalismo de projeto. O número de portas necessárias para obter este mintermo variou entre 3 e 18 o que mostra que, como na investigação anterior, a CGP obtém circuitos complexos para realizar funções simples. A quantidade de portas necessárias para atender os mintermos são apresentados na Tabela 4.7. Foi possível observar também redundâncias fenotípicas nos produtos, isto é, produtos entre portas que possuem o mesmo

conteúdo.

Além disso, um fato interessante merece ser destacado. Conforme observado na primeira investigação, a CGP possui facilidade para obter F=0 e F=1. Nesta investigação de mintermos isolados, percebe-se que na maioria das vezes a CGP primeiramente obtém F=0 e depois acerta o mintermo específico que está sendo buscado. Isso é explicável através do fato de que F=0 dará uma aptidão de 15 para todas as investigações, pois o único erro será justamente o mintermo que está sendo buscado. Portanto este problema pode ser considerado fácil pois conforme demonstrado na Seção 4.1.2 a CGP obtém F=0 com facilidade e concentra os esforços da busca para obter o único mintermo desejado. Além disso, como a busca é por um único mintermo leva-se em consideração apenas uma única combinação de níveis lógicos de entradas (as do mintermo) para um único nível lógico 1 na saída. Os resultados desta investigação indicam que dados desbalanceados não afetam o desempenho da CGP.

Tabela 4.7: Número de portas necessárias para obter cada um dos mintermos.

Mintermo	**Melhor**	**Q1**	**Med**	**Q3**	**Média**	**DP**	**Pior**	**DAM**	**IIQ**	**TS**
$\overline{A}\overline{B}CD$	4	7	9	10	9,07	3,16	18	2	3	100%
$\overline{A}\overline{B}C\overline{D}$	6	8	12	15	11,57	3,65	19	3	7	100%
$\overline{A}\overline{B}C\overline{D}$	6	8	9	12,75	10,17	3,47	21	2	4,75	100%
$\overline{A}\overline{B}CD$	5	8	11	12	10,93	3,67	22	2	4	100%
$\overline{A}\overline{B}\overline{C}\overline{D}$	5	8	9	11,75	10,1	3,78	20	1,5	3,75	100%
$\overline{A}\overline{B}\overline{C}D$	5	7,25	9	12	9,47	3,16	16	3	4,75	100%
$\overline{A}\overline{B}C\overline{D}$	4	6,25	9	12	9,57	3,48	17	3	5,75	100%
$\overline{A}BCD$	4	6	8	10	8,77	3,51	18	2	4	100%
$A\overline{B}\overline{C}\overline{D}$	4	8	11,5	14,75	11,43	4,14	18	3,5	6,75	100%
$A\overline{B}\overline{C}D$	6	9,25	11	14	11,47	3,72	19	3	4,75	100%
$A\overline{B}C\overline{D}$	6	7,25	11	13	10,97	3,65	18	3	5,75	100%
$A\overline{B}CD$	5	7	9	10,75	9,3	3,15	17	2	3,75	100%
$AB\overline{C}\overline{D}$	5	9	11	13	11,5	4,46	22	2	4	100%
$AB\overline{C}D$	4	6,25	9	12	9,33	3,69	17	3	5,75	100%
$ABC\overline{D}$	5	6	10	12	9,37	3,81	20	3	6	100%
ABCD	3	5,25	8	10	8,03	3,49	18	2	4,75	100%

4.1.5 INVESTIGAÇÃO 4 - UM ÚNICO *BIT* 0 COM POSIÇÃO VARIÁVEL

Na Seção 4.1.4 o problema em questão era sobre desbalanceamento de dados onde a classe majoritária era o nível lógico 0. Nesta investigação é proposto o contrário, isto é, tem-se uma tabela verdade composta por muitos 1 e apenas um único 0. Para as 16 possíveis

posições da tabela verdade, variou-se, uma a uma, a posição que o *bit* 0 encontrava-se. A Tabela 4.8 apresenta o número de avaliações necessárias para obter as tabelas verdade para cada posição (Pos). A mesma informação é apresentada nos *boxplots* da Figura 4.12.

Tabela 4.8: Número de avaliações necessárias para a investigação com *bit* 0 em posição variável.

Pos	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	10	84	297,5	1100	742,13	1138,63	5280	220	1016	100%
2	5	273	1001,5	2144,5	1991,17	2661,27	11004	829,5	1871,5	100%
3	27	329	1210	2417	1535,47	1423	6032	958	2088	100%
4	316	1020,75	2281,5	4263	3217,9	2877,47	12021	1376	3242,25	100%
5	26	373,75	678,5	2455,75	1343,6	1243,17	3943	517	2082	100%
6	81	448,75	1027	2307,5	1494,43	1388,11	4803	812,5	1858,75	100%
7	172	674,75	2202,5	5044	3051,03	2673,71	9384	1587,5	4369,25	100%
8	65	340,75	2100	4192,25	2988,8	3333,53	13261	1818,5	3851,5	100%
9	25	419,5	886,5	2385,25	1914,3	2476,83	10959	727	1965,75	100%
10	23	590,25	1036,5	3645,75	2282,9	2378,11	7656	921	3055,5	100%
11	48	1500,5	1906,5	3245,5	2739,07	2542,44	11112	638	1745	100%
12	55	637,5	1436,5	3207	2569,2	3314,74	14225	1167,5	2569,5	100%
13	8	357,75	1284,5	2515,5	2259	2641,86	10070	1067	2157,75	100%
14	58	736	1941	3362,75	3193,43	5155,15	27607	1360,5	2626,75	100%
15	45	594	1489,5	2810,5	2458,4	2509,72	12465	1006	2216,5	100%
16	32	656	2019,5	3960,25	2477,97	2111,1	7301	1513,5	3304,25	100%

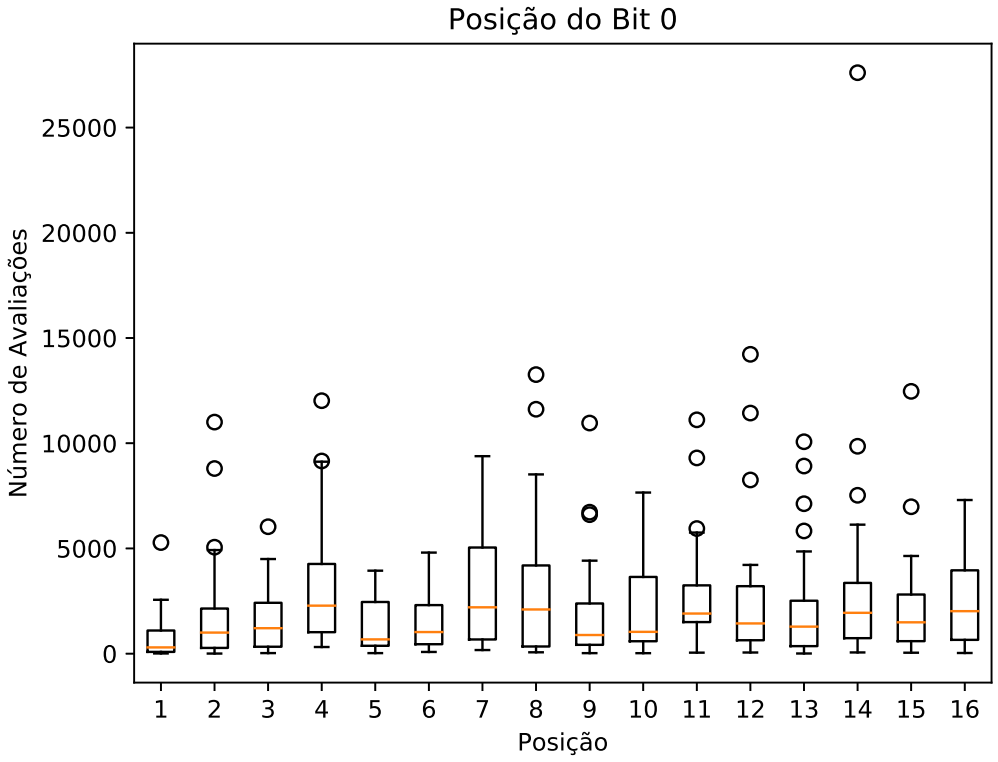


Figura 4.12: *Boxplots* para um bit 0 em posições variadas da tabela verdade.

O comportamento, no entanto, se mostrou muito semelhante para as duas abordagens, isto é, cada uma das investigações apresenta comportamento diferente, mas todas

se mostraram relativamente fáceis para a CGP obter os circuitos. É importante ressaltar que essa investigação consiste em uma tabela verdade onde a função quase representa F = 1. Observando o processo evolutivo da CGP nesta investigação é possível perceber que, justamente, a CGP obtém F = 1 inicialmente, o que resulta em uma aptidão de 15 e depois acerta o *bit* 0 específico. Esse comportamento também foi observado para a análise dos mintermos separadamente onde F = 0 resultava em uma aptidão de 15. Novamente, a investigação apresentada nesta seção indica que o desbalanceamento de dados não é um problema para a obtenção de soluções via CGP.

4.1.6 INVESTIGAÇÃO 5 - COMPOSIÇÕES NÃO SIMPLIFICANTES ENTRE SI

4.1.6.1 2 mintermos

Similarmente à análise apresentada na investigação anterior, a CGP apresentou facilidade em resolver os problemas testados. De maneira geral, a CGP apresentou maior facilidade em resolver os problemas com 2 mintermos do que somente um. Dentre as 8 composições geradas, 4 delas tiveram seus mintermos escolhidos de maneira aleatória (as 4 últimas da Tabela 4.9. As demais composições foram criadas de tal maneira que um dos mintermos fosse constante ($\overline{ABC}D$) e somente o segundo fosse variado. A justificativa para essa escolha é verificar se manter um mintermo constante influencia na busca pelo segundo mintermo. As composições utilizadas são apresentadas na Tabela 4.9 e o número de avaliações necessárias para obter cada uma das composições é apresentado na Tabela 4.10 e nos *boxplots* da Figura 4.13.

Tabela 4.9: Composições de 2 mintermos não simplificantes entre si utilizados.

Composição	**Função**
1	$\overline{ABC}D + \overline{A}BC\overline{D}$
2	$\overline{ABC}D + A\overline{B}CD$
3	$\overline{ABC}D + AB\overline{C}D$
4	$\overline{ABC}D + ABCD$
5	$\overline{AB}C\overline{D} + AB\overline{C}\overline{D}$
6	$\overline{AB}CD + A\overline{BC}D$
7	$\overline{A}B\overline{C}\overline{D} + ABCD$
8	$\overline{A}BC\overline{D} + AB\overline{C}\overline{D}$

Tabela 4.10: Número de avaliações necessárias para composições de 2 mintermos não simplificantes entre si.

Comp.	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	949	1827,25	2873	6066,5	4913,17	4865,41	21052	1406	4239,25	100%
2	270	1358,5	2122	4604,75	3691,97	3844,08	14947	1155,5	3246,25	100%
3	50	677,25	1709	2949,25	2308,03	2282,45	10309	1063	2272	100%
4	532	2332	5087	9105	7887,07	8212,75	36871	2974,5	6773	100%
5	178	1494,75	2449,5	4836,25	4025,3	4086,76	17693	1728	3341,5	100%
6	86	973,5	1520,5	2202,75	1814,5	1232,63	4628	593	1229,25	100%
7	270	1283,25	1998	7417,5	6527,97	9421,33	38978	1235,5	6134,25	100%
8	12	699,25	1536	3246,5	2046,13	1830,18	8157	1219	2547,25	100%

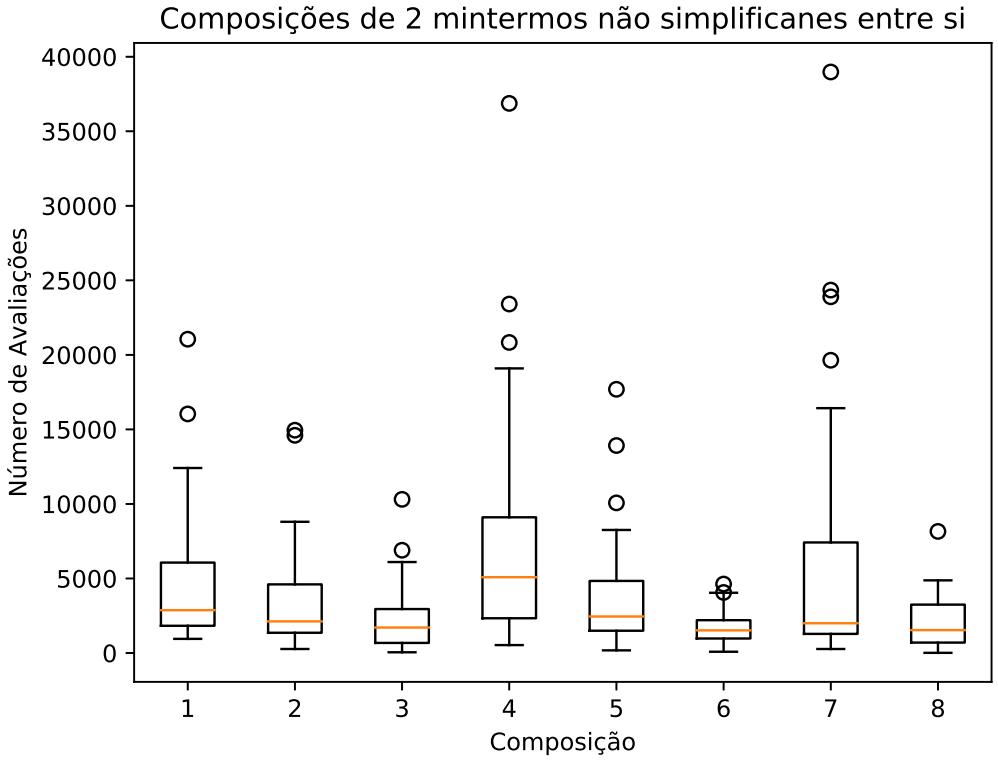


Figura 4.13: *Boxplots* para as composições de 2 mintermos não simplificantes entre si.

Conforme apresentado na Seção 4.1.4, o mintermo ABCD foi o mais fácil de ser obtido. Baseado nisto é possível acreditar que as composições utilizadas nesta investigação que incluem tal mintermo seriam mais fáceis de serem obtidos (composições 4 e 7). Entretanto isso não foi observado o que indica que o efeito de um único mintermo não é decisivo para o sucesso da CGP, mas sim a disposição dos mintermos na tabela verdade que geram as composições.

4.1.6.2 4 mintermos

Para quatro mintermos não simplificantes entre si o número de avaliações necessárias para obter circuitos factíveis aumenta consideravelmente em relação às investigações anteriores. Os mintermos que formam as composições de 4 mintermos são apresentados na Tabela 4.11. A quantidade de avaliações necessárias para obter os circuitos das composições é apresentado na Tabela 4.12 e nos *boxplots* da Figura 4.14.

Tabela 4.11: Composições de 4 mintermos não simplificantes entre si utilizados.

Composição	**Função**
1	$\overline{A}BC\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + ABCD$
2	$\overline{A}BC\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD$
3	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD$
4	$\overline{A}B\overline{C}D + \overline{A}BCD + AB\overline{C}D + ABC\overline{D}$
5	$\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}BCD + AB\overline{C}\overline{D}$
6	$\overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + A\overline{B}C\overline{D}$
7	$\overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD$
8	$A\overline{B}C\overline{D} + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + ABCD$

Tabela 4.12: Número de avaliações necessárias para composições de 4 mintermos não simplificantes entre si.

Comp.	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	277	144,75	2791	4482,75	3699,6	4155,18	23443	1524	3005	100%
2	179	540,25	1362	2006	1513,67	1152,5	4625	755,5	1465,75	100%
3	240	712,5	1246,5	2538,5	1767,1	1382,19	6082	588	1826	100%
4	57	524,5	1134,5	1920,75	1939,77	2479,71	10202	666	1396,25	100%
5	1217	2514,75	4753	8448,5	5871,93	4038,39	16413	2644	5933,75	100%
6	2721	4577,75	7400,5	10290	13531,32	17026,41	69545	3002,5	5712,25	93,33%
7	24	894	1597	5204,5	3280,9	3511,41	14844	1130,5	4310,5	100%
8	281	480,5	942	2031	1426,83	1363,86	6077	575	1550,5	100%

A composição 8 foi a mais fácil de ser obtida. Um fato interessante a ser destacado aqui é que a composição mais difícil, 5, é a única que apresenta somente um mintermo com o literal A (não barrado), o que leva a crer que, como é o único caso em que isso acontece, a CGP não pode reaproveitar essa função caso fosse obtida durante o processo evolutivo para os demais mintermos e, portanto, necessita obtê-la do zero. A fim de verificar esse comportamento, foi criada uma nova composição onde não só aparecesse um mintermo A não barrado, mas também um único mintermo B não barrado. Essa composição é apresentada como composição 6. Baseado nas informações levantadas pela composição 5, a composição 6 deve ser mais difícil do que todas as demais. E é o que de fato foi obtido. Além de o problema ter sido mais difícil de ser resolvido pela CGP, sua taxa de sucesso

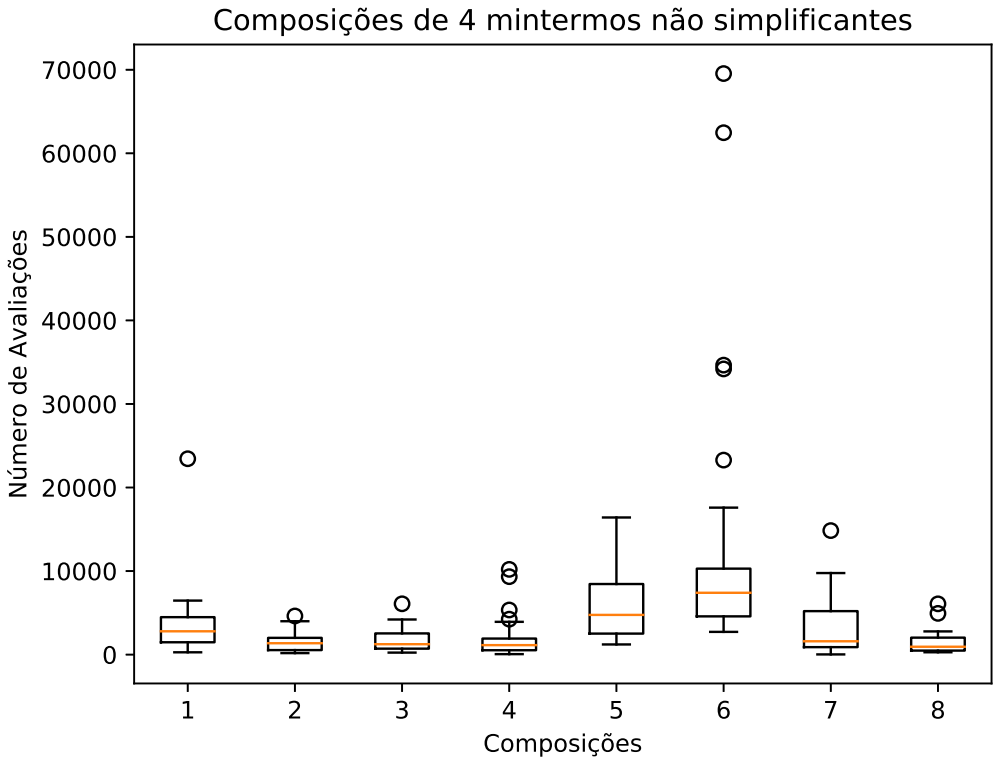


Figura 4.14: *Boxplots* das composições de 4 mintermos não simplificantes entre si.

foi de 93,33%, o que indica que 2 das 30 execuções não encontraram um circuito factível considerando o orçamento computacional disponível.

Outro fato interessante é que as composições que apresentaram melhor comportamento foram as que mantiveram um dos literais sempre em mesmo nível lógico: composição 2 (literal B sempre em 0) e composição 4 (literal B sempre em 1). A fim de verificar se manter um dos literais sempre em mesmo nível lógico favorece o desempenho da CGP, as composições 7 e 8 foram criadas, mantendo mantendo o literal A sempre em nível lógico 0 e 1, respectivamente. Os resultados mostraram que quando o nível lógico está sempre em 1 é de fato benéfico. O mesmo não foi observado para o nível lógico 0. Uma possível explicação para tal fato é que mintermos que possuem um literal de nível lógico zero necessitam de uma porta NOT em sua representação de soma de produtos ou alguma composição de portas lógicas que seja capaz de inverter o sinal deste literal. O mesmo não é necessário para literais com nível lógico um. De fato, o número de portas lógicas necessárias para obter o circuito da composição 7 foram maiores do que os necessários para a composição 8. A quantidade de portas lógicas para todas as composições pode ser vista na Tabela 4.13. De maneira geral, as composições que se comportaram melhor em número

Tabela 4.13: Número de portas lógicas necessárias para composições de 4 mintermos não simplificantes.

Composição	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	5	8,25	11	14	11,67	4,78	25	3	5,75	100%
2	4	7	10	12	9,93	3,4	17	2	5	100%
3	4	8	10	12	10,17	3,36	19	2	4	100%
4	4	6,25	8	10,75	9,03	3,57	17	2	4,5	100%
5	5	9,25	11	13,75	11,9	3,67	22	2	4,5	100%
6	8	11	12	15	13,07	3,52	20	2	4	93,33%
7	5	7,25	10,5	13	10,57	4,23	23	2,5	5,75	100%
8	3	6	8,5	11,75	9,13	4,31	18	2,5	5,75	100%

de avaliações apresentaram também menor quantidade de portas lógicas. Portanto pode-se perceber que para o sucesso da CGP, composições que não são simplificantes entre si são influenciadas pelo nível lógico que os literais se encontram. Esse dado traz informações sobre a possibilidade de obter os mintermos desejados de forma mais rápida, pois é possível aproveitar uma combinação de entradas já obtida durante o processo evolutivo da CGP.

4.1.6.3 6 mintermos

Para as composições de 6 mintermos a CGP apresentou maior dificuldade. Das 6 composições analisadas somente metade delas foi possível obter uma taxa de sucesso igual a 100%, apesar das demais terem ficado em 96,67%, o que representa 1 em 30 sementes sem que uma solução factível fosse encontrada. Os mintermos que formam as composições de 6 mintermos são apresentados na Tabela 4.14. A quantidade de avaliações necessárias para obter os circuitos das composições é apresentado na Tabela 4.15 e nos *boxplots* da Figura 4.15.

Tabela 4.14: Composições de 6 mintermos não simplificantes entre si utilizados.

Composição	**Função**
1	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D}$
2	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D} + ABCD$
3	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}BCD + A\overline{B}CD + ABC\overline{D}$
4	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D}$
5	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D}$
6	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + ABC\overline{D}$

Segundo os *boxplots* da Figura 4.15, é possível perceber que a CGP apresentou pior desempenho para obter a Composição 5 e melhor desempenho na Composição 3. En-

Tabela 4.15: Número de avaliações necessárias para composições de 6 mintermos não simplificantes.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	242	2057	2999	7585	7122,59	12729,5	69447	1834	5528	96,67%
2	521	1379	3070	5676	4207,41	3698,29	14636	1766	4297	96,67%
3	267	1589,25	2608	4891,5	4757,9	5918,11	26248	1205	3302,25	100%
4	799	1917,5	3383	8843,5	7306,87	9306,26	36542	1762	6926	100%
5	741	2954	5308	11726	8847,48	10039,02	45231	3297	8772	96,67%
6	406	1137,25	2250,5	5580,25	4046,9	4632,38	22588	1623,5	4443	100%

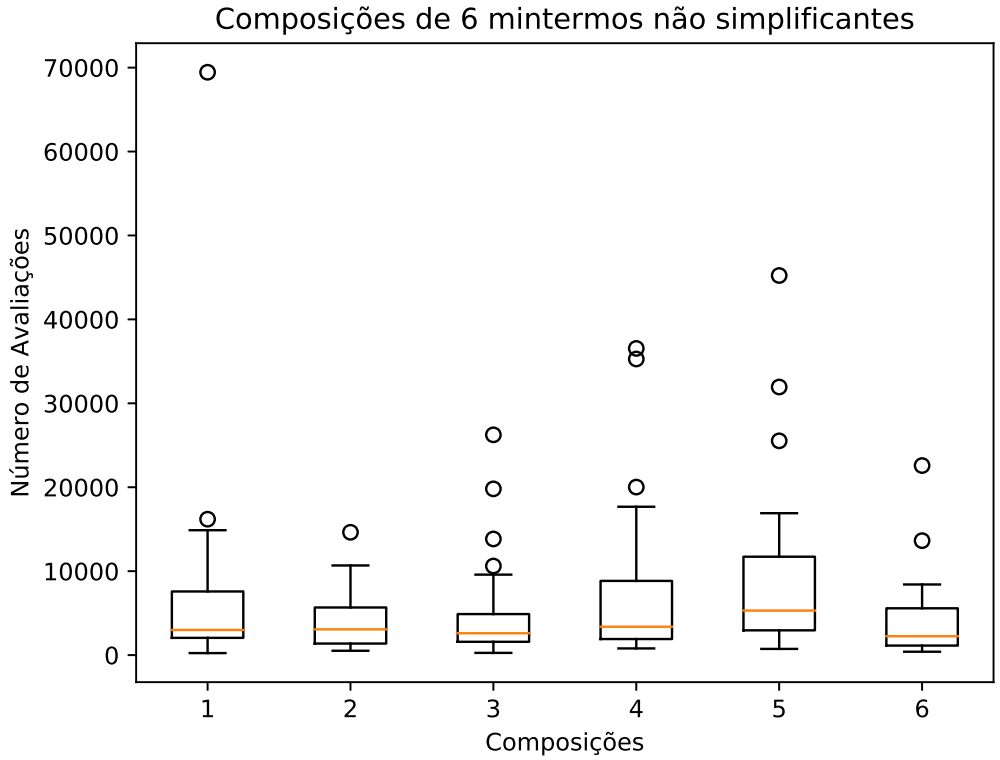


Figura 4.15: *Boxplots* das composições de 6 mintermos não simplificantes entre si.

tretanto, é importante ressaltar que as composições apresentam diferenças somente em 2 mintermos. Outro fato a ser destacado é que a Composição 5 difere somente em um mintermo da Composição 4 e resultou em comportamento similar. O mintermo que os difere é o $AB\overline{C}D$ para a Composição 4 e o $ABC\overline{D}$ para a Composição 5. Nos *boxplots* da Figura 4.11 da Seção 4.1.4 o mintermo $AB\overline{C}D$ é mais facilmente obtido do que o mintermo ABCD, o que leva a crer que a influência individual dos mintermos, em composições não simplificantes entre si, é levada em consideração quando abordadas em composições maiores diferentemente do ocorrido para as composições de 2 mintermos apresentadas na Seção 4.1.6.1 e de 4 mintermos apresentadas na Seção 4.1.6.2. Não é possível formar uma composição de 6 mintermos não simplificantes entre si utilizando os mintermos que obtiveram melhores resultados quando analisados individualmente, pois eles se tornam simplificantes. Desta forma, qualquer experimento que tente responder essa questão terá o viés da possibilidade de formação desta composição. Então pode-se concluir que em composições que possuem grande quantidade de mintermos a CGP tem seu desempenho interferido pela influência de cada mintermo separado quando estes estão presentes nas composições.

4.1.6.4 8 mintermos

Considerando que o problema exemplo é um circuito de 4 entradas, existem 16 combinações possíveis. Dado este fato só é possível criar duas composições de 8 mintermos não simplificantes entre si. Estas duas composições são apresentadas na Tabela 4.16.

Tabela 4.16: Composições de 8 mintermos não simplificantes entre si utilizados.

Comp	**Função**
1	$\overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}D + ABCD$
2	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}D + ABC\overline{D}$

Diferentemente dos comportamentos observados nas investigações de 2, 4 e 6 mintermos não simplificantes entre si, as composições de 8 mintermos apresentaram facilidade de serem obtidos pela CGP. Poucas avaliações foram necessárias e ambas obtiveram 100% de taxa de sucesso. A quantidade de avaliações necessárias para obter as duas composições são apresentadas na Tabela 4.17 e nos *boxplots* da Figura 4.16.

Como o comportamento das composições de 8 mintermos diferiu significativamente das composições de 2, 4 e 6, os resultados indicam que deve existir alguma relação não

Tabela 4.17: Número de avaliações necessárias para composições de 8 mintermos não simplificantes.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	22	250,25	523,5	1003	917,83	1166,65	5470	287	752,75	100%
2	27	268,5	482,5	1361,75	979,47	1118,09	4394	345,5	1093,25	100%

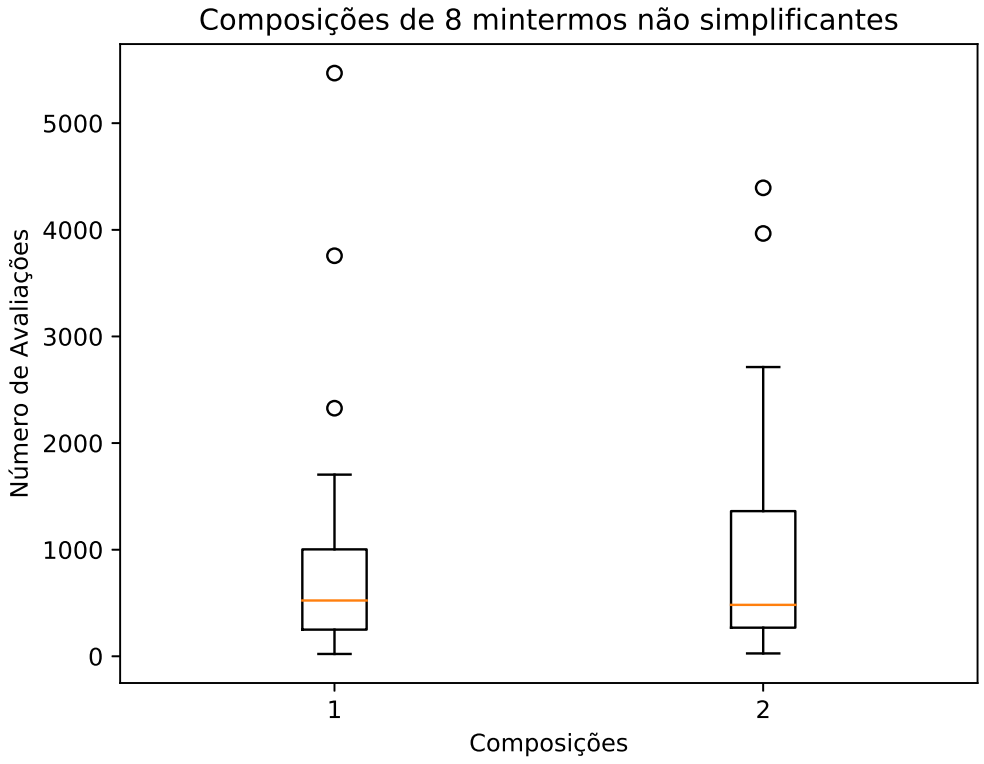


Figura 4.16: *Boxplots* das composições de 8 mintermos não simplificantes entre si.

só do balanceamento entre as quantidades de *bits* 0 e *bits* 1 na tabela verdade mas também em como estes *bits* estão dispostos na tabela. Essa suposição é discutida mais à frente. Entretanto é possível concluir que a CGP obtém soluções com maior facilidade em composições que utilizam muitos mintermos. Isso é refletido pela taxa de sucesso.

4.1.7 INVESTIGAÇÃO 6 - COMPOSIÇÕES SIMPLIFICANTES ENTRE SI

Nesta investigação propõe-se o uso de problemas cujos mintermos sejam agrupados de tal maneira que sejam simplificantes entre si. Entende-se por composições simplificantes aquelas cujos mintermos variam somente 1 *bit* entre si da mesma forma que o código de Gray. Esses agrupamentos permitem que essas composições formadas por diversos mintermos possam ser simplificadas e, consequentemente, resultem em expressões booleanas

mais simples do que as obtidas pela soma de produtos. O objetivo destas investigações é, portanto, verificar se a CGP apresenta melhor comportamento quando é possível simplificar as expressões dos problemas em questão ou se a disposição dos mintermos na tabela verdade não é levada em consideração durante a busca.

4.1.7.1 2 mintermos

Para analisar o comportamento da CGP com 2 mintermos simplificantes entre si foram utilizados 8 composições escolhidas de maneira aleatória conforme apresentado na Tabela 4.18. A CGP apresenta 100% de taxa de sucesso em todas as composições e obteve valores significativamente melhores em todas as métricas quando comparados às composições 2 mintermos não simplificantes entre si, chegando à uma melhora de quase 7 vezes na média do número de avaliações. A quantidade de avaliações necessárias para a obtenção destas composições é apresentada na Tabela 4.19 e nos *boxplots* da Figura 4.17. Aparentemente, quando a simplificação ocorre na variável mais significativa, os resultados são mais fáceis de serem obtidos. Além disso, os resultados indicam que o sucesso da CGP é influenciado pela disposição dos mintermos nas composições.

Tabela 4.18: Composições de 2 mintermos simplificantes entre si utilizados.

Composição	**Função**	**Simplificação**
1	$\overline{A}BC\overline{D} + A\overline{B}C\overline{D}$	$\overline{B}C\overline{D}$
2	$\overline{A}BC\overline{D} + \overline{A}BCD$	$\overline{A}BC$
3	$\overline{A}BC\overline{D} + \overline{A}B\overline{C}D$	$\overline{A}B\overline{D}$
4	$\overline{A}BC\overline{D} + \overline{A}B\overline{C}D$	$\overline{A}C\overline{D}$
5	$\overline{A}BC\overline{D} + \overline{A}BCD$	$\overline{A}BC$
6	$A\overline{B}CD + ABCD$	ACD
7	$AB\overline{C}\overline{D} + AB\overline{C}D$	$AB\overline{C}$
8	$\overline{A}B\overline{C}D + AB\overline{C}D$	$B\overline{C}D$

4.1.7.2 4 mintermos

Para as análises com 4 mintermos simplificantes entre si foram utilizadas 6 composições escolhidas de maneira aleatória conforme apresentado na Tabela 4.20. Todas as composições apresentaram 100% de taxa de sucesso e os valores obtidos foram significativamente melhores em todas as métricas quando comparados às composições 4 mintermos não simplificantes entre si, chegando a uma melhora de quase 23 vezes na média do número de

Tabela 4.19: Número de avaliações necessárias para as composições de 2 mintermos simplificantes.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	5	116,25	352	889,25	659,57	790,9	3562	291	773	100%
2	32	165	477	768,75	721	893,13	3594	310	603,75	100%
3	8	148,25	461	911	631,6	618,46	2431	364	762,75	100%
4	22	188	375,5	937,75	581,63	519,3	1974	290,5	749,75	100%
5	10	327,5	837,5	1243,5	866,7	626,23	2165	508,5	916	100%
6	19	110,75	373,5	680,75	482,03	456,27	1704	274,5	570	100%
7	19	125,5	388,5	642,75	525,9	595,93	2740	265,5	517,25	100%
8	17	118,5	236,5	708,75	417,77	390,57	1293	136	590,25	100%

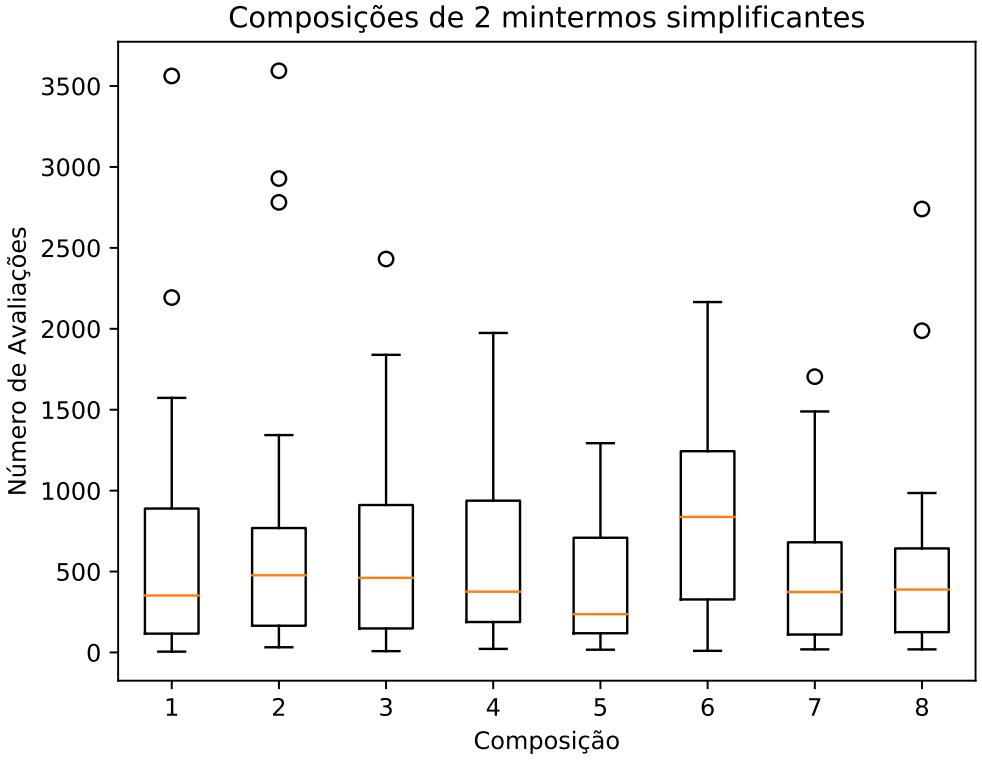


Figura 4.17: *Boxplots* das composições de 2 mintermos simplificantes entre si.

avaliações. A quantidade de avaliações necessárias para a obtenção destas composições é apresentado na Tabela 4.21 e nos *boxplots* da Figura 4.18. Novamente a CGP apresentou maior facilidade em obter resultados quando os mintermos são simplificantes entre si ressaltando a importância da disposição dos mintermos nas composições.

4.1.7.3 6 mintermos

Para as análises com 6 mintermos com blocos simplificantes entre si foram utilizadas 5 composições escolhidas de maneira aleatória conforme apresentado na Tabela 4.22. As

Tabela 4.20: Composições de 4 mintermos simplificantes entre si utilizados.

Composição	**Função**	**Simplificação**
1	$\overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}BC\overline{D} + \overline{A}BCD$	$\overline{A}C$
2	$\overline{A}\overline{B}\overline{C}D + \overline{A}BCD + A\overline{B}\overline{C}D + ABCD$	BD
3	$A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D} + ABCD$	AC
4	$\overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + A\overline{B}\overline{C}D + A\overline{B}CD$	$\overline{B}D$
5	$\overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}D$	$B\overline{C}$
6	$A\overline{B}\overline{C}D + A\overline{B}CD + AB\overline{C}D + ABCD$	AD

Tabela 4.21: Número de avaliações necessárias para as composições de 4 mintermos simplificantes.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	4	59,5	131,5	270,25	235,67	280,26	1095	88,5	210,75	100%
2	1	32,5	50,5	143,5	126,4	167,58	604	29	111	100%
3	25	80,5	126,5	241,75	167,33	142,18	734	80,5	161,25	100%
4	4	41,25	111,5	209,25	177,23	195,15	692	77	168	100%
5	11	82	197,5	295,5	236,3	211,97	907	117,5	213,5	100%
6	4	18,5	53	321,5	158,6	196,52	653	46,5	303	100%

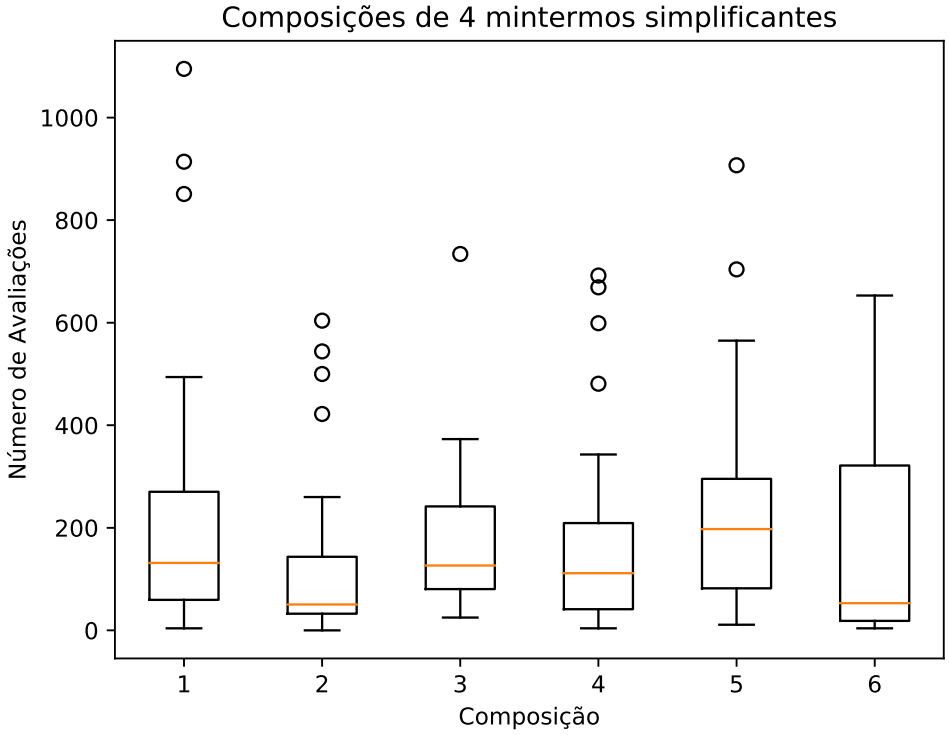


Figura 4.18: *Boxplots* das composições de 4 mintermos simplificantes entre si.

simplificações de expressões boolenas na eletrônica digital levam em consideração os agrupamentos que são múltiplos de potência de dois e, portanto, não é possível criar uma única composição que utilize 6 mintermos e que seja totalmente simplificável. Desta forma o

objetivo desta investigação é justamente analisar como a CGP realiza as simplificações entre os mintermos, pois é possível agrupar tanto 3 pares de mintermos quando 1 bloco de 4 mintermos e um par de mintermos.

Tabela 4.22: Composições de 6 mintermos simplificantes entre si utilizados.

Composição	**Função**	**Simplificação**
1	$A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}D + ABC\overline{D} + ABCD$	AD + AC
2	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD + AB\overline{C}D + ABC\overline{D} + ABCD$	BD + BC
3	$\overline{A}B\overline{C}\overline{D} + \overline{A}BC\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + ABC\overline{D}$	$B\overline{D} + A\overline{D}$
4	$\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD$	$\overline{A}D + \overline{A}C$
5	$\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD$	$\overline{B}D + \overline{B}C$

Para todas as composições a CGP apresentou 100% de taxa de sucesso e obtiveram valores melhores em todas as métricas quando comparados às composições 6 mintermos não simplificantes entre si, chegando a uma melhora de quase 8 vezes na média do número de avaliações. A quantidade de avaliações necessárias para a obtenção destas composições é apresentada na Tabela 4.23 e nos *boxplots* da Figura 4.19. Os resultados indicam que a CGP é capaz de agrupar os blocos de simplificação das duas maneiras citadas anteriormente: 1 bloco de 4 mintermos e 1 par e também em 3 pares de mintermos. Isso indica que a possibilidade de haver simplificação facilita a busca da CGP.

Tabela 4.23: Número de avaliações necessárias para as composições de 6 mintermos simplificantes.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	16	136,75	262	789,75	555,43	622,09	2634	182	653	100%
2	12	138,75	423	600,25	784,2	1090,74	4500	282,5	461,5	100%
3	78	313,75	784	1079,75	999,63	1059,37	4602	466,5	766	100%
4	10	248	401,5	1308,75	1095,63	1819,89	9735	361	1060,75	100%
5	10	248	401,5	1308,75	1095,63	1819,89	9735	361	1060,75	100%

As composições 1 e 2 foram as que apresentaram melhores resultados quando comparados às demais composições. Estas duas composições são bastante similares diferindo somente no fato da variável mais significativa ser A na composição 1 e B na composição 2. Os resultados indicam que de fato quando existem variáveis mais significativas nas simplificações a CGP é capaz de obtê-las mais facilmente, conforme destacado anterior-

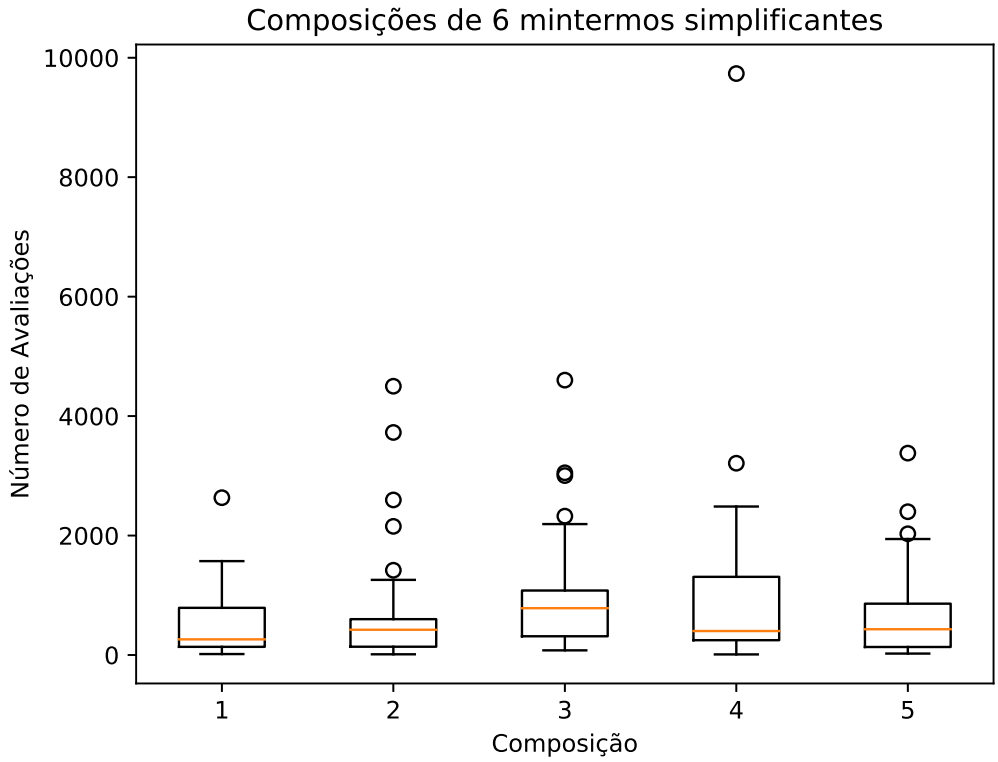


Figura 4.19: *Boxplots* das composições de 6 mintermos simplificantes entre si.

mente. A composição 5 obteve comportamento similar e também possui a característica de manter a variável mais significativa ($\overline{B}$).

4.1.7.4 8 mintermos

Para as análises com 8 mintermos simplificantes entre si foram utilizadas 5 composições escolhidas de maneira aleatória conforme apresentado na Tabela 4.24. Essa investigação torna-se um pouco redundante pois ela representa, de fato, uma função que expressa uma entrada primária do programa ou seu complemento, conforme investigado na Seção 4.1.3. Entretanto os dados aqui apresentados servem como comparação direta com os blocos de 8 mintermos não simplificantes entre si.

Para todas as composições a CGP apresentou 100% de taxa de sucesso e obtiveram valores significativamente melhores em todas as métricas quando comparados às composições 8 mintermos não simplificantes entre si, chegando a uma melhora de quase 14 vezes na média do número de avaliações. A quantidade de avaliações necessárias para a obtenção destas composições é apresentado na Tabela 4.24 e nos *boxplots* da Figura 4.20. Novamente os dados indicam que a CGP se comporta melhor quando os mintermos estão

Tabela 4.24: Composições de 8 mintermos simplificantes entre si utilizados.

Composição	**Função**	**Simplificação**
1	$A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + A\overline{B}CD +$ $AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D} + ABCD$	A
2	$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}CD +$ $A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + AB\overline{C}D$	$\overline{C}$
3	$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}CD +$ $\overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}BCD$	$\overline{A}$
4	$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}CD +$ $A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + ABC\overline{D}$	$\overline{D}$
5	$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}CD +$ $A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + A\overline{B}CD$	$\overline{B}$

dispostos de tal maneira que a expressão do circuito seja simplificável.

Tabela 4.25: Número de avaliações necessárias para as composições de 8 mintermos simplificantes.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	1	14,5	22	89,75	62,43	78,29	340	18,5	75,25	100%
2	7	30,75	57	103,5	85,6	91,5	430	35,5	72,75	100%
3	1	22,5	51,5	106,25	79,27	84,13	327	45	84	100%
4	1	11	39,5	73,75	51,53	50,92	202	31,5	62,75	100%
5	1	12,75	42,5	106,75	64,27	75,58	344	34,5	94	100%

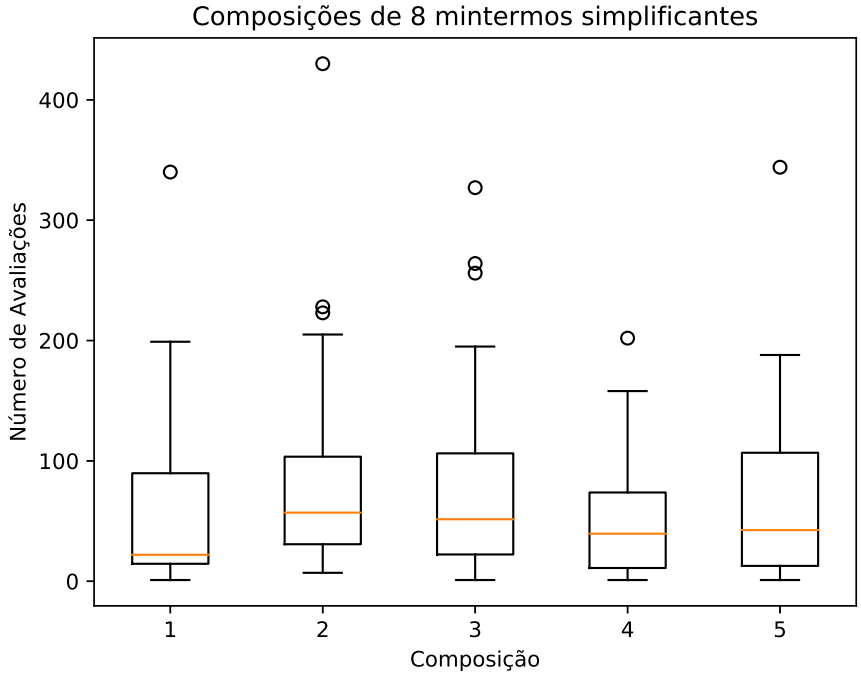


Figura 4.20: *Boxplots* das composições de 8 mintermos simplificantes entre si.

4.1.8 INVESTIGAÇÃO 7 - COMPOSIÇÕES HÍBRIDAS

As composições híbridas envolvem diferentes agrupamentos de mintermos que podem ser simplificados. O objetivo destas investigações é analisar como a CGP se comporta e quais são os agrupamentos que ela utiliza para realizar as simplificações.

4.1.8.1 Bloco de 4 mintermos e um bloco de 2 mintermos

Para composições de 4 mintermos e um bloco de 2 mintermos a CGP apresentou facilidade em encontrar soluções. Para todas as 4 composições apresentadas na Tabela 4.26 a CGP obteve 100% de taxa de sucesso. O número de avaliações necessárias para obter cada uma das composições é apresentado na Tabela 4.27 e nos *boxplots* da Figura 4.21.

Tabela 4.26: Composições de 4 e 2 mintermos não simplificantes entre si utilizados.

Composição	**Função**	**Simplificação**
1	$\overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}\overline{D} + AB\overline{C}D$	$A\overline{C} + \overline{A}BC$
2	$A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}\overline{D} + AB\overline{C}D + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D$	$A\overline{C} + \overline{A}BC$
3	$\overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD$	$B\overline{C} + A\overline{B}C$
4	$\overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD$	$\overline{A}C + A\overline{B}C$

Tabela 4.27: Número de avaliações necessárias para as composições de 4 e 2 mintermos não simplificantes entre si.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	188	428,75	768	1648,75	1316,27	1346,16	5842	406	1220	100%
2	80	352	735,5	1314,25	1037,8	1000,68	4533	416	962,25	100%
3	106	504	932	2748,25	2051,93	2553,22	10557	681,5	2244,25	100%
4	79	326,25	701	1510,5	1310,43	1628,99	8310	540	1184,25	100%

Para as quatro composições a CGP apresentou comportamentos semelhantes entre si, principalmente quando a mediana é analisada. Aparentemente, a composição 3 é a mais difícil das 4 mas o motivo é desconhecido. Além disso, as composições utilizadas nesta investigação mostraram-se mais difíceis de serem obtidas do que as composições de 2 mintermos simplificantes entre si, bem como as composições de 4 mintermos simplificantes entre si.

Algumas coisas importantes merecem destaque no processo evolutivo, dentre elas na maior parte das execuções, aproximadamente 40%, a CGP primeiramente obteve o bloco

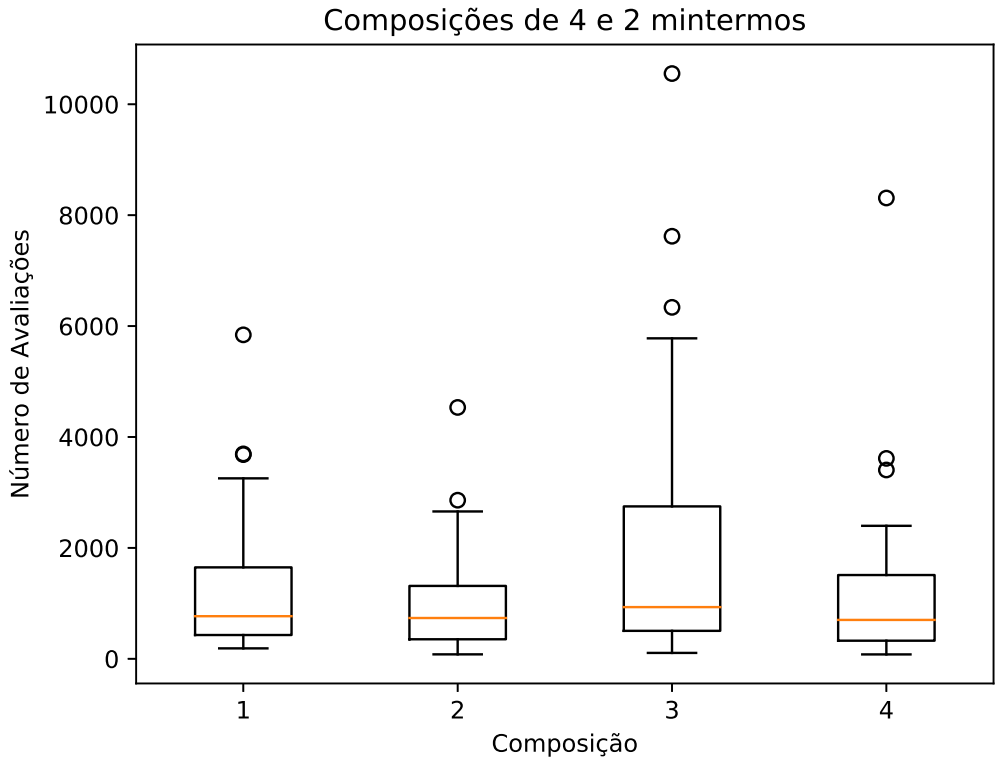


Figura 4.21: *Boxplots* das composições de 4 e 2 mintermos não simplificantes entre si.

de 4 mintermos para depois obter o bloco de 2 mintermos. Além disso, em aproximadamente 10% das execuções a CGP obteve primeiramente o bloco de 2 mintermos, e em 13% obteve uma tabela verdade somente com *bits* 1, isto é, F = 1.

4.1.8.2 Bloco de 4 mintermos e um mintermo isolado

Nesta investigação o objetivo é analisar o comportamento da CGP quando adiciona-se um mintermo cuja ausência faria com que a função *booleana* fosse bastante simplificável. No caso, tendo 4 mintermos simplificantes entre si com 4 variáveis de entrada, obtém-se uma simplificação cujo resultado é uma função de 2 variáveis. Entretanto, ao incluir um novo mintermo, a simplificação máxima que pode ocorrer é a dos 4 mintermos que resultam em uma função de 2 variáveis e o próprio mintermo isolado. Para esta investigação foram utilizadas 5 composições que são apresentadas na Tabela 4.28.

A CGP apresentou dificuldade em algumas composições e só foi capaz de obter 100% de circuitos funcionais na composição 1. O número de avaliações necessárias para obter cada uma das composições é apresentado na Tabela 4.29 e nos *boxplots* da Figura 4.22.

As quatro composições apresentaram comportamento bastante semelhante como é pos-

Tabela 4.28: Composições de 4 mintermos com 1 mintermo isolado utilizados.

Composição	**Função**	**Simplificação**
1	$\overline{A}BC\overline{D} + \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + A\overline{B}C\overline{D}$	$\overline{A}C + A\overline{B}C\overline{D}$
2	$\overline{A}BC\overline{D} + \overline{A}BCD + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D} + ABC\overline{D}$	$\overline{A}C + ABC\overline{D}$
3	$\overline{A}BC\overline{D} + \overline{A}BCD + ABC\overline{D} + ABCD + \overline{A}B\overline{C}\overline{D}$	$BC + \overline{A}B\overline{C}\overline{D}$
4	$\overline{A}BC\overline{D} + \overline{A}BCD + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D}$	$\overline{B}C + ABC\overline{D}$

Tabela 4.29: Número de avaliações necessárias para as composições de 4 mintermos com 1 mintermo isolado.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	59	2165,5	10222	23018,25	16603,67	19463,03	74749	884,5	20852,75	100%
2	308	4266,25	11863	27584,25	20122,29	23079,12	98258	10206	23318	93,33%
3	328	2471	8087	23929	18979,83	24037,95	78839	7078	21458	96,67%
4	585	4254,5	10698	26515,5	22043,57	24744,76	99264	8827,5	22261	93,33%

sível perceber nos *boxplots* da Figura 4.22. Conforme esperado, o fato da função utilizada não ser completamente simplificável resultou no baixo desempenho da CGP em obter circuitos factíveis. É importante ressaltar que nesta investigação a CGP primeiramente obtinha o bloco de 4 mintermos e depois concentrava os esforços na busca pelo mintermo isolado.

4.1.8.3 Bloco de 8 mintermos com um mintermo retirado

Esta investigação é semelhante à anterior. Entretanto, neste caso a simplificação que resultaria em um único literal será transformado em uma função composta por um somatório de 3 produtos de 2 variáveis. As composições utilizadas para esta investigação são apresentados na Tabela 4.30. O número de avaliações necessárias para obter cada uma das composições é apresentado na Tabela 4.31 e nos *boxplots* da Figura 4.23.

Tabela 4.30: Composições de 8 mintermos com um mintermo retirado.

Composição	**Função**	**Simplificação**
1	$\overline{A}B\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}D$	$\overline{C}D + A\overline{C} + \overline{B}C$
2	$\overline{A}B\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + AB\overline{C}\overline{D} + AB\overline{C}D + ABC\overline{D}$	$A\overline{B} + A\overline{C} + A\overline{D}$
3	$\overline{A}B\overline{C}\overline{D} + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD$	$A\overline{B} + \overline{B}D + \overline{B}C$
4	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + A\overline{B}C\overline{D} + A\overline{B}CD + AB\overline{C}D + ABCD$	$\overline{C}D + AD + \overline{B}D$
5	$\overline{A}B\overline{C}D + \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}CD + AB\overline{C}D + ABCD$	$CD + \overline{A}D + BD$

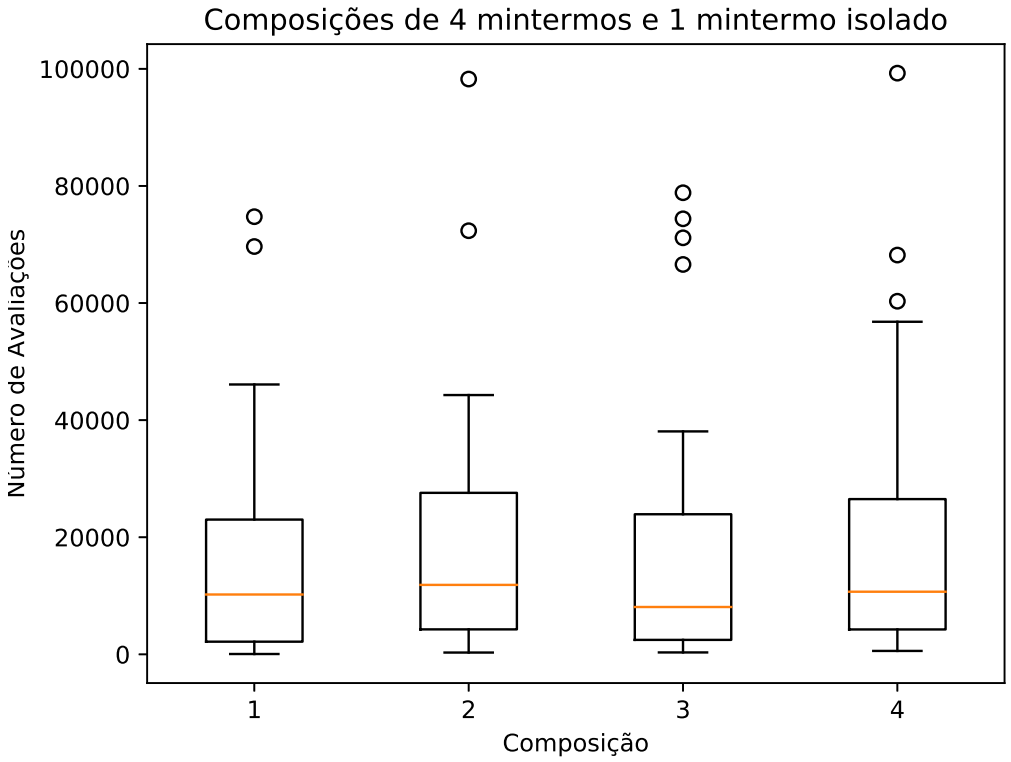


Figura 4.22: *Boxplots* das composições de 4 mintermos com 1 mintermo isolado.

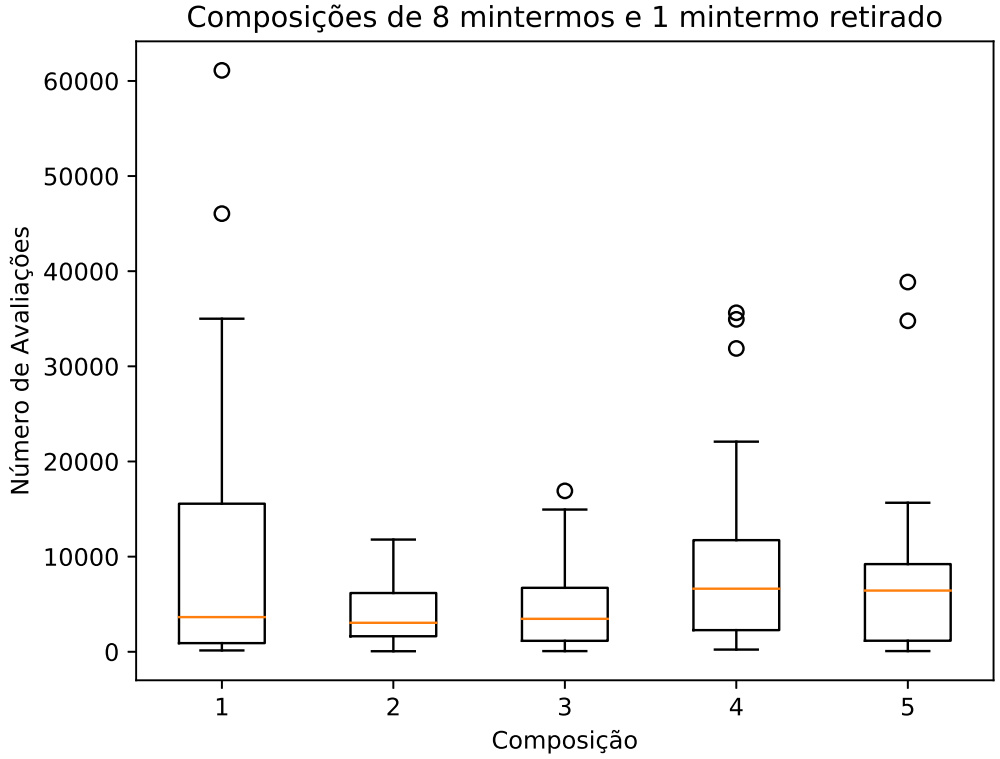


Figura 4.23: *Boxplots* das composições de 8 mintermos com 1 mintermo retirado.

Tabela 4.31: Número de avaliações necessárias para as composições de 8 mintermos com 1 mintermo retirado.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	138	906,5	3641,5	15562	11199,07	15516,79	61110	3230,5	14655,5	100%
2	54	1629	3044	6173,5	4116,77	3500,85	11793	1652	4544,5	100%
3	68	1149,25	3460,5	6720	4572,07	4279,22	16910	2696,5	5570,75	100%
4	229	2272,75	6628,5	11731,5	9476,57	10089,56	35631	4491	9458,75	100%
5	71	1159	6431,5	9213	7544,87	9197,45	38861	5190,5	8054	100%

É possível perceber que a composição 1 foi mais difícil de ser obtida do que as demais, apesar das medianas estarem relativamente próximas e a CGP obter 100% de taxa de sucesso em todas as composições. A única peculiaridade que a composição 1 apresenta é o fato de que todos os termos da simplificação possuírem o literal C negado. Em outros testes realizados, essa característica da composição 1 não se mostrou um fator que afete negativamente o desempenho da CGP, pois a facilidade estava ligada à variável mais significativa.

4.1.9 INVESTIGAÇÃO 8 - ESTUDOS DA TABELA 4.1

Nesta investigação, diversos dos mintermos cujo nível lógico era 0 foram transformados em nível lógico 1, um de cada vez, a fim de formar blocos maiores de simplificação. Após isso, o comportamento através da transformação de dois e depois três mintermos cujo nível lógico era 0 para nível lógico 1 foram analisados. Os mintermos adicionados são mostrados na Tabela 4.32. O número de avaliações necessárias para obter cada uma das composições é apresentado na Tabela 4.33 e nos *boxplots* da Figura 4.24.

Os resultados mostram que quando os mintermos da Investigação 10 são adicionados, obtém-se a tabela verdade mais fácil de ser obtida pela CGP.

Apesar do mintermo adicionado na composição 4 auxiliar na criação de um bloco de 4 mintermos simplificantes entre si a CGP teve certa dificuldade em obtê-lo. Entretanto, para todas as composições quando comparadas com a primeira (tabela verdade proposta inicialmente), a CGP foi capaz de obter 100% de circuitos factíveis devido à adição dos mintermos que auxiliam nas simplificações.

Tabela 4.32: Investigações em modificações feitas na Tabela 4.1.

Investigação	**Adicionados**	**Função**
1	-	$\overline{A}CD + \overline{B}\overline{C}D + AB\overline{C} + BCD + \overline{A}CD + \overline{A}B\overline{D} + A\overline{B}C\overline{D}$
2	$A\overline{B}CD$	$A\overline{C} + \overline{B}\overline{C} + \overline{A}CD + BCD + \overline{A}B\overline{D} + A\overline{B}D$
3	$\overline{A}B\overline{C}D$	$\overline{A}B + \overline{C}D + \overline{A}\overline{C} + \overline{A}D + B\overline{C} + BD + A\overline{B}C\overline{D}$
4	$A\overline{B}CD$	$AD + \overline{B}D + \overline{A}C\overline{D} + B\overline{C}D + A\overline{B}C + \overline{A}BC$
5	$\overline{A}B\overline{C}\overline{D}$	$\overline{A}\overline{B} + \overline{A}C + \overline{A}\overline{D} + A\overline{C}D + AB\overline{C} + BCD + \overline{B}C\overline{D}$
6	$ABC\overline{D}$	$AB + BC + B\overline{D} + \overline{B}\overline{C}D + AC\overline{D} + \overline{A}C\overline{D} + \overline{A}CD$
7	$A\overline{B}C\overline{D} + \overline{A}B\overline{C}D$	$\overline{C} + B\overline{A}D + A\overline{D}$
8	$\overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D}$	$\overline{A} + B + \overline{C}D + C\overline{D}$
9	$\overline{A}B\overline{C}\overline{D} + A\overline{B}CD$	$C + \overline{A}\overline{B} + AB + \overline{A}\overline{D} + AD$
10	$A\overline{B}C\overline{D} + \overline{A}B\overline{C}D + \overline{A}B\overline{C}\overline{D}$	$\overline{C} + \overline{A} + BC + \overline{B}\overline{D}$
11	$A\overline{B}CD + \overline{A}B\overline{C}\overline{D} + A\overline{B}C\overline{D}$	$\overline{B} + \overline{C}\overline{D} + CD + \overline{A}C + A\overline{C}$

4.2 CONCLUSÕES PRELIMINARES

A CGP é uma ferramenta largamente utilizada pela literatura no projeto de circuitos lógicos combinacionais (CLCs) e seus resultados mostram-se competitivos com os das técnicas tradicionais. Entretanto o problema da escalabilidade continua sendo um gargalo para a evolução de circuitos mais complexos. Apesar disso a CGP encontra maior facilidade quando o projeto envolve circuitos cujas expressões podem ser simplificadas. Isso é de se esperar, pois quando as funções são simplificadas, não é necessário explorar tanto o espaço

Tabela 4.33: Número de avaliações necessárias para as variações da Tabela 4.1.

Comp	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
1	875	3072,75	5635	110722,75	11713,71	15657,66	65144	3223,5	7650	93,33%
2	473	2941,75	4685,5	10377,25	6679	5298,72	18970	2339,5	7435,5	100%
3	318	2177,75	2660	3906	3655,3	3040,6	14376	691	1728,25	100%
4	554	3485	6205	9873,25	7239,53	5437,21	22235	3558,5	6388,25	100%
5	405	2010,75	3561,5	5790,5	4817,2	4155,78	18918	1891	3779,75	100%
6	580	2049,5	3713	7328,5	5695	4843,05	17224	2358	5279	100%
7	472	2461,5	4149,5	5707	7384,33	13000,78	70655	1691,5	3245,5	100%
8	109	1569	2847	4997,75	3634,73	3120,57	10699	1719,5	3428,75	100%
9	306	3556,25	4788,5	8425,75	6804,8	5200,83	21201	1910,5	4869,5	100%
10	182	894,25	1704	2329,75	1865,03	1320,57	5567	806,5	1435,5	100%
11	235	1676	3850,5	6165	4642,2	3759,17	16183	2303	4489	100%
12	74	811,5	1433	3781,75	2588,13	2612,77	9717	1084,5	2970,25	100%

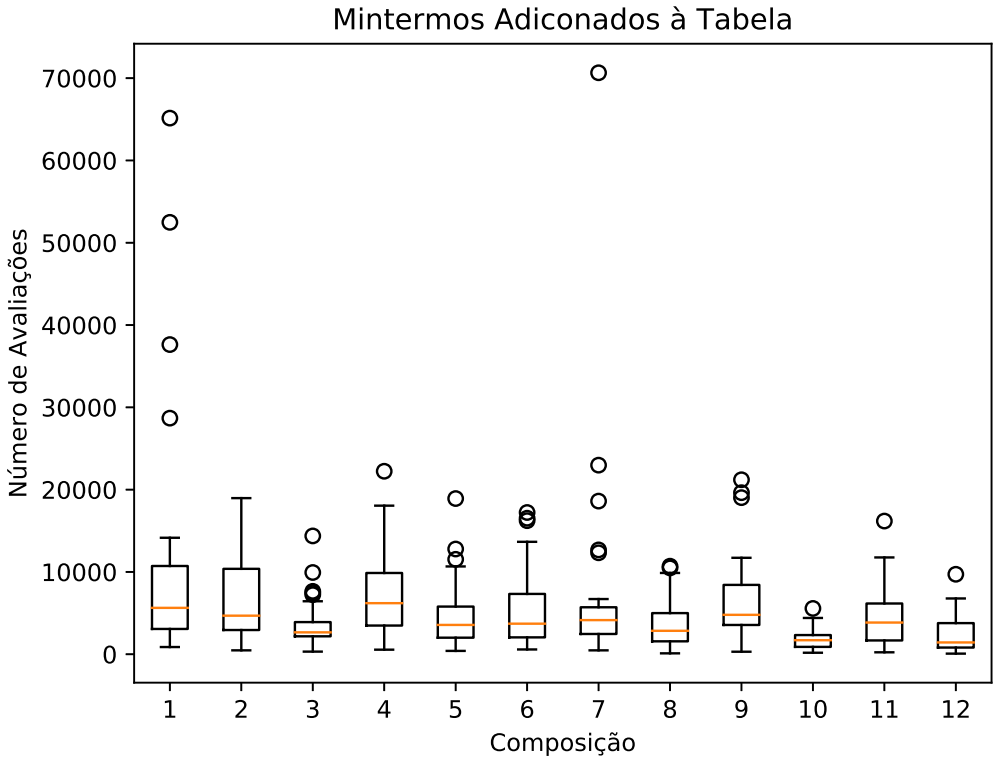


Figura 4.24: Número de avaliações necessárias para obter a tabela verdade proposta com todas as modificações.

de busca quanto quando existem diversas combinações diferentes, conforme levantado nos experimentos da Seção 4.1.4 (página 77) e reforçado nos experimentos da Seção 4.1.7 (página 89), pois a expressão desejada é reduzida para um problema que envolve menor quantidade de combinações.

Além disso, conforme relatado na literatura, a CGP gera projetos de circuitos não convencionais. Poucas vezes são vistas simplificações comumente utilizadas na eletrônica digital nos circuitos obtidos pela CGP. Existe também uma forte tendência na utilização de portas XOR, a qual muitas vezes foge à concepção do projetista humano.

Os experimentos mostraram uma característica importante sobre o desbalanceamento de dados. Quando existem muitos representantes de uma única categoria (muitos zeros ou muitos uns), a CGP tende a obter inicialmente uma solução de F = 0 ou F = 1 para depois acertar os demais, conforme apresentado na Seção 4.1.4 (página 77) e na Seção 4.1.5 (página 80). Entretanto mesmo que os dados das saídas do circuito estejam balanceados, o desempenho da CGP não é afetado. Dados balanceados como na investigação de entradas primárias ou seus complementos apresentados na Seção 4.1.3 (página 75) e dados não balanceados como nas investigações simplificantes entre si de 2 e 4 mintermos das Seções 4.1.7.1 (página 90) e 4.1.7.2 (página 90) foram problemas nos quais a CGP obteve bom desempenho. Isso por que depende da disposição dos dados uma vez que quando dispostos de maneira que criem blocos de simplificação apresenta maior facilidade, conforme observado na investigação de 8 mintermos simplificantes e não simplificantes entre

si apresentados na Seção 4.1.7.4 (página 94) e Seção 4.1.6.4 (página 88) respectivamente. Então, de maneira geral, a CGP apresenta facilidade quando os mintermos existentes formam blocos que são simplificáveis e apresenta dificuldades quando esses mintermos não se simplificam. Além disso, a CGP sempre se mostrou eficiente para obter circuitos cuja tabela verdade representa F = 0, F = 1 ou quando a função expressa alguma entrada primária do circuito inclusive para maior número de entradas conforme apresentado na Seção 4.1.2 (página 74).

Ainda, parece haver relação entre a capacidade de obter soluções via CGP e o problema em questão, quando a simplificação ocorre na variável mais significativa (a variável que apresenta maior ordem de grandeza no sistema binário). Os resultados indicam que estes são mais fáceis de serem obtidos, isto é, quanto mais significativa é a variável simplificada, melhores são os resultados. Isso aparenta ser válido para blocos de composições simplificantes de 2, 4, 6 e 8, apresentados na Seção 4.1.7 (página 89).

Portanto, percebe-se que a CGP apresenta maior facilidade para obter circuitos factíveis quando as funções que os descrevem são simplificáveis. Como as funções são mais simples, os circuitos que as realizam também são mais simples e, portanto, existe menor necessidade de exploração do espaço de busca. Desta maneira, pode-se concluir que a CGP apresenta facilidade em encontrar soluções cujas tabelas verdade são simplificáveis.

Como não se pode modificar a tabela verdade desejada, uma vez que o circuito resultante também será diferente do procurado, pode-se, entretanto, utilizar o multiplexador a fim de facilitar a obtenção da mesma. Como os multiplexadores são capazes de dividir a tabela verdade de acordo com seu número de entradas, a tabela desejada pode ser obtida através de vários subcircuitos diferentes conectados às entradas do multiplexador e controlados através de um outro subcircuito conectado ao controle do multiplexador. Desta maneira propõe-se duas abordagens que utilizam multiplexadores:

- adicionar o multiplexador como elemento lógico no conjunto de funções da CGP: considerando um multiplexador de 2 entradas teremos um elemento lógico de 3 entradas (entrada A, entrada B e controle) e uma saída, diferentemente das portas lógicas binárias ou unárias utilizadas anteriormente. Através disso, podem-se criar tabelas verdade intermediárias que não são obtidas com as portas lógicas tradicionais durante a exploração do espaço de busca e que pode resultar em aumento da obtenção de circuitos factíveis. Além disso, pode-se tirar proveito da geração de

don't cares no circuito de controle.

- utilizar o multiplexador acoplado em cada uma das saídas do circuito e fazer uma evolução em 3 etapas:
  - obter subcircuitos nas entradas de tal maneira que, quando consideradas em conjunto, satisfaçam a tabela verdade (uma das entradas do multiplexador deve ser igual à desejada, para toda a função);
  - maximizar o número de correspondências (situação na qual as entradas do multiplexador são iguais entre si e iguais à função desejada) entre os subcircuitos das entradas pois isso aumenta o número de *don't cares* no circuito de controle; e
  - obter o subcircuito de controle.

Estas propostas são detalhadas nas Seções 5.4 (página 116) e 5.5 (página 117) respectivamente.

5 MÉTODOS PROPOSTOS

Neste capítulo, são apresentados os métodos propostos que consistem em (i) um operador de recombinação que une os melhores subgrafos dos indivíduos da CGP (SILVA; BERNARDINO, 2018), (ii) um operador de mutação guiada que atua no pior subgrafo do indivíduo, (iii) uma abordagem que modifica a estratégia evolutiva comumente usada na CGP, e (iv) duas abordagens que utilizam multiplexadores.

5.1 OPERADOR DE RECOMBINAÇÃO

O desenvolvimento de um operador de recombinação eficiente para a CGP tem sido amplamente investigado, mas não existe uma grande quantidade de abordagens utilizando este tipo de operador quando aplicado ao projeto de circuitos lógicos combinacionais (MILLER, 2011; HUSA; KALKREUTH, 2018).

Durante a busca de factibilidade pela CGP a função de aptidão é calculada em termos da quantidade de *bits* que são acertados da tabela verdade a qual deve-se maximizar. Uma vez que o erro é zero, isto é, todos os *bits* são os desejados, o indivíduo é considerado factível.

Considerando um circuito de 3 entradas e 3 saídas, tem-se $2^3 = 8$ *bits* para cada saída, totalizando 24 *bits* que deverão ser acertados. A função de aptidão leva em consideração a aptidão global do circuito, isto é, a soma dos acertos de todas as saídas deste circuito. Entretanto, o processo evolutivo é capaz de, em alguns momentos, melhorar significativamente uma única saída e piorar as demais de tal maneira que esse ganho na aptidão não seja o suficiente para manter o indivíduo na população. Tendo em vista este aspecto, o operador de recombinação proposto (chamado aqui de X-CGP) cria um super indivíduo composto pelos subgrafos que codificam as melhores saídas dentre toda população. Desta forma, não é necessário um mecanismo de seleção por torneio, pois o super indivíduo será sempre melhor ou equivalente ao melhor indivíduo da população. O operador X-CGP é aplicado durante a fase de seleção parental e, portanto, não necessita de uma taxa de recombinação.

A abordagem é similar àquela proposta em (WALKER *et al.*, 2006), entretanto a nossa proposta é para a evolução de todo o circuito em um único cromossomo, maximizando

indiretamente o compartilhamento de nós ativos e gerando soluções potencialmente mais compactas em termos de área e número de elementos lógicos. Essa maximização de compartilhamento de nós acontece, pois como todas as saídas do circuito são codificadas em um único cromossomo e a maior parte dos nós ativos é concentrada próxima às entradas primárias do circuito, estas acabam por servir como base para os nós ativos que se encontram mais adiante do cromossomo. Uma modificação num nó próximo das entradas geralmente leva a grandes variações fenotípicas no indivíduo.

Walker *et al.* (2006) utilizaram representações multi-cromossômicas da CGP, que diferem somente na divisão de genótipos por saída requerida pelo problema, isto é, diferentemente da CGP padrão de cromossomo único nas abordagens multi-cromossomos para cada saída do programa existe um genótipo que a codifica.

O X-CGP funciona da seguinte maneira:

1. Para o pai e a descendência: determinar cm_o, o número de correspondências corretas em relação à tabela verdade para cada saída o. Esse valor está entre 0 e 2^{ni}, onde ni é o número de entradas. Esses valores são armazenados para cada indivíduo.

2. Identificar o indivíduo com o maior valor de cm_o. Essa solução é o indivíduo base para o procedimento de recombinação.

3. Para cada saída j do indivíduo base na qual este não possui a melhor saída, incorporar o subgrafo utilizado para computar a saída correspondente a partir do indivíduo com o melhor cm_j no indivíduo base.

O crossover proposto cria um indivíduo que é composto pelos subgrafos que codificam as melhores saídas presentes nos indivíduos da população atual. Os indivíduos com o maior número de correspondências em relação à tabela verdade para cada saída são selecionados (*best_inds*). O indivíduo base (*best_ind*) é aquele, dos selecionados anteriores, com o maior número de melhores saídas. Com essa solução candidata é possível determinar quais saídas serão recombinadas. Os subgrafos do indivíduo base que não estão nos melhores indivíduos previamente selecionados são então substituídos por aqueles cujos subgrafos geram as melhores saídas. Esse procedimento de substituição é denominado aqui de incorporação. Um pseudocódigo do X-CGP é apresentado no Algoritmo 1 e o fluxograma na Figura 5.1.

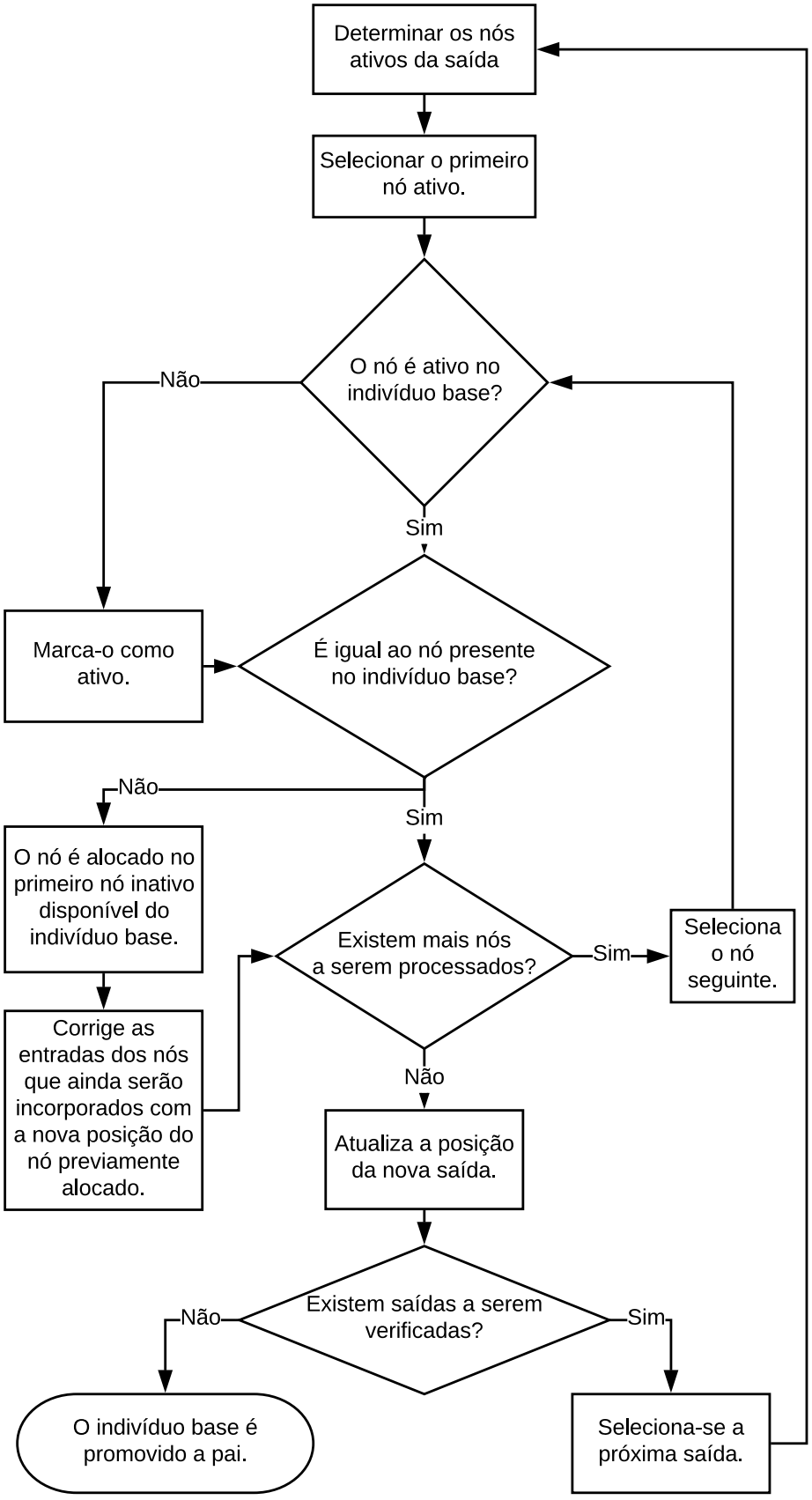


Figura 5.1: Fluxograma do X-CGP.

Algoritmo 1: O crossover proposto.

```
best_inds ← indivíduos com a maior quantidade de correspondências corretas para
  cada saída i;
best_ind ← indivíduo mais frequente em best_inds;
for cada saída i do
    if best_ind não está em best_inds_i then
        active_path ← todos os nós ativos da i-ésima saída do best_inds_i;
        for cada índice de nó k ∈ active_path do
            if nó k ≠ k-ésimo nó de best_ind then
                available_node ← índice do primeiro nó inativo de best_ind após o
                  índice k;
                best_ind_available_node ← nó k;
            end if
        end for
    end if
end for
```

Dois indivíduos com 3 entradas (I0, I1, and I2) e três saídas (O0, O1, and O2) são utilizados aqui pare representar o X-CGP, como mostrado na Figura 5.2 (a) e (b). Os círculos numerados representam os nós do indivíduo. Nós em cinza são ativos e os brancos inativos. As setas representam as conexões entre os nós. Neste sentido, o caminho para a saída O2 do primeiro indivíduo é formado por I2 e os nós 8 e 11. Neste exemplo, cada saída apresenta correspondências em relação à tabela verdade de acordo com a Tabela 5.1.

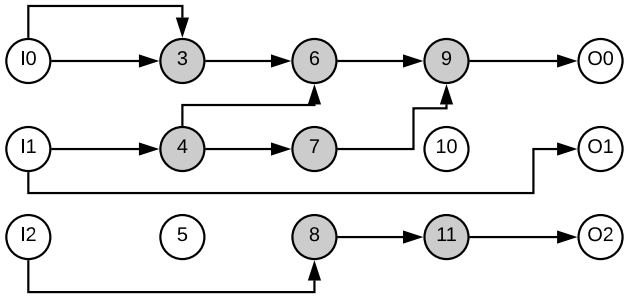
Tabela 5.1: Correspondências em relação à tabela verdade.

Indivíduo	O0	O1	O2	Fitness
1	7	3	6	16
2	5	8	4	17

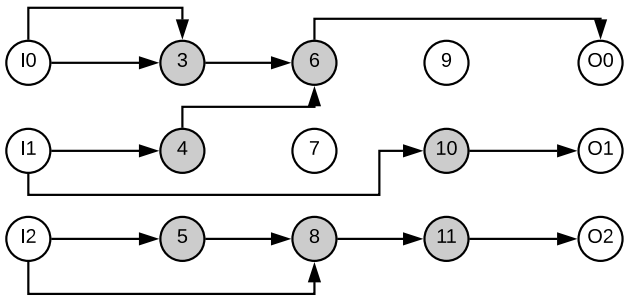
O primeiro indivíduo apresenta as melhores saídas O0 e O2. A saída O1 é melhor no segundo indivíduo. Então, o indivíduo base do X-CGP é o primeiro e o caminho para a saída O1 do segundo indivíduo será incorporado neste indivíduo base.

O indivíduo resultante é apresentado na Figura 5.2 (c), onde os nós em verde são os originais do indivíduo base (primeiro indivíduo) e o nó azul é aquele cujo subgrafo foi incorporado ao indivíduo base a partir do segundo indivíduo. As linhas tracejadas formam o novo caminho a fim de calcular a saída O1 do novo indivíduo.

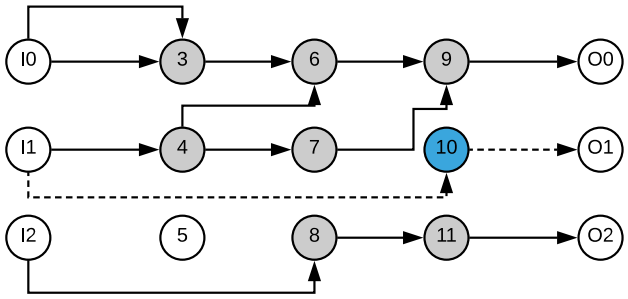
Vale ressaltar na Tabela 5.1 que a CGP tradicional escolheria o segundo indivíduo, uma vez que é o melhor indivíduo da população com 17 correspondências em relação à



(a) Indivíduo um.



(b) Indivíduo dois.



(c) Indivíduo resultante.

Figura 5.2: (a) e (b) são dois indivíduos da população. (c) é o novo indivíduo gerado pelo *crossover* proposto; a saída O1 é do segundo indivíduo (b) é incorporado para o genótipo do primeiro (a).

tabela verdade. Na CGP, este indivíduo sobreviveria para a próxima geração. Por outro lado, o operador de recombinação proposto gera um novo indivíduo que acerta 21 dos 24 *bits* da tabela verdade.

O procedimento de incorporação dos subgrafos é a parte principal do X-CGP. Na Figura 5.3, considere agora dois indivíduos onde O0 (sugrafo vermelho) é melhor no indivíduo um, O1 (subgrafo verde) é igual em ambos e O2 (subgrafo azul) é melhor no indivíduo dois, e que o indivíduo base selecionado é o indivíduo 1. Note que o nó de índice 4 é utilizado tanto pelo indivíduo base (indivíduo um) como pelo subgrafo da saída O2 do indivíduo dois que deverá ser incorporado. Sendo assim, o procedimento de incorporação buscará o primeiro nó inativo a fim de alocar este nó. Quando este nó for realocado para o nó de índice 5, todos os demais nós que compunham o subgrafo que

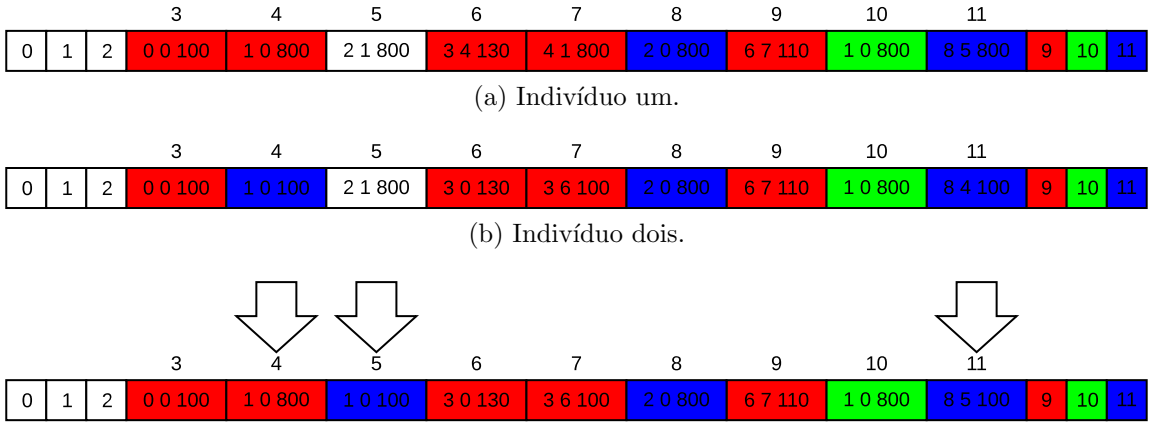
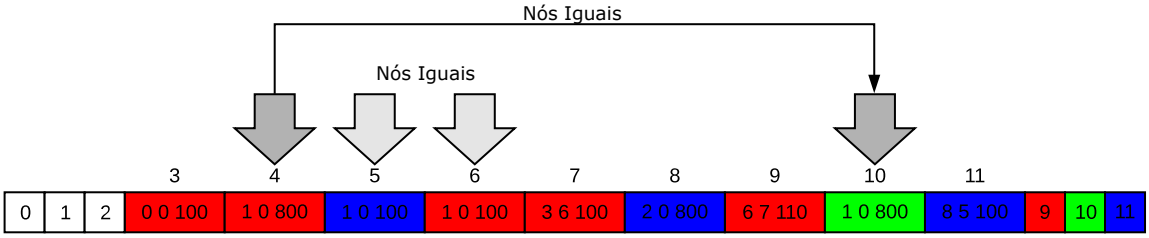


Figura 5.3: Procedimento de incorporação dos subgrafos considerando que O0 é melhor no indivíduo 1 (a), O1 é igual em ambos os indivíduos e O2 é melhor no indivíduo 2 (b).

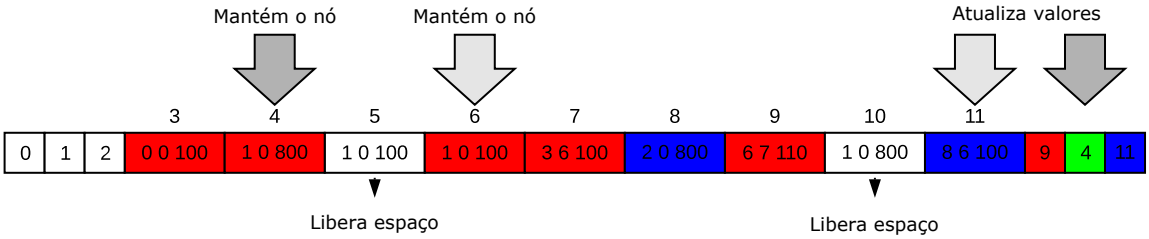
codificava a saída O2 deverão ter suas entradas corrigidas para o novo índice (de 4 para 5), como é possível ver no nó de índice 11 do indivíduo final.

Em certos momentos pode haver indisponibilidade de nós inativos para que os subgrafos sejam incorporados. Para isso existe o procedimento de reaproveitamento de nós, pois a realocação de nós no genótipo pode criar redundâncias, isto é, nós que codificam a mesma função com as mesmas entradas porém componentes de saídas distintas. Este procedimento visa justamente aproveitar apenas um nó cuja presença seja maior que 1 e liberar os demais nós. Depois de identificar os nós iguais, basta selecionar um deles para ser mantido e percorrer o genótipo a fim de atualizar todos os demais nós que faziam referência ao nó que será removido para o nó que será mantido. Depois disso, remove-se o nó redundante e libera-se espaço no genótipo, conforme ilustra a Figura 5.4, onde o nó 4 é igual ao 10 e o nó 5 é igual ao 6. Mantém-se os nós 4 e 6 pois ambos codificam o mesmo subgrafo. Os nós 5 e 10 são liberados e o nó 11 e a saída O1 têm seus valores corrigidos para 6 e 4 respectivamente.

Existe ainda a possibilidade de mesmo após este procedimento não existirem nós disponíveis para o procedimento de incorporação e, neste caso, o algoritmo é finalizado. Em experimentos computacionais foi testado o uso de um genótipo de tamanho variável que incrementava em 10% o seu tamanho toda vez que não havia espaço disponível para o procedimento de incorporação. Entretanto, percebeu-se que os genótipos cresciam indefinidamente, principalmente quando os circuitos que estavam sendo buscados possuíam um grande número de saídas. Além disso, o procedimento de incorporação gera um aumento significativo no número de nós ativos no genótipo tornando a situação mais crítica, pois



(a) Nós com genes de entrada e de função iguais.



(b) Exemplo onde dois nós se tornam inativos.

Figura 5.4: Procedimento de liberação de nós redundantes.

quando o genótipo aumenta de tamanho, rapidamente os nós que eram inativos tornam-se ativos. Apesar dos bons resultados obtidos por este operador de recombinação, o espaço disponível no genótipo ainda continua sendo uma questão importante a ser explorada no futuro.

5.2 OPERADOR DE MUTAÇÃO GUIADA

Como mencionado anteriormente, a função de aptidão adotada na CGP é a quantidade de acertos em relação à tabela verdade. Toda vez que a solução candidata possui um *bit* de saída igual ao *bit* descrito na tabela verdade essa solução tem seu valor de aptidão incrementado em um. Desta forma, o valor máximo de aptidão de uma solução candidata é:

$$\text{Aptidão Máxima} = 2^{n_i} \times n_o \tag{5.1}$$

onde n_i é o número de entradas e n_o é o número de saídas. Quando uma solução atinge tal valor ela é denominada factível.

Em diversos momentos nos experimentos computacionais, a CGP encontrava-se retida em mínimos locais, isto é, em posições nas quais o valor de aptidão era muito próximo do máximo mas nunca o alcançava. Sabe-se que o SAM (Seção 3.6.2.4, página 55) garante

que a mutação sempre ocorrerá em um nó ativo, entretanto, quando uma determinada saída do circuito já tem sua aptidão máxima alcançada, qualquer mutação que ocorra nessa saída será prejudicial, decrescendo a qualidade da solução nesta determinada saída ou sendo inútil no sentido de obter um circuito factível. Tendo em vista esta questão, esta proposta consiste em um operador de mutação guiado no subgrafo que codifica a saída que menos acerta a tabela verdade. Essa característica é importante pois reduz o número de avaliações desnecessárias da função de aptidão. O operador de mutação guiado é denominado *Guided Active Mutation* (GAM).

Neste operador de mutação, primeiramente determina-se quais são as saídas com menor quantidade de acertos em relação à tabela verdade. Em caso de empate, seleciona-se uma delas aleatoriamente. Após determinar qual saída sofrerá mutação, determinam-se quais são seus nós ativos. Depois, aplica-se o operador de mutação na maneira tradicional: seleciona-se um nó aleatoriamente e realiza a mutação, seja em genes de entrada ou genes função. Neste ponto, existe uma diferença do GAM em relação ao SAM: o GAM sempre atua em nós ativos e nunca em nós inativos. Como resultado, garante-se que os nós ativos que compõem a pior saída do circuito sofrerão mutação e, portanto, terão uma diferença fenotípica de seu progenitor, como o SAM, mas nunca terão nós inativos modificados. O fluxograma de funcionamento do GAM é apresentado na Figura 5.5 e no Algoritmo 2.

Algoritmo 2: O operador GAM.

```
worst_out ← saída com os piores valores em relação à tabela verdade;
active_nodes ← nós ativos no subgrafo que codifica worst_out;
while o número de nós ativos mutados não é alcançado do
    modifica um nó ativo selecionado aleatoriamente
end while
```

Nesta abordagem apenas um nó é modificado para cada nova solução candidata, mas isso pode ser modificado na instrução `while` do Algoritmo 2.

A Figura 5.6 ilustra o operador GAM com um indivíduo de três entradas (I0, I1, e I2) e três saídas (O0, O1, e O2). Os círculos numerados representam os nós do indivíduo (portas do circuito). Os nós em cinza são ativos e aqueles em branco são os inativos. As setas representam as conexões diretas entre os nós e as linhas contínuas conectam os nós ativos. O fenótipo (circuito) é composto pelas entradas, nós ativos e conexões entre eles, e saídas. Desta forma, o caminho para a saída O2 do primeiro indivíduo é formado por I2, e os nós 5, 8 e 11.

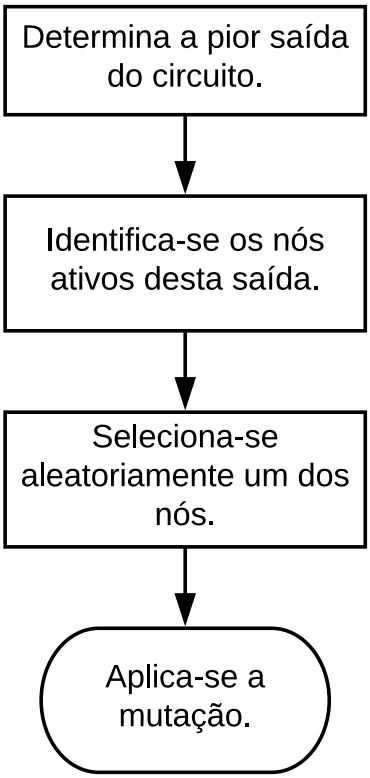


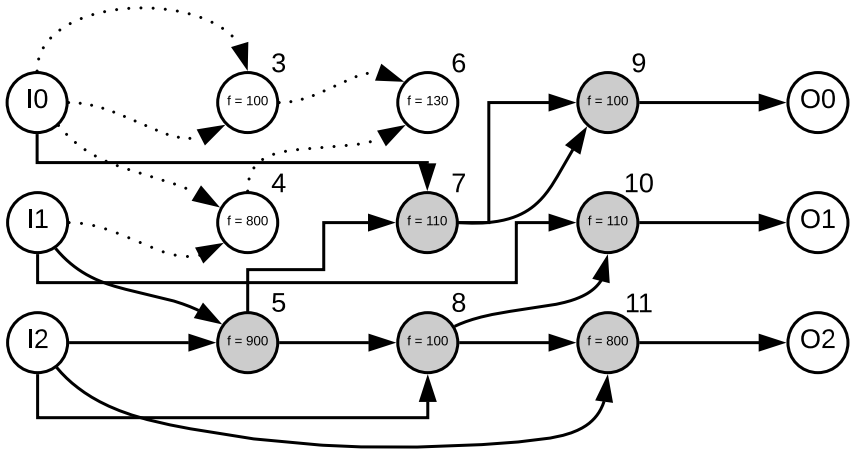
Figura 5.5: Fluxograma do operador de mutação GAM.

Para esse exemplo ilustrativo, o número de valores corretos das saídas dos indivíduos das gerações 1 e 2 são apresentados na Tabela 5.2. Na geração 1, a saída O1 é a pior, uma vez que acerta somente 3 de 8 possíveis *bits* da tabela verdade. As demais saídas do circuito (O0 e O2) acertam 8 e 6 valores da tabela verdade. Então, GAM atua apenas em nós ativos no subgrafo da saída O1, que são os nós vermelhos da Figura 5.6 (b).

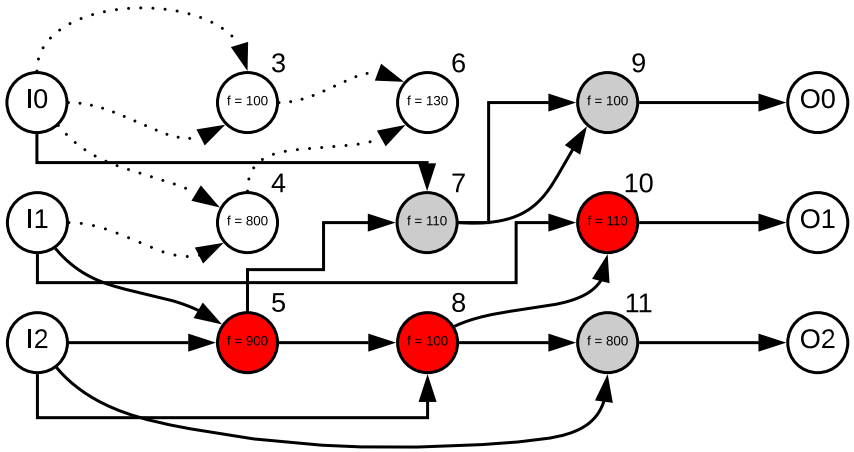
Considere que a descendência criada durante a geração 1 é o pai da geração 2, como apresentado na Figura 5.6 (c). Esse indivíduo foi criado a partir de uma mutação no nó 10. Tal mutação melhorou a saída O1, que possui 6 valores corretos agora. Neste caso, a pior saída deste indivíduo pode ser O1 ou O2. Então, o procedimento do GAM selecionará aleatoriamente uma delas e a mutação será aplicada. Os subgrafos dessas saídas são apresentados na Figura 5.6 (c) and (d). Estes passos são repetidos em todas as gerações até que o critério de parada seja atingido.

Tabela 5.2: Correspondências em relação à tabela verdade por geração.

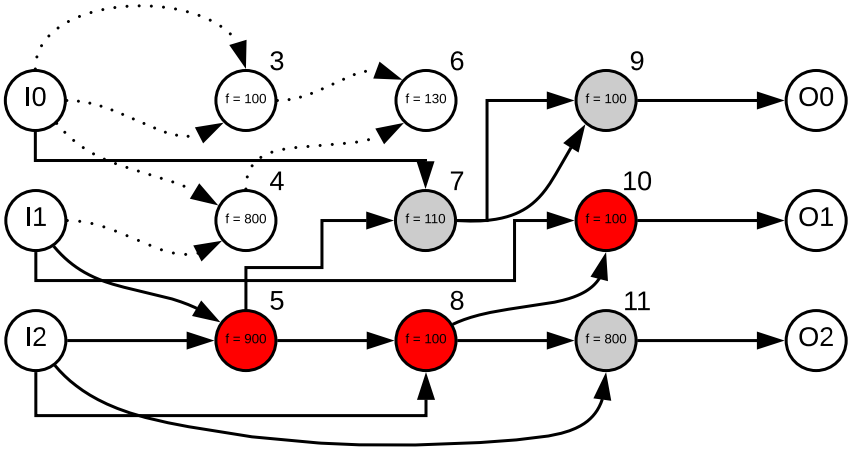
Geração	O0	O1	O2	Fitness
1	8	3	6	17
2	8	6	6	20



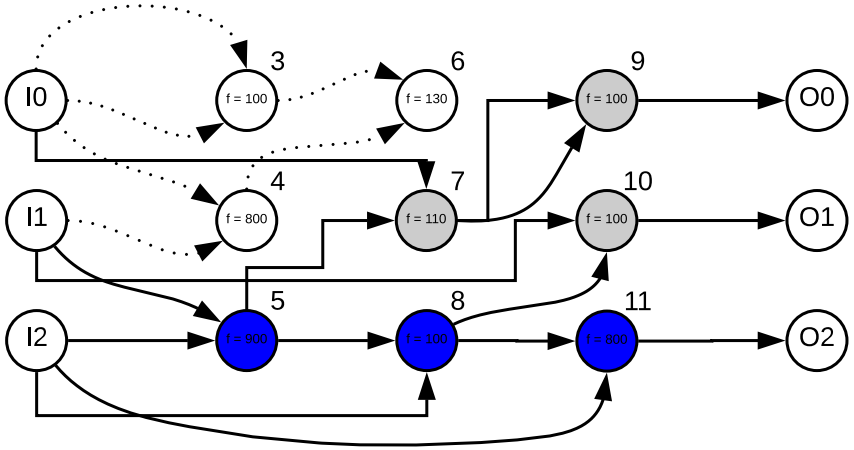
(a) Pai da geração 1.



(b) Subgrafo que compõe a saída O1.



(c) Pai da geração 2.



(d) Subgrafo que compõe a saída O2.

Figura 5.6: (a) é o pai da geração 1 e seu subgrafo que compõe a saída O1 é apresentado em (b), em vermelho. (c) é o pai da geração 2 criado por uma mutação no nó 10 do pai anterior com seu respectivo subgrafo em vermelho.. Ambos subgrafos em vermelho de (c) e em azul de (d) acertam 6 de 8 bits da tabela verdade. Um deles será escolhido aleatoriamente para receber o operador GAM e gerar a descendência subsequente.

O fato do GAM não atuar em nós inativos pode ser questionado pois as mutações nestes nós favorecem a deriva gênica neutra (NGD) (TURNER; MILLER, 2015), entretanto assim como o SAM, o GAM garante diferença fenotípica em subgrafos que possuem a chance de serem melhorados mas nunca atuará nos nós inativos presentes. Isso constitui a diferença fundamental entre o SAM e o GAM. Além disso, poder-se-ia utilizar o SAM somente ao subgrafo que codifica a pior saída e favorecer a NGD. Entretanto, experimentos mostraram que o SAM não obteve resultados melhores que o GAM.

É importante ressaltar que mesmo que a pior saída selecionada para aplicação do GAM possua nós compartilhados com outras saídas, ainda sim a mutação é feita nestes nós. Caso a mutação piore uma saída considerada boa e melhore uma saída considerada ruim o operador de recombinação proposto anteriormente na Seção 5.1 (página 105) é capaz de tirar proveito desta modificação genotípica benéfica, quando este operador é usado em conjunto com o GAM, como é apresentado na seção que segue.

5.2.1 USO DO GAM COM CROSSOVER

Como mencionado na seção anterior, o GAM atua no subgrafo que codifica a pior saída do circuito. Entretanto, este subgrafo pode conter nós que são compartilhados com outras saídas e caso este nó sofra mutação, pode acontecer da pior saída ser melhorada e de uma saída que era anteriormente considerada boa ser piorada. Este problema pode acarretar em uma piora da qualidade geral da solução tornando o operador GAM ineficiente. Sendo assim, propõe-se o uso do X-CGP descrito na Seção 5.1 (página 105) em conjunto com o GAM. Com isso, mesmo que o GAM piore uma saída que era boa e melhore uma outra, o procedimento do X-CGP é capaz de manter o subgrafo original da saída boa e aproveitar a mutação benéfica.

5.3 MODIFICAÇÃO DAS ESTRATÉGIAS EVOLUTIVAS DA CGP: SAM E GAM

Em experimentos preliminares as abordagens que utilizam o GAM mostraram-se eficientes em reduzir o número de avaliações necessárias para encontrar uma solução factível. Contudo, essas abordagens obtiveram baixas taxas de sucesso (quantidade de execuções que alcançaram soluções factíveis) quando comparadas com aquelas com o SAM. Visando

uma abordagem que aumente as taxas de sucesso e que necessite de poucas avaliações da função objetivo para encontrar uma solução factível, propõe-se aqui o uso conjunto do SAM e do GAM com operador de recombinação. Essa abordagem é chamada aqui de X-SAMGAM. A X-SAMGAM modifica a estratégia evolutiva (ES) (1+λ) comumente utilizada na CGP por uma ES-(1+λ_{SAM}+λ_{GAM}), onde os novos indivíduos são gerados utilizando tanto o SAM quanto o GAM. Aqui, $\lambda_{SAM} = \lambda_{GAM} = 2$ e um pseudoalgoritmo desta abordagem é apresentado no Algoritmo 3.

Algoritmo 3: Modificação da ES 1+λ.

```
μ = 1;
λ_SAM = 2;
λ_GAM = 2;
Critério de Parada ← número máximo de avaliações ou encontrar factibilidade;
for μ + λ_SAM + λ_GAM do
    Gera Indivíduo Aleatório(i);
    Avalia Indivíduo(i);
end for
μ ← melhor indivíduo da população inicial;
while Critério de Parada não é atingido do
    for i em λ do
        if i = 0 ou i = 1 then
            Aplica mutação (SAM) no pai (μ) e gera um novo indivíduo k_i;
        end if
        if i = 2 ou i = 3 then
            Aplica mutação (GAM) no pai (μ) e gera um novo indivíduo k_i;
        end if
        Avalia Indivíduo(k_i);
    end for
    μ ← Aplica Recombinação(μ + λ_SAM + λ_GAM);
end while
```

5.4 MULTIPLEXADOR COMO ELEMENTO LÓGICO

A proposta de uso do multiplexador como elemento lógico não modifica o algoritmo de busca e, portanto, não consiste em uma nova técnica, mas sim da adaptação de um elemento lógico no conjunto de funções da CGP que não é comumente utilizado na literatura. A literatura relata o uso de multiplexadores fixos no genótipo (MILLER, 1999) ou acoplados às saídas do circuito (MANFRINI *et al.*, 2014). Para este método, a modificação necessária para a utilização do multiplexador como elemento lógico na CGP é simples.

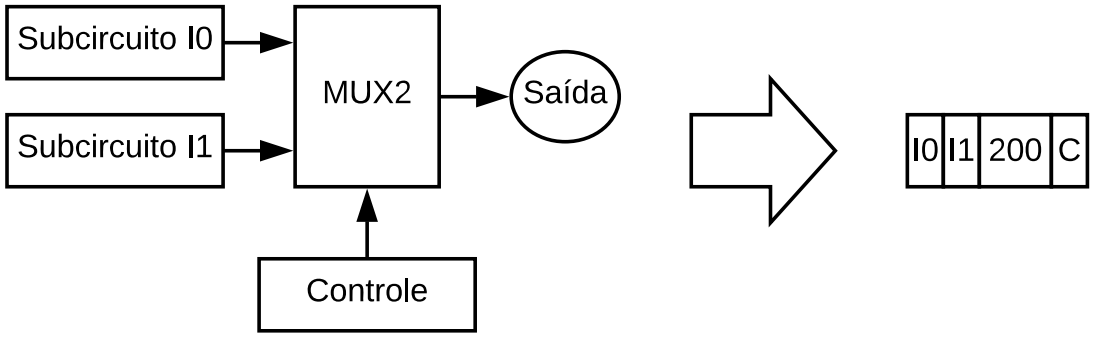


Figura 5.7: Representação de um nó cuja função é multiplexador. Todos os nós do genótipo possuirão 4 genes e o gene de controle só é considerado ativo quando a função executada é um multiplexador.

Antes haviam elementos lógicos unários (NOT e WIRE) e binários (AND, OR e XOR) e agora insere-se um elemento ternário devido à necessidade de uma entrada de controle no multiplexador. Todos os elementos lógicos ainda possuem uma única saída.

Além disso, a Figura 5.7 apresenta a codificação do nó do multiplexador onde I0 e I1 são as entradas do multiplexador, 200 é a função utilizada para o MUX e C é o controle. É importante ressaltar que não se considera uma mutação ativa quando a mesma ocorre no gene de controle e a função executada não é multiplexador. O mesmo acontece quando a mutação ocorre na segunda entrada de nós que realizam uma função unária como o NOT ou WIRE.

A nova função MUX é utilizada com o X-SAMGAM, cuja variante recebe o nome de X-SAMGAM-MUX pois a variante X-SAMGAM foi a que obteve os melhores resultados nos experimentos computacionais.

Manfrini *et al.* (2014) utilizaram multiplexadores para a evolução de circuitos lógicos combinacionais. Entretanto, esta abordagem considera o multiplexador como um elemento lógico fixo que é acoplado nas saídas do circuito e evoluem-se simultaneamente tanto as entradas quanto o controle desse multiplexador.

5.5 EVOLUÇÃO EM 3 ETAPAS

Para a evolução em 3 etapas, propõe-se o uso do multiplexador de 2 entradas (e um controle) acoplado em cada uma das saídas do circuito desejado. Desta maneira, cada saída será composta por 2 subcircuitos de entrada e um subcircuito de saída em um único cromossomo. Esse método surgiu a partir dos resultados obtidos nas investigações do Capítulo 4 principalmente no que diz respeito aos blocos simplificantes apresentados na

Seção 4.1.7 (página 89) pois a presença de tais blocos facilita a busca da CGP.

A evolução inicia-se buscando somente os subcircuitos de entrada de tal maneira que, quando analisados conjuntamente sejam capazes de atender a tabela verdade. Dessa forma a tabela verdade desejada é dividida entre as entradas do multiplexador. Uma vez obtidos os subcircuitos de entrada, a segunda etapa consiste em maximizar o número de similaridades entre as entradas, ou seja, fazer com que as entradas acertem os mesmos *bits* da tabela verdade em questão. Quanto maior a quantidade de similaridades maior será a facilidade para obter o subcircuito de controle, pois desta maneira, tanto faz escolher entre a entrada A ou a entrada B do multiplexador, uma vez que essa situação gera *don't cares* no subcircuito de controle. Por fim, na terceira etapa, evolui-se o subcircuito de controle que terá o trabalho de guiar qual é o subcircuito responsável pelo nível lógico desejado na saída para cada linha da tabela verdade. Um caso particular que pode ocorrer é quando obtém-se 100% de similaridade entre as entradas I0 e I1. Neste caso, existem somente *don't cares* na tabela do subcircuito de controle e, portanto, qualquer que seja esse subcircuito as restrições da tabela verdade serão atendidas.

Desta maneira, o esforço da CGP concentra-se apenas naqueles *bits* cuja similaridade não foi obtida e, portanto, necessita ser controlada. Nesta proposta, dedica-se uma quantidade de número de avaliações para a primeira e segunda etapas do processo, expressas em porcentagem da quantidade máxima de avaliações permitida. Experimentos preliminares indicam que quanto mais recursos são disponibilizados para essas etapas, menor será a quantidade de recursos necessários para a parte de controle, tendo em vista a maximização de similaridades entre as entradas do multiplexador e a consequente geração de *dont' cares* para o subcircuito de controle, simplificando a expressão que deve ser obtida. Por este motivo, o método de evolução em 3 etapas é indicado para circuitos mais complexos (mais entradas) pois circuitos que são facilmente obtidos pela CGP não serão resolvidos enquanto as etapas 1 e 2 não terminarem. Neste trabalho adotou-se um orçamento computacional total igual à 30% dos recursos máximos disponíveis para as primeira e segunda etapas em conjunto. Este número foi escolhido experimentalmente de tal maneira que as similaridades obtidas pela segunda etapa fossem maiores que 50%. A representação do funcionamento da evolução em 3 etapas é apresentado na Figura 5.8 e de um indivíduo da evolução em 3 etapas na Figura 5.9 e as etapas são detalhadas a seguir.

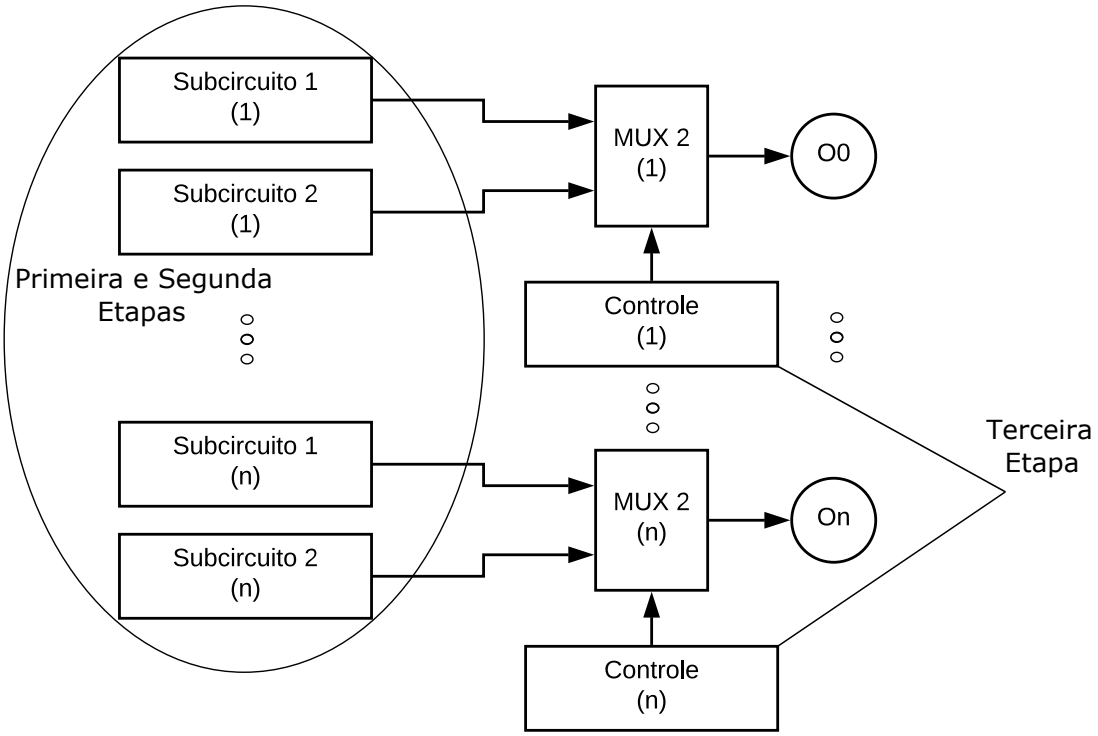


Figura 5.8: Ordem de evolução dos subcircuitos na abordagem de evolução em 3 etapas. Primeiramente, obtém-se um circuito factível levando em consideração a união das tabelas-verdade dos subcircuitos que compõem I0 e I1 para todas as saídas do circuito. Após isso, na segunda etapa, maximiza-se a similaridade entre I0 e I1. Por fim obtém-se o subcircuito de controle cuja tabela verdade torna-se mais fácil de ser obtida conforme aumenta-se as similaridades entre I0 e I1.

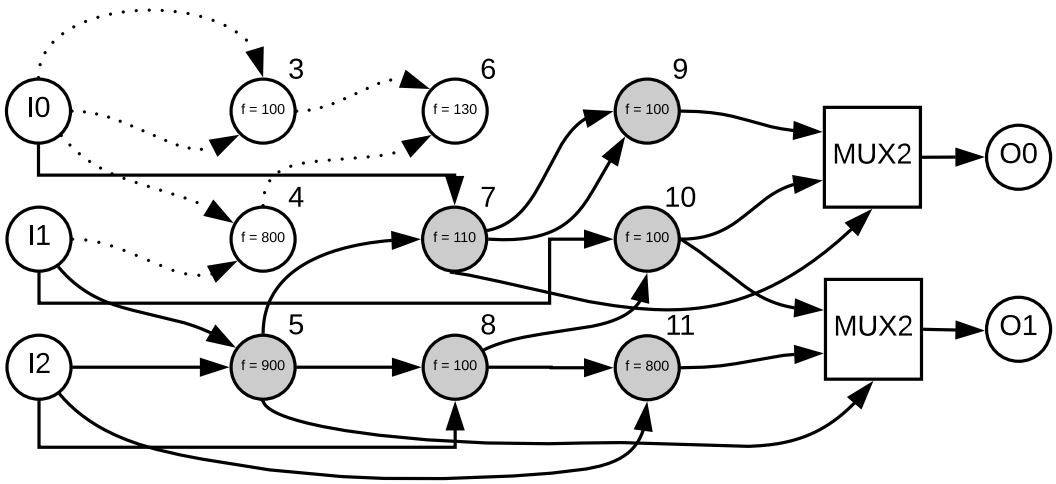


Figura 5.9: Representação do indivíduo da evolução em 3 etapas. Nós em cinza são ativos, brancos inativos, entradas inferiores nos multiplexadores são os subcircuitos de controle. As linhas contínuas definem o fenótipo.

5.5.1 PRIMEIRA ETAPA

Nesta etapa o objetivo é encontrar a tabela verdade desejada a partir de duas subtabelas, sujeito a:

$$f(I0) \text{ ou } f(I1) = f(D) \tag{5.2}$$

onde, $f(I0)$ é a função que gera a tabela verdade da entrada I0 do multiplexador, $f(I1)$ é a função que gera a tabela verdade da entrada I1 do multiplexador e $f(D)$ é a função da tabela verdade desejada.

Um exemplo ilustrativo onde no início do processo evolutivo as entradas I0, I1 e a tabela desejada são apresentadas na Tabela 5.3. A última coluna apresenta com um "X" e com a entrada que faz com que as linhas da tabela satisfaçam a tabela verdade desejada, isto é, caso I0 ou I1 sejam iguais a D. Ao longo do processo evolutivo as entradas vão sendo modificadas até que em determinado momento o critério desejado é obtido, conforme apresentado na Tabela 5.4. Neste momento, a primeira etapa do método proposto é finalizado pois um circuito considerado factível já foi encontrado, levando em consideração as entradas I0 e I1.

Tabela 5.3: Tabelas verdade no início do processo evolutivo.

I0	I1	D	Satisfaz
0	1	0	X (I0)
0	1	0	X (I0)
1	1	0	-
1	1	1	X (I0 e I1)
1	1	1	X (I0 e I1)
1	1	0	-
0	0	1	-
1	1	0	-
1	0	1	X (I0)
1	1	0	-
1	0	1	X (I0)
0	0	1	-
0	1	1	X (I1)
0	1	0	X (I0)
0	0	1	-
0	1	0	X (I0)

Tabela 5.4: Tabelas verdade no momento de satisfação do critério.

I0	I1	D	Satisfaz
0	1	0	X (I0)
0	1	0	X (I0)
1	0	0	X (I1)
1	1	1	X (I0 e I1)
0	1	1	X (I1)
0	1	0	X (I0)
1	1	1	X (I0 e I1)
1	0	0	X (I1)
1	1	1	X (I0 e I1)
0	0	0	X (I0 e I1)
1	1	1	X (I0 e I1)
0	1	1	X (I1)
0	1	1	X (I1)
0	1	0	X (I0)
1	0	1	X (I0)
0	0	0	X (I0 e I1)

5.5.2 SEGUNDA ETAPA

Nesta etapa o objetivo é maximizar o número de *bits* iguais entre as entradas I0 e I1 e manter o critério de factibilidade da etapa anterior. Sendo assim, para cada linha k da tabela verdade:

$$\max(I0_k = I1_k) \tag{5.3}$$

onde, $I0_k$ representa um *bit* (linha k) da tabela verdade de I0 e $I1_k$ representa um *bit* (linha k) da tabela verdade de I1.

Considerando o exemplo apresentado na primeira etapa, no momento que o critério é satisfeito, obtém-se a situação apresentada na Tabela 5.5. Neste caso, existem 6 *bits* iguais.

Tabela 5.5: Tabelas verdade no momento de satisfação do critério.

I0	I1	D	Iguais
0	1	0	-
0	1	0	-
1	0	0	-
1	1	1	X
0	1	1	-
0	1	0	-
1	1	1	X
1	0	0	-
1	1	1	X
0	0	0	X
1	1	1	X
0	1	1	-
0	1	1	-
0	1	0	-
1	0	1	-
0	0	0	X

Tabela 5.6: Tabelas verdade no fim da segunda etapa.

I0	I1	D	Iguais
0	0	0	X
0	0	0	X
1	0	0	-
1	1	1	X
1	1	1	X
0	1	0	-
1	1	1	X
1	0	0	-
1	1	1	X
0	0	0	X
1	1	1	X
0	1	1	-
1	1	1	X
0	0	0	X
1	1	1	X
0	0	0	X

É importante ressaltar que entradas I0 e I1 que representam as funções F=0 e F=1 sempre satisfarão o critério de factibilidade da primeira etapa. Entretanto, como não existem similaridades entre I0 e I1 a segunda etapa possibilita a eliminação desta situação.

Continuando o processo evolutivo maximiza-se o número de similaridades e obtém-se, por exemplo, uma tabela verdade como a apresentada na Tabela 5.6. Neste caso temos 12 *bits* iguais entre as entradas I0 e I1. Quando a quantidade de avaliações permitidas

para a primeira e segunda etapas é atingido, passa-se para a terceira etapa.

5.5.3 TERCEIRA ETAPA

O objetivo da terceira etapa é evoluir o subcircuito de controle do multiplexador que selecionará qual é a entrada (I0 ou I1) responsável por obter o valor desejado na saída (D). A terceira etapa inicia-se gerando a nova tabela verdade que deve ser evoluída para o controle, pois, para cada similaridade obtida anteriormente, gera-se um *don't care* na tabela verdade do circuito de controle, facilitando a busca. Sendo assim, a tabela verdade que deseja-se obter agora é apresentada na Tabela 5.7 onde "X" representa *don't care*, 0 significa que o valor desejado está em I0 e 1 significa que o valor desejado está em I1. Os *don't cares* por sua vez são linhas que não requerem nenhuma avaliação e, portanto, o esforço computacional é direcionado para as saídas com valores 0 ou 1.

Tabela 5.7: Tabelas verdade que deve ser obtida no controle.

I0	I1	D	Tabela Controle
0	0	0	X
0	0	0	X
1	0	0	1
1	1	1	X
1	1	1	X
0	1	0	0
1	1	1	X
1	0	0	1
1	1	1	X
0	0	0	X
1	1	1	X
0	1	1	1
1	1	1	X
0	0	0	X
1	1	1	X
0	0	0	X

Vale destacar que a função desejada é F = $B\overline{D}+A\overline{D}+\overline{B}CD$. Entretanto com o método de evolução em 3 etapas a partir da simplificação da tabela verdade de controle necessita-se obter somente F = C devido à presença dos *don't cares*. Conforme demostrado nos estudos de dificuldades da CGP com as composições simplificantes entre si apresentados na Seção 4.1.7 (página 89), essa simplificação será essencial para que a CGP encontre um circuito factível, ainda mais quando a função que necessita-se obter é uma entrada

primária do circuito.

Todas as modificações geradas nas tabelas verdade intermediárias através do multiplexador simplificam a expressão lógica que deverá ser obtida e, portanto, facilita a busca da CGP por circuitos factíveis. Além disso, essa modificação nas tabelas verdade intermediárias viabilizam a evolução de circuitos mais complexos (maior quantidade de entradas), pois simplifica a expressão lógica.

6 EXPERIMENTOS COMPUTACIONAIS

Os experimentos computacionais foram realizados a fim de analisar comparativamente o desempenho dos métodos propostos em relação à CGP tradicional além de apresentar um estudo de parâmetros do X-CGP (CGP com recombinação). A nomenclatura utilizada e a descrição dos métodos são:

- CGP: CGP padrão utilizando somente o operador de mutação SAM;
- X-CGP: CGP com recombinação e utilizando somente o operador de mutação SAM;
- X-CGP-OP: recombinação, utilizando somente o operador de mutação SAM com parâmetros otimizados;
- GAM: CGP padrão utilizando somente o operador de mutação GAM;
- SAMGAM: CGP padrão com as estratégias evolutivas modificadas onde a descendência é gerada pelo SAM e pelo GAM;
- X-GAM: CGP com recombinação e utilizando somente o operador de mutação GAM;
- X-SAMGAM: CGP com recombinação e com as estratégias evolutivas modificadas onde a descendência é gerada pelo SAM e pelo GAM;
- X-SAMGAM-MUX: CGP com recombinação e com estratégias evolutivas modificadas onde a descendência é gerada pelo SAM e pelo GAM e com elemento lógico MUX no conjunto de funções;
- X-CGP-3E: CGP com recombinação e com estratégias evolutivas modificadas onde a descendência é gerada pelo SAM e pelo GAM, com elemento lógico MUX no conjunto de funções e com evolução em 3 etapas.

A medida em que os métodos propostos superam o desempenho da CGP tradicional, o método base de comparação é modificado para a melhor abordagem até então obtida. Primeiramente, compara-se o operador de recombinação proposto com a CGP tradicional. Após isso os experimentos relacionados ao GAM são apresentados e comparados ao X-CGP. Em seguida, os resultados dos experimentos do X-SAMGAM são apresentados. Por

fim, apresentam-se os experimentos relacionados às abordagens com multiplexadores. Os códigos de todos os métodos estão disponíveis na internet[1].

6.1 DESCRIÇÃO DOS PROBLEMAS INVESTIGADOS

Para todos os métodos propostos, foram utilizados conjuntos variados de problemas a fim de verificar a eficácia das abordagens. Os problemas são diversos, possuindo quantidades variadas de número de entradas e número de saídas bem como sua natureza: somadores, multiplicadores, decodificadores e comparadores e circuitos fictícios. Os problemas utilizados são comumente explorados na literatura e, além disso, foram utilizados problemas *benchmark* de síntese lógica do conjunto LGSynth'91[2]. A descrição dos problemas e os parâmetros utilizados são apresentados em cada seção dos experimentos.

6.2 CROSSOVER (X-CGP)

Nesta seção avalia-se o desempenho do X-CGP através da comparação dos resultados obtidos por uma CGP padrão com o SAM e com a abordagem do X-CGP além de apresentar um estudo de otimização de parâmetros do X-CGP (denominado X-CGP-PO) cujos resultados são comparados com os dois métodos citados anteriormente.

6.2.1 PROBLEMAS

Os problemas utilizados para os experimentos do X-CGP consistem de problemas utilizados na literatura, circuitos aritméticos e de problemas *benchmark* do LGSynth 91'. Todos os problemas são apresentados na Tabela 6.1 contendo o nome adotado, número de entradas (ni) e saídas (no), origem do problema (referência), os parâmetros utilizados na CGP, número de colunas (nc) e número máximo de avaliações (Max. Aval.) e o número de execuções independentes (EI). Para todos os problemas utilizou-se nr = 1 e lb = nc e o conjunto de funções $\Gamma = \{AND, OR, NOT, XOR, WIRE\}$.

[1] `https://github.com/ciml/ciml-lib`

[2] https://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.pdf

Tabela 6.1: Problemas utilizados nos experimentos do X-CGP e para o operador de mutação GAM (ni: número de entradas, no: número de saídas, nc: número de colunas, EI: quantidade de execuções independentes e Max. Aval: número máximo de avaliações da função objetivo).

Problema	ni	no	Origem	nc	Max. Aval.	EI
Problema 2	4	3	(COELLO *et al.*, 2004)	100	100.000	30
Problema 4	4	2	(COELLO *et al.*, 2000)	100	100.000	30
Somador 2x2	4	4	(WALKER *et al.*, 2006)	100	100.000	30
Somador 3x3	6	6	(WALKER *et al.*, 2006)	500	1.000.000	30
Somador 4x4	8	8	(WALKER *et al.*, 2006)	500	1.000.000	10
Multiplicador 2x2	4	4	(COELLO *et al.*, 2000)	100	100.000	30
Multiplicador 3x3	6	6	(WALKER *et al.*, 2006)	500	500.000	10
C17	5	2	LGSynth 91'	100	100.000	30
dc1	4	7	LGSynth 91'	200	200.000	30
newbyte	5	8	LGSynth 91'	300	300.000	30
rd53	5	3	LGSynth 91'	100	100.000	30
wim	4	7	LGSynth 91'	100	100.000	30

6.2.2 AJUSTE DE PARÂMETROS DO X-CGP

Uma análise de parâmetros é realizada, onde o desempenho do X-CGP foi avaliado quando a quantidade de descendência gerada (λ) e a quantidade de nós ativos mutados são variados. A proposta do operador de recombinação e a análise de parâmetros foi publicado em Silva e Bernardino (2018). Conforme ressaltado na Seção 3.6.2.4 (página 55), originalmente o operador SAM é utilizado para a mutação de somente um nó ativo para a geração de cada um dos descendentes. Apesar de ser ressaltado nas conclusões do artigo (GOLDMAN; PUNCH, 2013) a possibilidade de mutar mais de um nó ativo, tais experimentos não foram realizados. Desta forma, essa possibilidade é explorada e adota-se a sigla MAM(n) (*Multiple Active Mutation*) para denotar a utilização do SAM mutando uma quantidade de **n** nós ativos. O número de nós ativos modificados pelo MAM(n) estão entre 1 e 10, bem como a quantidade de descendentes gerados.

Para a análise das combinações de parâmetros foi utilizada uma ferramenta denominada *Performance Profiles* (PPs) (BARBOSA *et al.*, 2010) e consiste em uma ferramenta analítica para a visualização e interpretação de resultados experimentais. Os PPs são analisados conforme a área normalizada sob a curva (AUC) uma vez que esta é uma medida geral de desempenho, onde AUCs maiores são preferíveis.

Cinco dos circuitos mais comuns encontrados na literatura de computação evolutiva foram utilizados para os experimentos (COELLO *et al.*, 2000, 2003, 2000, 2004). A Ta-

Tabela 6.2: Problemas utilizados para o ajuste de parâmetros (ni: número de entradas, no: número de saídas, nc: número de colunas e Max. Aval.: número máximo de avaliações permitidas).

Problemas	ni	no	nc	Max. Aval.
1 - Exemplo 4(COELLO *et al.*, 2000)	4	3	100	100.000
2 - Multiplicador 2x2-bit	4	4	100	100.000
3 - Exemplo 4(COELLO *et al.*, 2004)	4	2	100	100.000
4 - Somador 2x2-bit	4	4	100	100.000
5 - CM85A	8	3	300	1.000.000

bela 6.2 apresenta os problemas e o número de colunas (nc), número de entradas (ni), número de saídas (no) e o número máximo de avaliações da função objetivo permitido (Max. Aval.). Os parâmetros adotados são $n_r = 1$ e lb = nc e o conjunto de funções $\Gamma = \{AND, OR, NOT, XOR, WIRE\}$. Foram realizadas 30 execuções independentes para cada configuração para cada problema, com $n_c = 100$, exceto para o problema CM85A onde 3 execuções independentes foram realizadas para cada configuração e $n_c = 300$ devido à grande quantidade de entradas quando comparada aos demais circuitos.

Conforme as AUCs, os piores resultados são obtidos quando são utilizados valores menores de lambda, como 1 ou 2. As três melhores configurações observadas nos experimentos são:

- MAM(1); $\lambda = 9$,
- MAM(1); $\lambda = 3$, e
- MAM(3); $\lambda = 5$.

Os PPs dessas três variantes são apresentados na Figura 6.1, onde pode-se observar que:

- MAM(1)-L3 (mutação em um nó ativo e geração três descendências) e MAM(3)-L5 (mutação em três nós ativos e geração de cinco descendências) são as variantes com o melhor desempenho na maioria dos problemas (maior $\rho(1)$);
- MAM(1)-L9 (mutação em um nó ativo e geração de nove descendências) é a variante mais confiável (menor τ tal que $\rho(\tau) = 1$); e
- MAM(1)-L9 (mutação em um nó ativo e geração de nove descendências) apresenta o melhor desempenho geral (maior AUC).

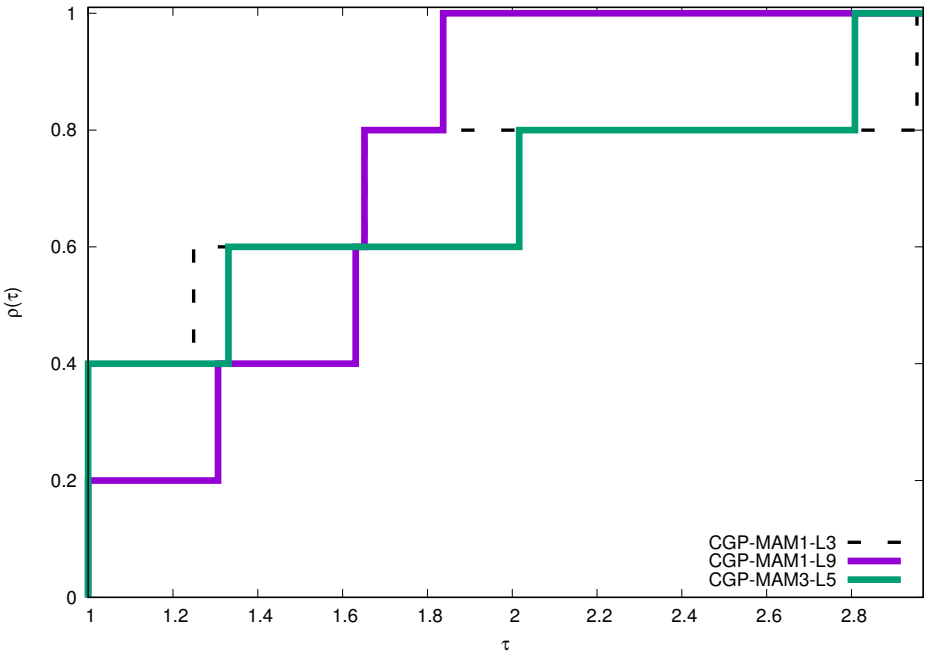


Figura 6.1: Performance Profiles das três melhores variantes do X-CGP. Os valores normalizados de AUC são 1.0, 0.9977 e 0.9952 para MAM(1)-L9, MAM(1)-L3 e MAM(3)-L5, respectivamente.

Além disso, é importante ressaltar que dentre essas três variantes somente MAM(3)-L5 foi capaz de obter taxa de sucesso (quantidade de execuções independentes capaz de obter solução factível) igual a 100% para todos os problemas testados e, por isso, foi considerada a melhor combinação de parâmetros para X-CGP pois está e a variante que apresentou maior desempenho na maioria dos problemas. Ainda que o MAM(1)-L9 obteve o melhor desempenho geral, esta variante não foi capaz de obter taxa de sucesso igual a 100% para todos os problemas testados.

6.2.3 ANÁLISE COMPARATIVA DOS RESULTADOS

Os resultados da comparação entre a CGP padrão, X-CGP e X-CGP-OP (X-CGP com parâmetros otimizados) são apresentados nas Tabelas 6.3 e 6.4. Os melhores resultados são apresentados em negrito. Além disso, a Tabela 6.5 apresenta a análise estatística com os p-valores (Kruskal-Wallis). Quando o mesmo é igual ou inferior a 0,05 o teste de Dunn também é realizado e os resultados comparativos dos p-valores entre os métodos são apresentados.

Tabela 6.3: Resultados dos experimentos computacionais do X-CGP em comparação à CGP padrão com SAM e ao X-CGP-OP (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
Problema 2	CGP	2929	6735,75	12428,5	21663	18487,2	17740,42	66868	6116	14927,25	**100%**
	X-CGP	1340	4388,75	**9161**	**10973,25**	**9576,33**	**7171,33**	35013	**3640,5**	**6584,5**	**100%**
	X-CGP-OP	**1223**	**2416**	9372	11499	9666,41	8604,44	**32429**	4012	9083	96,67%
Problema 4	CGP	1200	**4309,5**	8714,5	13916	9774,37	6309,94	**24698**	5044,5	9606,5	**100%**
	X-CGP	**496**	4698,75	**6879,5**	15756,5	11016,97	10273,52	45887	5650	11057,75	**100%**
	X-CGP-OP	623	4429,5	7331,5	**10005,75**	**8190,17**	**6213,68**	31032	**3018,5**	**5576,25**	**100%**
Somador 2x2	CGP	**1045**	7375	15654	26230	20302,48	17652,63	81447	10195	18855	96,67%
	X-CGP	1492	**3425**	**7260,5**	**15409**	15879,4	23035,91	96160	**4451,5**	**11984**	**100%**
	X-CGP-OP	1385	6780,75	10776,5	19658,5	**14512**	**10975,69**	**45180**	5881	12877,75	**100%**
Somador 3x3	CGP	29440	58684,5	115530	250002,25	165112,6	145547,46	690850	66993	191317,75	**100%**
	X-CGP	16144	63440	116055	**138632**	132628,03	123675,58	583450	**43959**	**75192**	96,67%
	X-CGP-OP	**13578**	**53659,75**	**87985**	166326,75	**117609,17**	**88340,89**	**298065**	46912	112667	**100%**
Somador 4x4	CGP	**75547**	**228674**	**374058**	662629	**438516,56**	270874,2	**847028**	211436	433955	**90%**
	X-CGP	104871	292312,75	437399	699021,75	478616,83	291620,36	861852	240359	406709	70%
	X-CGP-OP	193341	269401	419898	**647400**	477820,67	**264260,25**	935914	**187703**	**377999**	**90%**
Mult 2x2	CGP	1515	6357,5	10826,5	18466,75	12581	**8322,45**	**29927**	5676,5	12109,25	**100%**
	X-CGP	1308	3513,25	6481	9576	8741,3	9513,93	43616	3085,5	**6062,75**	**100%**
	X-CGP-OP	**921**	**2530**	**3565**	**8854,25**	**7135,87**	8495,32	41890	**1847,5**	6324,25	**100%**
Mult 3x3	CGP	207942	424810,75	459738	610679	489397,67	**187724,13**	734807	123383,5	185868,25	**60%**
	X-CGP	**137652**	**243092**	**353399,5**	584862	398427	223377,46	677772	176534	341770	**60%**
	X-CGP-OP	353697	367371	381045	**399784,5**	**384422**	325454,17	**418524**	**27348**	**32413,5**	30%

Tabela 6.4: Resultados dos experimentos computacionais do X-CGP em comparação à CGP padrão com SAM e ao X-CGP-OP (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
C17	CGP	**105**	**298,75**	**987**	**2170**	**1442,8**	**1240,88**	**4457**	**935**	**1871,25**	**100%**
	X-CGP	114	588	1345	3697	2660,47	2950,54	12874	972,5	3109	**100%**
	X-CGP-OP	145	635,25	2277	3873	2561,9	2043,88	7042	1635,5	3237,75	**100%**
dc1	CGP	75547	95732	129444	148295	127685,47	**39101,29**	191827	27230	**52563**	56,67%
	X-CGP	10436	**23583**	46073	97424	**60642,56**	44262,55	**164319**	25299	73841	**90%**
	X-CGP-OP	**10360**	27462,5	**42550**	**94175**	67128,15	63442,81	178770	**19040**	66712,5	**90%**
newbyte	CGP	47497	94662,5	135883,5	190567,25	143319,09	65310,27	245155	54481	95904,75	73,33%
	X-CGP	**10100**	**32108**	59980,5	98569,5	71636,33	53872,82	265984	33334,5	66461,5	**100%**
	X-CGP-OP	21615	41554	**53293**	**85005**	**64725,1**	**34425,17**	**145445**	**15513**	**43451**	96,67%
rd53	CGP	19880	34093	44322	75472	53458,77	25043,53	91365	17225	41379	43,33%
	X-CGP	14178	32479,5	41564	56918,5	47575,93	23301,2	96483	15352	24439	**50%**
	X-CGP-OP	**13620**	**23028,25**	**35015**	**42057,75**	**36073,5**	**18968,41**	**75157**	**10372,5**	**19029,5**	40%
wim	CGP	72537	73100,5	73664	86016,5	81523,33	**14599,65**	**98369**	**1127**	**12916**	10%
	X-CGP	10148	21891	**33212**	**63714**	42836,22	25784,97	99852	19640	41823	76,67%
	X-CGP-OP	**4975**	**20405**	37483	64805	**41058,6**	26801,63	98435	17383	44400	**83,33%**

Tabela 6.5: Resultados dos testes estatísticos entre CGP(1), X-CGP(2) e X-CGP-OP(3).

Problema	p-valor	(1) vs (2)	(1) vs (3)	(2) vs (3)
Problema 2	0,0425	0,0358	0,0249	0,8723
Problema 4	0,5531	-	-	-
Somador 2x2	0,0563	-	-	-
Somador 3x3	0,5657	-	-	-
Somador 4x4	0,8786	-	-	-
Mult 2x2	0,0019	4,8E-5	4,5E-4	0,1664
Mult 3x3	0,4310	-	-	-
C17	0,0886	-	-	-
dc1	9,2E-5	4,8E-5	2,8E-4	0,6256
newbyte	3,1E-5	7,3E-5	3,9E-5	0,8496
rd53	0,1466	-	-	-
wim	0,0665	-	-	-

Em 5 dos 12 problemas o X-CGP foi melhor na mediana. Com exceção do Somador 4x4, o X-CGP apresentou maiores taxas de sucesso ou, no mínimo, iguais aos da CGP padrão. De maneira geral, o desvio padrão da CGP tradicional é melhor que o do X-CGP. Os intervalos interquartis da CGP padrão também foram melhores indicando uma maior robustez da CGP padrão. Entretanto, o X-CGP é capaz de reduzir o número de avaliações necessárias para obter uma solução factível e de aumentar a taxa de sucesso na maioria dos problemas. Além disso, os p-valores indicam que existe diferença estatística entre a CGP padrão e o X-CGP e entre a CGP padrão e o X-CGP-OP para 4 dos 12 problemas. O mesmo não é observado quando compara-se a X-CGP com a X-CGP-OP, indicando que a X-CGP é robusto em relação à mudança de parâmetros.

Apesar da otimização de parâmetros do X-CGP (CGP com operador de recombinação), o método X-CGP-OP (CGP com operador de recombinação e parâmetros otimizados) não foi capaz de superar os resultados obtidos pelo X-CGP em alguns problemas. Inclusive, no Problema 2 e no newbyte, não foi possível obter 100% de taxa de sucesso com tal método apesar das demais métricas mostrarem-se competitivas. Entretanto, em outros problemas maiores, como no Somador 3x3, o X-CGP-OP foi capaz de obter 100% de factibilidade e nos problemas Multiplicador 2x2 e dc1, obteve melhores resultados em praticamente todas as métricas quando comparadas às obtidas pelos outros métodos. Além disso, no problema wim, aumentou consideravelmente a taxa de sucesso. Então, conclui-se que o X-CGP com os parâmetros otimizados obtém bons resultados, entretanto em alguns momentos ele não é capaz de obter resultados tão bons quanto aos parâmetros tradicionais. Portanto, pode-se

pensar que o conjunto de problemas utilizados para o treinamento e ajuste de parâmetros do X-CGP não foi tão representativo. Com isso, ao usar os parâmetros considerados os melhores em outros problemas, o resultado não apresenta variação chegando inclusive a piorar em algumas situações, conforme é apresentado na Tabela 6.4 onde os melhores resultados são apresentados em negrito.

Além disso, é possível perceber que a CGP padrão se comporta melhor quando os problemas possuem menor quantidade de saídas, como no C17. Para o Somador 3x3 os resultados são bastante similares para os dois métodos mas os melhores resultados são obtidos utilizando-se o X-CGP-OP. Uma possível explicação é que o X-CGP tira proveito da grande quantidade de saídas dos circuitos para que possa criar um super indivíduo. Quando existem poucas saídas o X-CGP não possui tantos subgrafos para aprimorar o super indivíduo e consequentemente fica mais dependente da capacidade de variação do genótipo via mutação. Logo, com a diminuição do número de saídas, o X-CGP tende a ficar semelhante à CGP padrão.

6.3 OPERADOR DE MUTAÇÃO GUIADA (GAM)

Os resultados referentes ao operador de mutação guiada (GAM) são comparados com a CGP utilizando somente o GAM e CGP com o operador de recombinação utilizando GAM (X-GAM). Além disso, os resultados do GAM e do X-GAM são comparados com a abordagem X-CGP. Os problemas e os parâmetros são os mesmos utilizados no experimento anterior e são apresentados na Tabela 6.1.

6.3.1 ANÁLISE COMPARATIVA DOS RESULTADOS

Os resultados do X-CGP e do GAM e X-GAM são apresentados nas Tabelas 6.6 e 6.7. Os melhores resultados são apresentados em negrito. Além disso, a Tabela 6.8 apresenta a análise estatística com os p-valores (Kruskal-Wallis). Quando o mesmo é igual ou inferior a 0,05, o teste de Dunn também é realizado e os resultados comparativos dos p-valores entre os métodos são apresentados.

O uso do operador de mutação GAM reduz significativamente o número de avaliações necessárias para se obter uma solução factível quando comparado à CGP padrão na maioria dos problemas. Porém obtém baixas taxas de sucesso e não foi capaz de encontrar

soluções factíveis para o problema wim. Além disso não foi capaz de encontrar soluções factíveis para o problema wim. Quando as abordagens com o operador de recombinação são consideradas, o X-GAM é capaz de obter os melhores resultados na mediana para a maioria dos problemas, 8 de 12, no que diz respeito ao número de avaliações necessárias e também é capaz de aumentar a taxa de sucesso quando comparado ao GAM somente mas não superam as do X-CGP. Além disso, o X-GAM apresenta baixo desvio padrão enaltecendo a robustez do método. Apesar de em alguns problemas o X-CGP apresentar maior taxa de sucesso, o operador de recombinação também promoveu um aumento na taxa de sucesso do GAM. Além disso, os p-valores indicam que existe diferença estatística em 4 dos 12 problemas. Entre a X-CGP e o GAM a diferença ocorre em 2 dos 12 problemas. Quando leva-se em consideração a X-CGP e o X-GAM a diferença é observada para os problemas Mult 2x2 e Somador 3x3. Por sua vez, 3 dos 4 problemas que obtiveram p-valor menor ou igual a 0,05 apresentam diferença estatística quando compara-se o GAM e o X-GAM. A exceção é o Somador 3x3. É possível perceber que o uso do operador de recombinação gera diferença estatística nos problemas dc1 e newbyte. Isso foi observado tanto para as comparações entre a X-CGP e o GAM quanto para o GAM com o X-GAM. Tendo em vista o aspecto de baixa quantidade de número de avaliações necessárias para obter factibilidade com as abordagens GAM e as altas taxas de sucesso obtidas pelas abordagens X-CGP, o experimento seguinte apresenta os resultados da abordagem que combina o operador de recombinação, e as mutações SAM e GAM.

6.4 ESTRATÉGIA SAM + GAM

Nesta seção, são apresentados os resultados referentes à abordagem que busca unir as características de baixo número de avaliações necessárias para obter uma solução factível através do GAM e do X-GAM e das altas taxas de sucesso obtidas pelo X-CGP, chamado aqui de X-SAMGAM. Os resultados obtidos pelo X-SAMGAM são comparados ao X-CGP e ao X-GAM. Os problemas e parâmetros utilizados são os mesmos apresentados na Tabela 6.1 e o conjunto de funções também é o mesmo utilizado nos experimentos anteriores. Neste experimento o objetivo é verificar se X-SAMGAM, que gera descendências tanto pelo SAM quanto pelo GAM é capaz de superar as abordagens que geram descendências somente pelo SAM (X-CGP) e somente pelo GAM (X-GAM).

Tabela 6.6: Resultados dos experimentos computacionais do GAM em comparação ao X-CGP e ao X-GAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
Problema 2	X-CGP	1340	4388,75	9161	10973,25	9576,33	7171,33	35013	3640,5	**6584,5**	**100%**
	GAM	1189	4155	7654	20664	11637,52	9656,04	32395	3822	16509	96,67%
	X-GAM	**940**	**2996**	**5247**	**10302,5**	**6732,74**	**4594,22**	**18604**	**3097**	7306,5	96,67%
Problema 4	X-CGP	**496**	4698,75	6879,5	15756,5	11016,97	10273,52	45887	5650	11057,75	**100%**
	GAM	694	**3313,5**	**5430,5**	**11251,25**	**9461,21**	14627,79	78947	**3403**	**7937,75**	93,33%
	X-GAM	1823	3567	6857	14414,75	10684,93	**10157,88**	**45829**	4519,5	10847,75	**100%**
Somador 2x2	X-CGP	1492	3425	7260,5	15409	15879,4	23035,91	96160	4451,5	11984	**100%**
	GAM	**527**	3561,5	9153	14055,25	10443,58	8344,02	36069	5439	10493,75	86,67%
	X-GAM	1028	**2560**	**4935**	**9039,5**	**6333,44**	**4802,28**	**20439**	**2755**	**6479,5**	90%
Somador 3x3	X-CGP	16144	63440	116055	138632	132628,03	123675,58	583450	43959	75192	**96,67%**
	GAM	8689	40388	64230	159368	97163,59	79717,46	**283768**	35947	118980	**96,67%**
	X-GAM	**6400**	**26565,5**	**40992**	**65141**	**61156,56**	**67669,41**	356259	**17312**	**38575,5**	90%
Somador 4x4	X-CGP	104871	292312,75	437399	699021,75	478616,83	291620,36	861852	240359	406709	**70%**
	GAM	231488	387621	437725,5	758974,5	548464,33	285246,18	946870	130236,5	371353,5	60%
	X-GAM	**90605**	**134505,5**	**161704**	**261259**	**200791,29**	**111684,81**	**361701**	**48743**	**126753**	**70%**
Mult 2x2	X-CGP	1308	3513,25	6481	9576	8741,3	9513,93	43616	3085,5	6062,75	**100%**
	GAM	1222	4943,75	7146	10337,5	11312,21	14155,29	70623	2512,5	5393,75	93,33%
	X-GAM	**344**	**1805**	**3740**	**5196**	**3876,62**	**2344,26**	**10337**	**1504**	**3391**	90%
Mult 3x3	X-CGP	**137652**	243092	353399,5	584862	398427	223377,46	677772	176534	341770	**60%**
	GAM	313032	412824,75	544708,5	785208	596328,33	251560,29	941283	190703,5	372383,25	**60%**
	X-GAM	177203	**234253,5**	**336932**	**429931**	**350628**	**155308,79**	**592463**	**114173**	**195677,5**	**60%**

Tabela 6.7: Resultados dos experimentos computacionais do GAM em comparação ao X-CGP e ao X-GAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
C17	X-CGP	114	588	**1345**	3697	2660,47	2950,54	12874	972,5	3109	**100%**
	GAM	110	**510,25**	1626,5	**2977,25**	**2149,93**	**1912,24**	**6224**	1201	**2467**	**100%**
	X-GAM	**52**	590,75	1348	3061,75	2343,97	2597,12	9317	**945**	2471	**100%**
dc1	X-CGP	10436	23583	46073	97424	60642,56	44262,55	164319	25299	73841	90%
	GAM	64629	108036	122754,5	154309	129364,38	36787,05	196068	26162	46273	53,33%
	X-GAM	**8288**	**17929**	**32229**	**62549**	**43009,71**	**32354,71**	**111752**	**16949,5**	**44620**	**93,33%**
newbyte	X-CGP	10100	32108	59980,5	98569,5	71636,33	**53872,82**	**265984**	33334,5	66461,5	**100%**
	GAM	61923	134085,25	175813,5	202144,75	170577,75	62426,05	294118	42036	68059,5	80%
	X-GAM	**6979**	**19226**	**30941,5**	**53433,25**	**51154,79**	62790,11	266540	**13492,5**	**34207,25**	90%
rd53	X-CGP	14178	**32479,5**	**41564**	**56918,5**	47575,93	**23301,2**	96483	**15352**	**24439**	**50%**
	GAM	33317	49881	78558,5	92917,75	71282,33	27643,35	99237	18458,5	43036,75	20%
	X-GAM	**11678**	32529,75	47253,5	65173	**47461,2**	24432,91	**88820**	18918	32643,25	33,33%
wim	X-CGP	10148	21891	**33212**	63714	42836,22	25784,97	99852	19640	41823	76,67%
	GAM	-	-	-	-	-	-	-	-	-	0%
	X-GAM	**7436**	**20632**	38268	**54676**	**40005,88**	**24715,19**	**97092**	**17636**	**34044**	**83,33%**

Tabela 6.8: Resultados dos testes estatísticos entre X-CGP(1), GAM(2) e X-GAM(3).

Problema	p-valor	(1) vs (2)	(1) vs (3)	(2) vs (3)
Problema 2	0,2040	-	-	-
Problema 4	0,3750	-	-	-
Somador 2x2	0,1063	-	-	-
Somador 3x3	0,0264	0,1641	0,0072	0,1422
Somador 4x4	0,0956	-	-	-
Mult 2x2	7,4E-4	0,4032	0,0047	3,1E-4
Mult 3x3	0,1055	-	-	-
C17	0,9355	-	-	-
dc1	9,4E-7	8,6E-5	0,1463	1,9E-7
newbyte	1,2E-8	1,5E-5	0,0856	3,8E-9
rd53	0,2357	-	-	-
wim	0,6722	-	-	-

Além disso apresenta-se a capacidade de otimização das soluções obtidas pelo X-SAMGAM e por uma CGP padrão com SAM em termos de redução do número de portas lógicas quando considera-se o uso do SAM.

6.4.1 ANÁLISE COMPARATIVA DOS RESULTADOS

Os resultados do X-CGP, X-GAM e X-SAMGAM são apresentados nas Tabelas 6.9 e 6.10. Os melhores resultados são apresentados em negrito. Além disso, a Tabela 6.11 apresenta a análise estatística com os p-valores (Kruskal-Wallis). Quando o mesmo é igual ou inferior a 0,05, o teste de Dunn também é realizado e os resultados comparativos dos p-valores entre os métodos são apresentados. Por fim, apresenta-se a comparação da capacidade de redução do número de portas lógicas utilizando o operador de mutação SAM entre a abordagem X-SAMGAM e a abordagem CGP padrão com SAM e os p-valores na Tabela 6.12. A coluna Portas Lógicas apresenta o número de portas lógicas da primeira solução factível (Primeira) e da solução final (Melhor). Além disso, a coluna Redução apresenta a redução relativa do número de portas lógicas entre a primeira e a melhor solução.

Conforme esperado, o X-SAMGAM foi capaz de obter maiores taxas de sucesso para todos os problemas testados com exceção do problema wim onde o método proposto não foi capaz de obter soluções factíveis e o comportamento foi bastante similar entre o X-CGP e o X-GAM sendo que o X-GAM foi capaz de obter maior quantidade de circuitos factíveis.

No multiplicador 2x2 o número de avaliações necessárias para obter uma solução factível é superior do que ao do X-GAM e inferior ao X-CGP. Entretanto, a taxa de factibilidade obtida pelo X-SAMGAM é de 100%. As mesmas características de obter-se menor número de avaliações necessárias para obter uma solução factível e o aumento da taxa de sucesso são observados para os problemas dc1, newbyte, somador 4x4, mult 3x3, somador 2x2, somador 3x3, mostrando que o X-SAMGAM consegue de fato combinar as melhores características do X-CGP e do GAM.

No problema 4, que possui somente duas saídas, o X-GAM foi o melhor e, além das três abordagens obterem 100% de factibilidade, o X-SAMGAM só conseguiu obter melhores resultados no melhor caso (menor número de avaliações necessárias para obter uma solução factível). Para o C17 o X-SAMGAM apresentou os melhores resultados apesar de todas as técnicas obterem 100%.

Os p-valores indicam que existe diferença estatística em 3 dos 12 problemas. Entre a X-CGP e o X-SAMGAM a diferença ocorre somente para o Problema 2. Além disso não existe diferença estatística entre o X-GAM e o X-SAMGAM.

A Tabela 6.12 mostra que as primeiras soluções obtidas pela CGP padrão obtém o menor número de portas lógicas em 10 dos 12 problemas apresentados. No que diz respeito à melhor solução a CGP padrão também obtém os melhores resultados em 9 dos 12 problemas. Tanto o X-SAMGAM quanto a CGP padrão apresentaram os maiores índices de redução relativa em 6 dos 12 problemas cada. Além disso, os p-valores indicam que só existe diferença estatística na distribuição de portas lógicas para o problema dc1 na primeira solução factível e no Problema 4 para melhor solução. Observa-se ainda que no caso da melhor solução, a diferença entre as médias dos números de portas é de $6,27 - 5,7 = 0.57$, ou seja, menos do que 1 porta.

De maneira geral, o X-SAMGAM consegue alcançar os resultados para os quais foi projetado, isto é, diminuir o número de avaliações necessárias para obter uma solução factível através do uso do GAM e aumentar a taxa de sucesso com o SAM. É importante ressaltar que o operador de recombinação tem contribuição fundamental para obter soluções factíveis devido à criação do super indivíduo. Portanto, o X-SAMGAM é considerado a melhor variante obtida até o momento e servirá de base para a comparação nos próximos experimentos.

Tabela 6.9: Resultados dos experimentos computacionais do X-CGP, X-GAM e X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
Problema 2	X-CGP	1340	4388,75	9161	10973,25	9576,33	7171,33	35013	3640,5	6584,5	**100%**
	X-GAM	940	2996	5247	10302,5	6732,74	**4594,22**	18604	3097	7306,5	96,67%
	X-SAMGAM	**788**	**1895,25**	**3667**	**8234,75**	**5550,2**	4703,94	**16548**	**2069**	**6339,5**	**100%**
Problema 4	X-CGP	496	4698,75	6879,5	15756,5	11016,97	10273,52	45887	5650	11057,75	**100%**
	X-GAM	1823	**3567**	**6857**	**14414,75**	**10684,93**	**10157,88**	**45829**	**4519,5**	10847,75	**100%**
	X-SAMGAM	**300**	3958	9592	14568,5	13598,07	17385,63	93745	5597	**10610,5**	**100%**
Somador 2x2	X-CGP	1492	3425	7260,5	15409	15879,4	23035,91	96160	4451,5	11984	**100%**
	X-GAM	1028	**2560**	**4935**	**9039,5**	**6333,44**	**4802,28**	**20439**	**2755**	6479,5	90%
	X-SAMGAM	**628**	4273,5	6746	10039,75	9308,4	8942,7	44280	3159,5	**5766,25**	**100%**
Somador 3x3	X-CGP	16144	63440	116055	138632	132628,03	123675,58	583450	43959	75192	96,67%
	X-GAM	**6400**	**26565,5**	**40992**	**65141**	**61156,56**	67669,41	356259	**17312**	**38585,5**	90%
	X-SAMGAM	10876	35639	60872,5	129370	84372,1	**67392,02**	**230021**	34381	93731	**100%**
Somador 4x4	X-CGP	104871	292312,75	437399	699021,75	478616,83	291620,36	861852	240359	406709	70%
	X-GAM	90605	**134505,5**	**161704**	**261259**	**200791,29**	**111684,81**	**361701**	**48743**	**126753**	70%
	X-SAMGAM	**57532**	275140	337915	464557	340039,22	141961,38	481296	113354	189417	**90%**
Mult 2x2	X-CGP	1308	3513,25	6481	9576	8741,3	9513,93	43616	3085,5	6062,75	**100%**
	X-GAM	**344**	**1805**	**3740**	**5196**	**3876,62**	**2344,26**	**10377**	**1504**	**3391**	90%
	X-SAMGAM	872	2445	4214,5	6087,25	5862,67	5912,5	29794	1899,5	3642,25	**100%**
Mult 3x3	X-CGP	137652	243092	353399,5	584862	398427	223377,46	677772	176534	341770	60%
	X-GAM	177203	234253,5	336932	**429931**	350628	**155308,79**	**592463**	**114173**	**195667,5**	60%
	X-SAMGAM	**13758**	**211066**	**315683**	541873	**321663,2**	218355,5	857683	174432	330807	**90%**

Tabela 6.10: Resultados dos experimentos computacionais do X-CGP, X-GAM e X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
C17	X-CGP	114	588	1345	3697	2660,47	2950,54	12874	972,5	3109	**100%**
	X-GAM	**52**	590,75	1348	3061,75	2343,97	2597,12	9317	645	2471	**100%**
	X-SAMGAM	114	**382,5**	**919**	**2085**	**1700,27**	**2006,19**	**9117**	**584,5**	**1702,5**	**100%**
dc1	X-CGP	10436	23583	46073	97424	60642,56	44262,55	164319	25299	73841	90%
	X-GAM	8288	**17929**	**32229**	**62549**	**43009,71**	**32354,71**	**111752**	**16949,5**	**44620**	93,33%
	X-SAMGAM	**7320**	28896,75	41762	86543	59396,37	43569,51	172432	22704	57646,25	**100%**
newbyte	X-CGP	10100	32108	59980,5	98569,5	71636,33	53872,82	265984	33334,5	66461,5	**100%**
	X-GAM	6976	**19226**	**30941,5**	53433,25	**51154,79**	62790,11	266540	**13492,5**	**34207,25**	90%
	X-SAMGAM	**4040**	23801,25	51272	**40566**	53457,53	**37690,99**	**158489**	24136,5	46764,75	**100%**
rd53	X-CGP	14178	32479,5	41564	56918,5	47575,93	**23301,2**	96483	15352	**24439**	50%
	X-GAM	11678	32529,75	47253,5	65173	47461,2	24432,91	**88820**	18918	32643,25	33,33%
	X-SAMGAM	**10109**	**24108**	**31790**	**49178**	**38349,71**	23489,39	89182	**13698**	25070	**56,67%**
wim	X-CGP	10148	21891	**33212**	63714	42836,22	25784,97	99852	19640	41823	76,67%
	X-GAM	**7436**	**20632**	38268	**54676**	**40005,88**	**24715,19**	**97092**	**17636**	**34044**	**83,33%**
	X-SAMGAM	-	-	-	-	-	-	-	-	-	0%

Tabela 6.11: Resultados dos testes estatísticos entre X-CGP(1), X-GAM(2) e X-SAMGAM(3).

Problema	p-valor	(1) vs (2)	(1) vs (3)	(2) vs (3)
Problema 2	0,0288	0,1868	0,0077	0,2034
Problema 4	0,8479	-	-	-
Somador 2x2	0,2351	-	-	-
Somador 3x3	0,0236	0,0086	0,0652	0,3009
Somador 4x4	0.2158	-	-	-
Mult 2x2	0,0242	0,0072	0,0807	0,3396
Mult 3x3	0,6082	-	-	-
C17	0,4197	-	-	-
dc1	0,1817	-	-	-
newbyte	0,0513	-	-	-
rd53	0,3251	-	-	-
wim	0,4459	-	-	-

6.5 MUX COMO ELEMENTO LÓGICO

6.5.1 PROBLEMAS

Para as abordagens que utilizam o multiplexador como elemento lógico, chamado aqui de X-SAMGAM-MUX e também para o método de evolução em 3 etapas, chamado aqui de X-CGP-3E, foram utilizados outros problemas com maior número de entradas. Apesar de todos os problemas utilizados anteriormente terem sido executados com o X-SAMGAM-MUX, outros problemas foram adicionados. Todos os problemas utilizados são apresentados na Tabela 6.13 que contém o nome adotado para o problema, número de entradas (ni) e saídas (no), origem do problema (referência), os parâmetros utilizados na CGP, número de colunas (nc), número máximo de avaliações(Max. Aval.) e o número de execuções independentes (EI). Para todos os problemas utilizou-se nr = 1 e lb = nc e o conjunto de funções $\Gamma = \{AND, OR, NOT, XOR, WIRE, MUX\}$. A coluna Métodos apresenta quais problemas foram utilizados para cada método onde MUX se refere ao uso do multiplexador como elemento lógico e $3E$ à evolução em 3 etapas. Existe um conjunto menor de problemas que utilizam somente o método X-CGP-3E, pois ele foi desenvolvido para problemas mais complexos (com maior quantidade de entradas). Problemas nos quais a CGP obteve bom desempenho não são indicados para o X-CGP-3E, pois necessita-se utilizar 30% dos recursos computacionais disponibilizados para as primeira e segunda etapas. Esta quantidade de avaliações disponibilizadas é muitas vezes superior à quantidade de

Tabela 6.12: Resultados da otimização entre a CGP padrão e o X-SAMGAM.

Problema	Método	Portas Lógicas		Redução (%)	p-valor	
		Primeira	Melhor		Primeira	Melhor
C17	CGP	**9,7**	**3**	**69,07**	0,4804	0,0781
	X-SAMGAM	10,17	3,2	68,53		
dc1	CGP	**48,41**	**20,12**	58,44	0,0298	0,4851
	X-SAMGAM	55,13	21	**61,91**		
Mult 2x2	CGP	**19,53**	**5,27**	73,02	0,0954	0,2037
	X-SAMGAM	21,67	5,6	**74,16**		
Mult 3x3	CGP	**61**	**27,5**	**54,92**	0,8726	0,0916
	X-SAMGAM	62,17	30,83	50,41		
newbyte	CGP	**59,55**	15,86	73,37	0,0505	0,5428
	X-SAMGAM	66,4	**15,3**	**76,96**		
Problema 2	CGP	**16,2**	**4,03**	75,12	0,3166	0,7295
	X-SAMGAM	17,27	4,13	**76,09**		
Problema 4	CGP	**14,9**	**5,7**	**61,74**	0,9053	0,0460
	X-SAMGAM	15,37	6,27	59,21		
rd53	CGP	26,46	8,62	**67,42**	0,0584	0,7732
	X-SAMGAM	**23,35**	**8,56**	63,34		
Somador 2+2	CGP	**20,1**	4,11	79,55	0,9149	0,5747
	X-SAMGAM	20,37	**3,8**	**81,35**		
Somador 3+3	CGP	49,13	**8,17**	**83,37**	0,4730	0,9643
	X-SAMGAM	**47,2**	8,23	82,56		
Somador 4+4	CGP	**58,67**	**12,44**	**78,80**	0,8946	0,0671
	X-SAMGAM	59,78	15	74,91		
wim	CGP	**34,67**	**17**	50,97	0,1371	0,5077
	X-SAMGAM	42,74	18	**57,88**		

Tabela 6.13: Problemas utilizados nos experimentos que utilizam abordagens com multiplexadores.

Problema	ni	no	Origem	nc	Max, Aval.	EI	Métodos
Problema 2	4	3	(COELLO *et al.*, 2004)	100	100.000	30	MUX
Problema 4	4	2	(COELLO *et al.*, 2000)	100	100.000	30	MUX
Somador 2x2	4	4	(WALKER *et al.*, 2006)	100	100.000	30	MUX
Somador 3x3	6	6	(WALKER *et al.*, 2006)	500	1.000.000	30	MUX
Somador 4x4	8	8	(WALKER *et al.*, 2006)	500	1.000.000	10	MUX
Multiplicador 2x2	4	4	(COELLO *et al.*, 2000)	100	100.000	30	MUX
Multiplicador 3x3	6	6	(WALKER *et al.*, 2006)	500	500.000	10	MUX
C17	5	2	LGSynth 91'	100	100.000	30	MUX
dc1	4	7	LGSynth 91'	200	200.000	30	MUX
newbyte	5	8	LGSynth 91'	300	300.000	30	MUX
rd53	5	3	LGSynth 91'	100	100.000	30	MUX
wim	4	7	LGSynth 91'	100	100.000	30	MUX
dk27	9	9	LGSynth 91'	200	400.000	10	MUX/3E
clpl	11	5	LGSynth 91'	300	400.000	10	MUX/3E
br1	12	8	LGSynth 91'	500	1.000.000	3	MUX/3E
newtpla1	10	2	LGSynth 91'	200	200.000	10	MUX/3E
newtpla2	10	4	LGSynth 91'	400	500.000	10	MUX/3E

avaliações necessárias para obter um circuito factível pelos demais métodos ou, devido à facilidade da CGP resolver os problemas envolvidos, muitos recursos computacionais são desperdiçados nas primeira e segunda etapas do método X-CGP-3E.

6.5.2 ANÁLISE COMPARATIVA DOS RESULTADOS

Os resultados são apresentados nas Tabelas 6.14 e 6.15. Os melhores resultados são apresentados em negrito. Além disso a Tabela 6.16 apresenta a análise estatística com os p-valores (Kruskal-Wallis).

O método X-SAMGAM-MUX obteve 100% de soluções factíveis em todos os problemas com exceção do Multiplicador 3x3 que obteve 90% assim como o método X-SAMGAM. Inclusive no problema wim onde o método X-SAMGAM não obteve soluções. Além disso, o X-SAMGAM-MUX apresenta menor número de avaliações necessárias para obter uma solução factível quando comparada ao X-SAMGAM. De maneira geral, o X-SAMGAM só se comporta melhor no melhor caso. Os valores obtidos na mediana para o método X-SAMGAM-MUX são 69% melhores que os obtidos pelo X-SAMGAM. Nesta métrica, somente para o problema rd53 o X-SAMGAM obteve melhor desempenho. Isso mostra que o multiplexador como elemento lógico consegue criar pequenas modificações nas tabelas verdade intermediárias facilitando a busca da CGP. De acordo com a Tabela 6.16

os p-valores indicam que existe diferença estatística em 5 dos 12 problemas. Não foram feitos experimentos visando a minimização do número de portas quando o MUX é considerado, pois esta é mais complexa do que as demais portas utilizadas. De fato, o MUX pode ser escrito como uma combinação de portas primárias. Entretanto, pensando em custo, o MUX requer menos transistores (elemento principal do qual as portas lógicas são formadas) do que a combinação de portas equivalente a ele. Fica então a necessidade de se adotar uma função objetivo (por exemplo, o número de transistores) diferente daquela largamente usada na literatura a fim de avaliar esta proposta em termos de custo. Além disso, o método do X-SAMGAM-MUX apesar de reduzir o número de avaliações necessários para obter uma solução factível não é capaz de resolver o problema da escalabilidade. Isso é perceptível quando analisa-se a razão entre a mediana do número de avaliações e a quantidade de entradas do problema em questão.

Tabela 6.14: Resultados dos experimentos computacionais do X-SAMGAM-MUX em comparação com o X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
Problema 2	X-SAMGAM	788	1895,25	3667	8234,75	**5550,2**	**4703,94**	**16548**	2069	6339,5	**100%**
	X-SAMGAM-MUX	**651**	**1519,25**	**2979**	**6378**	5656,8	6056,26	25761	**1619,5**	**4858,75**	**100%**
Problema 4	X-SAMGAM	300	3958	9592	14568,5	13598,07	17385,63	93745	5597	10610,5	**100%**
	X-SAMGAM-MUX	**76**	**2134,25**	**3659**	**7076**	**4986,17**	**4219,99**	**16316**	**1951**	**4941,75**	**100%**
Somador 2x2	X-SAMGAM	**628**	4273,5	6746	10039,75	9308,4	8942,7	44280	3159,5	5766,25	**100%**
	X-SAMGAM-MUX	763	**1787,75**	**4191**	**6477**	**5055,83**	**4606,31**	**20262**	**2386**	**4689,25**	**100%**
Somador 3x3	X-SAMGAM	10876	35639	60872,5	129370	84372,1	**67392,02**	**230021**	34381	93731	**100%**
	X-SAMGAM-MUX	**4832**	**34493**	**45823,5**	**76549,75**	**70052,43**	75851,24	391341	**17000,5**	**42056,75**	**100%**
Somador 4x4	X-SAMGAM	57532	275140	337915	464557	340039,22	**141961,38**	**481296**	113354	189417	90%
	X-SAMGAM-MUX	**45464**	**89046**	**125186,5**	**212687**	**182271,5**	145669,6	526224	**55856**	**123641**	**100%**
Mult 2x2	X-SAMGAM	**872**	2445	4214,5	**6087,25**	5862,67	5912,5	29794	1899,5	**3642,25**	**100%**
	X-SAMGAM-MUX	996	**2289**	**3026**	7658	**5314,03**	**5153,87**	**23728**	**1511**	5369	**100%**
Mult 3x3	X-SAMGAM	**13758**	211066	315683	541873	321663,2	218355,5	857683	174432	330807	**90%**
	X-SAMGAM-MUX	51607	**77101**	**160020**	**197804**	**142554,11**	**75636,54**	**232137**	**72117**	**120703**	**90%**

Tabela 6.15: Resultados dos experimentos computacionais do X-SAMGAM-MUX em comparação com o X-SAMGAM. (Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

Problema	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
C17	X-SAMGAM	114	382,5	919	2085	1700,27	2006,19	9117	584,5	1702,5	**100%**
	X-SAMGAM-MUX	**78**	**225**	**797**	**1383**	**1218,77**	**1531,11**	**7151**	**572,5**	**1158**	**100%**
dc1	X-SAMGAM	**7320**	28896,75	41762	86543	59396,37	43569,51	172432	22704	57646,25	**100%**
	X-SAMGAM-MUX	7552	**14228,25**	**19691**	**36690**	**28605,8**	**22202,39**	**97788**	**8865,5**	**22461,75**	**100%**
newbyte	X-SAMGAM	**4040**	23801,25	51272	70566	53457,53	37690,99	158489	24136,5	46764,75	**100%**
	X-SAMGAM-MUX	13992	**19361**	**32093,5**	**43242,5**	**37084,83**	**22145,78**	**97856**	**12655,5**	**23881,5**	**100%**
rd53	X-SAMGAM	10109	**24108**	**31790**	**49178**	**38349,71**	23489,39	89182	**13698**	**25070**	56,67%
	X-SAMGAM-MUX	**10088**	26216,5	38937,5	61166,25	43238,27	**22460,88**	**88481**	18036	34949,75	**100%**
wim	X-SAMGAM	-	-	-	-	-	-	-	-	-	0%
	X-SAMGAM-MUX	**5548**	**11912**	**21210**	**37847**	**27996,03**	**22761,09**	**91305**	**10006**	**25935**	**100%**

Tabela 6.16: Resultados dos testes estatísticos entre X-SAMGAM e X-SAMGAM-MUX.

Problema	p-valor
Problema 2	0,6361
Problema 4	0,0021
Somador 2x2	0,0089
Somador 3x3	0,2805
Somador 4x4	0,0412
Mult 2x2	0,5154
Mult 3x3	0,0229
C17	0,2675
dc1	8,3E-4
newbyte	0,1103
rd53	0,4561
wim	0,1411

6.6 EVOLUÇÃO EM 3 ETAPAS

Nesta seção são apresentados os resultados dos experimentos computacionais através do método de evolução em 3 etapas (X-CGP-3E). Os resultados são comparados com a melhor abordagem até então obtida que é o X-SAMGAM-MUX e com a CGP padrão com SAM. A CGP padrão é trazida de volta para comparações pois não foi analisado o comportamento deste método para circuitos com grande número de entradas. Os problemas utilizados são os apresentados na Tabela 6.13 cuja coluna Método apresenta a sigla 3E.

6.6.1 ANÁLISE COMPARATIVA DOS RESULTADOS

Os resultados são apresentados na Tabela 6.17. Aqui, adotam-se as siglas X-SG-MUX para o método onde o multiplexador como elemento lógico é utilizado, X-CGP-3E para o método de evolução em 3 etapas e CGP para a CGP padrão com SAM. Além disso, os recursos computacionais disponibilizados para as primeira e segunda etapas do método X-CGP-3E foi de 30% do número máximo de avaliações permitidas para todas as técnicas analisadas nesse conjunto de experimentos.

Os resultados obtidos pelo X-CGP-3E são superiores aos dos demais métodos considerados. Além disso, o X-CGP-3E foi capaz de obter 100% de circuitos factíveis em todos os problemas. Com excessão dos problemas clpl e newtpla2, somete o método 3E foi capaz de obter circuitos factíveis. Por sua vez, para o problema clpl o número de avaliações necessárias para obter circuitos factíveis pelo método X-SG-MUX foi inferior ao obtido

pelo X-CGP-3E, entretanto o X-SAMGAM-MUX só foi capaz de obter circuitos factíveis em 80% das execuções independentes. O mesmo efeito não foi observado para o problema newtpla2.

Quando leva-se em consideração a complexidade do circuito (quantidade de entradas), o método de evolução em 3 etapas requer menor quantidade de número de colunas e número de avaliações disponíveis para obter circuitos com grande números de entradas. Isso se deve ao fato de haver o período de maximização de similaridades das entradas. Isso é bastante perceptível quando utilizamos 1.000.000 de avaliações em circuitos de 8 entradas como o Somador 4x4 e apenas 200.000 para um circuito de 10 entradas como o *benchmark* newtpla1. O método X-CGP-3E é capaz, portanto, de evoluir circuitos maiores quando comparados à abordagem da CGP tradicional além de diminuir o número de avaliações necessárias para obter circuitos factíveis. É importante ressaltar que o método X-CGP-3E é indicado para problemas grandes, tendo em vista o número de avaliações que são utilizados nas primeira e segunda etapas. Isso significa dizer que, caso utilize-se um número máximo de avaliações de 1.000.000, 30% deste valor é dedicado às primeira e segunda etapas, ou seja, 300.000 avaliações. Entretanto, apesar do método de evolução em 3 etapas ser o único método capaz de encontrar soluções factíveis em todas as execuções independentes para todos os problemas, ele também não resolve o problema da escalabilidade pois a razão entre a mediana do número de avaliações e a quantidade de entradas dos problemas não é constante.

Tabela 6.17: Resultados dos experimentos computacionais do X-CGP-3E em comparação ao X-SG-MUX e ao X-SAMGAM. (P: Problema, Melhor: menor número de avaliações, Q1: primeiro quartil, Med: mediana, Q3: terceiro quartil, DP: desvio padrão, Pior: maior número de avaliações, DAM: desvio absoluto da mediana, IIQ: intervalo inter-quartil, TS: taxa de sucesso).

P	Método	Melhor	Q1	Med	Q3	Média	DP	Pior	DAM	IIQ	TS
dk27	CGP	-	-	-	-	-	-	-	-	-	0%
	X-SG-MUX	-	-	-	-	-	-	-	-	-	0%
	X-CGP-3E	**123745**	**138718,75**	**156512,5**	**182351,25**	**158624,5**	**2,76E+4**	**198965**	**24492,5**	**43632,5**	**100%**
clpl	CGP	211350	211350	**211350**	**211350**	**211350**	-	**211350**	-	211350	10%
	X-SG-MUX	**100466**	**148272,5**	216526	287237	218983,5	1,08E+5	342416	84189	138964,5	80%
	X-CGP-3E	180001	213370,25	228051,5	348216	268011,3	**8,13E+4**	388787	**42535,5**	**134845,75**	**100%**
br1	CGP	-	-	-	-	-	-	-	-	-	0%
	X-SG-MUX	-	-	-	-	-	-	-	-	-	0%
	X-CGP-3E	**403575**	**486726,25**	**569877,5**	**653028,75**	**569877,5**	**2,35E+5**	**736180**	**166302,5**	**166302,5**	**100%**
newtpla1	CGP	-	-	-	-	-	-	-	-	-	0%
	X-SG-MUX	-	-	-	-	-	-	-	-	-	0%
	X-CGP-3E	**60001**	**60440,25**	**62552,5**	**67053,5**	**65262,8**	**6,25E+3**	**76163**	**2525,5**	**6613,25**	**100%**
newtpla2	CGP	-	-	-	-	-	-	-	-	-	0%
	X-SG-MUX	189299	241964	278260	319108	273980	6,08E+4	341269	40848	77144	50%
	X-CGP-3E	**156588**	**171903**	**209780**	**240556,75**	**212944,1**	**4,82E+4**	**298121**	**37462,5**	**68653,75**	**100%**

7 CONCLUSÕES E TRABALHOS FUTUROS

Na eletrônica digital o projeto e otimização de circuitos lógicos são tarefas de grande importância, seja na necessidade de implementar uma nova funcionalidade ou de atender outros critérios, tais como espaço e tamanho a serem minimizados, consumo de potência, *delay*, tolerância a falhas, entre outros. Para o projeto tradicional utiliza-se, além das regras de simplificação da álgebra de Boole, ferramentas gráficas e computacionais. Apesar do uso tais técnicas, o projeto é um processo de grande complexidade que demanda tempo e experiência de conjuntos de regras específicas por parte dos projetistas, principalmente quando os circuitos são grandes. Na década 90 a então área denominada de *hardware* evolutivo começou a receber mais atenção mostrando a capacidade de obter projetos inovadores a partir dos métodos de computação evolucionista. Dentre as técnicas de computação evolucionista relatadas na literatura a Programação Genética Cartesiana (CGP) é apontada como o método mais eficiente para a evolução de circuitos digitais.

Esta dissertação trouxe novas abordagens para a evolução de circuitos lógicos combinacionais via Programação Genética Cartesiana (CGP). Neste trabalho foram apresentados dois novos operadores genéticos para CGP: um operador de mutação (denominado GAM) e um operador de recombinação. Além disso, foi apresentada uma proposta que combina dois operadores de mutação (SAM e GAM) modificando a estratégia evolutiva da CGP e duas propostas que utilizam multiplexadores.

A literatura relata poucas tentativas bem sucedidas de operadores de recombinação para CGP, sendo ressaltado que tais operadores mostraram-se prejudiciais aos subgrafos da CGP. O operador de recombinação proposto é capaz de unir os subgrafos que codificam as melhores saídas dentre todos os indivíduos da população. Desta forma cria-se um super-indivíduo cuja aptidão é sempre maior ou igual à aptidão do melhor indivíduo presente na população. Esse operador de recombinação é capaz de aprimorar consideravelmente a exploração do espaço de busca da CGP. A capacidade de sempre utilizar os melhores subgrafos via recombinação é fator decisivo para a factibilidade.

O operador de mutação proposto, denominado GAM, atua sempre no pior subgrafo do indivíduo selecionado. Um dos nós ativos de seu subgrafo é selecionado e modificado. Além disso, nenhum nó inativo sofre mutação. O GAM mostrou-se capaz de reduzir

a quantidade de avaliações necessárias para obter um circuito factível apesar de obter baixas taxas de sucesso. Entretanto, quando este operador de mutação é utilizado juntamente com o operador de recombinação (abordagem X-GAM), o desempenho melhora consideravelmente.

Apesar disso, foi possível combinar as altas taxas de sucesso das abordagens X-CGP com a baixa quantidade de avaliações da função objetivo com as abordagens que utilizam GAM, gerando uma nova abordagem (X-SAMGAM) que modifica a ES (estratégia evolutiva) comumente utilizada pela CGP. Este método requereu menos cálculos de função objetivo para encontrar circuitos lógicos combinacionais factíveis quando comparado aos métodos anteriores: CGP, X-CGP, GAM e X-GAM. Além disso, a otimização do projeto foi avaliada neste caso e os resultados indicaram que o X-SAMGAM obteve soluções compatíveis com aquelas encontradas pela CGP padrão.

Esta dissertação traz também um estudo sobre as dificuldades da CGP quando aplicada ao projeto de circuitos lógicos combinacionais. As investigações se concentraram no comportamento da CGP na busca por circuitos factíveis através de um variado conjunto de expressões lógicas. Os resultados indicam a CGP apresenta maior facilidade em obter circuitos cujas expressões podem ser simplificadas. Essa conclusão serviu de base para propor os métodos baseados em multiplexadores: X-SAMGAM-MUX que utiliza multiplexadores como elemento lógico dentro do conjunto de funções da CGP e o X-CGP-3E que consiste em uma evolução em 3 etapas. Os multiplexadores são úteis, pois conseguem trabalhar com tabelas verdade intermediárias através de seus controles e que são flexíveis quando suas entradas são iguais pois neste caso gera-se condições de irrelevância na tabela verdade de controle.

A abordagem X-SAMGAM-MUX combina a modificação da ES da abordagem anterior com o uso de multiplexadores como elemento lógico no conjunto de funções da CGP. O MUX utilizado é um elemento que possui 2 entradas primárias, um controle e uma única saída. O uso de multiplexadores como elemento lógico mostrou-se bastante eficiente para a diminuição do número de avaliações da função objetivo para encontrar circuitos factíveis além de aumentar consideravelmente a taxa de sucesso dos problemas utilizados. Este bom desempenho deve-se à capacidade dos multiplexadores gerarem tabelas verdade intermediárias, fazendo com que a busca da CGP seja consideravelmente facilitada. Os resultados mostram que o método X-SAMGAM-MUX obteve melhora de 69% em média

para os problemas testados no que diz respeito ao número de avaliações necessárias para obter uma solução factível além de ter sido capaz de obter soluções para o problema wim diferentemente do X-SAMGAM.

O método de evolução em 3 etapas, denominado X-CGP-3E, funciona através do acoplamento de um multiplexador de 2 entradas em cada uma das saídas do circuito em questão. Durante as primeira e segunda etapas o objetivo é encontrar uma solução totalmente funcional quando consideradas as duas entradas do multiplexador. Neste momento o circuito de controle não é levado em consideração. Uma vez obtido um circuito factível considerando as entradas do multiplexador a segunda etapa maximiza o número de similaridades entre essas entradas, pois, neste caso, gera-se situações de irrelevância (*don't cares*) na tabela verdade do controle do multiplexador que será evoluído na terceira etapa. Consequentemente, na terceira etapa, ao evoluir o circuito de controle do multiplexador, os *don't cares* são utilizados a fim de criar blocos de simplificação que se mostraram benéficos para o bom desempenho da busca da CGP. Além disso, o método X-CGP-3E apresentou bom desempenho para evoluir circuitos maiores (em número de entradas) quando comparados aos demais métodos aqui apresentados e à CGP padrão. Todas as abordagens relatadas na literatura que superaram a evolução de multiplicadores 5x5-bits consistem em variantes híbridas da CGP que não a utilizam como heurística principal. Inclusive para a evolução de multiplicadores 4×4-bits comumente utiliza-se computação paralela. O uso destes multiplexadores acoplados às saídas do circuito fazem com que as primeira e segunda etapas do método proposto busquem simplificar a tabela verdade do controle através da maximização do número de *don't cares* na mesma. Este método foi capaz de evoluir um circuito de 12 entradas e 8 saídas.

Os métodos propostos aumentaram a capacidade de exploração do espaço de busca da CGP, entretanto alguns pontos devem ser melhorados. No que diz respeito ao X-CGP faz-se necessário um novo método para alocação dos nós dos subgrafos que devem ser incorporados. Este pode ser um problema crítico no uso do X-CGP, principalmente quando o circuito em questão possui muitas saídas e, desta forma um único cromossomo necessita alocar todas estas saídas. Quando uma melhora ocorre em uma única saída e o subgrafo que a codifica possui sua quantidade de nós ativos aumentado por conta de uma mutação, este subgrafo deve ser alocado no mesmo espaço máximo (tamanho do cromossomo) juntamente com as demais saidas. Além disso, é interessante realizar um

novo estudo de parâmetros que utilize uma maior quantidade de circuitos de tal modo que obtenha-se um conjunto representativo e que os parâmetros obtidos possam ser, de fato, melhores que os parâmetros tradicionais.

Em relação à investigação de dificuldades da CGP faz-se necessária uma investigação mais profunda sobre o comportamento da CGP e também do X-CGP para circuitos maiores. Conforme os experimentos mostraram, o X-CGP tende a não apresentar um comportamento superior ao da CGP padrão quando os circuitos possuem poucas saídas.

Nas abordagens com multiplexadores, estudos preliminares foram realizados no método X-CGP-3E utilizando uma função de aptidão com bonificação para quando obtém-se *bits* 1 na tabela de controle. Como a CGP apresenta facilidade para obter circuitos cujas funções booleanas são simplificáveis, maximizar o número de *bits* 1 pode auxiliar na obtenção de blocos de simplificação. Além disso, no que tange à quantidade de recursos computacionais disponibilizados para as primeira e segunda etapas, faz-se necessário um estudo sobre a quantidade ideal, tendo em vista que o valor de 30% foi escolhido empiricamente em experimentos preliminares.

No que diz respeito à metaheurística CGP em si, é interessante estender a capacidade da CGP através de permitir a evolução de circuitos lógicos sequenciais, cuja presença de elementos de memória tais como *flip-flops* são naturalmente opostos à codificação em grafos direcionados acíclicos da CGP. Além disso, adaptar os métodos propostos aqui para uma otimização envolvendo múltiplos e conflitantes objetivos. Circuitos tolerantes à falhas são de suma importância no mundo real tais como em equipamentos eletromédicos ou em equipamentos de controle de tráfego, por exemplo. Quando envolve-se otimização multiobjetivo em circuitos as técnicas humanas são bastante limitadas e a literatura especializada já mostra uma grande capacidade de obtenção de bons circuitos neste quesito via computação evolucionista. Pretende-se também utilizar as técnicas aqui apresentadas para o projeto de circuitos visando a minimização do custo de produção. O número de elementos lógicos é uma das funções objetivo mais comuns da literatura. Entretanto, diferentes portas possuem diferentes custos, levando a uma aproximação inadequada do custo de produção. Uma alternativa é adotar o número de transistores que formam a solução candidata como função objetivo. Por fim, utilizar a CGP na modelagem de redes de regulação gênica através de circuitos lógicos combinacionais também é uma linha de pesquisa importante.

REFERÊNCIAS

BARBOSA, H. J.; BERNARDINO, H. S.; BARRETO, A. M. Using performance profiles to analyze the results of the 2006 cec constrained optimization competition. In: IEEE. **Proceedings of the Congress on Evolutionary Computation (CEC)**, 2010. p. 1–8.

BRAYTON, R. K.; HACHTEL, G. D.; MCMULLEN, C.; SANGIOVANNI-VINCENTELLI, A. **Logic minimization algorithms for VLSI synthesis**, 1984.

BREMNER, P.; SAMIE, M.; DRAGFFY, G.; PIPE, T.; WALKER, J. A.; TYRRELL, A. M. Evolving digital circuits using complex building blocks. In: SPRINGER. **International Conference on Evolvable Systems**, 2010. p. 37–48.

CAI, X.; SMITH, S. L.; TYRRELL, A. M. Positional independence and recombination in cartesian genetic programming. In: SPRINGER. **European Conference on Genetic Programming**, 2006. p. 351–360.

CLEGG, J.; WALKER, J. A.; MILLER, J. F. A new crossover technique for cartesian genetic programming. In: ACM. **Proceedings of the 9th annual conference on Genetic and evolutionary computation**, 2007. p. 1580–1587.

COELLO, C.; AGUIRRE, A.; BUCKLES, B. Evolutionary multiobjective design of combinational logic circuits. In: IEEE. **Proceedings of The Second NASA/DoD Workshop on Evolvable Hardware**, 2000. p. 161–170.

COELLO, C. A.; CHRISTIANSEN, A. D.; AGUIRRE, A. H. Automated design of combinational logic circuits using genetic algorithms. In: **Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms**, 1997. p. 335–338.

COELLO, C. A. C.; AGUIRRE, A. H. Design of combinational logic circuits through an evolutionary multiobjective optimization approach. **AI EDAM**, Cambridge University Press, v. 16, n. 1, p. 39–53, 2002.

COELLO, C. A. C.; ALBA, E.; LUQUE, G. Comparing different serial and parallel heuristics to design combinational logic circuits. In: IEEE. **Proceedings of NASA/DoD Conference on Evolvable Hardware**, 2003. p. 3–12.

COELLO, C. A. C.; CHRISTIANSEN, A. D.; AGUIRRE, A. H. Use of evolutionary techniques to automate the design of combinational circuits. **International Journal of Smart Engineering System Design**, TAYLOR AND FRANCIS, v. 2, p. 299–314, 2000.

COELLO, C. A. C.; GUTIÉRREZ, R. L. Z.; GARCÍA, B. M.; AGUIRRE, A. H. Automated design of combinational logic circuits using the ant system. **Engineering Optimization**, Taylor & Francis, v. 34, n. 2, p. 109–127, 2002.

COELLO, C. A. C.; LUNA, E. H.; AGUIRRE, A. H. Use of particle swarm optimization to design combinational logic circuits. In: SPRINGER. **International Conference on Evolvable Systems**, 2003. p. 398–409.

COELLO, C. A. C.; ZAVALA, R. L. G.; GARCÍA, B. M.; AGUIRRE, A. H. Ant colony system for the design of combinational logic circuits. In: SPRINGER. **International Conference on Evolvable Systems**, 2000. p. 21–30.

COELLO, C. C.; LUNA, E. H.; AGUIRRE, A. H. A comparative study of encodings to design combinational logic circuits using particle swarm optimization. In: IEEE. **Proc. of NASA/DoD Conference on Evolvable Hardware**, 2004. p. 71–78.

DARWIN, C. **On the Origin of Species by Means of Natural Selection, 1859**, 1969.

EIBEN, A. E.; SMITH, J. E. *et al.* **Introduction to evolutionary computing**, 2003.

FRIEDMAN, G. J. **Selective feedback computers for engineering synthesis and nervous system analogy.** Dissertação (Mestrado) — UCLA, Engineering, 1956.

GARCÍA, B. M.; COELLO, C. A. C. An approach based on the use of the ant system to design combinational logic circuits. **Mathware and Soft Computing**, v. 9, n. 2-3, p. 235–250, 2002.

GOLDBARG, E.; GOLDBARG, M.; LUNA, H. **Otimização combinatória e metaheurísticas: algoritmos e apliacações**, 2017.

GOLDBERG, D. E. **others, Genetic algorithms in search, optimization, and machine learning, vol. 412**, 1989.

GOLDMAN, B. W.; PUNCH, W. F. Reducing wasted evaluations in cartesian genetic programming. In: SPRINGER. **European Conference on Genetic Programming**, 2013. p. 61–72.

GOLDMAN, B. W.; PUNCH, W. F. Analysis of cartesian genetic programming's evolutionary mechanisms. **IEEE Trans. on Evolutionary Computation**, IEEE, v. 19, n. 3, p. 359–373, 2015.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**, 1975.

HUSA, J.; KALKREUTH, R. A comparative study on crossover in cartesian genetic programming. In: SPRINGER. **European Conference on Genetic Programming**, 2018. p. 203–219.

IMAMURA, K.; FOSTER, J. A.; KRINGS, A. W. The test vector problem and limitations to evolving digital circuits. In: IEEE. **Proceedings of The Second NASA/DoD Workshop on Evolvable Hardware**, 2000. p. 75–79.

KALGANOVA, T. Circuit layout evolution: an evolvable hardware approach. In: IET, 1999.

KALGANOVA, T.; MILLER, J. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In: IEEE. **Proceedings of the First NASA/DoD Workshop on Evolvable Hardware**, 1999. p. 54–63.

KALGANOVA, T.; MILLER, J.; LIPNITSKAYA, N. Multiple-valued combinational circuits synthesized using evolvable hardware approach. Citeseer, 1998.

KALGANOVA, T.; MILLER, J. F.; FOGARTY, T. C. Some aspects of an evolvable hardware approach for multiple-valued combinational circuit design. In: SPRINGER. **International Conference on Evolvable Systems**, 1998. p. 78–89.

KALKREUTH, R.; RUDOLPH, G.; DROSCHINSKY, A. A new subgraph crossover for cartesian genetic programming. In: SPRINGER. **European Conference on Genetic Programming**, 2017. p. 294–310.

KAUFMANN, P.; PLATZNER, M. Advanced techniques for the creation and propagation of modules in cartesian genetic programming. In: ACM. **Proc. of the Conference on Genetic and Evolutionary Computation**, 2008. p. 1219–1226.

KOZA, J. R. **Genetic programming: On the programming of computers by means of natural selection. MA**, 1992.

LOUIS, S. J. **Genetic algorithms as a computational tool for design**. Tese (Doutorado) — Indiana University Bloomington, IN, 1993.

MANFRINI, F.; BARBOSA, H. J.; BERNARDINO, H. S. Optimization of combinational logic circuits through decomposition of truth table and evolution of sub-circuits. In: IEEE. **Proceedings of the Congress on Evolutionary Computation (CEC)**, 2014. p. 945–950.

MANFRINI, F. A.; BERNARDINO, H. S.; BARBOSA, H. J. A novel efficient mutation for evolutionary design of combinational logic circuits. In: SPRINGER. **International Conference on Parallel Problem Solving from Nature**, 2016. p. 665–674.

MANFRINI, F. A. L. *et al.* Estratégias de busca no projeto evolucionista de circuitos combinacionais. Universidade Federal de Juiz de Fora (UFJF), 2017.

MCCLUSKEY, E. J. Minimization of boolean functions. **Bell Labs Technical Journal**, Wiley Online Library, v. 35, n. 6, p. 1417–1444, 1956.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)**, 1996. ISBN 3-540-60676-9.

MILLER, J. What bloat? cartesian genetic programming on boolean problems. In: SAN FRANCISCO, CALIFORNIA, USA. **2001 Genetic and Evolutionary Computation Conference Late Breaking Papers**, 2001. p. 295–302.

MILLER, J. F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: MORGAN KAUFMANN PUBLISHERS

INC. **Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2**, 1999. p. 1135–1142.

MILLER, J. F. **Cartesian genetic programming**, 2011. 344 p.

MILLER, J. F.; HARTMANN, M. Evolving messy gates for fault tolerance: some preliminary findings. In: IEEE. **eh**, 2001. p. 0116.

MILLER, J. F.; JOB, D.; VASSILEV, V. K. Principles in the evolutionary design of digital circuits—part i. **Genetic programming and evolvable machines**, Springer, v. 1, n. 1-2, p. 7–35, 2000.

MILLER, J. F.; JOB, D.; VASSILEV, V. K. Principles in the evolutionary design of digital circuits—part ii. **Genetic programming and evolvable machines**, Springer, v. 1, n. 3, p. 259–288, 2000.

MILLER, J. F.; SMITH, S. L. Redundancy and computational efficiency in cartesian genetic programming. **IEEE Transactions on Evolutionary Computation**, IEEE, v. 10, n. 2, p. 167–174, 2006.

MILLER, J. F.; THOMSON, P. Combinational and sequential logic optimisation using genetic algorithms. IET, 1995.

MILLER, J. F.; THOMSON, P.; FOGARTY, T. **Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study**, 1997.

MOORE, P. W.; VENAYAGAMOORTHY, G. K. Evolving digital circuits using hybrid particle swarm optimization and differential evolution. **International Journal of neural systems**, World Scientific, v. 16, n. 03, p. 163–177, 2006.

PÉREZ, E. I.; COELLO, C. A. C.; HERNÁNDEZ-AGUIRRE, A.; RAMÍREZ, A. V. Genetic algorithms and case-based reasoning as a discovery and learning machine in the optimization of combinational logic circuits. In: SPRINGER. **Mexican International Conference on Artificial Intelligence**, 2002. p. 128–137.

POLI, R.; LANGDON, W. B.; MCPHEE, N. F.; KOZA, J. R. **A field guide to genetic programming**, 2008.

SEKANINA, L. Evolutionary design of digital circuits: Where are current limits? In: IEEE. **First NASA/ESA Conference on Adaptive Hardware and Systems (AHS)**, 2006. p. 171–178.

SILVA, J. E. da; BERNARDINO, H. Cartesian genetic programming with crossover for designing combinational logic circuits. In: IEEE. **Proc. of the 7th Brazilian Conference on Intelligent Systems (BRACIS)**, 2018. p. 145–150.

SILVA, J. E. H. da. Biased mutation and tournament selection approaches for designing combinational logic circuits via cartesian genetic programming. **Anais do Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)**, p. 835–846, 2018. Disponível em: <http://portaldeconteudo.sbc.org.br/index.php/eniac/article/view/4471>.

SIMON, D. **Evolutionary optimization algorithms**, 2013.

STEPNEY, S.; ADAMATZKY, A. **Inspired by Nature: Essays Presented to Julian F. Miller on the Occasion of His 60th Birthday**, 2018.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais: princípios e aplicações**, 2003.

TURNER, A. J.; MILLER, J. F. Neutral genetic drift: an investigation using cartesian genetic programming. **Genetic Programming and Evolvable Machines**, Springer, v. 16, n. 4, p. 531–558, 2015.

VASICEK, Z. Cartesian gp in optimization of combinational circuits with hundreds of inputs and thousands of gates. In: SPRINGER. **European Conference on Genetic Programming**, 2015. p. 139–150.

VASICEK, Z. Bridging the gap between evolvable hardware and industry using cartesian genetic programming. In: **Inspired by Nature**, 2018. p. 39–55.

VASICEK, Z.; SEKANINA, L. How to evolve complex combinational circuits from scratch? In: IEEE. **Proceedings of the Conference on Evolvable Systems (ICES)**, 2014. p. 133–140.

VASICEK, Z.; SEKANINA, L. Evolutionary approach to approximate digital circuits design. **IEEE Transactions on Evolutionary Computation**, IEEE, v. 19, n. 3, p. 432–444, 2015.

VASSILEV, V. K.; MILLER, J. F.; FOGARTY, T. C. Digital circuit evolution: the ruggedness and neutrality of two-bit multiplier landscapes. IET, 1999.

WALKER, J. A.; MILLER, J. F. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. **IEEE Transactions on Evolutionary Computation**, IEEE, v. 12, n. 4, p. 397–417, 2008.

WALKER, J. A.; MILLER, J. F.; CAVILL, R. A multi-chromosome approach to standard and embedded cartesian genetic programming. In: ACM. **Proceedings of the 8th annual conference on Genetic and evolutionary computation**, 2006. p. 903–910.

YU, T.; MILLER, J. Neutrality and the evolvability of boolean function landscape. In: SPRINGER. **European Conference on Genetic Programming**, 2001. p. 204–217.