Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas/Engenharia
Programa de Pós-Graduação em Modelagem Computacional

Francisco Augusto Lima Manfrini

Estratégias de Busca no Projeto Evolucionista de Circuitos Combinacionais

Juiz de Fora
2017

Francisco Augusto Lima Manfrini

Estratégias de Busca no Projeto Evolucionista de Circuitos Combinacionais

Tese apresentada ao Programa de Pós-Graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora, na área de concentração em Modelagem Computacional, como requisito parcial para obtenção do título de Doutor em Modelagem Computacional.

Orientador: D.Sc. Helio José Corrêa Barbosa

Coorientador: D.Sc. Heder Soares Bernardino

Juiz de Fora

2017

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

Manfrini, Francisco Augusto Lima.
Estratégias de Busca no Projeto Evolucionista de Circuitos Combinacionais / Francisco Augusto Lima Manfrini. -- 2017.
143 f. : il.

Orientador: Helio José Corrêa Barbosa
Coorientador: Heder Soares Bernardino
Tese (doutorado) - Universidade Federal de Juiz de Fora, ICE/Engenharia. Programa de Pós-Graduação em Modelagem Computacional, 2017.

1. Programação Genética Cartesiana. 2. Computação Evolucionista. 3. Hardware Evolutivo. 4. Circuitos Lógicos. I. Barbosa, Helio José Corrêa, orient. II. Bernardino, Heder Soares, coorient. III. Título.

Francisco Augusto Lima Manfrini

Estratégias de Busca no Projeto Evolucionista de Circuitos Combinacionais

Tese apresentada ao Programa de Pós-Graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora, na área de concentração em Modelagem Computacional, como requisito parcial para obtenção do título de Doutor em Modelagem Computacional.

Aprovada em: 23 de Fevereiro de 2017

BANCA EXAMINADORA

Prof. D.Sc. Helio José Corrêa Barbosa - Orientador
Universidade Federal de Juiz de Fora
Laboratório Nacional de Computação Científica

Prof. D.Sc. Heder Soares Bernadino - Coorientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Douglas Adriano Augusto
Fundação Oswaldo Cruz

Prof. D.Sc. Leonardo Goliatt da Fonseca
Universidade Federal de Juiz de Fora

Profa. D.Sc. Luciana Conceição Dias Campos
Universidade Federal de Juiz de Fora

Prof. D.Sc. Nelson Francisco Favilla Ebecken
Universidade Federal do Rio de Janeiro

Prof. D.Sc. Wilian Soares Lacerda
Universidade Federal de Lavras

AGRADECIMENTOS

A Deus, o maior de todos os cientistas, que tem o poder de dar a vida!

A minha esposa Rozângela e ao meu filho Marcos, que têm o poder de renovar todas as minhas forças com apenas um sorriso.

Aos meus pais que se sacrificaram e nunca mediram esforços em prol da minha educação.

Aos meus irmãos, cunhados e sobrinhos pelos momentos felizes.

Aos meus orientadores Prof. Helio e Prof. Heder, pela confiança, companheirismo e competência.

Aos amigos do mestrado e doutorado em Modelagem Computacional, em especial ao Anderson, Bernardo e Rafael, pelas longas discussões que também contribuíram para minha formação.

Aos professores do Programa de Pós Graduação em Modelagem Computacional, pelos ensinamentos e profissionalismo.

Ao IFET pela minha liberação nos dois últimos anos do doutorado e por sempre apoiar a qualificação dos servidores.

"The love for all living creatures is the most noble attribute of man."
Charles Darwin

RESUMO

A computação evolucionista tem sido aplicada em diversas áreas do conhecimento para a descoberta de projetos inovadores. Quando aplicada na concepção de circuitos digitais o problema da escalabilidade tem limitado a obtenção de circuitos complexos, sendo apontado como o maior problema em hardware evolutivo. O aumento do poder dos métodos evolutivos e da eficiência da busca constitui um importante passo para melhorar as ferramentas de projeto. Este trabalho aborda a computação evolutiva aplicada ao projeto de circuito lógicos combinacionais e cria estratégias para melhorar o desempenho dos algoritmos evolutivos. As três principais contribuições resultam dessa tese são: (*i*) o desenvolvimento de uma nova metodologia que ajuda a compreensão das causas fundamentais do sucesso/fracasso evolutivo; (*ii*) a proposta de uma heurística para a semeadura da população inicial; os resultados mostram que existe uma correlação entre a topologia da população inicial e a região do espaço de busca explorada; e (*iii*) a proposta de um novo operador de mutação denominado *Biased SAM*; verificou-se que esta mutação pode guiar de maneira efetiva a busca. Nos experimentos realizados o operador proposto é melhor ou equivalente ao operador de mutação tradicional. Os experimentos computacionais que validaram as respectivas contribuições foram feitos utilizando circuitos *benchmark* da literatura.

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

ABSTRACT

Evolutionary computation has been applied in several areas of knowledge for discovering Innovative designs. When applied to a digital circuit design the scalability problem has limited the obtaining of complex circuits, being pointed as the main problem in the evolvable hardware field. Increased power of evolutionary methods and efficiency of the search constitute an important step towards improving the design tool. This work approaches the evolutionary computation applied to the design of combinational logic circuits and creates strategies to improve the performance of evolutionary algorithms. The three main contributions result from this thesis are: (*i*) the developement of a methodology that helps to understand the success/failure of the genetic modifications that occur along the evolution; (*ii*) a heuristic proposed for seeding the initial population; the results showed there is a correlation between the topology of the initial population and the region of the search space which is explored. (*iii*) a proposal of a new mutation operator referred to as Biased SAM; it is verified that this operator can guide the search. In the experiments performed the mutation proposed is better than or equivalent to the traditional mutation. The computational experiments that prove the efficiency of the respective contributions were made using benchmark circuits of the literature.

Key-words: Cartesian Genetic Programming, Evolutionary Computation, Evolvable Hardware, Combinational Logic Circuits.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CGP	*Cartesian Genetic Programming*
CLC	*Circuito Lógico Combinacional*
CDA	*Conversor Digital Analógico*
E.E	*Estratégia Evolutiva*
FOB	*Função Objetivo*
FPGA	*Field Programmable Gate Array.*
NDG	*Neutral Genetic Drift*
n_c	*número de colunas*
n_r	*número de linhas*
ON	Ligada, utilizado para designar Neutralidade ligada.
OFF	Desligada, utilizado para designar Neutralidade desligada.
pm	pontos de mutação
lb	*levels-back*
SAM	*Single Active Mutation*

LISTA DE SÍMBOLOS

μ	Número de progenitores da estratégia evolutiva.
λ	Número de filhos da estratégia evolutiva.
Γ	Conjunto de operadores lógicos
$\square$	Mediana do número de nós ativos para a semeadura quando *Levels-back*= 25%
$\circ$	Mediana do número de nós ativos para a semeadura quando *Levels-back*= 50%
$*$	Mediana do número de nós ativos para a semeadura quando *Levels-back*= 100%

SUMÁRIO

1 INTRODUÇÃO

A luta pela sobrevivência das espécies ao longo de milhões de anos pode ser considerada o maior desafio já enfrentado pelos seres vivos, o qual foi resolvido de maneira eficaz pela natureza através da evolução das espécies. A teoria da evolução tem como fundamento os postulados estabelecidos pela teoria da seleção natural, publicada por Charles Darwin em seu clássico livro *"The origin of species (1859)"* (DARWIN, 1969).

De acordo com a teoria de Darwin, a evolução dos seres vivos ocorre devido à seleção natural e à adaptação ao ambiente. Assim, mudanças genéticas e variações bem sucedidas em indivíduos, numa determinada população, podem se propagar ao longo das gerações, por sua vez, variações indesejáveis tendem à extinção. Indivíduos mais adaptados ao meio ambiente têm maior probabilidade de sobrevivência e, com isso, as suas características genéticas também têm mais chances de serem propagadas para as gerações futuras.

A contínua evolução dos seres vivos e a sua adaptação ao meio ambiente podem ser consideradas como um processo de otimização. Esse modelo biológico inspirou o desenvolvimento de diversos algoritmos evolucionistas, tais como: Programação Evolucionária por Fogel em 1966 (FOGEL, 1966), Algoritmos Genéticos por Holland em 1975 (HOLLAND, 1975) , Estratégias de Evolução por Rechenberg em 1973 (EIGEN, 1973) e a Programação Genética por Koza em 1992 (KOZA, 1992).

A aplicação dos algoritmos genéticos em problemas de otimização torna-se interessante, principalmente quando a complexidade do espaço inviabiliza a utilização de técnicas clássicas (EIBEN; SMITH, 2003). Ressalta-se que o processo evolutivo não é limitado à otimização de parâmetros, o que faz com que possibilidades interessantes surjam no processo de otimização (MILLER; THOMSON; FOGARTY, 1997).

Esse paradigma vem sendo utilizada para resolver problemas práticos em diversos ramos da otimização. Segundo Koza (KOZA, 2010), através da computação evolucionária é possível projetar circuitos digitais de uma forma radicalmente diferente das técnicas tradicionais. Essa abordagem pode ser expressa como uma visão "caixa preta" do problema, onde mediante os dados de entrada e das saídas desejadas o algoritmo se encarrega de projetar a melhor configuração do circuito que atenda aos requisitos do projeto. A característica fundamental dessa técnica é que os detalhes dentro da "caixa preta" são codificados em cromossomos e submetidos aos processos habituais dos algoritmos evolutivos (MILLER; THOMSON; FOGARTY, 1997).

O projeto de circuitos eletrônicos é, geralmente, uma tarefa que exige conhecimento complexo de grandes coleções de regras específicas do domínio (MILLER; JOB; VASSILEV, 2000). Por sua vez, nas técnicas baseadas em algoritmos evolutivos, tal projeto além de ser menos dependente do especialista (automatizado), pode ser otimizado

quando comparado com as técnicas clássicas. A Figura 1 ilustra a comparação entre essas duas abordagens.

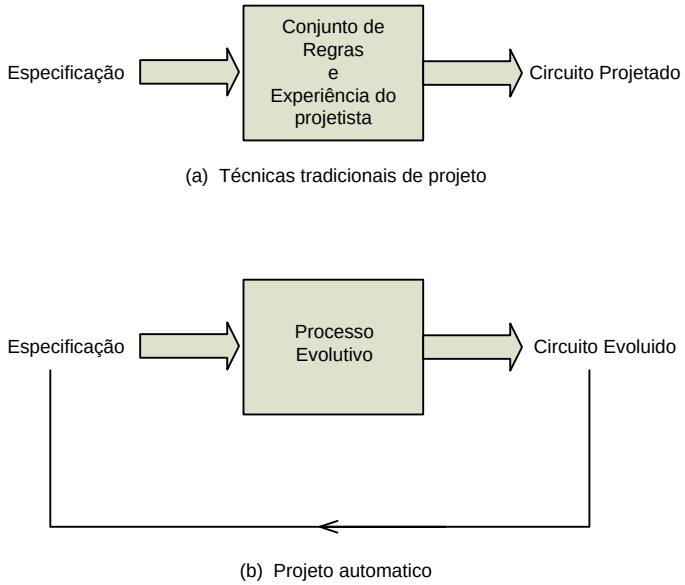


Figura 1 – Processo tradicional para o projeto de circuitos (a) e etapas da evolução de um circuito (b).

A literatura destaca a tese de George J. Friedman (FRIEDMAN, 1956), publicada em 1956, como a primeira tentativa de aplicar técnicas evolutivas para projetar circuitos eletrônicos. Entretanto somente a partir da década de 90 começaram a surgir trabalhos mostrando que é possível projetar circuitos eletrônicos de uma forma radicalmente diferente.

Através da aplicação de algoritmos evolucionistas é possível obter resultados competitivos e algumas vezes superiores aos projetos elaborados por especialistas. Essa linha de pesquisa foi denominada *Hardware Evolutivo* (VASICEK; SEKANINA, 2011) .

Engenheiros e projetistas estão limitados a um conjunto de regras convencionais e consideram sistemas hierarquicamente estruturados. Projetos automatizados oferecem a possibilidade de transcender a essas limitações e explorar um espaço muito mais rico de possibilidades (MILLER; JOB; VASSILEV, 2000).

Koza em seu artigo *Human - competitive results by genetic programming* (KOZA, 2010) descreveu 76 exemplos de resultados obtidos pela programação genética, os quais competem com os resultados produzidos por humanos. O autor ainda classificou a síntese de circuitos eletrônicos como um ramo promissor, prevendo que o aumento da disponibilidade do poder computacional deve resultar na produção de um crescente fluxo de inovações, acarretando também em resultados mais intrigantes e impressionantes.

Entretanto o projeto de circuitos via computação evolucionista não tem sido competitivo devido ao problema da escalabilidade (VASICEK; SEKANINA, 2014b; MILLER,

2011; VASICEK, 2015). Do ponto de vista da representação, o problema está relacionado ao aumento do tamanho dos cromossomos necessários para representar soluções complexas, o que acarreta num aumento do espaço de soluções candidatas, dificultando a busca (VASICEK, 2015).

Outra dificuldade está relacionada ao tempo de cálculo da função de avaliação, que cresce exponencialmente com o número de entradas (supondo circuitos combinatórios) (MILLER, 2011). Vasicek e Sekanina (VASICEK; SEKANINA, 2015) ressaltam que o tempo de projeto é um dos fatores que determinam a aplicabilidade de um método de concepção.

Para alcançar resultados inovadores no projeto de circuitos digitais via computação evolucionista, tal como sugerido por Koza (KOZA, 2010), é fundamental o desenvolvimento de estratégias que tornem a busca mais eficiente. Goldman (GOLDMAN; PUNCH, 2015) destaca que o entendimento de como os operadores genéticos interagem com a solução (mecanismo evolutivo) é um importante passo para melhorar a busca, mas a complexidade do mapeamento genótipo-fenótipo torna esse entendimento um desafio.

Neste contexto, uma das contribuições deste trabalho é o desenvolvimento de uma metodologia para extrair informações proveniente do processo evolutivo. Ressalta-se que a partir das informações obtidas é possível criar estratégias para uma busca mais eficiente.

1.1 OBJETIVOS

1.1.1 **Objetivo Geral**

O objetivo geral é investigar o projeto de circuitos lógicos combinacionais via computação evolucionista e os detalhes envolvidos no mecanismo evolutivo, visando a criação de estratégias para melhorar a eficiência da busca.

1.1.2 **Objetivos Específicos**

- Investigar as forças evolutivas exploradas pela CGP (*Cartesian Genetic Programming*), em especial a Deriva Gênica Neutra.
- Verificar as modificações na conformação dos indivíduos ao longo da evolução, tais como: troca dos operadores, topologia, número de nós ativos, entre outras.
- Investigar a existência da correlação entre a semeadura da população inicial e o espaço de busca explorado.
- Propor um novo operador de mutação para o projeto de circuitos lógicos combinacionais.
- Desenvolver algoritmos evolutivos para projetar circuitos lógicos combinacionais.

- Validar os algoritmos desenvolvidos, comparando-os com os melhores resultados encontrados na literatura.

1.2 ORGANIZAÇÃO DO TEXTO

O restante deste trabalho está dividido como segue. O Capítulo 2 apresenta a revisão bibliográfica relacionada ao projeto de circuitos combinacionais via computação evolucionista. O Capítulo 3 descreve os fundamentos teóricos do projeto de circuitos lógicos. O Capítulo 4 apresenta os conceitos relacionados aos algoritmos genéticos, programação genética e programação genética cartesiana. O Capítulo 5 descreve os métodos propostos, sendo que inicialmente é detalhada a metodologia criada para investigar o processo evolutivo. Essa metodologia desenvolvida motivou a criação de uma heurística para a semeadura da população inicial e de um operador de mutação enviesado, os quais são descritos na sequência do capítulo. O Capítulo 6 apresenta os experimentos computacionais que validaram a metodologia proposta. O Capítulo 7 apresenta as conclusões e possíveis trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

2.1 HISTÓRICO

O marco histórico que deu início à era moderna da eletrônica, se deu em 1947 com a invenção do transistor por William Shockley, John Bardeen e Walter Brattian, no Bell Laboratories, situado em New Jersey. Os três cientistas compartilharam o prêmio Nobel de Física em 1956 pela invenção.

Com a invenção do transistor, surgiu também o Vale do Silício, onde foram instaladas diversas empresas, como Shockley Semiconductor, Fairchild e Intel. Um dos primeiros feitos da Intel foi o desenvolvimento de transistores MOS, uma tecnologia que outras empresas demoraram vários anos para alcançar. Em 1958 Jack Kilby ganhou o Prêmio Nobel de Física pelo desenvolvimento do primeiro circuito integrado. Em 1971 a Intel lançou o primeiro microprocessador integrado denominado 4004, com aproximadamente 2300 transistores (VAHID; WILEY, 2006).

Nas últimas décadas, em especial a partir de 1970, a indústria de semicondutores obteve altas taxas de crescimento no volume de produção e na complexidade do sistema de hardware (CAMPOS, 2007). Os avanços tecnológicos permitiram a integração de grande capacidade computacional em dimensões ínfimas, o que permitiu a disseminação da eletrônica e da computação.

Atualmente, circuitos eletrônicos fazem parte do nosso cotidiano e estão presentes em todos os de tipos equipamentos tecnológicos, como veículos, aparelhos eletrodomésticos, computadores e equipamentos de comunicação e entretenimento. Todos esses equipamentos digitais são construídos sobre uma mesma base. Seja o mais poderoso microprocessador ou o mais simples relógio digital, todos são constituídos pelos mesmos elementos básicos: as portas lógicas (RECH, 2006).

Como consequência do crescimento da complexidade dos sistemas eletrônicos, algumas metodologias de projeto tornaram-se obsoletas, enquanto outras, ainda atuais, não conseguem manter uma taxa de crescimento equivalente (CAMPOS, 2007). As técnicas clássicas de projeto de circuitos muitas vezes se tornam de difícil execução, exigindo muito tempo e esforço dos projetistas devido à grande complexidade dos circuitos envolvidos (GUIMARÃES, 2006).

Técnicas tradicionais de projeto utilizam metodologia *top down* (MILLER; JOB; VASSILEV, 2000), e tendem a decompor o sistema em sub-sistemas; as técnicas evolutivas, por sua vez, estão livres para trabalhar de forma *bottom up*. Da crescente complexidade dos circuitos a serem projetados, ressulta a necessidade o desenvolvimento de ferramentas mais sofisticadas, que sejam mais autônomas, reduzindo a intervenção humana e a dependência de especialistas.

Diante dessa necessidade, ferramentas de projeto automático de circuitos eletrônicos baseadas em inteligência computacional passaram a ser desenvolvidas e adotadas (KARAKATIC; PODGORELEC; HERICKO, 2013), sendo a computação evolucionista apontada como a mais bem sucedida (KARAKATIC; PODGORELEC; HERICKO, 2013).

Em 1993 Louis desenvolveu um AG para projetar circuitos lógicos combinacionais e propôs uma codificação matricial para o cromosomo, conforme Figura 2. Cada elemento da matriz é considerado um gene, o qual é composto por uma porta lógica (AND, NOT, OR, XOR e WIRE) [1] e suas entradas, conforme Figura 3.

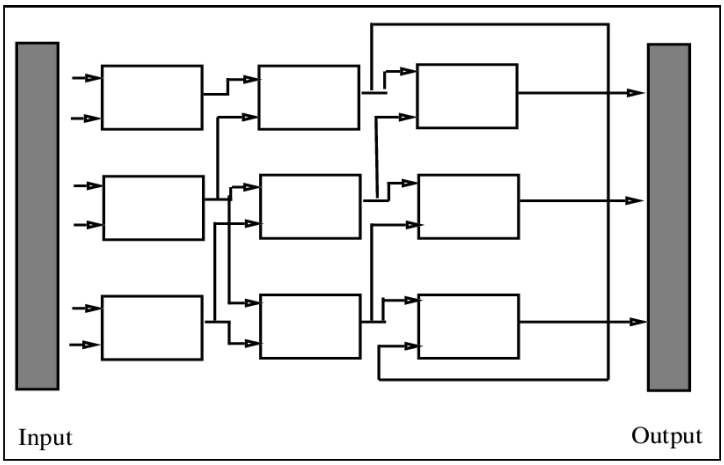


Figura 2 – Codificação matricial proposta por Louis - figura extraída de (LOUIS, 1993)

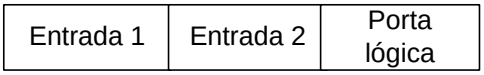


Figura 3 – Gene utilizado na codificação de Louis

Louis projetou circuitos somadores, que possuem mais de uma saída, e circuitos detectores de paridade com apenas uma saída. Na sua implementação as portas lógicas que não contribuem para a configuração do circuito são eliminadas. Um exemplo do circuito detector de paridade por Louis é mostrado na Figura 4.

A avaliação do cromossomo foi baseada em uma função de aptidão dinâmica. No início da pesquisa, apenas a validade das saídas são levadas em consideração, o AG basicamente explora o espaço de busca a procura de circuitos que atendam todos os requisitos da tabela verdade. Uma vez que as soluções aparecem a função aptidão se modifica, de tal forma que todos os projetos válidos são recompesados para cada WIRE que surge, ou seja, o AG busca maximizar o número de WIREs (equivalente a minimizar o número de portas lógicas).

[1] Para simplificação WIRE é considerado uma porta lógica, onde a saída é igual a entrada.

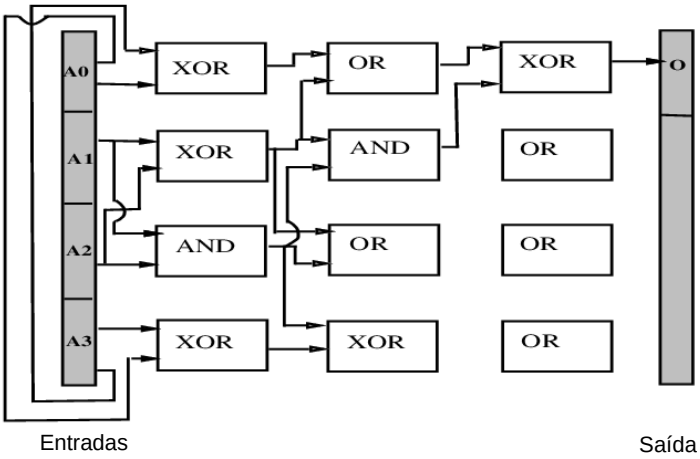


Figura 4 – Detector de paridade obtido por Louis - figura extraída de (LOUIS, 1993)

Miller (MILLER; THOMSON; FOGARTY, 1997; MILLER; JOB; VASSILEV, 2000), também utilizou a representação matricial na codificação, mas considerou que cada uma das células lógicas representadas na Figura 2 podem assumir a funcionalidade de qualquer porta lógica ou, alternativamente, de um multiplexador 2 x 1. Com a inclusão da possibilidade do multiplexador, o espaço de busca foi ampliado.

Ressalta-se que a função objetivo não fez distinção entre os custos de uma porta lógica e de um multiplexador, todos foram considerados como um elemento lógico. Miller destaca que para muitos circuitos haverá células lógicas que não estão efetivamente conectadas à saída do circuito e, portanto, devem ser removidas. O AG proposto evoluiu vários circuitos aritméticos interessantes, como somadores e multiplicadores, obtendo melhores resultados quando comparados às técnicas tradicionais.

Apesar do trabalho pioneiro de Friedman em 1956 (FRIEDMAN, 1956), somente a partir de 1993 com a publicação de Louis (LOUIS, 1993), começaram a aparecer resultados interessantes em projeto de circuitos lógicos combinacionais. A seguir serão relatados os principais trabalhos da literatura que utilizaram a codificação proposta por Louis em diferentes meta-heurísticas.

- Coello (COELLO; CHRISTIANSEN; AGUIRRE, 2000b) utilizou a codificação matricial proposta por Louis e implementou um AG denominado NGA. Para projetar um CLC a partir de uma tabela verdade, Coello sugere aumentar gradativamente a dimensão da matriz, começando pelo número de colunas, até que uma solução seja encontrada. Para os exemplos de 4 variáveis abordados pelo NGA, os resultados foram obtidos com uma matriz 5x5, o que gerou um espaço de busca da ordem de 10^{52}.

- Coello (COELLO; AGUIRRE; BUCKLES, 2000) implementou também um AG mul-

tiobjetivo com sub-populações denominado MGA, e resolveu os mesmos problemas anteriormente abordados pelo NGA. Além do objetivo de maximizar o número de WIRES presentes na matriz, o MGA considera cada restrição de igualdade imposta pela tabela verdade como um objetivo (COELLO; CHRISTIANSEN; AGUIRRE, 2000b). Dessa forma, em um sistema com n variáveis, tem-se uma tabela verdade com $m = 2^n$ linhas, e portanto, $m + 1$ objetivos a serem considerados. Em cada geração a população é dividida em $m + 1$ sub-populações e cada uma delas busca atender uma restrição separadamente. Os resultados do MGA foram superiores aos obtidos pelo NGA.

- Moore (MOORE; VENAYAGAMOORTHY, 2005) desenvolveu um algoritmo híbrido, denominado QEPSO, baseado no algoritmo de inspiração quântica (HAN; KIM, 2002), nas técnicas evolucionárias e na meta-heurística Otimização por Enxame de Partículas (PSO) (EBERHART; SHI, 2001). Para avaliação do indivíduo foram considerados dois objetivos, encontrar um circuito viável e minimizar o número de portas lógicas. Os circuitos projetados foram equivalentes aos do MGA, quando analisados em termos da quantidade de portas lógicas. Para as mesmas condições, o QEPSO sempre encontrava o melhor resultado, diferente do MGA.

- Otimização por Colônia de Formigas (ACO) também tem sido aplicada ao projeto de CLC, alguns exemplos mostrados em (GARCÍA; COELLO, 2002) os resultados do ACO superou o AG binário. Mostafa (ABD-EL-BARR et al., 2003) comenta que no ACO, além da cooperação entre os agentes (formigas), características de outras heurísticas também podem ser facilmente incorporadas para melhorar a solução e propõe, ainda, um ACO modificado.

- Coello (COELLO; ALBA; LUQUE, 2003) fez um estudo comparativo de heurísticas seriais e paralelas usadas para projetar CLCs, constatando-se que a hibridização de um algoritmo genético com um algoritmo de busca local (recozimento simulado) é benéfico, e que a paralelização não apenas introduz um aumento da velocidade, mas permite melhorar as soluções encontradas. O algoritmo híbrido implementado foi denominado GASA2.

Em 1997, Miller (MILLER; THOMSON; FOGARTY, 1997) destaca a existência de apenas 4 ou 5 grupos de pesquisas em *Hardware Evolutivo*. Entre 2000 e 2007, o grupo liderado por Coello publicou diversos trabalhos (COELLO; CHRISTIANSEN; AGUIRRE, 2000b; COELLO; LUNA; AGUIRRE, 2003; GARCÍA; COELLO, 2002; COELLO et al., 2000; COELLO; AGUIRRE, 2002; ALBA et al., 2007; COELLO; ALBA; LUQUE, 2003; AGUIRRE; COELLO, 2004), sendo apontado como uma referência na área (KARAKATIC; PODGORELEC; HERICKO, 2013).

A Figura 5 mostra alguns circuitos digitais encontrados na literatura[2], que foram otimizados através das técnicas evolutivas, e outros algoritmos bio-inspirados como Otimização por Colônia de Formigas (ACO) e Otimização por Enxame de Partículas (PSO).

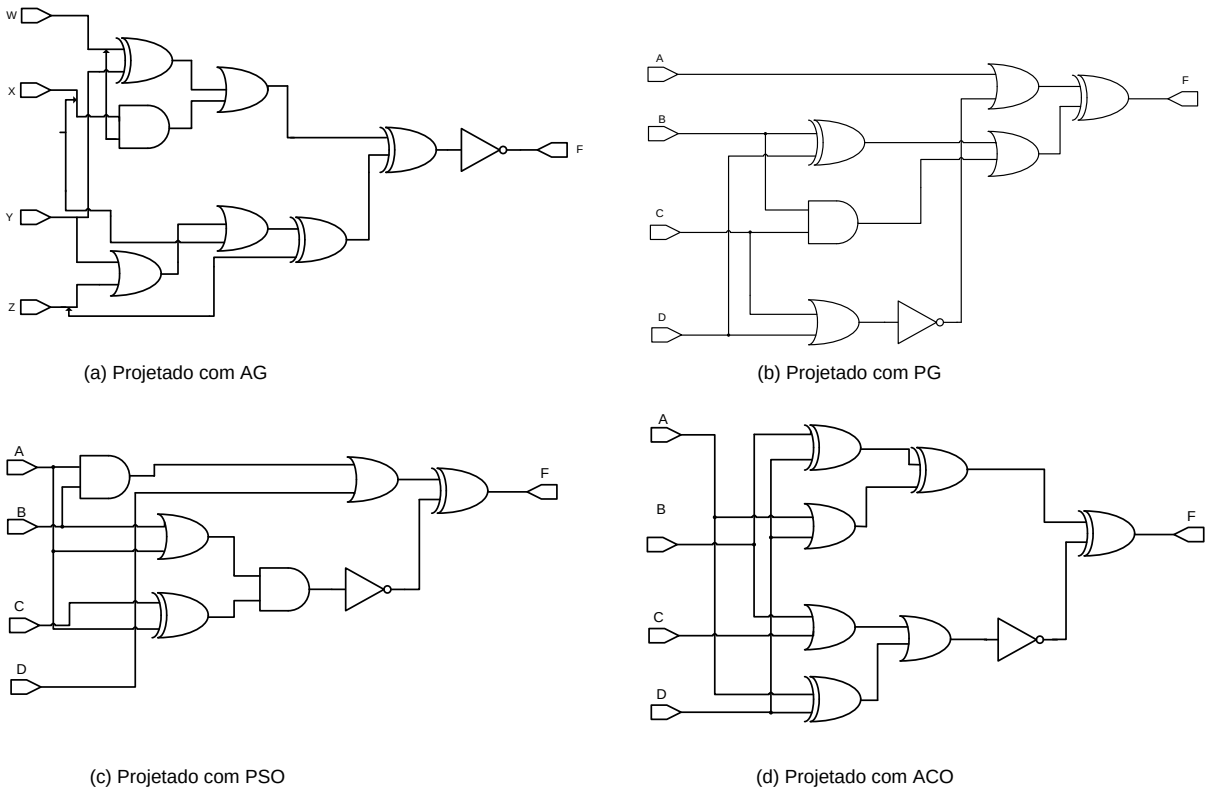


Figura 5 – Resultados obtidos com diferentes meta-heurísiticas para o projeto de circuitos lógicos combinacionais (COELLO; CHRISTIANSEN; AGUIRRE, 2000b; KARAKATIC; PODGORELEC; HERICKO, 2013; COELLO; LUNA; AGUIRRE, 2003; ABD-EL-BARR et al., 2003)

.

É importante destacar que a literatura relata vários trabalhos sobre a utilização de algoritmos evolucionistas e outras meta-heurísticas aplicados no projeto de circuitos digitais combinacionais. No entanto, com exceção da programação genética, onde se têm uma representação em árvore, diversos trabalhos utilizam a representação proposta na dissertação de Louis (LOUIS, 1993) de 1993, para a codificação do circuito (COELLO; CHRISTIANSEN; AGUIRRE, 2000b; KARAKATIC; PODGORELEC; HERICKO, 2013; COELLO; LUNA; AGUIRRE, 2003; MILLER; THOMSON; FOGARTY, 1997; COELLO; CHRISTIANSEN; AGUIRRE, 2000a; GARCÍA; COELLO, 2002; COELLO et al., 2000; COELLO; CHRISTIANSEN; AGUIRRE, 1997; COELLO; AGUIRRE, 2002; ALBA et al., 2007; MILLER; JOB; VASSILEV, 2000; MOORE; VENAYAGAMOORTHY, 2005; ANJOMSHOA; MAHANI, 2013; ANJOMSHOA; MAHANI; SADEGHIFARD, 2014).

[2] As tabelas verdade referentes a cada circuito são distintas; portanto os circuitos não podem ser comparados entre si.

2.2 PROJETO EVOLUTIVO

Em regra, os sistemas físicos, como antena, computadores, celulares, são elaborados por engenheiros e projetistas, a partir da utilização de um conjunto de normas e princípios. Esses projetos utilizam uma metodologia *top-down*, que começa com uma especificação detalhada do sistema. Essa concepção apresenta um grande contraste quando comparada ao mecanismo da evolução natural, que produziu uma extraordinária diversidade nos seres vivos (MILLER; JOB; VASSILEV, 2000).

Esse contraste evidenciado a partir da comparação entre a metodologia do especialista e o processo evolutivo encontra-se ilustrado na Figura 6.

De um lado, nota-se que o especialista, baseando-se em um conjunto de componentes e regras, utiliza um algoritmo determinístico para a obtenção dos projetos, que estão representados na figura como uma sub-região do espaço geral de projetos. Em algumas situações, através da inspiração e criatividade dos especialistas, esse sub-conjunto é ampliado.

Quanto à metodologia evolutiva, assim como nos seres vivos, as modificações acontecem de forma *bottom-up*. A partir da interação entre o amplo espaço de projeto, as etapas Montagem/ Teste e o Algoritimo Evolutivo, possíveis soluções vão sendo criadas e testadas, explorando assim uma diversidade muito maior de possíveis soluções.

É importante ressaltar que no processo do especialista o conjunto de componentes utilizados muitas vezes é restrito, sendo que tais restrições não estão presentes quando se utiliza o processo evolutivo.

Como exemplo a Figura 7 ilustra antenas projetadas com algoritmos evolutivos no centro de pesquisas da NASA (HORNBY et al., 2015). Nota-se que essas antenas não possuem nenhuma simetria e estão fora do domínio dos especialistas. Vale destacar que tais antenas tiveram um desempenho melhor em comparação com as antenas tradicionais. Hornby (HORNBY et al., 2015) relata que ao longo do processo evolutivo, foi possível testar milhares de novos projetos, os quais não seriam passíveis de serem obtidos por projetistas.

2.2.1 **Hardware Evolutivo**

Quando algoritmos evolutivos e outros bio-inspirados são desenvolvidos para a concepção de circuitos, tal aplicação é denominada hardware evolutivo. De acordo com Gordon e Bentley (ZOMAYA, 2006) o hardware evolutivo origina-se da interseção da ciência da computação, engenharia e biologia.

Conforme a ferramenta utilizada para a evolução dos circuitos, a metodologia pode ser classificada como intrínseca ou extrínseca. O trabalho de DeGaris é referenciado

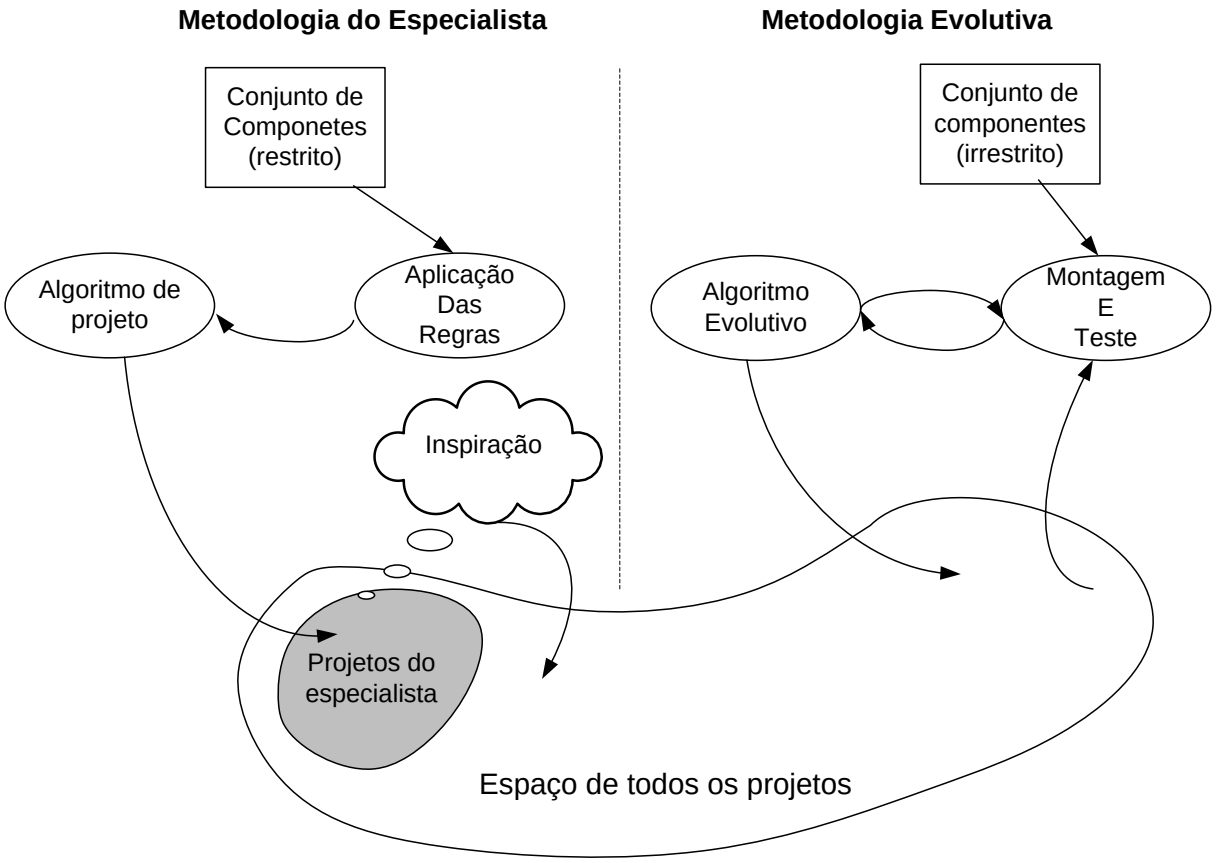
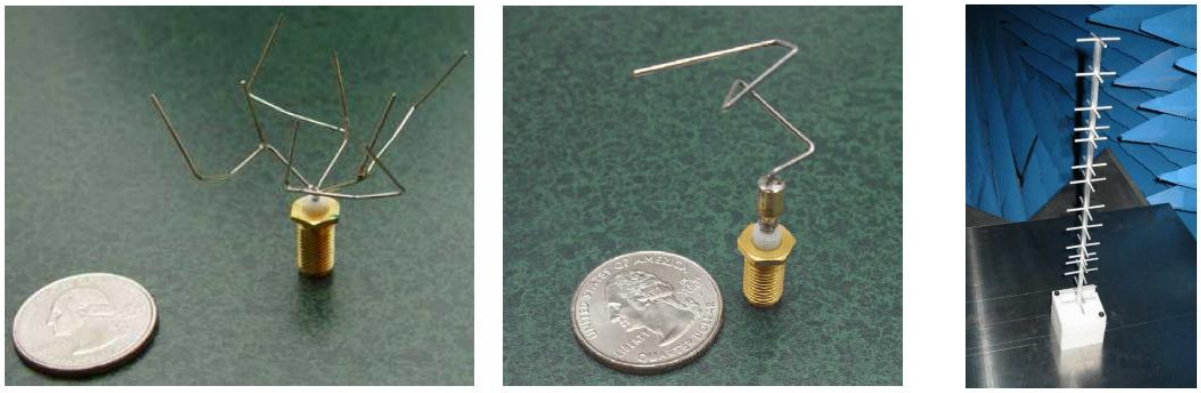


Figura 6 – Metodologia tradicional x Metodologia Evolutiva, esquema adaptado de (MILLER; JOB; VASSILEV, 2000)



a) b) c)

Figura 7 – Antenas projetas com algoritmos evolutivos para missões espaciais da NASA (HORNBY et al., 2015)

na literatura como responsável por introduzir esses conceitos (AGUIRRE; BUCKLES; COELLO, 2001; ZEBULUM, 1999; REN et al., 2011).

2.2.1.1 *Evolução Extrínseca*

Todo o processo evolucionário é realizado em software e, ao final do processo, obtém-se o circuito otimizado, que poderá ser implementado em hardware. Dessa forma, todas as soluções candidatas, ao longo das gerações, são avaliadas por simulação e apenas o resultado final é decodificado e implementado em hardware.

2.2.1.2 *Evolução Intrínseca*

O processo evolucionário é feito via hardware e cada solução candidata (circuito) é avaliada diretamente através da resposta obtida pela aplicação de sinais de entrada. Para que o processo seja implementado é necessário uma plataforma reconfigurável, utilizando-se, usualmente, os *chips* FPGAs (*Field Programmable Gate Array*).

Os FPGAs são dispositivos classificados como Dispositivos Lógicos Programáveis (PLD). Um FPGA possui internamente vários blocos lógicos, também conhecidos como células lógicas, que podem ser conectados ou desconectados, de acordo com o circuito desejado. Essa configuração das conexões dos blocos lógicos é considerada a programação do FPGA.

Na implementação de Miller (MILLER; THOMSON; FOGARTY, 1997), os cromossomos representam a funcionalidade digital de cada célula e as suas conexões. Luo (REN et al., 2011) fornece as diretrizes de um projeto para a evolução intrínseca. Por ser implementado em hardware, a evolução intrínseca, além de levar em consideração todas as características elétricas do circuito, é muito mais rápida que a evolução extrínseca.

No trabalho de Miller (MILLER; THOMSON; FOGARTY, 1997), em que foram projetados circuitos aritméticos (somadores e multiplicadores) melhores do que as topologias convencionais, é destacada uma desvantagem da evolução intrínseca: como o processo é totalmente livre de restrições, e qualquer configuração interna é possível, as propriedades físicas (não-digitais) do dispositivo FPGA influenciam no resultado, isso pode impedir que o resultado seja reproduzido em outro FPGA.

2.3 PROGRAMAÇÃO GENÉTICA CARTESIANA - CGP

A CGP pode ser definida como uma técnica de programação genética em que os programas são modelados como grafos acíclicos (DAG) direcionados, cujos nós são organizados em uma matriz (MILLER; SMITH, 2006; MILLER, 2011). Sua utilização conduziu um desenvolvimento significativo em projetos evolutivos de uma forma geral (VASICEK; SEKANINA, 2016).

Na CGP diversas estruturas computacionais podem ser facilmente representadas (MILLER, 2011) e apesar de ter sido inicialmente desenvolvida para representar circuitos digitais, problemas de outros domínios tem sido abordados, tais como: diagnóstico de doenças (AHMAD et al., 2012; JAN; KHURSHID; KHATTAK, 2015), robótica (HARDING; MILLER, 2005), filtros digitais (MILLER, 1999b), representação de redes neurais (KHAN; MILLER; HALLIDAY, 2011) e processamento de imagem (MILLER; SMITH, 2006; MILLER, 2011; MILLER; TURNER, 2015).

A característica fundamental que difere a CGP da programação genética tradicional, que adota uma representação em árvore, é a sua capacidade de explorar o material genético não codificante análogo ao *junk dna*. Na CGP uma grande parcela do genótipo não é mapeada para o fenótipo, mas durante o processo evolutivo os genes não codificantes podem se tornar ativos e contribuir na semântica. Isso faz com que novas soluções sejam exploradas e torna a CGP uma ferramenta poderosa para escapar de ótimos locais (TURNER; MILLER, 2015a).

Além disso, quando comparada com a PG (Programação Genética) tradicional, a CGP apresenta ainda mais duas vantagens a serem destacadas: a representação adequada para problemas com múltiplas saídas e o fato da CGP não apresentar o *bloat*, fenômeno associado ao crescimento desordenado do programa, em que as soluções se tornam cada vez maiores sem melhora substancial na qualidade (GOLDMAN; PUNCH, 2013).

2.3.1 **CGP aplicada ao Hardware Evolutivo**

A CGP (*Cartesian Genetic Programming*) tem sido apontada como o método mais eficiente para evoluir circuitos digitais (VASICEK; SEKANINA, 2016) sendo aplicada em diferentes vertentes, as quais serão descritas a seguir.

2.3.1.1 *Projeto Evolucionário*

Essa abordagem tem como objetivo encontrar soluções factíveis para um dado problema. No caso de CLCs esse objetivo é alcançado quando um circuito que atenda a todas as restrições impostas pela tabela verdade é encontrado. Entretanto, o problema mais sério em hardware evolutivo, conhecido como problema da escalabilidade (MILLER, 2011; VASICEK; SEKANINA, 2014b; VASICEK, 2015), tem restringido a aplicabilidade do projeto evolucionário.

O problema da escalabilidade está relacionado ao fato de o tempo de avaliação de uma solução candidata crescer exponencialmente com o aumento das variáveis de entradas. Essa limitação tem motivado o desenvolvimento de estratégias com a finalidade de aumentar a eficiência da CGP (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015; MANFRINI; BERNARDINO; BARBOSA, 2016b; MANFRINI; BERNARDINO; BARBOSA, 2016a).

Brian Goldman e William Punch (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015) testaram vários mecanismos de mutação e dentre as possibilidades apresentadas a mutação denominada *Single* mostrou bons resultados, evitando o desperdício de avaliações e aumentando a eficiência da busca. No site da CGP (MILLER, 2017 (accessed February 14, 2017)) Miller destaca que o trabalho desenvolvido em (GOLDMAN; PUNCH, 2015) é muito interessante. Vale destacar que uma das contribuições dessa tese foi a criação de um operador de mutação eficiente (MANFRINI; BERNARDINO; BARBOSA, 2016a), derivado do operador *Single*.

2.3.1.2 *Otimização*

Vasicek e Sekanina (VASICEK; SEKANINA, 2012; VASICEK; SEKANINA, 2014b; VASICEK, 2015) desenvolveram um interessante método de otimização de circuitos, sendo que o trabalho (VASICEK, 2015) além de ser citado por Miller (MILLER, 2017 (accessed February 14, 2017)) como uma excelente contribuição, foi classificado como o melhor artigo da EUROGP 2015.

Ao contrário do projeto evolutivo, a otimização evolutiva começa com uma população semeada por um circuito totalmente funcional. A característica mais importante dessa abordagem é que cada solução candidata criada por meio dos operadores genéticos deve ser funcionalmente equivalente ao progenitor. O método leva em consideração que o progenitor e sua prole compartilham muitos sub-circuitos. Com isso, para verificar se uma solução candidata é factível, basta analisar se os sub-circuitos não compartilhados são funcionalmente equivalentes.

Os experimentos mostram que o método proposto por Vasicek e Sekanina reduz significativamente o número de portas lógicas em circuitos projetados usando as técnicas tradicionais.

2.3.1.3 *Projeto aproximado*

A computação aproximada está associada à capacidade de alguns sistemas em tolerar a perda da qualidade em prol da diminuição da complexidade ou de algum outro benefício. Em sistemas computacionais móveis, por exemplo, aplicações que envolvam processamento de mídia (áudio, vídeo e imagem), um resultado perfeito não é necessário, uma solução inferior ao ótimo pode ser suficiente (HAN; ORSHANSKY, 2013; VASICEK; SEKANINA, 2015).

Nos sistemas digitais a computação aproximada permite um relaxamento funcional, tornando os sistemas mais eficientes em termos de consumo energético, velocidade ou complexidade. Como resultado tem-se uma gama de *trade-off* entre erro e eficiência (HRBACEK; MRAZEK; VASICEK, 2016). Dessa forma, a computação aproximada

Tabela 1 – Áreas das portas lógicas em relação a área de uma porta NAND

Porta	Área
AND	1.333
OR	1.333
XOR	2
NAND	1
NOR	1
XOR	3
NOT	0.667

tornou-se uma abordagem promissora para uma concepção de sistemas digitais eficientes (HAN; ORSHANSKY, 2013; VASICEK; SEKANINA, 2015).

O projeto de circuitos aproximados tem utilizado o consumo energético como o principal objetivo a ser minimizado, contudo esse consumo está diretamente correlacionado à área do circuito, sendo assumido que a otimização da área é equivalente à otimização energética (VASICEK; SEKANINA, 2015; VASICEK; SEKANINA, 2016; HRBACEK; MRAZEK; VASICEK, 2016). Por sua vez, a área de um circuito candidato é calculada em relação à área de uma porta NAND. Para o conjunto Γ = {AND, OR, XOR, NAND, NOR , XNOR, NOT} as correspondentes áreas são mostradas na Tabela 1.

Outro objetivo a ser minimizado é o *delay*, ou retardo de propagação, que está associado ao atraso na mudança de nível das portas lógicas ($0 \rightarrow 1$ ou $1 \rightarrow 0$). O retardo de propagação determina o tempo de resposta do circuito, ou seja, o tempo necessário para que a saída do circuito seja atualizada após uma mudança nos valores das entradas. Para o cálculo desse *delay* é levado em consideração o atraso através do caminho mais longo entre a entrada e saída do circuito (ERCEGOVAC; MORENO; LANG, 1998).

3 FUNDAMENTOS DE CIRCUITOS LÓGICOS COMBINACIONAIS

3.1 SISTEMAS DIGITAIS

Um sistema digital se diferencia dos sistemas analógicos no tipo de sinal que processam. Enquanto um sistema analógico trabalha com sinais contínuos, os digitais têm um número finito de valores discretos. A maioria das grandezas físicas, como por exemplo a temperatura, a velocidade, entre outras, são analógicas e para que sejam processadas por circuitos digitais é necessário digitalizar as informações[1]. A Figura 8 mostra a digitalização de um sinal analógico.

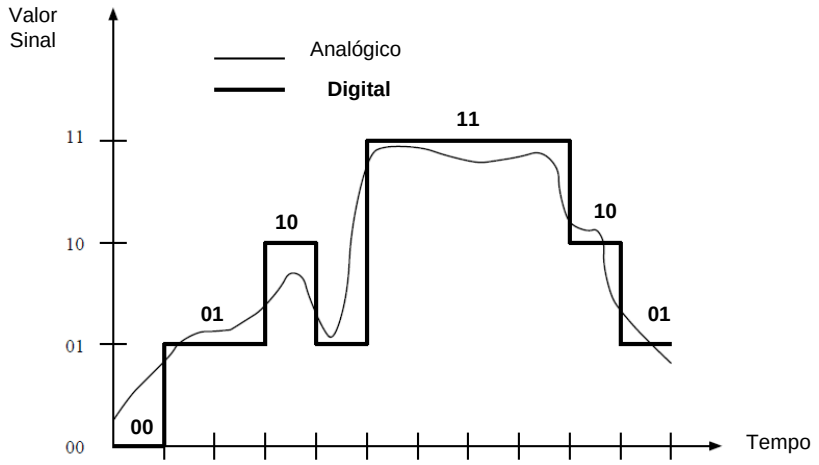


Figura 8 – Sinal analógico X digital

Existem vários benefícios decorrentes da utilização de sistemas digitais em comparação com os analógicos, dentre eles (TOCCI, 2011):

- Facilidade de fabricação de circuitos integrados. Os circuitos analógicos possuem componentes que não podem ser integrados do ponto de vista econômico, como capacitores de alto valor e indutores.
- Facilidade de projeto. Os valores exatos de tensão e corrente não são relevantes, apenas a faixa na qual eles se encontram.
- Operação programável. Circuitos digitais podem ser projetados através de linguagens de descrição de hardware.
- Maior imunidade a ruídos. Variações espúrias na tensão são toleradas dentro de uma determinada faixa.

[1] Existem circuitos específicos para essa transformação, denominados Conversor Digital Analógico (CDA).

Os circuitos que compõem os sistemas digitais podem ser classificados de acordo com as suas características em 2 tipos:

- **Circuitos Lógicos Combinacionais (CLCs):** Possuem como característica principal o fato da saída atual, $z(t)$ depender exclusivamente da entrada atual $x(t)$, ou seja:

$$z(t) = F(x(t))$$

  Ressalta-se que nesse trabalho será abordado essa classe de circuitos digitais.

- **Circuitos Lógicos Sequenciais**: A saída atual $z(t)$ depende da entrada atual e das entradas anteriores $x(0,t)$, ou seja:

$$z(t) = F(x(0,t))$$

Para representar a informação nos sistemas digitais é utilizado o sistema binário, o que faz com que o circuito precise de apenas dois níveis de tensão elétrica[2]: $0V$ ($0\ Volts$) e $5V$.

3.2 DESENVOLVIMENTO DE SISTEMAS DIGITAIS

O desenvolvimento de um sistema digital começa pela sua especificação, a qual fornece a descrição e os detalhes do seu funcionamento. No nível físico, a implementação é feita através de circuitos formados basicamente por transistores. Devido à grande quantidade de transistores para se projetar um sistema digital, é preciso definir diferentes níveis de abstrações, os quais são formados por módulos de complexidade crescente, que podem variar desde portas lógicas a processadores complexos (ERCEGOVAC; MORENO; LANG, 1998).

A implementação de um projeto pode seguir dois tipos de hierarquia, *Top-down* descendente e *Bottom-up* ascendente, conforme representado na Figura 9. O nível de abstração mais elevado pode ser considerado como os próprios circuitos integrados. A descrição do projeto começa no nível de sistema; e na medida em que se caminha para hierarquias inferiores, informações mais detalhadas ficam mais evidentes.

Na metodologia descendente, o sistema é decomposto diversas vezes em subsistemas mais simples até um determinado nível, onde módulos previamente disponíveis permitam a realização do projeto. Na abordagem ascendente, subsistemas do nível inferior são conectados para formarem subsistemas mais elaborados, até que a especificação funcional do sistema seja atingida. Nas duas abordagens, o resultado final depende da experiência do projetista e não existe um procedimento sistemático que possa garantir a otimalidade do projeto (ERCEGOVAC; MORENO; LANG, 1998).

[2] Dependendo da tecnologia de fabricação, outros valores de tensão estão relacionados ao nível lógico 1.

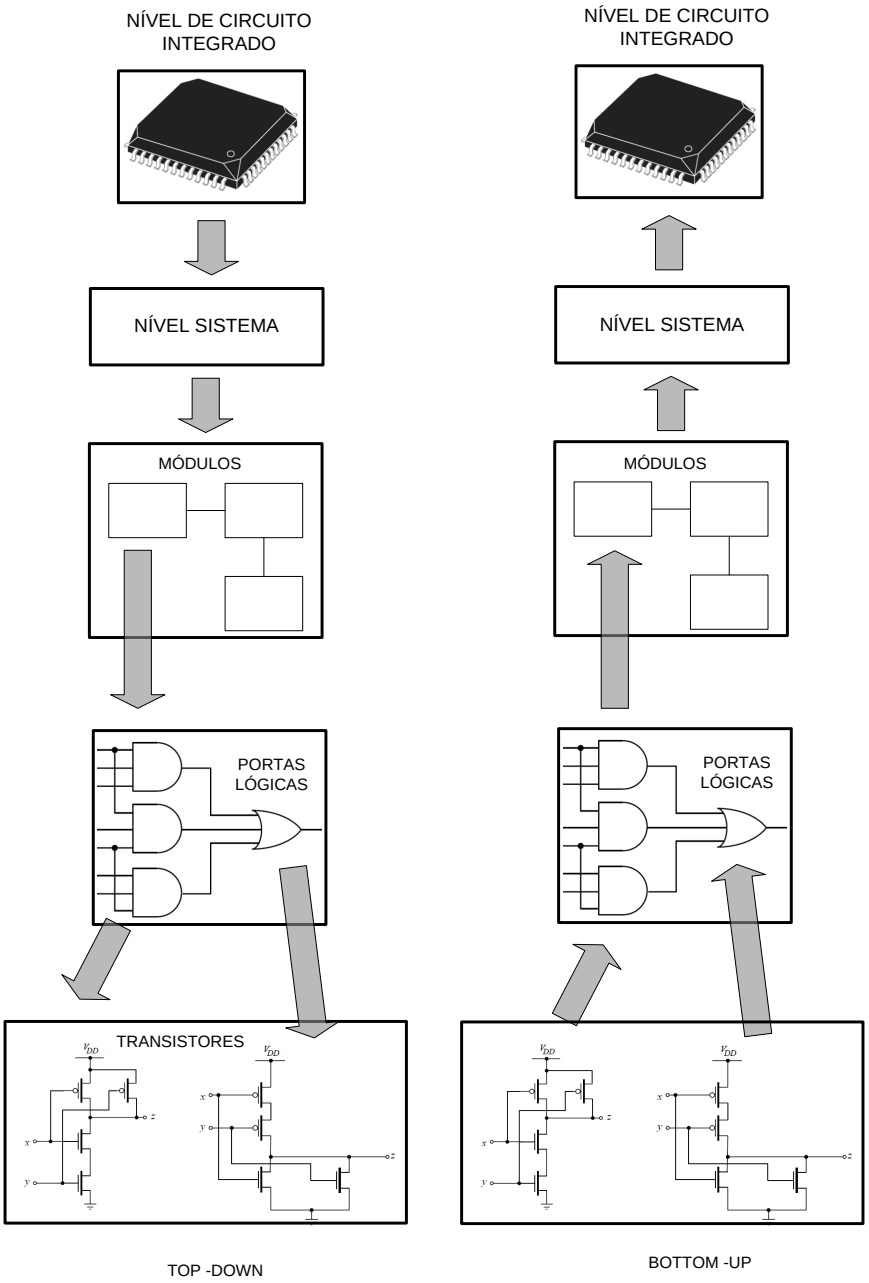


Figura 9 – Implementação hierárquica

3.3 DESCRIÇÃO DE CIRCUITOS LÓGICOS

Os sistemas digitais, desde o mais simples ao mais complexo, são compostos pelos elementos denominados portas lógicas. As portas lógicas são os blocos construtivos de um circuito digital e a partir delas podem ser implementados circuitos digitais complexos (VAHID; WILEY, 2006).

Uma forma utilizada para descrever a funcionalidade de um circuito lógico é através de uma tabela, denominada tabela verdade, que expressa todas as combinações das variáveis de entrada e as respectivas saídas do circuito. A figura 10 mostra um exemplo

de uma tabela verdade com 3 variáveis de entrada, denotadas por A, B e C, e uma saída, representada por F. Ressalta-se que para se projetar um circuito, cada linha da tabela verdade pode ser considerada uma restrição que deverá ser atendida.

	A B C	F
0	000	0
1	001	0
2	010	1
3	011	0
4	100	0
5	101	1
6	110	1
7	111	1

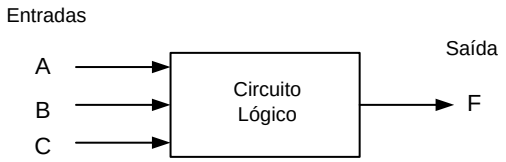


Figura 10 – Exemplo de uma tabela verdade, que descreve a relação *entrada/saída* de um circuito lógico.

Uma forma compacta de expressar as informações presentes na tabela verdade é através de mintermos, De forma simples, os mintermos estão relacionados as saídas "1" da tabela verdade, os detalhes dessa associação e a definição formal de mintermos estão no Apêndice A.0.1. No exemplo apresentado na Figura 10 os mintermos estão associados às linhas $2, 5, 6$ e 7 e podem ser representados pela função: $F(A, B, C) = \sum m(2, 5, 6, 7)$. Por simplicidade, essa notação utilizando a função dos mintermos será utilizada nessa tese para expressar as informações contidas nas tabelas verdade.

Outro modo para analisar o comportamento de um circuito lógico é através da álgebra booleana (BOOLE, 1854). Em uma equação booleana, as variáveis podem assumir os valores 0 ou 1. Num circuito lógico, as variáveis booleanas não representam efetivamente números mas o estado do nível de tensão, sendo 5V para "Alto" e 0V para "Baixo". As operações na álgebra booleana podem ser AND, OR, NOT, XOR e XNOR entre outras operações lógicas.

As funções binárias básicas da álgebra booleana são descritas a seguir:

- **NOT:** O valor de saída é o valor complementar da variável de entrada A, ou seja: $F(0) = 1$ e $F(1) = 0$. Algebricamente denotada por: $F = \overline{A}$.
- **AND:** A função $F(X)$ possui o valor binário 1 se e somente se todas as variáveis de entrada possuem o valor 1, e 0 caso contrário. Algebricamente denotada por: $F = A \cdot B$.

  Vale ressaltar que na álgebra boolena o operador $\cdot$ têm um significado diferente da álgebra convencional.
- **NAND:** É o inverso da AND, algebricamente denotada por: $F = \overline{(A \cdot B)}$.

- **OR:** A função $F(X)$ possui o valor binário 1 se pelo menos uma das entradas for 1, e 0 caso contrário. Algebricamente denotada por: $F = A + B$.
- **NOR:** É o inverso da OR, algebricamente denotada por: $F = \overline{(A + B)}$.
- **XOR:** A função $F(X)$ possui o valor binário 1 se as entradas forem diferentes, ou seja: $A = 0$ e $B = 1$ ou $A = 1$ e $B = 0$. Algebricamente denotada por: $F = A \oplus B$.
- **XNOR:** A função $F(X)$ possui o valor binário 1 se as entradas forem iguais, ou seja: $A = 1$ e $B = 1$ ou $A = 0$ e $B = 0$. Algebricamente denotada por: $F = A \odot B$.

Cada função booleana tem associada a ela uma porta lógica e uma tabela verdade. A Figura 11 mostra essa associação para as portas NOT, OR, NOR, AND, NAND, XOR e XNOR.

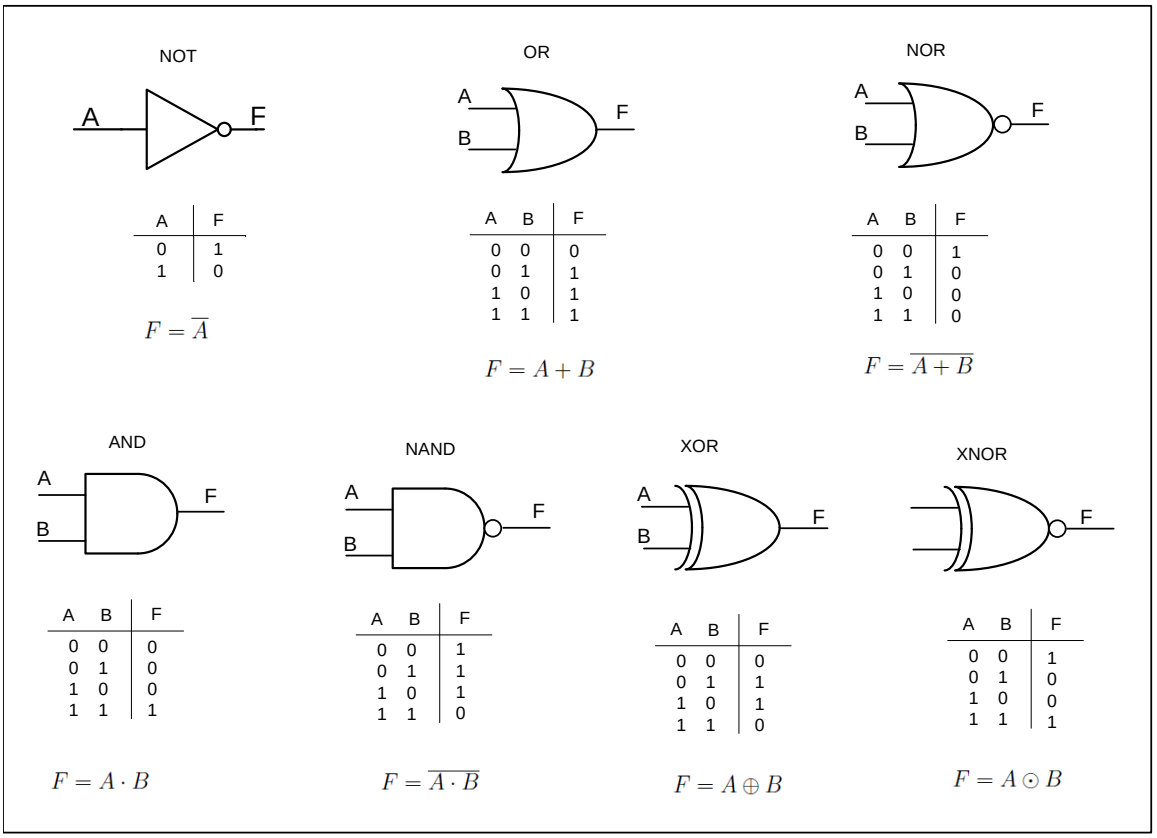


Figura 11 – Símbolos e tabela verdade das portas lógicas

Um circuito lógico é formado pela conexão de diversas portas lógicas, sendo que é possível projetar qualquer circuito lógico utilizando apenas as portas NOT, AND e OR. Esse tipo de implementação pode ser de duas formas: Soma de Produtos (SOP) e Produto de Somas (POS), os detalhes dessas metodologias de implementação estão descritas no Apêndice A.

3.4 ALGORITMOS PARA PROJETAR CIRCUITOS LÓGICOS COMBINACIONAIS

A álgebra booleana, assim como as definições e conceitos envolvidos na simplificação lógica através do mapa de Karnaugh, formam a base para o projeto de circuitos lógicos combinacionais (UYEMURA, 1999). Entretanto, a simplificação booleana através do mapa de Karnaugh é uma ferramenta limitada a sistemas de 5 ou 6 variáveis, sendo que sistemas maiores exigem uma abordagem computacional (TOCCI, 2011).

O método computacional mais conhecido (VAHID; WILEY, 2006) é o algoritmo de Quine-McCluskey (MCCLUSKEY, 1956; MCCLUSKEY, 1961), chamado também de método tabular. Esse método divide a busca pelos agrupamentos em diversos níveis, de acordo com o número de variáveis. Inicialmente, identificam-se as duplas, em seguida as quadras, os octetos e assim, sucessivamente, os agrupamentos vão sendo encontrados. A fase final do algoritmo tem como objetivo verificar quais são os agrupamentos essenciais.

Uma limitação do método de Quine-McCluskey é o custo computacional, que aumenta exponencialmente com o número de variáveis (PEDRONI, 2008; VAHID; WILEY, 2006). Uma função com n variáveis pode ter até 2^n mintermos, fazendo com que a enumeração de todos os agrupamentos e a identificação dos essenciais fique inviável para uma função com muitas variáveis (VAHID; WILEY, 2006). Uma função de 32 variáveis, por exemplo, pode ter até 2^{32} mintermos, ou seja, até cerca de 4 bilhões de mintermos.

Diante da complexidade na geração de todos os agrupamentos essenciais, surgiram métodos heurísticos capazes de resolver esses passos com um menor custo computacional. Essas ferramentas de otimização lógica, ao invés de enumerar todos os mintermos, buscam encontrar simplificações para a função através de um processo iterativo, que realiza pequenas modificações na função a cada iteração (VAHID; WILEY, 2006).

O método heurístico denominado Espresso (BRAYTON et al., 1984), desenvolvido na Universidde da Califórnia em Berkeley, é um dos mais conhecidos (REIS et al., 2000), sendo apontado pela literatura coma a base das ferramentas comerciais de otimização lógica (VAHID; WILEY, 2006).

4 COMPUTAÇÃO EVOLUCIONISTA

O paradigma da evolução natural serviu de inspiração para Holland e seus alunos desenvolverem, em meados de 1960 na Universidade de Michigan, os Algoritmos Genéticos (AGs). Os propósitos dos estudos de Holland foram analisar os mecanismos de adaptação dos organismos biológicos e implementar um modelo computacional com os mesmos princípios básicos (CUNHA; TAKAHASHI; ANTUNES, 2012). De acordo com a ideia principal aplicada nos algoritmos evolutivos, dada uma população de indivíduos, a pressão ambiental provoca a seleção natural (sobrevivência dos mais aptos), e isso aumenta a aptidão da população (EIBEN; SMITH, 2003).

4.1 CONCEITOS DE ALGORITMOS GENÉTICOS

A utilização dos conceitos da teoria da evolução no modelo computacional ocorre de maneira simplificada. Inicialmente é gerada uma população de possíveis soluções para o problema (indivíduos). Através de um processo iterativo, busca-se evoluir tais soluções aplicando operadores genéticos (seleção, cruzamento e mutação) (EIBEN; SMITH, 2003) e dessa forma é gerada uma nova população, conforme ilustra a Figura 12.

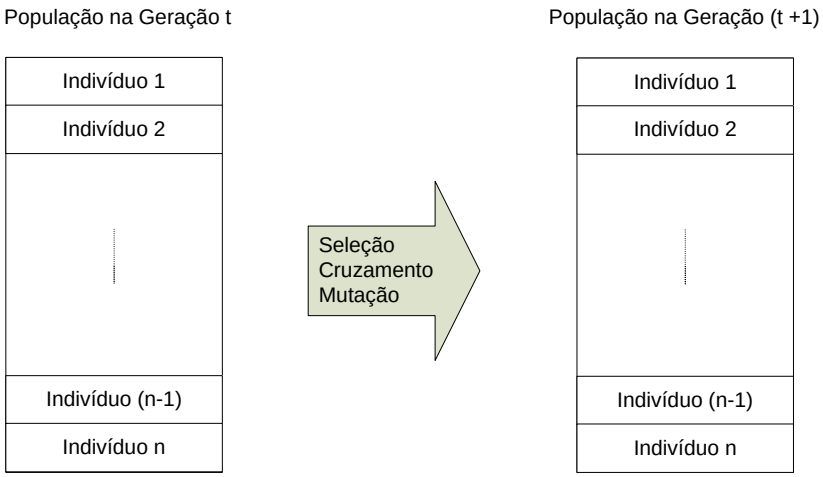


Figura 12 – Esquema da reposição populacional

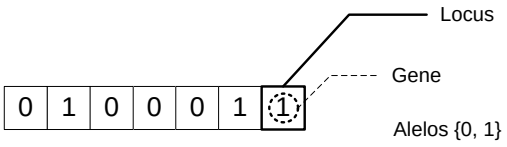
4.1.1 **Representação**

As soluções candidatas de um determinado problema, também chamadas de indivíduos, são codificadas através de um estrutura de dados denominada cromossomo, sendo o espaço dos cromossomos também denominado de espaço dos genótipos.

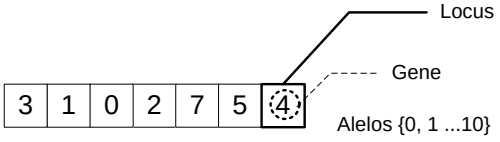
Originalmente Holland utilizou uma codificação binária para representar as possíveis soluções, dando origem a uma sequência de $'0'$ e $'1'$, conforme Figura 13(a). Os

constituintes básicos dos cromossomos são denominados genes, os quais podem ser interpretados como uma das características elementares da solução. A localização dos genes é denominada locus, e os valores possíveis que os genes podem assumir são os alelos.

Existem várias formas de representar a solução de um problema, a melhor delas depende do problema em questão e do espaço de busca. A Figura 13(b) mostra um exemplo de um cromossomo formado por números inteiros, onde os genes podem assumir valores de 0 a 10 (CUNHA; TAKAHASHI; ANTUNES, 2012).



(a) Cromossomo com representação binária



(b) Cromossomo com representação com inteiros

Figura 13 – Exemplos de representação de cromossomos

4.1.2 **Função Objetivo**

Um problema de otimização tem como objetivo maximizar ou minimizar uma função objetivo e essa função pode ser utilizada para atribuir uma aptidão para cada indivíduo. A qualidade das possíveis soluções é estabelecida de forma quantitativa, possibilitando a comparação entre as diversas soluções.

A definição da função objetivo nem sempre é uma tarefa fácil, principalmente quando se tem vários objetivos conflitantes para serem atendidos.

4.1.3 **Seleção**

No contexto da otimização, a seleção pode ser interpretada como sendo o mecanismo que direciona a busca pela solução ótima, ou seja, determina qual região do espaço de busca será explorada. No AG, o processo que conduz a criação de uma nova população tem início com a seleção dos progenitores. Essa escolha leva em consideração a aptidão dos indivíduos, sendo que aqueles de melhor qualidade possuem maior probabilidade de

serem selecionados (EIBEN; SMITH, 2003). Um conceito importante associado ao processo de seleção é a "Pressão de Seleção", que pode ser entendida como uma relação entre o mérito relativo de um indivíduo e a sua probabilidade de ser selecionado como progenitor. Uma escolha aleatória dos indivíduos, ou uma baixa pressão de seleção, faz com que o AG perca a capacidade de identificar regiões promissoras do espaço de busca.

Por outro lado, situações em que há uma pressão de seleção elevada, onde apenas um pequeno grupo de indivíduos é selecionado, podem levar a uma perda da diversidade da população, ocasionando uma convergência prematura do AG.

Diversos métodos de seleção têm sido apresentados na literatura (CUNHA; TAKAHASHI; ANTUNES, 2012). No AG canônico desenvolvido por Holland foi apresentado um esquema de seleção denominado roleta, mas atualmente o mais utilizado é o método denominado torneio (EIBEN; SMITH, 2003). A seguir são descritos o funcionamento e as características desses métodos.

4.1.3.1 *Seleção por Roleta*

A probabilidade de um indivíduo ser selecionado baseia-se na razão entre a sua qualidade e a soma das qualidades de todos os indivíduos da população, ou seja:

$$Psel = \frac{f}{\sum_{i=1}^{n_{pop}} f(i)}$$

onde;

P_{sel} probabilidade de seleção de um determinado indivíduo;

f aptidão (*fitness*) do indivíduo que se deseja determinar P_{sel};

f_i aptidão do indivíduo i;

n_{pop} número de indivíduos da população;

Na seleção por roleta, indivíduos com aptidão muito maior que os outros acabam sendo selecionados com uma frequência muito alta, o que ocasiona uma convergência prematura. Por outro lado, quando as aptidões são muito próximas, a distribuição de probabilidades se torna uniforme, gerando um processo quase aleatório (CUSTÓDIO, 2008).

4.1.3.2 *Seleção por Torneio*

Nesse método, um número k de indivíduos são escolhidos aleatoriamente e aquele que tiver maior aptidão é selecionado para a reprodução. O parâmetro k define o tamanho

do torneio e controla a pressão de seleção. No torneio, a pressão de seleção não está diretamente ligada ao valor absoluto da aptidão dos indivíduos; a informação necessária para efetuar a seleção é uma ordenaçao dos indivíduos, baseada na aptidão(MITCHELL, 1996).

4.1.4 **Cruzamento ou Recombinação**

O cruzamento é uma operação genética em que são obtidos os descendentes através da troca (ou recombinação) do material genético de cromossomos progenitores. Na prática, escolhe-se um ou mais pontos de corte e os descendentes são gerados a partir das sequências complementares dos progenitores. A Figura 14(a) mostra uma recombinação com um ponto de corte, que pode ser generalizada para n pontos de corte, conforme a Figura 14(b).

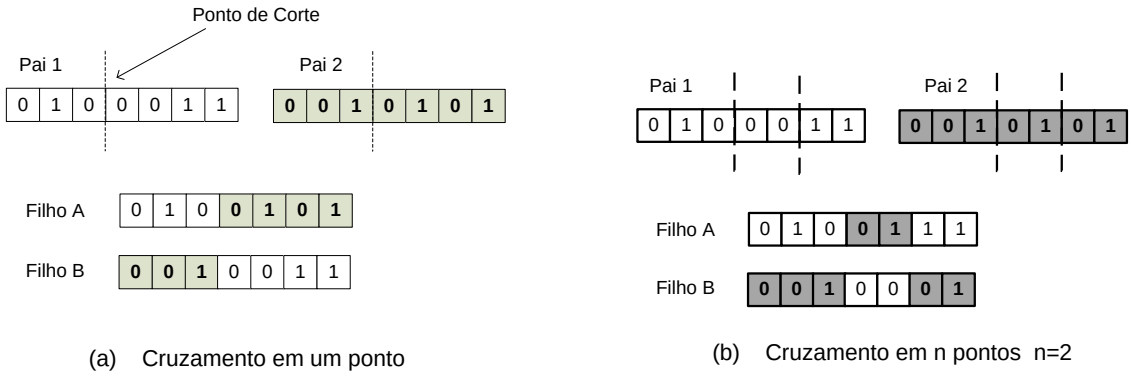


Figura 14 – Cruzamento

4.1.5 **Mutação**

O operador de mutação pode ser considerado o mais importante para o funcionamento do AG, pois ele é responsável pela diversidade do material genético (EIBEN; SMITH, 2003; HARVEY, 1993). A aplicação desse operador em um gene faz com que o mesmo troque de valor, como exemplificado na Figura 15. A alteração aleatória de um ou mais genes do cromossomo evita que a solução se prenda a ótimos locais e aumenta a probabilidade de se chegar a pontos do espaço de busca que ainda não foram explorados (EIBEN; SMITH, 2003; CUNHA; TAKAHASHI; ANTUNES, 2012). A mutação pode ser aplicada com uma probabilidade p_a em cada gene do cromossomo, sendo possível diminuir esse parâmetro ao longo das gerações com o intuito de fazer um ajuste fino na solução.

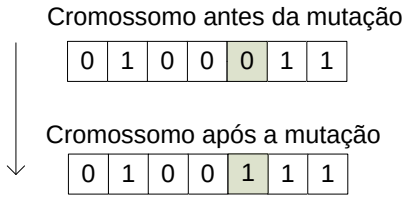


Figura 15 – Exemplo do operador de mutação

4.1.6 **Elitismo**

Ao longo das gerações a aptidão média da população aumenta, sendo que esse aumento é mais evidente no início da otimização e tende a dimimuir ou estagnar quando a população começa a convergir. Esse comportamento não acontece quando a aptidão do melhor indivíduo é analisada ao longo das gerações, uma vez que devido ao fato do processo ser probabilístico, não é raro a qualidade do melhor indivíduo diminuir de uma geração para a outra.

O operador de elitismo não permite que o melhor indivíduo piore de uma geração para a outra. A sua aplicação é feita da seguinte forma: ao término da avaliação dos descendentes que irão compor a nova população, verifica-se se o melhor filho tem aptidão maior do que o melhor pai, em caso negativo, o melhor indivíduo entre os progenitores é mantido na nova população. Para que o tamanho da população continue fixo, é necessário sacrificar um indivíduo; geralmente elimina-se um indivíduo aleatoriamente ou o que tem menor aptidão (CUNHA; TAKAHASHI; ANTUNES, 2012). Também é possível aplicar o elitismo mantendo os k melhores progenitores e eliminando k descendentes.

4.1.7 **Critério de Parada**

Por ser um algoritmo estocástico não é possível identificar se a solução encontrada por um AG é um ótimo global, por isso os operadores genéticos continuam atuando ao longo das gerações em busca da solução ótima. Portanto, faz-se necessário definir um ou mais critérios de parada, sendo que os critérios usuais são:

- **Número máximo de gerações:** Nesse esquema, quando se atinge um número predefinido de gerações o processo termina.
- **Aptidão do melhor indivíduo:** É feita a comparação da aptidão do melhor indivíduo ao longo das gerações; caso o valor se repita por n vezes consecutivas, o algoritmo é finalizado.
- **Tempo de execução:** O processo é interrompido ao ser atingindo um tempo máximo para execução.

4.1.8 **Algoritmo Genético Canônico**

No AG originalmente desenvolvido por Holland (HOLLAND, 1975), designado de SGA ou AG canônico, após o processo reprodutivo os descendentes substituem todos os indivíduos da população antiga, dando origem a uma nova população. A Figura 16 mostra o esquema do AG geracional.

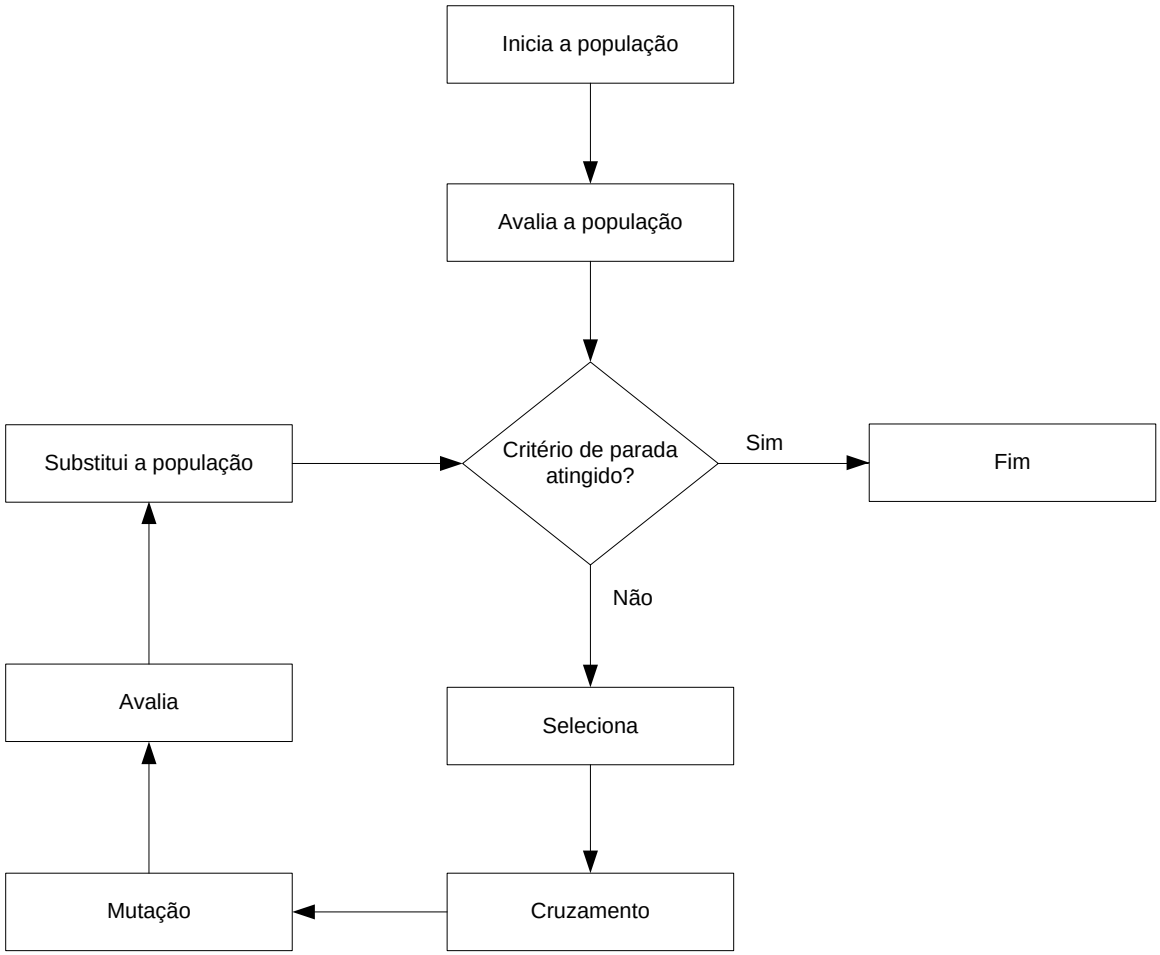


Figura 16 – Fluxograma do Algoritmo Genético

A Tabela 2 mostra as características principais do AG desenvolvido originalmente por Holland (EIBEN; SMITH, 2003), habitualmente chamado de SGA (*simple GA*) ou *canonical GA*.

Tabela 2 – Características do SGA

Representação	vetor binário
Recombinação	Cruzamento em 1 ponto
Mutação	troca de bit
Seleção	Roleta
modelo	geracional

4.1.9 **Algoritmo Genético *STEADY-STATE***

Em muitos problemas a avaliação dos indivíduos exige um grande esforço computacional. Uma tentativa de reduzir o custo computacional é através do *AG steady-state* proposto por De Jong (DEJONG, 1975).

Nesse modelo um parâmetro G estabelece a fração da população que será substituída, e pode variar da seguinte forma:

$$\frac{1}{P} \leq G \leq 1$$

onde P é o tamanho da população.

Portanto, a variação do parâmetro G ocorre desde $G = 1/P$, que representa a reprodução de apenas um indivíduo a cada geração, até $G = 1$, onde toda a população é substituída, o que equivale ao modelo geracional. Nota-se que no AG *steady-state* os progenitores e descendentes passam a coexistir e competem/cooperam entre si (WHITLEY et al., 1989).

4.2 PROGRAMAÇÃO GENÉTICA - PG

A programação genética (PG) é uma abordagem da computação evolucionista desenvolvida por Koza (KOZA, 1992). A PG é capaz de evoluir programas computacionais, sendo possível otimizar estruturas funcionais que realizam operações lógicas, condicionais e de desvio, entre outras. A capacidade de produzir soluções simbólicas é apontada como umas das principais características da PG (CUNHA; TAKAHASHI; ANTUNES, 2012).

4.2.1 **Representação canônica da Programação Genética**

A PG canônica utiliza uma representação em árvore em que os nós podem pertencer ao conjunto de funções ou terminais. Tais conjuntos determinam as ferramentas que serão utilizadas no processo evolutivo, as quais são definidas de acordo com o domínio do problema (CUNHA; TAKAHASHI; ANTUNES, 2012).

Os nós terminais situam-se nas folhas da árvore, conforme ilustra a Figura 17, sendo os responsáveis por fornecer as entradas do programa (variáveis e constantes). Os nós que fazem parte do conjunto de funções são os operadores, encarregados pelo processamento dos dados. Os referidos operadores recebem os parâmetros das entradas fornecidos pelos nós inferiores, processam os dados e retornam ao nó imediatamente superior. Como exemplo tem-se a árvore da Figura 17, cuja expressão gerada é: $x \cdot y + (x - 1)$.

4.2.1.1 *Exemplo de aplicação: Programa para determinar o fatorial de N*

Igwe e Pillary (IGWE; PILLAY, 2013) evoluíram vários programas para resolver problemas clássicos de programação. Como exemplo será mostrada a representação do

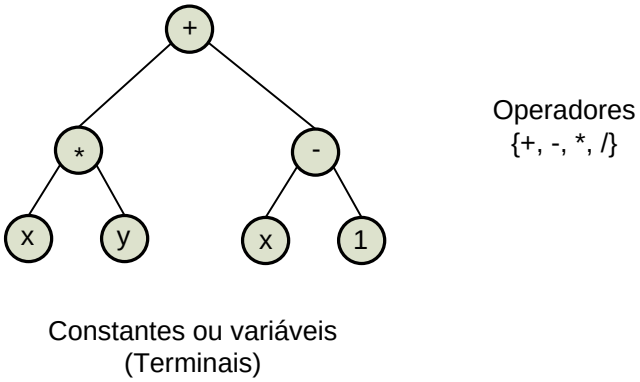


Figura 17 – Nós terminais e funcionais na representação canônica da PG

operador *if else* e do laço *for*, os quais foram utilizados na solução do programa evoluído, cujo objetivo é determinar $N!$.

A Figura 18(a) mostra o operador *if- else* construído com 3 sub-árvores (IGWE; PILLAY, 2013), onde a primeira sub-árvore representa a condição que deve ser testada. Caso seja verdadeira, a ação 1 é executada, caso contrário, executa-se a ação 2.

O laço *for* também pode ser construído com 3 sub-árvores (IGWE; PILLAY, 2013), conforme ilustra a Figura 18(b). A primeira sub-árvore representa o valor inicial do laço, a segunda sub-árvore o valor final e a terceira representa os comandos que serão executados a cada iteração. Duas variáveis são necessárias para o controle do *for*. Uma variável (a) de contagem que pode ser incrementada ou decrementada a cada iteração, sendo que tal variável é inicializada com o primeiro argumento do operador *for*. A outra variável (b) é responsável por armazenar o resultado de cada iteração, sendo inicializada com $b = 1$ ou $b = 0$ dependendo se o corpo do laço é uma multiplicação ou soma, respectivamente.

A Figura 19 mostra a árvore referente ao programa que calcula $N!$. O código fonte referente está descrito a seguir:

```
a = N
b = 1
for ((if (1 >= 1*N) 1 else N ) 1 b)
    b = a * b
    a = a - 1
end
```

A variável a é um contador decrescente, tem um valor de N na primeira iteração e o valor de 1 na última, por sua vez a variável b armazena o resultado da iteração anterior. Se $N = 1$ apenas uma iteração do laço *for* é executada, se $N > 1$ então N iterações serão realizadas.

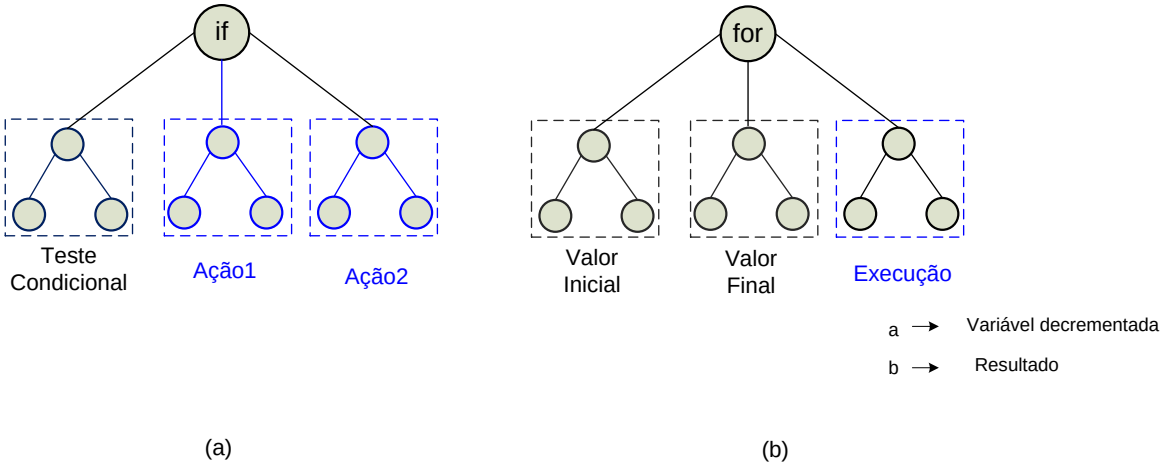


Figura 18 – Representação em árvore dos comandos *if-else* e *for*

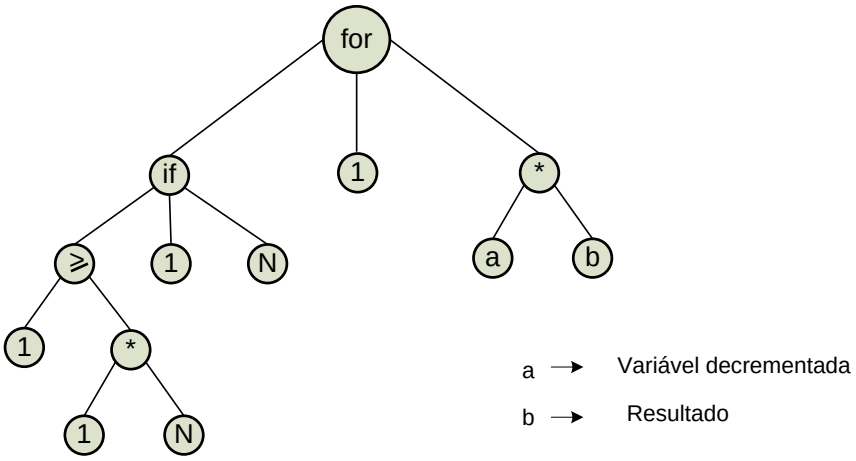


Figura 19 – Programa para cálculo do $N!$ obtido com a PG (IGWE; PILLAY, 2013).

4.3 PROGRAMAÇÃO GENÉTICA CARTESIANA

Em 1997 Miller et.al (MILLER; THOMSON; FOGARTY, 1997) desenvolveram um novo método para evoluir circuitos digitais, que posteriormente foi denominado *Cartesian Genetic Programming* (MILLER, 1999a). A designação *Cartesian* está associada à representação adotada, que utiliza uma matriz bi-dimensional de nós. Uma descrição ampla pode ser encontrada no livro publicado em 2011 por Miller (MILLER, 2011), o qual apresenta a CGP como uma forma geral de programação genética.

Na CGP os genes que constituem o cromossomo são números inteiros que representam as entradas dos nós e a respectiva operação que o nó executa. Apesar do genótipo ter um comprimento fixo, que está associado à dimensão da matriz, o tamanho do fenótipo pode variar desde zero nós até o número máximo de nós definido pelo genótipo. Um fenótipo teria zero nós se todas as saídas do programa fossem ligadas diretamente as entradas de dados.

A CGP tem três parâmetros escolhidos pelo usuário, número de colunas, número de linhas e *levels-back* denotados por n_c,n_r e lb, respectivamente. Os dois primeiros determinam a representação matricial e o parâmetro lb, controla a conectividade do grafo.

Levels-back fornece quantas colunas um determinado nó pode ser ligado, por exemplo se $lb = 1$ um determinado nó só pode ser ligado em um dos nós pertencentes a coluna imediatamente a sua esquerda. Se $lb = 2$ o referido nó pode ter suas entradas conectadas na saída de um dos nós pertencentes as duas colunas imediatamente a sua esquerda. A fim de não permitir *feedback*, os nós pertencentes à mesma coluna não podem ser conectados entre si. Essa restrição é importante, pois é uma das características dos circuitos lógicos combinacionais. Além dos nós poderem ser conectados nas colunas anteriores designadas pelo parâmetro lb, é permitido que os nós se liguem diretamente às entradas de dados.

Na CGP cada nó atua como um operador, esse operador pode ser aritmético $\Gamma = \{+, -, \div, \times\}$, lógico $\Gamma = \{$ AND, OR, NOT, XOR $\}$, entre outros, de acordo com a natureza do problema abordado. Ressalta-se que os operadores também são codificados por números inteiros. As entradas dos nós são processadas pelo respectivo operador dando origem a saída do nó.

Os detalhes passo a passo da construção do genótipo-fenótipo de um determinado indivíduo serão exemplificados nas Figuras 20, 21 e 22.

Considere um determinado problema com quatro variáveis de entradas e duas saídas, mapeado por uma representação matricial onde $n_c = 4$, $n_r = 2$, como mostra a Figura 20(a). Neste exemplo é adotado o parâmetro *levels-back*, $lb = 2$ e o conjunto de operadores são codificados por $\Gamma = \{1, 2, 3, 4\}$,

Cada elemento da matriz representa um nó e são numerados na sequência das variáveis de entrada, nesse exemplo assumem os valores de 5 a 12. A Figura 20(b), ilustra o início da construção do genótipo - fenótipo do indivíduo, os nós da primeira coluna são conectados aleatoriamente nas variáveis de entradas e executam as funções designadas por $\Gamma = 3$ e $\Gamma = 2$ respectivamente.

As Figura 20(c) e Figura 21(a) mostram as conexões dos nós da segunda e terceira colunas, respectivamente. Nota-se que o parâmetro *levels-back*= 2 ainda não impôs nenhuma restrição nas ligações. Para a ligação da quarta coluna, o *levels-back*= 2 permite que os nós 11 e 12 sejam ligados apenas em algum dos nós referentes as colunas 2 e 3 ou nas entradas de dados. Dessa forma os alelos que representam as entradas dos nós 11 e 12 podem assumir qualquer valor do conjunto $\alpha = \{1, 2, 3, 4, 7, 8, 9, 10\}$

Da mesma forma, as possibilidades de conexões das saídas S_1 e S_2 são restritas aos nós das colunas 3 e 4 e nas variáveis de entradas, ou seja os alelos que representam as saídas podem assumir um dos valores $\alpha = \{1, 2, 3, 4, 9, 10, 11, 12\}$, como exemplificado na Figura 21(c).

Dessa forma são gerados os sub-grafos das saídas como mostra as Figuras 22(a) e 22(b). Nota-se que os nós 8 e 11 não participam dos sub-grafos S_1 e S_2 e, portanto, são considerados nós neutros ou inativos. A Figura 22(c) destaca esses nós inativos e os seus respectivos sub-grafos. Ressalta-se que os nós 5, 6, 7 e 9 participam dos sub-grafos dos nós inativos, mas não são denominados inativos pois também participam das conexões das saídas.

A presença dos nós que não fazem parte do fenótipo, possibilitam uma grande diversidade no fenótipo. A CGP explora intensamente essa neutralidade. A literatura sugere que o tamanho do genótipo seja bem maior do que o necessário para resolver o problema, de tal forma que em alguns problemas 95% dos nós são inativos. Essa é uma das principais características da CGP (MILLER, 2011; TURNER; MILLER, 2015a; VASICEK, 2015). Quando não se tem informações sobre a quantidade de nós necessários para resolver o problema, Miller (MILLER, 2011; MILLER, 2017 (accessed February 14, 2017)) sugere um valor entre 100 e 1000.

Neste exemplo nota-se que a CGP pode modelar facilmente problemas com várias saídas; essa é uma das vantagens destacadas da representação cartesiana em relação à PG tradicional (MILLER, 2011).

Apesar da variação do número de linhas n_r e do número de colunas n_c permitir uma grande diversidade na representação da CGP, Miller (MILLER, 2011) destaca que quando número de linhas $= 1$ e *Levels-back* $= n_c$ a representação é mais genérica, pois pode representar qualquer grafo acíclico. Essa representação pode ser denominada *Generic CGP* (TURNER; MILLER, 2015b; MANFRINI; BERNARDINO; BARBOSA, 2016b) e tem sido adotada como padrão na literatura (MILLER, 2011; GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015; HILDER; WALKER; TYRRELL, 2010; VASICEK; SEKANINA, 2014a; VASICEK, 2015).

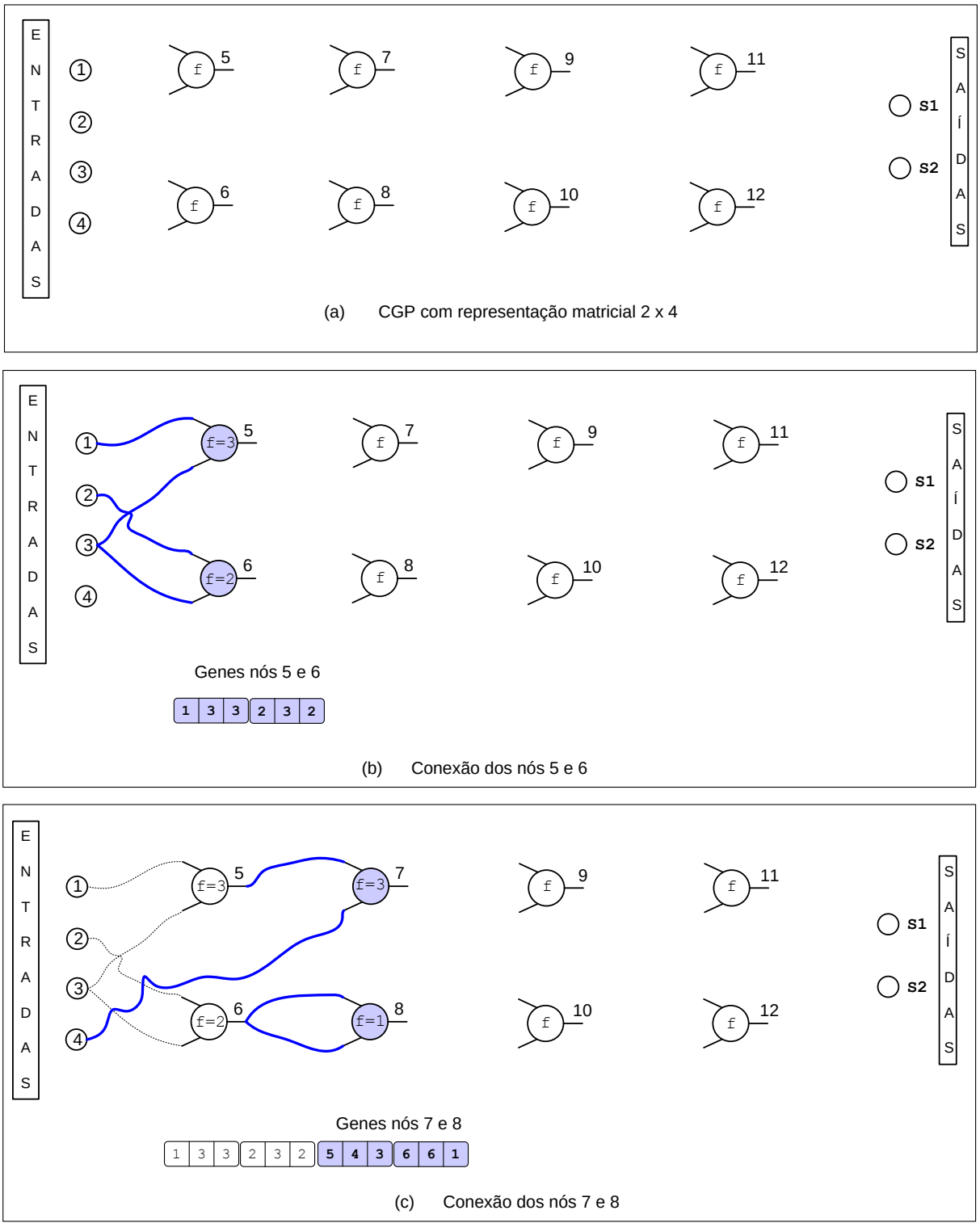


Figura 20 – Formação do fenótipo e genótipo dos nós 5 a 8.

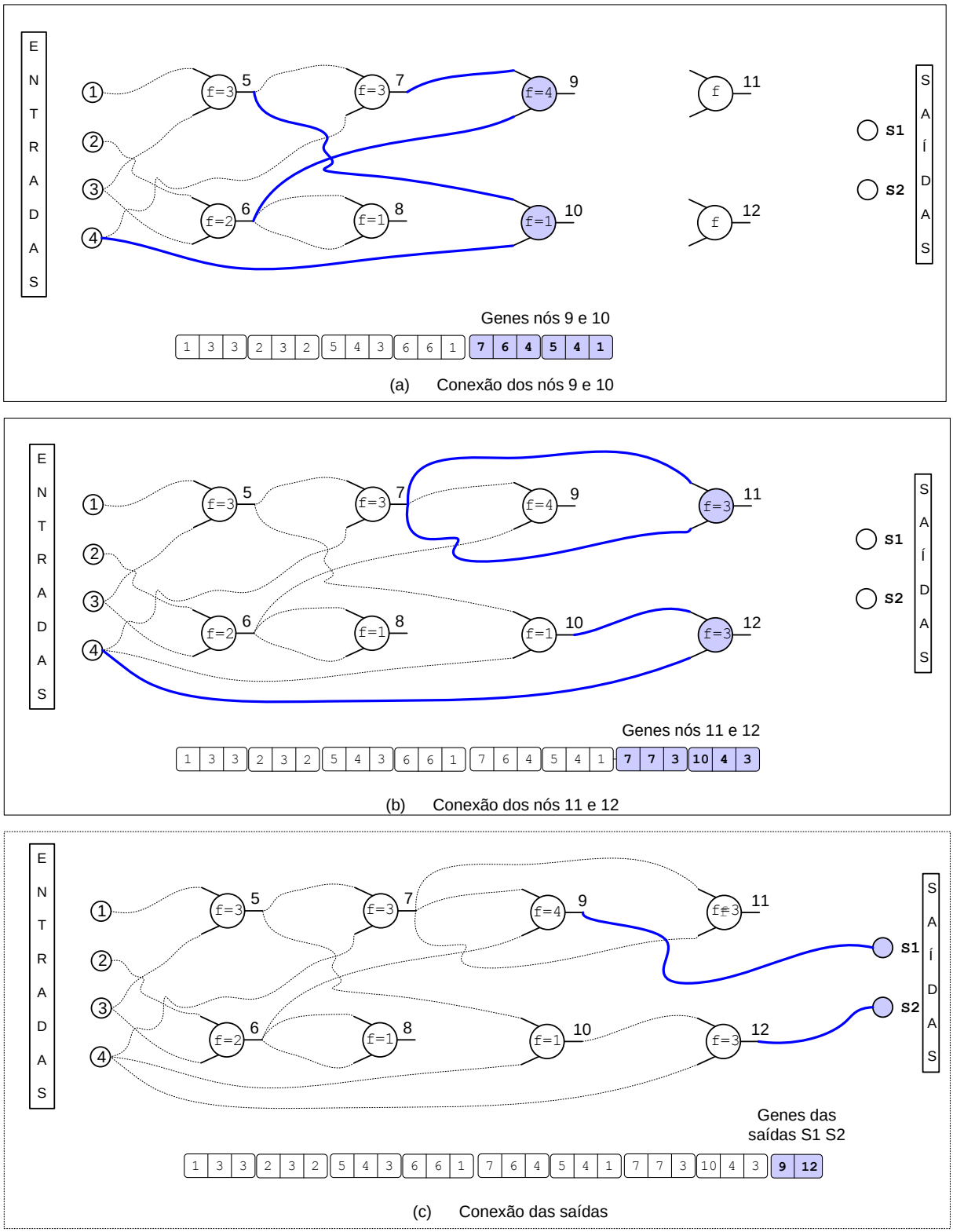
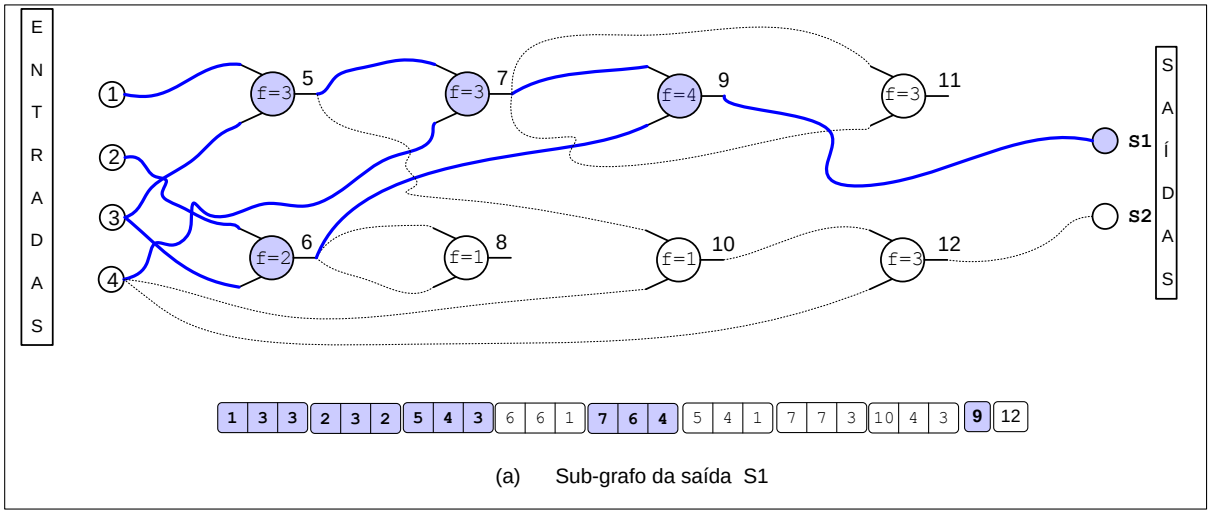
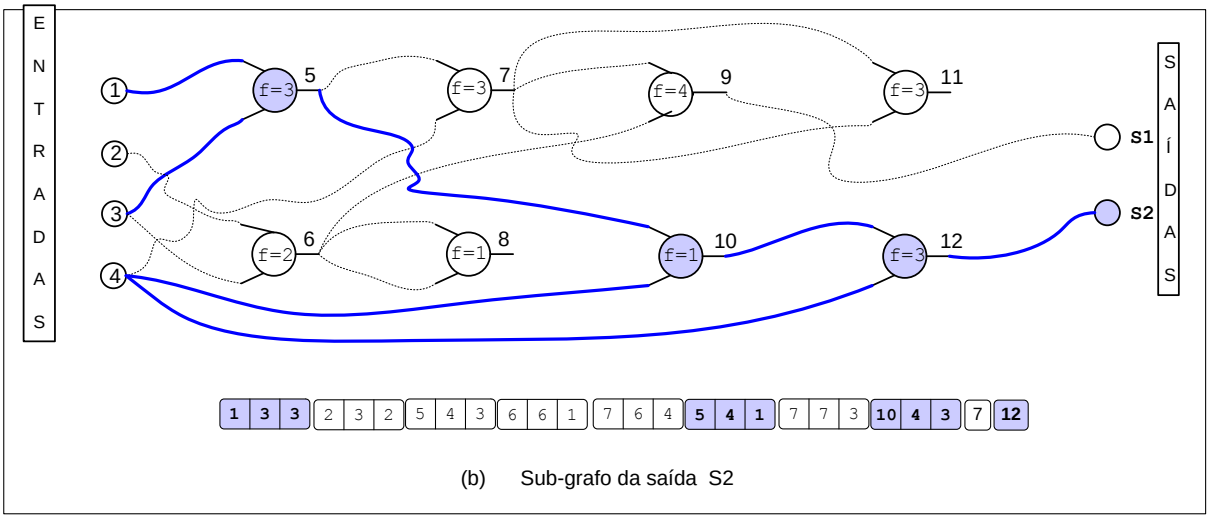


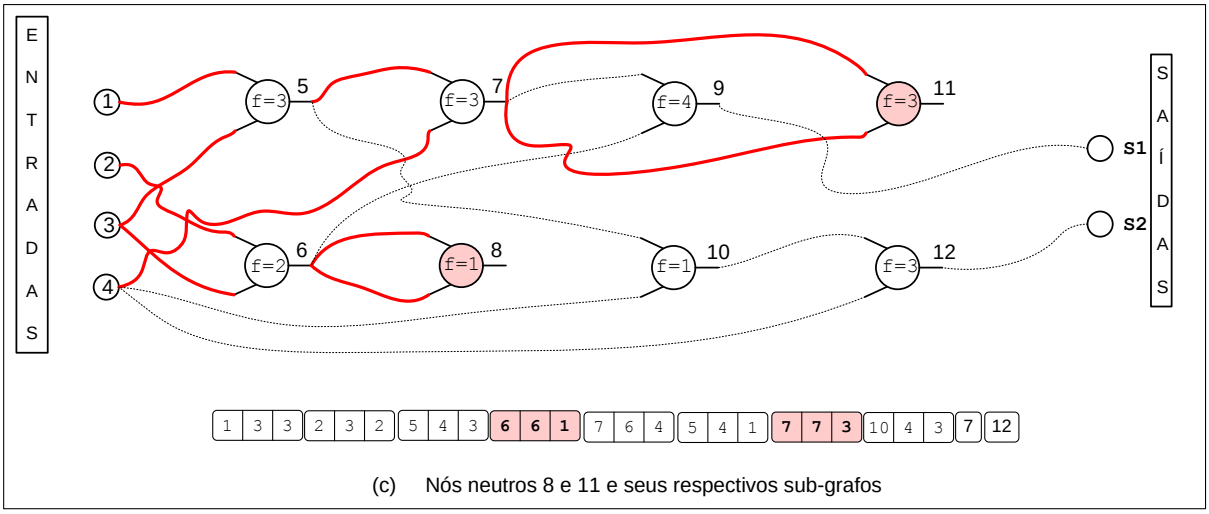
Figura 21 – Formação do fenótipo e genótipo dos nós 9 a 12 e as saídas S_1 e S_2.



(a) Sub-grafo da saída S1



(b) Sub-grafo da saída S2



(c) Nós neutros 8 e 11 e seus respectivos sub-grafos

Figura 22 – Sub-grafos das saídas S_1 e S_2. e sub-grafos da neutralidade

4.3.1 **Mutação - CGP**

Apesar de alguns trabalhos utilizarem um operador de mutação onde cada gene pode ser mutado com uma certa probabilidade (TURNER; MILLER, 2015a), permitindo assim que qualquer número de genes seja mutado de uma vez, o operador de mutação mais utilizado na CGP é um operador de mutação pontual (MILLER, 2011).

Na mutação pontual, um ou mais alelos são selecionados aleatoriamente e o valor de cada um desses alelos é modificado para outro valor válido. O número de genes a serem mutados é determinado pelo parâmetro denominado pontos de mutação pm.

Um exemplo de mutação pontual para $pm = 2$ é ilustrado na Figura 23, onde os nós 10 e 11 foram submetidos ao processo de mutação. Percebe-se que o alelo modificado referente ao nó 10 é um alelo que determina a ligação do nó, enquanto o alelo modificado do nó 11 representa a operação do nó. Nota-se ainda que o nó 11 é neutro, por isso não está influenciando nas saídas S_1 e S_2. A atuação do operador de mutação pode ser vista comparando as Figuras 23(a) e 23(b).

4.3.1.1 *Single Active Mutation - SAM*

Um algoritmo evolutivo que utilize o operador de mutação pontual, conforme descrito na Seção 4.3.1, não tem a garantia de que o fenótipo da prole e do progenitor sejam diferentes, pois existe a possibilidade de todos os nós mutados serem neutros. Nesse caso a avaliação dos novos indivíduos é desnecessária e não acrescenta nenhuma informação.

Para evitar esse desperdício de avaliações Brian Goldman e William Punch (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015) propuseram um método denominado *Single Active Mutation - SAM*, o qual assegura que um único gene ativo é mutado em cada prole gerada. Criar os descendentes usando SAM é um processo iterativo em que a mutação ocorre até que um gene ativo, selecionado aleatoriamente, seja mutado.

O operador SAM tem as seguintes propriedades:

- Exatamente um gene ativo é mutado para todos os descendentes.
- Zero ou mais genes inativos podem ser mutados.
- Nenhuma taxa de mutação é necessária.

4.3.2 **Algoritmo Evolutivo - CGP**

Uma estratégia evolutiva (E.E) simples conhecida como $(\mu + \lambda)$ (EIGEN, 1973), onde μ e λ representam o número de progenitores e descendentes respectivamente, é amplamente utilizada na CGP.

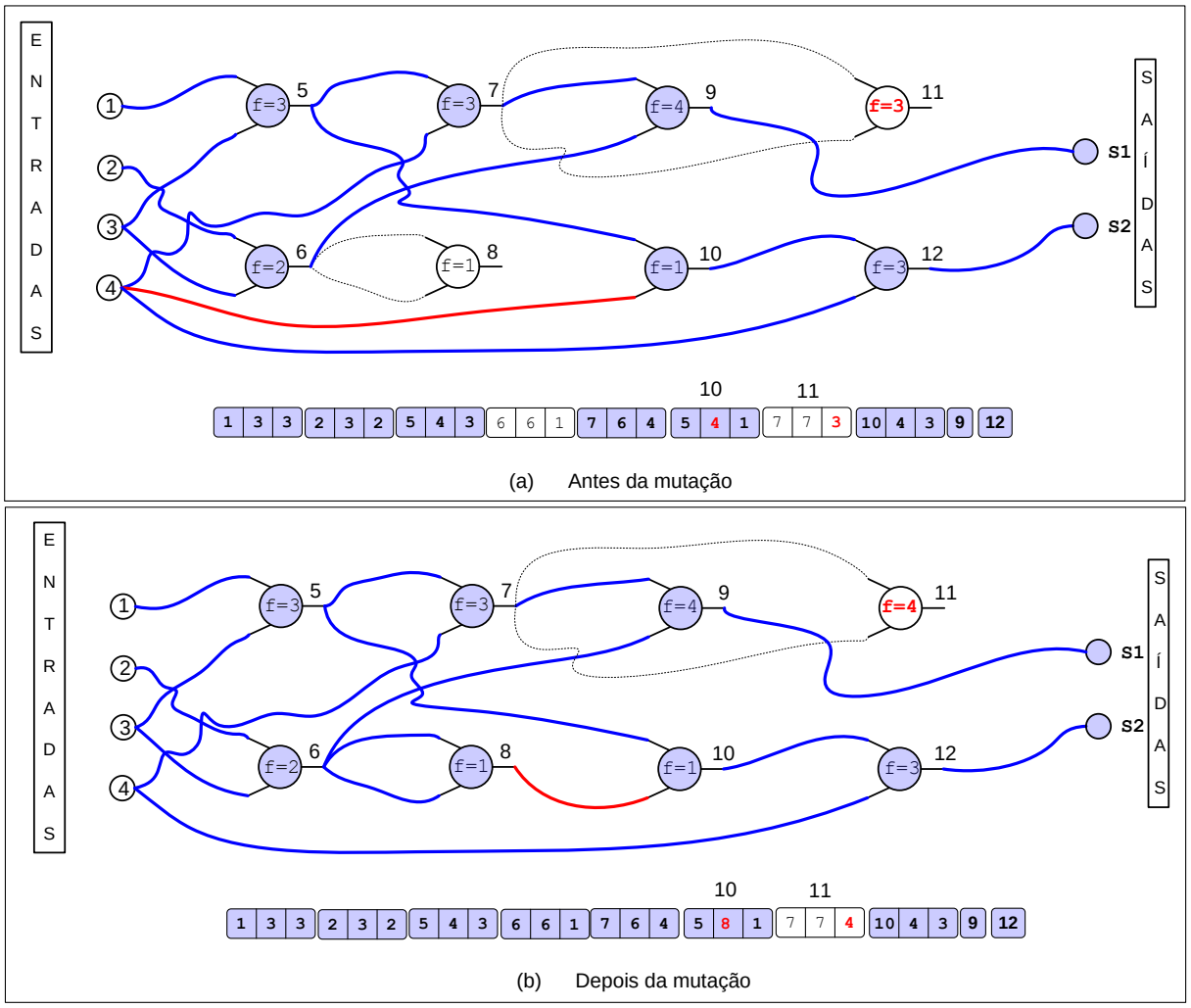


Figura 23 – Exemplo de mutação para $pm = 2$ (nó 10 e nó 11)

A E.E. $(\mu + \lambda)$ pode ser escrita da seguinte forma. Em uma determinada geração existe uma população de μ progenitores que geram λ descendentes por mutação. Em seguida os $(\mu + \lambda)$ indivíduos são avaliados e ordenados de acordo com os seus valores da função objetivo. Finalmente o processo de seleção faz com que apenas os μ melhores de todos os $(\mu + \lambda)$ indivíduos se tornem os progenitores da geração seguinte, isto é, a seleção é feita a partir dos $(\mu + \lambda)$ indivíduos. Na CGP normalmente μ é escolhido como 1 e $\lambda = 4$ (MILLER, 2011).

Originalmente as EEs baseavam-se em um único operador, a mutação para gerar novos indivíduos. Posteriormente foi introduzida a recombinação que era aplicada junto à mutação (SCHWEFEL, 1993). Particularmente na CGP a recombinação não têm sido utilizada, já que Miller (MILLER, 2011) realizou testes cujos resultados mostraram que a recombinação foi prejudicial.

Uma condição adicional é introduzida no algoritmo: quando a aptidão do melhor descendente é igual à aptidão do progenitor, o melhor descendente substitui o progenitor,

isso introduz diversidade na população (MILLER, 2011).

O pseudocódigo canônico da CGP é apresentado no algoritmo 1.

Algoritmo 1: Estratégia Evolutiva $(1 + 4)$

```
1  para todo i tal que 0 ≤ i < 4 faça
2  |  Gera indivíduo i
3  fim
4  progenitor ← melhor indivíduo
5  enquanto solução não foi encontrada ou geração < limite de gerações faça
6  |  para todo i, tal que 0 ≤ i < 4 faça
7  |  |  Aplica Mutação progenitor e gera filho i
8  |  fim
9  |  Seleciona o progenitor usando as regras:
10 |  se Algum descendente é melhor ou semelhante ao progenitor então
11 |  |  Melhor filho substitui o progenitor
12 |  senão
13 |  |  Progenitor é mantido
14 |  fim
15 fim
```

4.3.3 **Deriva Gênica Neutra - NDG**

Assim como a mutação e a pressão de seleção a Deriva Gênica Neutra (NDG) é uma força evolutiva (CUSTÓDIO, 2008). A NDG descreve a mudança no material genético inativo durante a evolução, sendo considerada um mecanismo poderoso de fuga de ótimos locais. Uma das características que diferencia a CGP de outras meta-heurísticas é a sua capacidade de explorar a NDG (TURNER; MILLER, 2015b).

Particularmente na computação evolucionista, um ótimo local pode ser considerado uma região do espaço de busca onde a maioria das mutações possíveis, ou até mesmo todas, resultam em uma solução de pior qualidade. Apesar da mutação ser um mecanismo de fuga de ótimos locais, ela isoladamente pode ser insuficiente, isso faz com que a fuga de ótimos locais se torne um desafio (CUNHA; TAKAHASHI; ANTUNES, 2012).

Na CGP os cromossomos possuem genes que são ignorados na construção do fenótipo. A NDG permite que mesmo as modificações feitas no genótipo que não alteraram o fenótipo sejam selecionados para as gerações futuras.

No algoritmo 1 a NDG está expressa na linha 10, onde o melhor descendente substitui o progenitor quando os dois possuem a mesma aptidão.

A seleção dos nós inativos para as gerações futuras permite a solução caminhar sobre o espaço de busca mesmo quando não ocorrem melhorias na aptidão. Isso fornece diversidade e possibilita que novas regiões sejam exploradas via mutação.

A GP também pode apresentar nós inativos quando representa as soluções candidatas usando estrutura de árvore: a Figura 24 apresenta um exemplo onde uma sub-árvore é multiplicada por zero e não influencia na saída do programa. Essa forma de redundância genética é denominada Deriva Genética Neutra Implícita, pois os genes são decodificados para o fenótipo, nesse sentido são ativos, porém não contribuem para a formação do fenótipo. Quando a deriva gênica está associada aos nós não codificantes é denominada Deriva Gênica Neutra Explícita (ENDG) (TURNER; MILLER, 2015b).

Umas das dificuldades no estudo da NDG é identificar quais genes são redundantes. Por exemplo, na GP determinar quais são os genes implicitamente redundantes é um desafio, uma vez que envolve a análise do fenótipo. Para a CGP, que apresenta redundância explícita, identificar os nós inativos é trivial, pois são os genes que não fazem parte dos sub-grafos das saídas.

A identificação dos nós ativos é importante para avaliar o indivíduo, como os inativos não contribuem para o fenótipo podem ser ignorados pela função de avaliação.

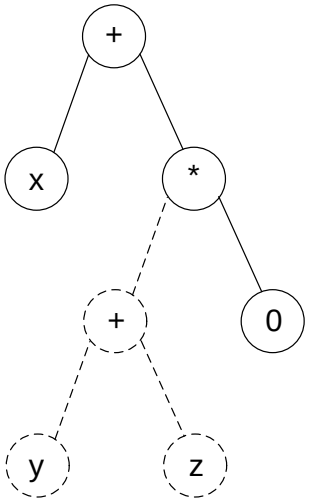


Figura 24 – Exemplo da deriva gênica neutra na PG representada em árvore. O operando zero faz com que a sub-árvore destacada seja neutra.

A justificativa da NDG introduzir diversidade na população, mesmo quando a solução está presa em um ótimo local será descrita no exemplo a seguir.

Considere que um algoritmo evolucionista atingiu um ótimo local. Ao longo das gerações a população irá convergir para a melhor solução encontrada, isso ocorre porque quando a solução está presa em um ótimo local, a maioria das mutações irão gerar uma prole pior que o progenitor, e portanto apenas descendentes geneticamente semelhantes ao progenitor serão selecionados e sobrevivem. Isso faz com que toda a população se torne geneticamente semelhante. Se todos os membros da população são parecidos, então o número de soluções oriundas de mutações são reduzidas. Assim, escapar de um ótimo local torna-se cada vez mais difícil. Quando os genes inativos estão presentes, os genes

ativos irão convergir, mas não há pressão para que os genes inativos convirjam, isso faz com que o material genético inativo se modifique aleatoriamente (deriva). Como a mutação pode tornar os genes inativos em ativos, pode-se concluir que o número de soluções ao longo das gerações é muito maior quando a NDG está presente (TURNER; MILLER, 2015b).

5 MÉTODOS PROPOSTOS

5.1 INVESTIGAÇÃO DO PROCESSO EVOLUTIVO

Com o objetivo de compreender as causas fundamentais do sucesso - fracasso relacionado ao processo evolutivo, é proposta uma metodologia que analisa os detalhes envolvidos nas modificações genéticas ao longo das gerações. Com essa metodologia é possível extrair informações intrínsecas ao processo; algumas informações obtidas podem ser desconhecidas até mesmo por especialistas.

A meta-heurística escolhida para ser utilizada no procedimento de investigação do mecanismo evolutivo é a CGP. Inicialmente, a influência da presença dos *'junk genes'* será investigada para diversas taxas de mutação. Na sequência, serão analisadas as modificações essenciais do mecanismo evolutivo, ou seja, as transformações que ocasionaram efetivamente a evolução.

5.1.1 **Descrição dos problemas investigados**

Na busca pela compreensão das causas de sucesso do mecanismo evolutivo foram escolhidos quatro exemplos estudados pela literatura (COELLO; CHRISTIANSEN; AGUIRRE, 2000b; KARAKATIC; PODGORELEC; HERICKO, 2013; SASAO, 1993; GARCÍA; COELLO, 2002; MANFRINI; BARBOSA; BERNARDINO, 2014). Em todos os experimentos utilizou-se a meta-heurística CGP, com a estratégia evolutiva $(1 + 4)$ e uma codificação matricial 1x100, sendo a taxa de retorno igual ao número de colunas. O algoritmo termina quando o melhor resultado da literatura é alcançado ou até que o número máximo de avaliações seja atingido.

A avaliação dos indivíduos é baseada em uma aptidão dinâmica. No início da pesquisa, apenas a validade das saídas são levadas em consideração, a CGP basicamente explora o espaço de busca a procura de circuitos que atendam a todos os requisitos da tabela verdade. Uma vez que a solução factível é encontrada a função aptidão se modifica, de tal forma que todos os projetos são recompesados de acordo com o número de nós inativos e a cada WIRE que surge, ou seja, a CGP busca maximizar o número de nós inativos e WIREs (equivalente a minimizar o número de portas lógicas).

Para esses exemplos foram permitidas 100000 avaliações e o conjunto de operadores utilizados foi (AND, OR, XOR, NOT e WIRE). O objetivo dos problemas propostos é encontrar um circuito otimizado que tenha os mintermos[1] dados por:

- **Exemplo 1:**

$$F_1(A, B, C, D) = \sum m(0, 1, 3, 6, 7, 8, 10, 13)$$

[1] A definição de mintermos pode ser encontrada no Apêndice A.0.1.

- **Exemplo 2:**

$$F_2(A,B,C,D) = \sum m(0,4,5,6,7,8,9,10,13,15)$$

- **Exemplo 3:**

$$F_3(A,B,C,D) = \sum m(2,3,8,9,11,12,13,14)$$

- **Exemplo 4:**

$$F_4(A,B,C,D) = \sum m(0,1,4,5,6,7,10,15)$$

5.1.2 Análise da Neutralidade

Para avaliar os benefícios do *'junk genes'* no mecanismo evolutivo o algoritmo da CGP foi executado 120 vezes em dois cenários diferentes. No primeiro cenário, denominado *"Neutralidade ON"*, foi permitida a mutação em qualquer tipo de nó, neutro ou ativo. Por sua vez, no segundo cenário, *"Neutralidade OFF"*, foi permitida a mutação apenas em nós ativos. Além disso, a mutação também variou ao longo das execuções: no início adotou-se a mutação em apenas um ponto ($pm = 1$) e a cada 20 execuções o pm foi incrementado de uma unidade. Dessa forma, foi possível analisar o desempenho da CGP com a neutralidade ON/OFF a partir de diferentes mutações ($1 \leq pm \leq 6$). A aptidão foi normalizada levando em consideração o melhor resultado da literatura. Para os quatro exemplos propostos os resultados da influência da neutralidade são mostrados nas Figuras 25, 26, 27 e 28, respectivamente.

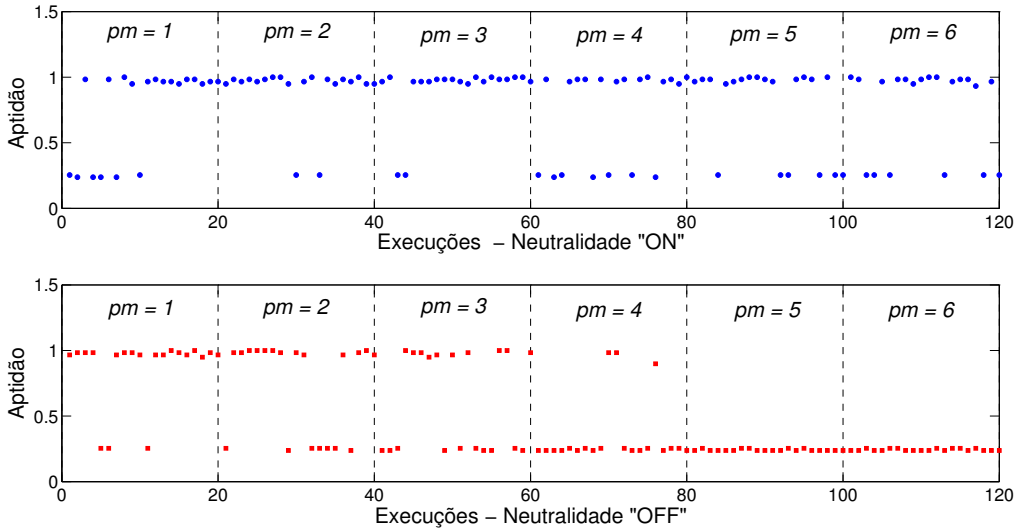


Figura 25 – Exemplo 1: Comparação de 120 execuções para Neutralidade ligada (ON) e Neutralidade desligada (OFF). Os pontos de mutação foram variados a cada 20 execuções, assumindo os valores $pm = 1$ até $pm = 6$.

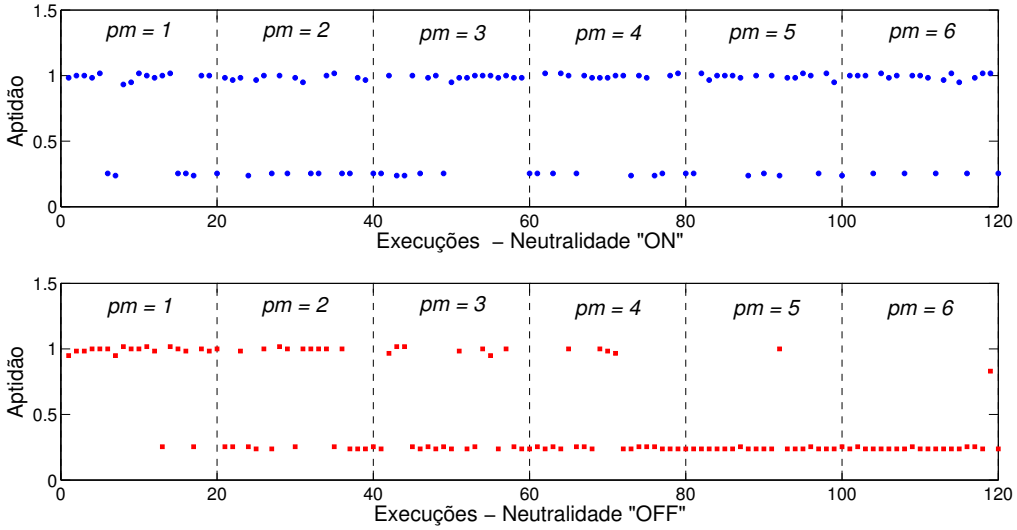


Figura 26 – Exemplo 2: Comparação de 120 execuções para Neutralidade ligada (ON) e Neutralidade desligada (OFF). Os pontos de mutação foram variados a cada 20 execuções, assumindo os valores $pm = 1$ até $pm = 6$.

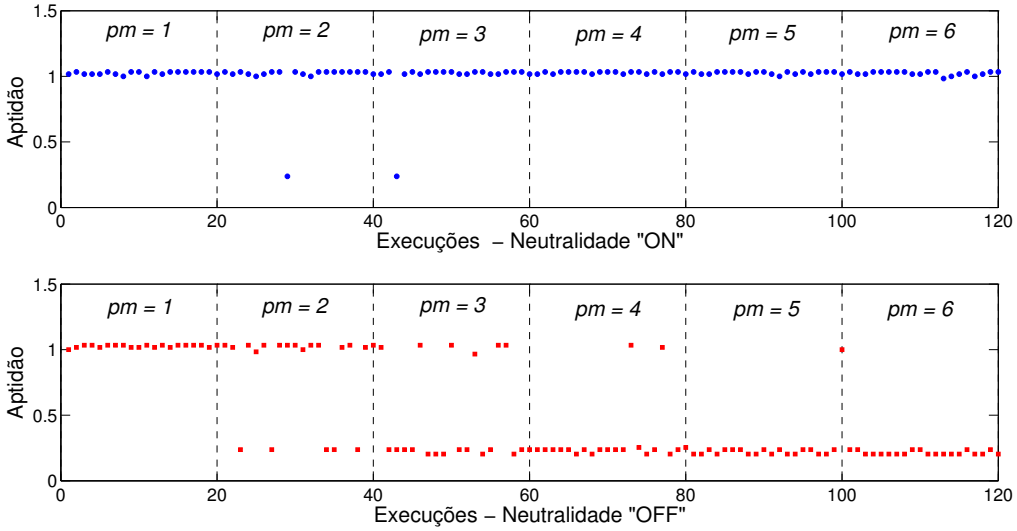


Figura 27 – Exemplo 3: Comparação de 120 execuções para Neutralidade ligada (ON) e Neutralidade desligada (OFF). Os pontos de mutação foram variados a cada 20 execuções, assumindo os valores $pm = 1$ até $pm = 6$.

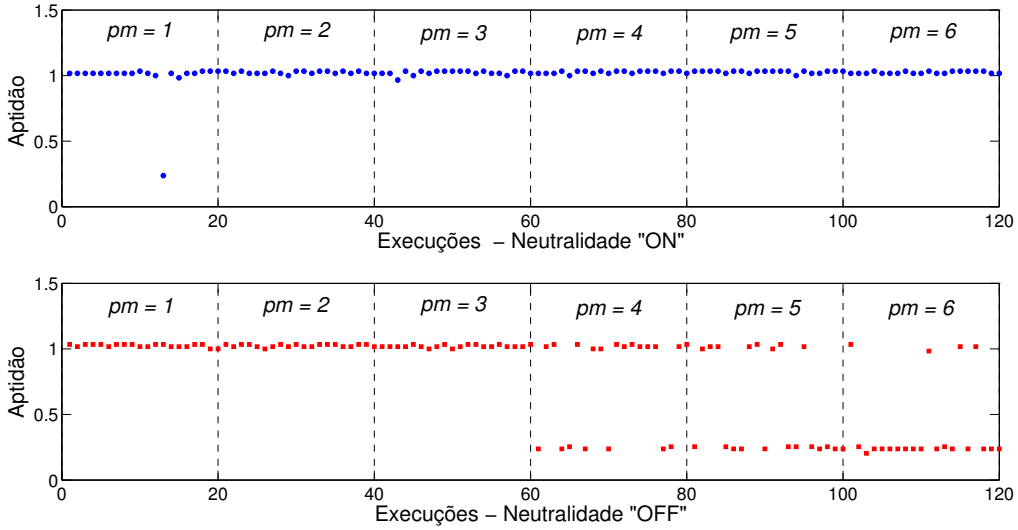


Figura 28 – Exemplo 4: Comparação de 120 execuções para Neutralidade ligada (ON) e Neutralidade desligada (OFF). Os pontos de mutação foram variados a cada 20 execuções, assumindo os valores $pm = 1$ até $pm = 6$.

Em todos os experimentos de análise da neutralidade representados nas Figuras 25, 26, 27 e 28, verifica-se que para baixos valores de *pm* o desempenho da CGP é similar nos dois cenários. Na medida em que os pontos de mutação aumentam, observa-se que diminui a quantidade de execuções em que o melhor indivíduo atinge o sucesso (resultado da literatura), ou seja, há uma degradação do desempenho da *"Neutralidade OFF"*.

O fato do cenário *"Neutralidade OFF"* não permitir que o operador de mutação atue em nós neutros faz com que as variações no fenótipo sejam sempre proporcionais aos valores de *pm*, ou seja, valores elevados de *pm* sempre ocasionam mudanças bruscas no fenótipo, o que degrada a solução.

Por outro lado, no cenário *"Neutralidade ON"* o operador de mutação atua tanto em nós ativos como em inativos, ou seja, o número de nós ativos modificados é variável. Altos valores de *pm* podem alterar bruscamente o fenótipo ou, simplesmente, causar mudanças suaves. Portanto, o cenário "*Neutralidade ON* é menos sensível ao valor de *pm* e possui a capacidade de regular a quantidade de nós ativos que sofrem mutação.

Portanto o desempenho da CGP no cenário "*Neutralidade ON* foi menos sensível a escolha do *pm*. Ressalta-se que nas estratégias evolutivas o controle do tamanho do passo da mutação é uma das componentes mais importantes para a eficiência do processo de busca (CUNHA; TAKAHASHI; ANTUNES, 2012).

5.1.3 **Mutações benéficas**

A análise da atuação dos operadores genéticos, bem como das modificações topológicas, pode contribuir para a descoberta de algum viés (ou alguma tendência) associado à melhora da aptidão. A ideia consiste em investigar as transformações genéticas benéficas, ou seja, as modificações que geram um aumento na aptidão do indivíduo. Como na CGP o único operador atuante é a mutação, serão investigadas as denominadas Mutações Benéficas.

Considerando a E.E. $(1 + \lambda)$ conforme descrito na Seção 4.3, a cada nova geração, caso o melhor descendente tenha aptidão maior que o progenitor (hipótese de mutação benéfica), as características do individuo pai e do melhor filho, bem como algumas informações relevantes, são armazenadas em uma matriz. Dessa forma, a cada ocorrência da mutação benéfica uma matriz denominada Matriz Evolução foi sendo atualizada conforme exemplo da Figura 29.

A primeira e a segunda colunas da Matriz Evolução, denominadas *Rodada* e *Número da avaliação*, respectivamente, fornecem informações sobre quando ocorreu a mutação benéfica. A terceira coluna registra se a mutação foi uma troca na conexão (mutação tipo conexão), ou se foi uma troca do operador genético (mutação tipo porta)[2]. Por sua vez, a quarta e quinta colunas armazenam a quantidade de portas dos indivíduos pai e filho respectivamente. Na sequência das colunas 6 a 11 são armazenados os genes do pai e filho onde o operador de mutação foi aplicado. Em seguida, as colunas 12 e 13 fornecem a variação da aptidão em decorrência da mutação benéfica.

Como exemplo, verifica-se na Figura 29 que na 5ª rodada, na 12ª chamada da função de avaliação, ocorreu uma mutação benéfica tipo porta, em que um operador AND codificado como 200 foi trocado por um operador OR, identificado como 300. Nota-se que essa transformação ocasionou um aumento da aptidão de 13 para 14. Como a mutação apenas trocou um operador AND $\rightarrow$ OR, não houve mudança no número de portas lógicas, conforme verifica-se nas colunas 4 e 5.

Para os exemplos descritos na Seção 5.1.1, foram construídas as respectivas Matrizes Evolução para 51 execuções com $pm = 1$. A partir dos dados obtidos, foram feitas diversas análises, as quais serão mostradas nas próximas seções.

5.1.3.1 *Análise da variação do número de portas lógicas dos indivíduos ao longo da evolução*

A variação do número de portas lógicas em função da aptidão do melhor indivíduo pode ser analisada através dos dados presentes nas colunas 5 e 13 da Matriz Evolução.

[2] As definições referentes ao tipo de mutação, conexão ou porta, estão formalizadas na Seção 5.1.3.3.

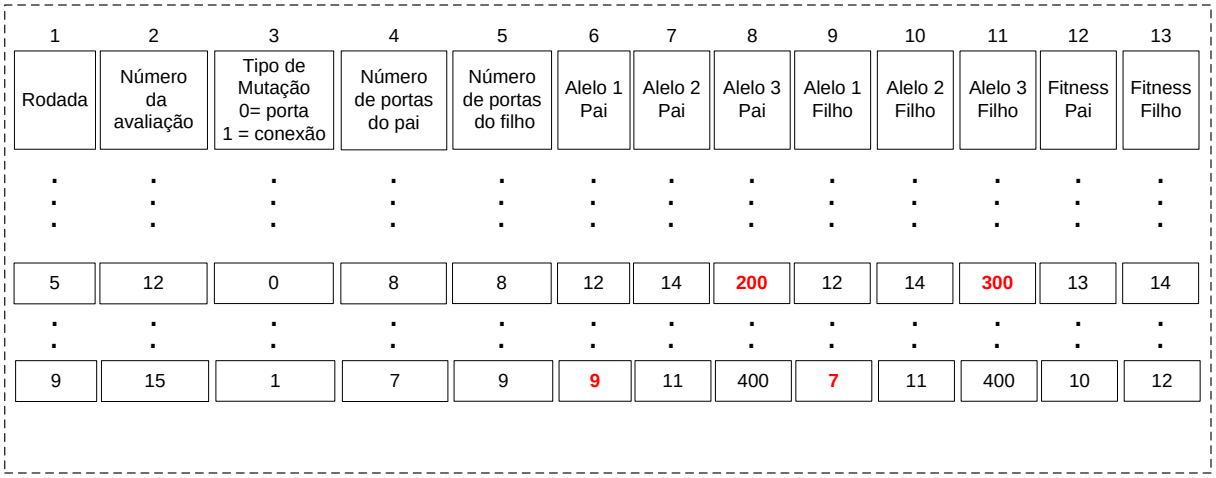


Figura 29 – Matriz Evolução: A cada ocorrência de uma mutação benéfica armazena as informações dos indivíduos Pai e Filho.

Para cada exemplo proposto foram plotados os gráficos referentes a cinco indivíduos de execuções distintas. As Figuras 30, 31, 32, 33 mostram o comportamento da *quantidade de portas lógicas* x *valor da função objetivo* ao longo das gerações.

Como os exemplos propostos possuem quatro variáveis de entradas, uma solução factível é encontrada quando valor da função objetivo (FOB) é igual a 16. Em decorrência do cálculo da FOB ser distinto conforme as regiões (infactível e factível). O comportamento de cada indivíduo foi plotado para as respectivas regiões.

Analisando as Figuras 30, 31, 32, 33 percebe-se que a quantidade de portas lógicas de cada um dos indivíduos aumentou na região infactível. Em relação à região factível, o comportamento foi oposto, ou seja, o número de portas lógicas diminuiu ao longo do processo. Esse comportamento na região factível já era esperado, uma vez que essa região tem como objetivo a minimização do número de portas lógicas.

5.1.3.2 *Número de portas lógicas associadas a cada aptidão*

Conforme discutido na Seção 4.3, na meta-heurística CGP um determinado indivíduo que entrou na população em decorrência de uma mutação benéfica não será necessariamente o progenitor da próxima mutação benéfica. Tal situação está ilustrada na Figura 34, onde o indivíduo 2 foi originado de uma mutação benéfica, sendo o primeiro indivíduo com $FOB = B$. Ocorre que o processo evolutivo prosseguiu e o indivíduo 2 foi substituído por outro indivíduo com a mesma aptidão. Após algumas mutações sem que houvesse melhoria na aptidão, a mutação sobre o indivíduo 4 originou um indivíduo com uma maior aptidão (indivíduo 5).

Nesse contexto, ao analisar as mutações benéficas, dois indivíduos são de interesse:

- O primeiro indivíduo a entrar na população com determinada aptidão (indivíduo 2).

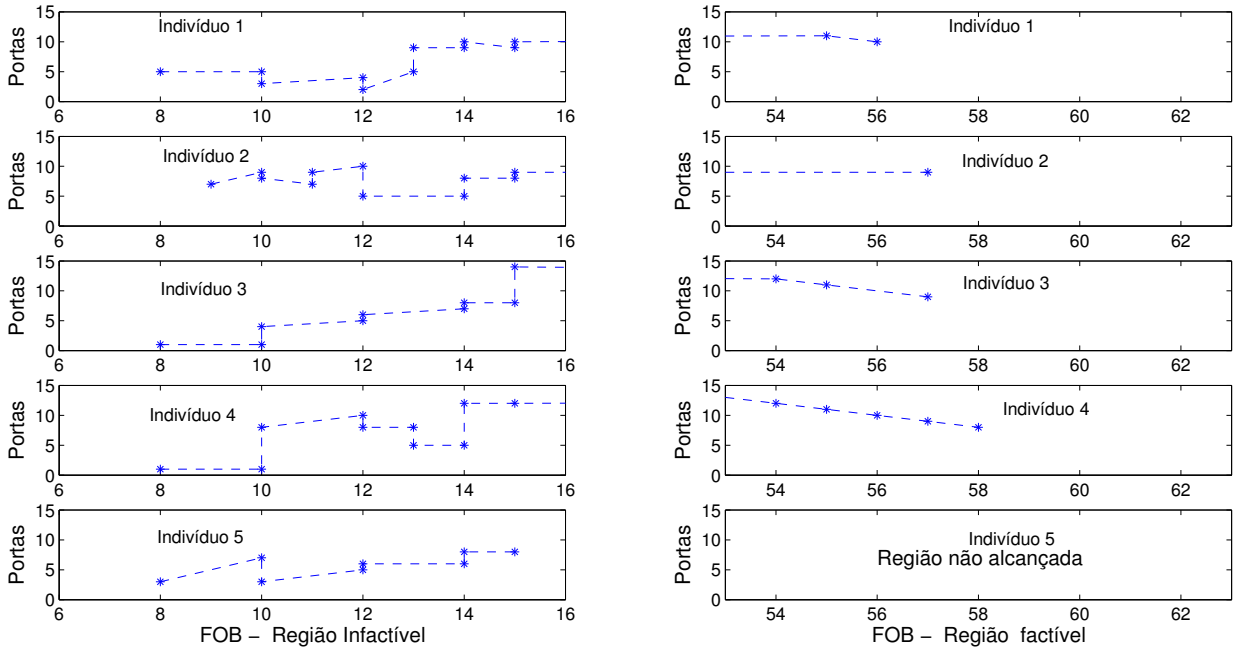


Figura 30 – Exemplo 1: Variação do número de portas lógicas em função da aptidão para 5 execuções independentes. Os gráficos à esquerda mostram a variação da quantidade de portas lógicas na região infactível e à direita na região factível.

Nos exemplos tratados nesse capítulo as informações sobre o número de portas desse primeiro indivíduo estão contidas na 5ª coluna da Matriz Evolução denominada "Número de portas do filho".

- O progenitor que permitirá um aumento na aptidão - representado pelo indivíduo 4. Nos exemplos tratados nesse capítulo as informações acerca do número de portas desse indivíduo estão contidas na 4ª coluna da Matriz Evolução denominada "Número de portas do pai".

Para analisar a variação de portas lógicas desses dois indivíduos foram plotados os gráficos referentes às colunas 4 e 5 da Matriz Evolução em função da aptidão, conforme mostrado na Figura 35. Ressalta-se que o eixo 'FOB dos indivíduos' apresenta valores repetidos para cada aptidão, sendo o primeiro valor referente ao "Número de portas do filho"e o segundo ao "Número de portas do pai". Esses gráficos foram feitos apenas para a região infactível, pois conforme mostrado na Seção 5.1.3.1 na região factível o número de portas lógicas decresce de acordo com o aumento da própria função objetivo.

5.1.3.3 *Frequência dos tipos de mutação*

Conforme mostrado na Seção 4.3, durante o processo evolutivo um determinado nó é escolhido aleatoriamente para ser submetido ao operador de mutação. Após essa seleção,

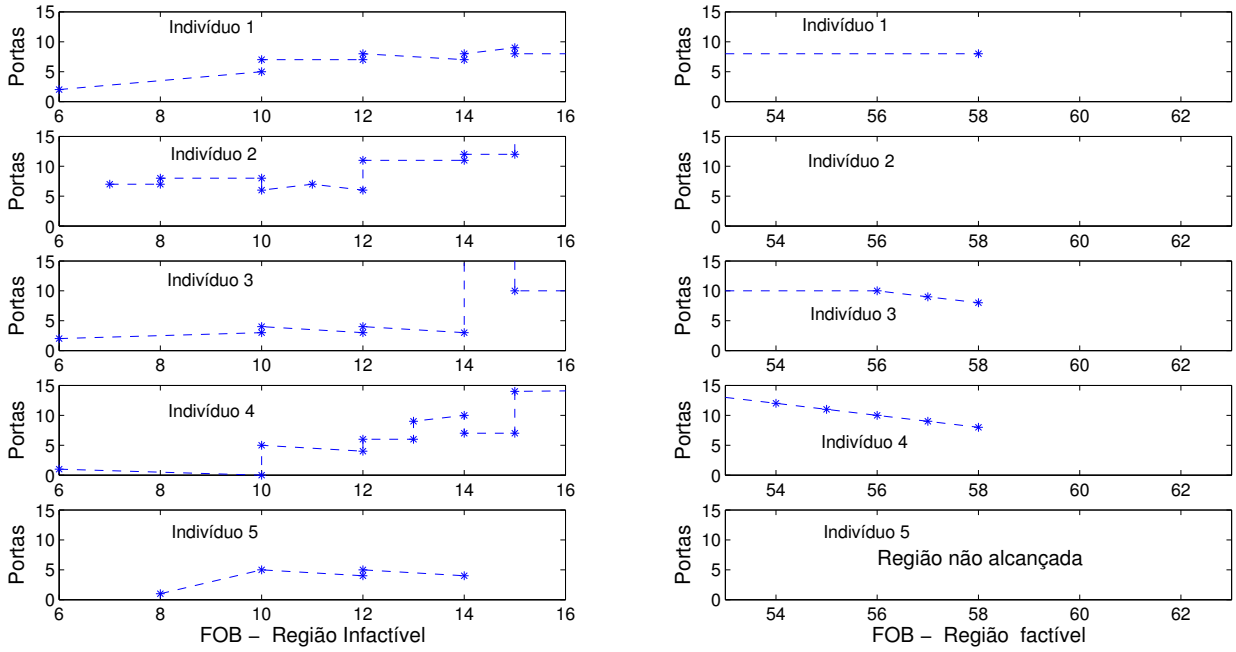


Figura 31 – Exemplo 2: Variação do número de portas lógicas em função da aptidão para 5 execuções independentes. Os gráficos à esquerda mostram a variação da quantidade de portas lógicas na região infactível e à direita na região factível.

determina-se qual alelo será submetido ao operador. Dessa forma devido à natureza do alelo escolhido, a mutação pode ser:

Mutação tipo conexão: quando um alelo relacionado à conexão do nó é submetido à mutação.

Mutação tipo Porta: quando um alelo relacionado a operação que o nó executa é submetido à mutação.

A frequência de ocorrências das mutações benéficas tipo porta ou tipo conexão foi analisada de acordo com a região de busca infactível ou factível, conforme mostrado nas Figuras 36 e 37, onde para cada FOB tem-se a quantidade de mutações de acordo com o tipo (porta ou conexão). Os dados dos gráficos representados nas Figuras 36 e 37 também podem ser interpretados da seguinte forma: Quando houve uma mutação benéfica e o novo indivíduo teve uma FOB maior, esse incremento na FOB foi ocasionado pelo tipo de mutação indicado.

As análises dos gráficos das Figuras 36 e 37 mostram que tanto na região infactível quanto na região factível há uma predominância de mutações tipo conexão em relação às mutações tipo porta. Considerando que cada nó possui dois alelos tipo conexão e um alelo tipo porta, esse viés já era esperado.

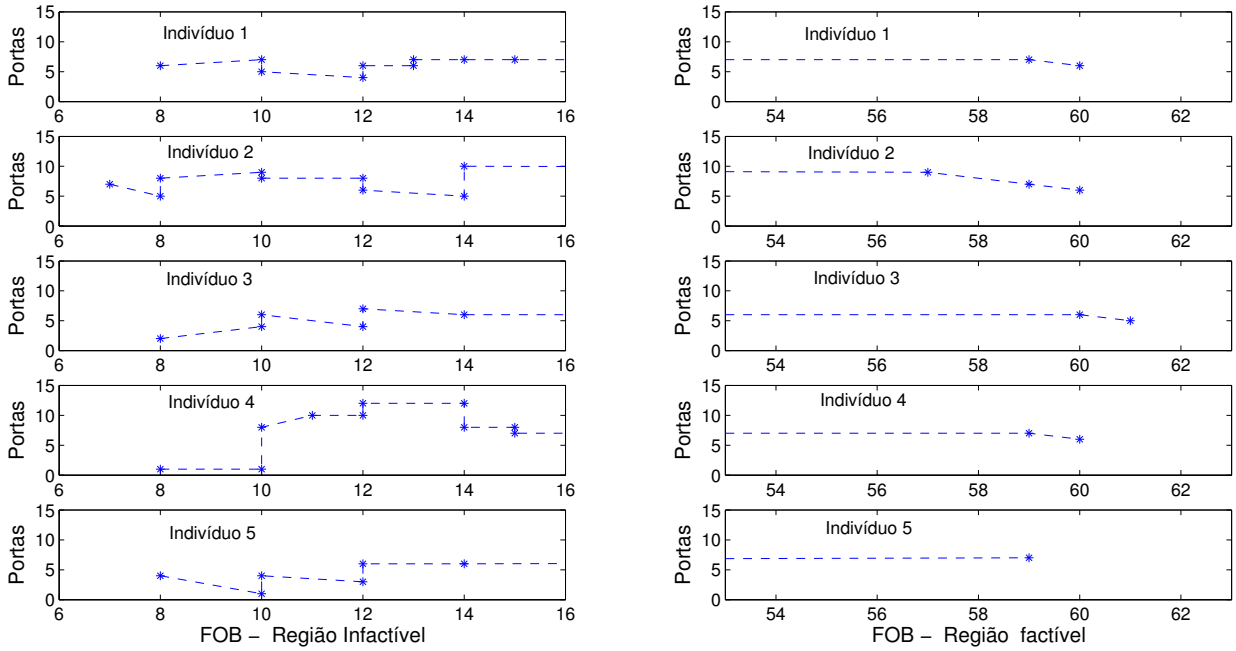


Figura 32 – Exemplo 3: Variação do número de portas lógicas em função da aptidão para 5 execuções independentes. Os gráficos à esquerda mostram a variação da quantidade de portas lógicas na região infactível e à direita na região factível.

5.1.3.4 *Primeiro indivíduo factível*

O sucesso da CGP é alcançado quando todas as restrições do problema em questão são satisfeitas, o que ocorre no momento em que o primeiro indivíduo factível é encontrado. A Figura 38 mostra a conformação desse indivíduo em relação à distribuição do tipo de porta para os problemas tratados nessa seção. Analisando os gráficos da Figura 38 nota-se que houve a mesma tendência para todos os exemplos, ou seja, houve uma predominância do operador XOR e uma menor utilização dos operadores WIRE e NOT.

5.1.3.5 *Matriz Probabilidade de Transição*

As colunas 8 e 11 da Matriz Evolução armazenam os dados das transformações benéficas entre os operadores. Dessa forma, toda vez que ocorre uma melhora na aptidão ocasionada por uma mutação tipo porta, o operador do pai e do filho são diferentes e a comparação das colunas 8 e 11 indicam qual a troca de operadores foi benéfica ao processo. A partir dessas informações é possível montar a denominada Matriz Probabilidade de Transição.

A Figura 39 ilustra a criação da Matriz Probabilidade de Transição ao longo do processo evolutivo. Cada célula da matriz representa a frequência de ocorrência de uma determinada mutação. Nota-se que quando ocorre uma mutação benéfica tipo porta, a

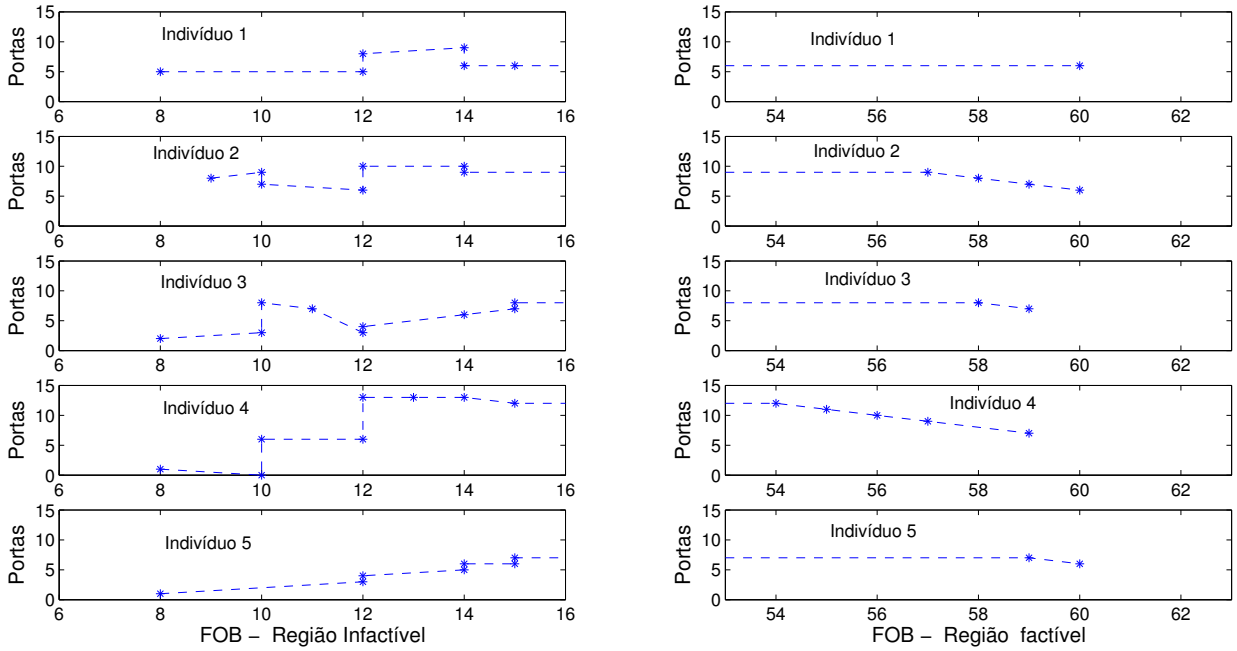


Figura 33 – Exemplo 4: Variação do número de portas lógicas em função da aptidão para 5 execuções independentes. Os gráficos à esquerda mostram a variação da quantidade de portas lógicas na região infactível e à direita na região factível.

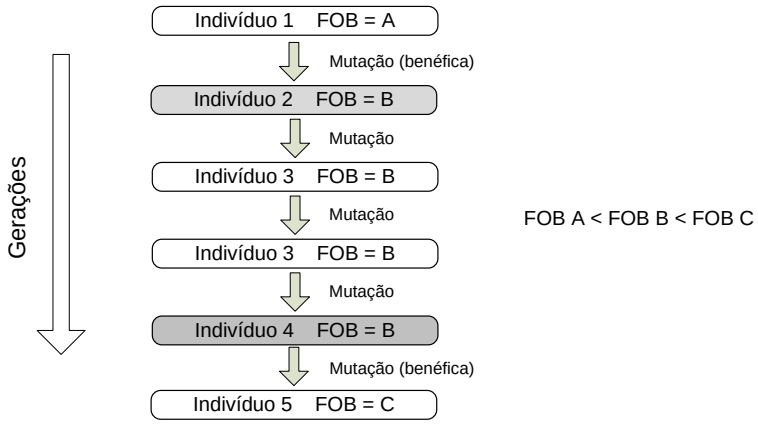


Figura 34 – Esquema ilustrando as trocas dos indivíduos em decorrência das mutações.

célula associada à mutação em questão é incrementada. Nesse sentido, as Figuras 39(a) 39(b) ilustram o preenchimento da célula correspondente à primeira mutação benéfica designada por XOR $\rightarrow$ OR. Ao final do processo, as células estão preenchidas com o número de ocorrência da respectiva mutação, conforme ilustra a Figura 39(c).

A Matriz Probabilidade de Transição pode ser representada conforme Figura 39(d). Para todos os exemplos foram plotadas as Matrizes de Probabilidade de Transição para as regiões factível e infactível, sendo que os resultados estão descritos nas Figuras 40 e 41. Analisando os gráficos da região infactível, mostrados na Figura 40, nota-se que em todos

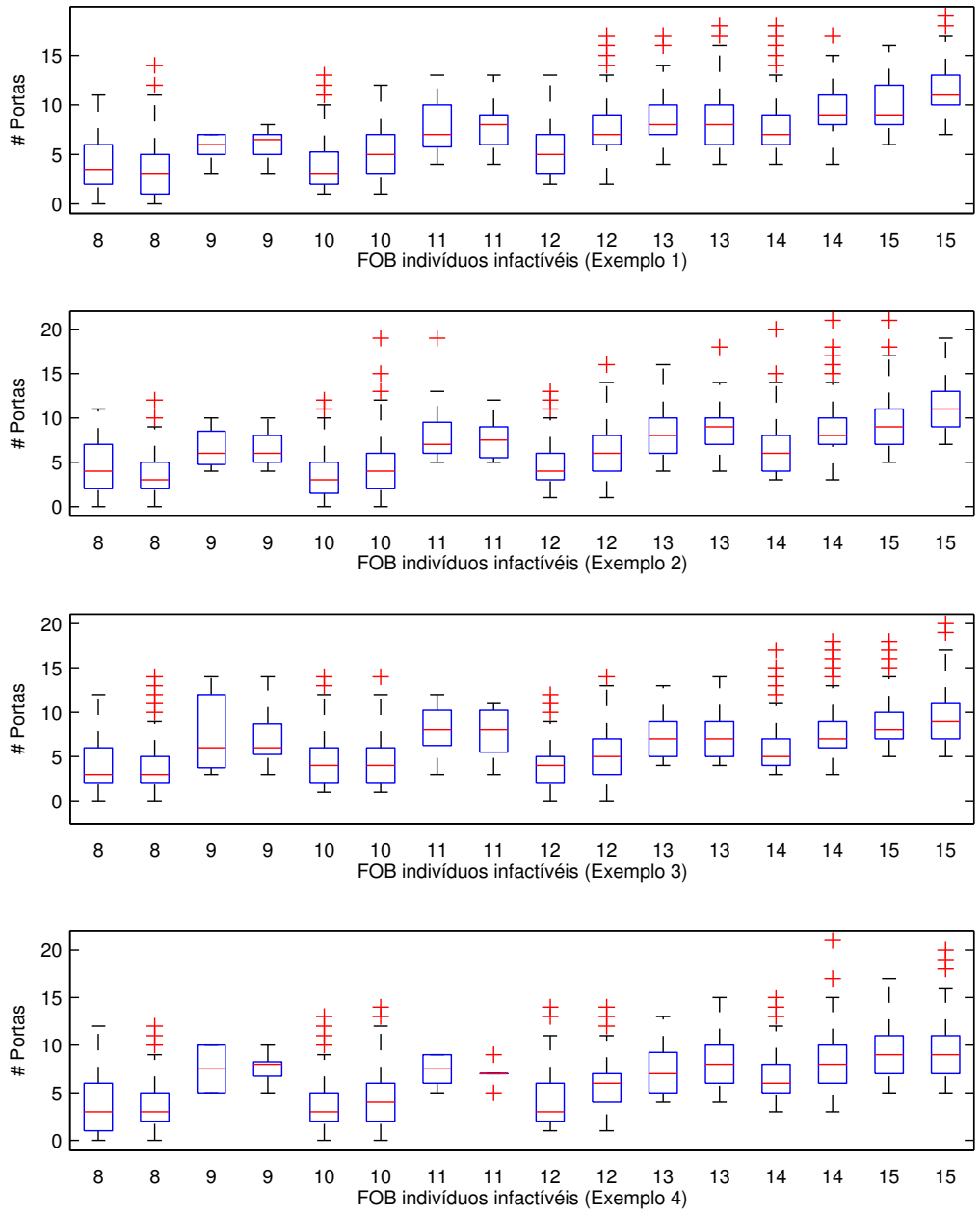


Figura 35 – Variação do número de nós ativos para os diversos valores de aptidão. Os valores duplicados de cada aptidão referem-se ao primeiro indivíduo que entrou na população (proveniente de uma mutação benéfica) e ao ultimo indivíduo da população, o qual gerou um descendente mais apto.

os exemplos houve algumas mutações benéficas que ocorreram com maior frequência, como por exemplo OR → XOR e NOT → XOR. Por outro lado, mutações benéficas dos tipos XOR → AND e OR → AND foram raras.

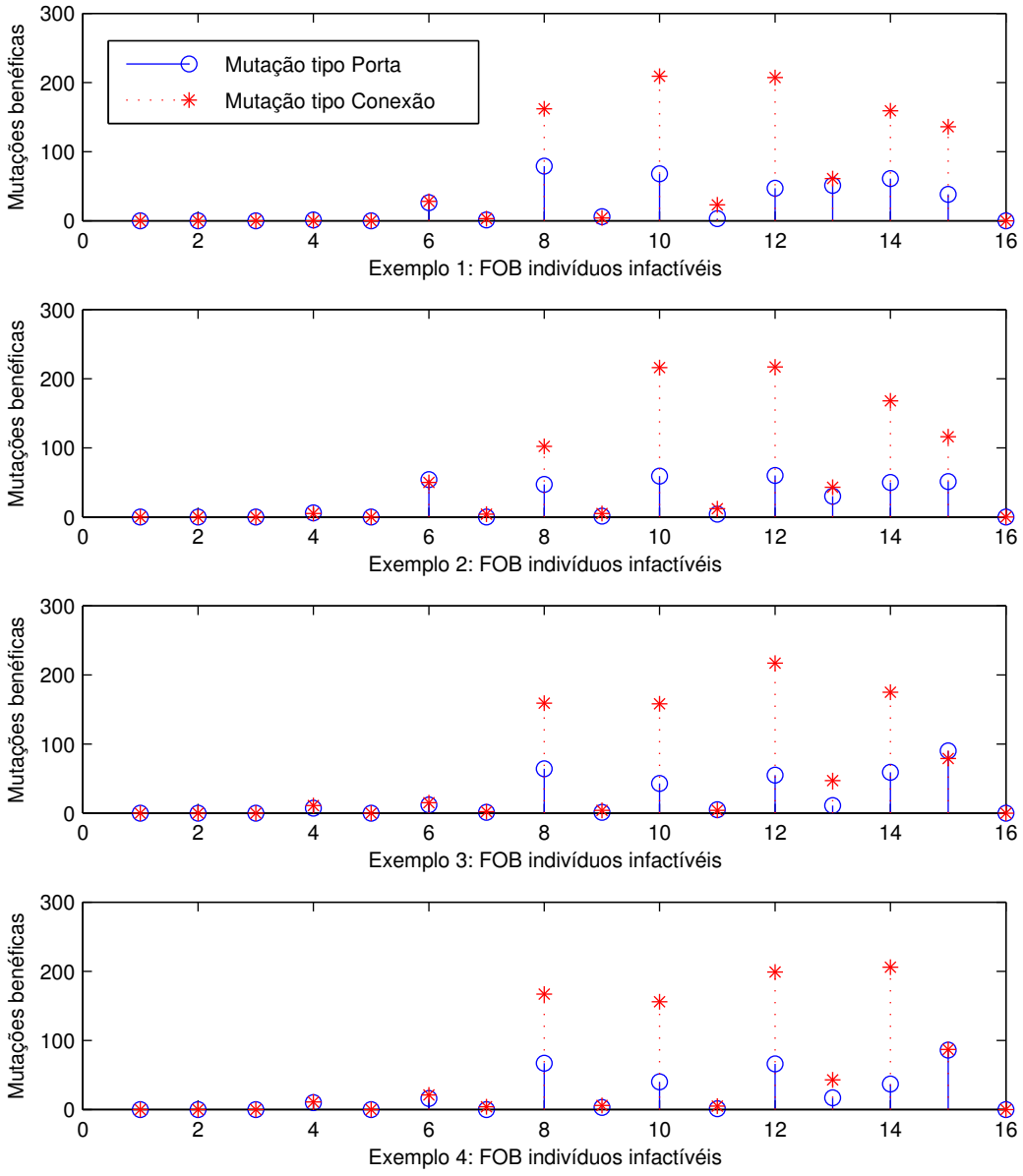


Figura 36 – Quantidade de mutações benéficas para cada tipo de mutação (porta e conexão) ao longo do processo evolutivo e na região infactível do espaço de busca.

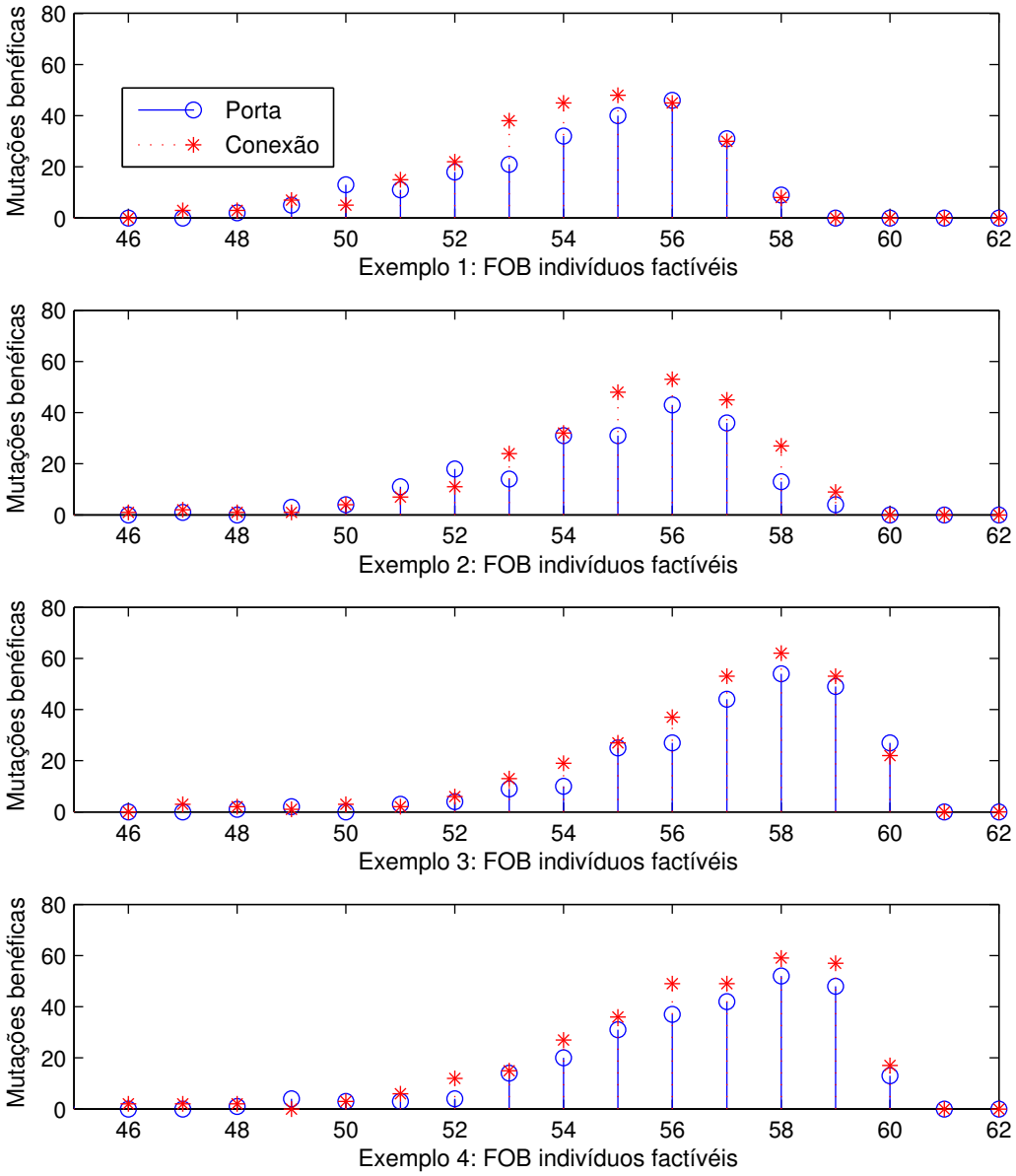


Figura 37 – Quantidade de mutações benéficas para cada tipo de mutação (porta e conexão) ao longo do processo evolutivo e na região factível do espaço de busca.

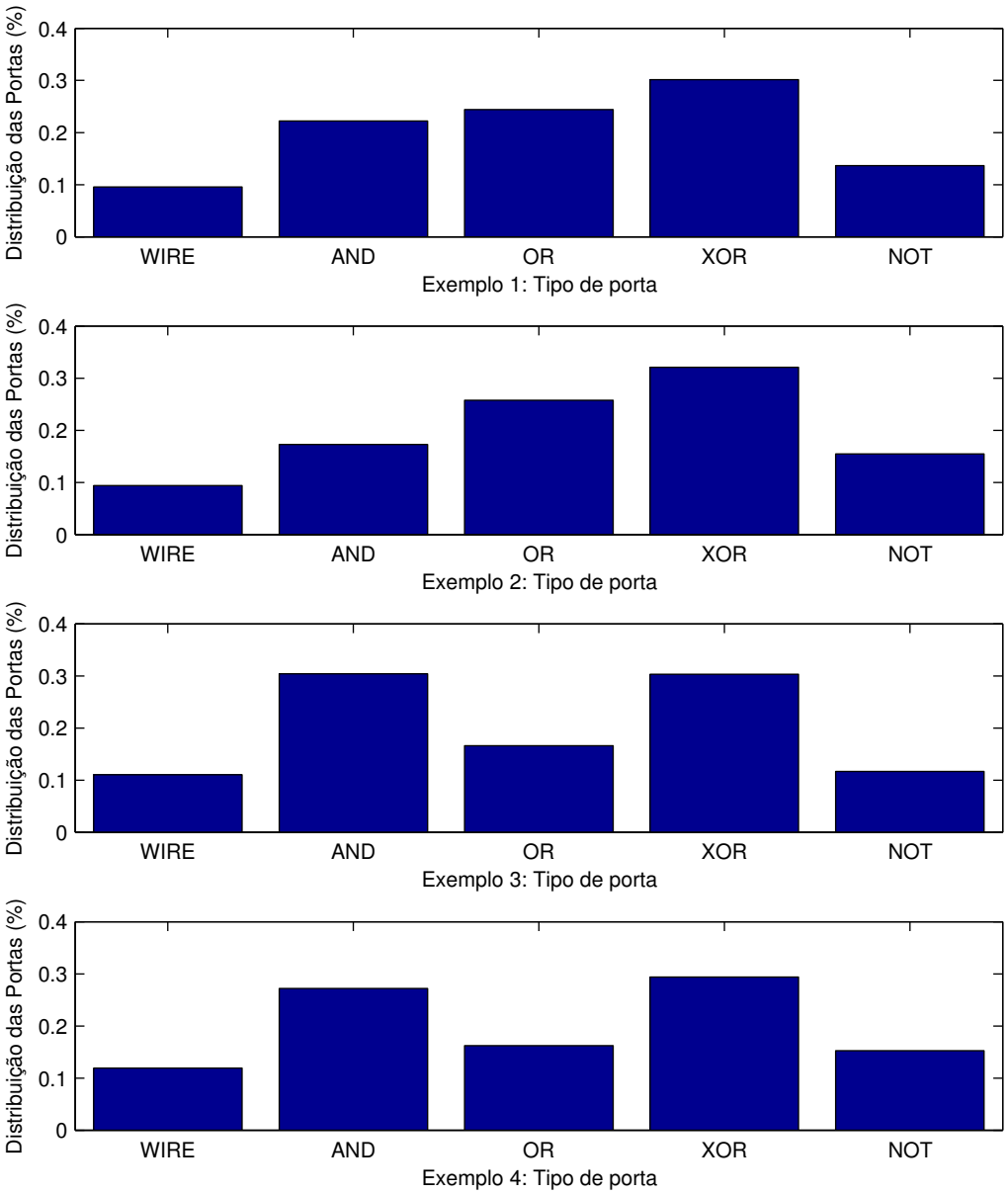


Figura 38 – Distribuição das portas lógicas no primeiro indivíduo factível.

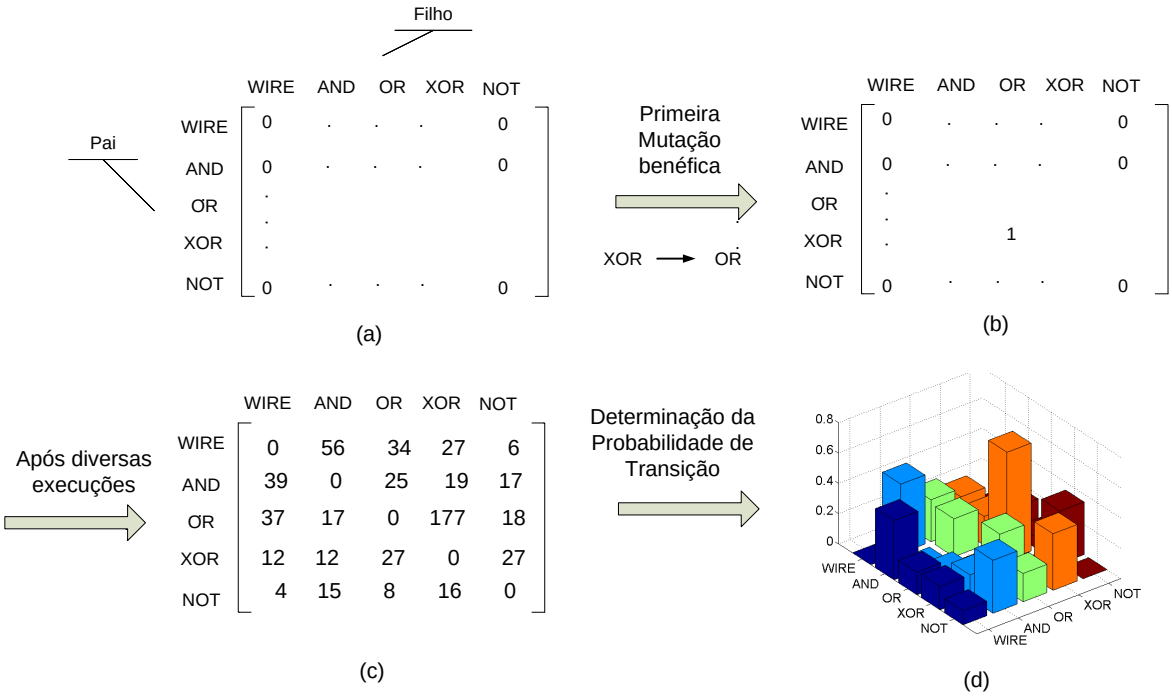


Figura 39 – Criação da Matriz Probabilidade de Transição. A cada ocorrência de uma mutação benéfica do tipo porta, a célula correspondente àquela transição é incrementada. A Matriz Probabilidade de Transição é então obtida ao final desse processo.

Na Figura 41, percebe-se as trocas entre os operadores que possuem 2 entradas (OR, AND e XOR) não contribui para o aumento na aptidão. Apenas a substituição de WIRES e raramente a entrada da porta NOT é benéfica para a região factível. Ressalta-se que os algoritmos de otimização da literatura fazem trocas entre os operadores sem levar em consideração essa informação.

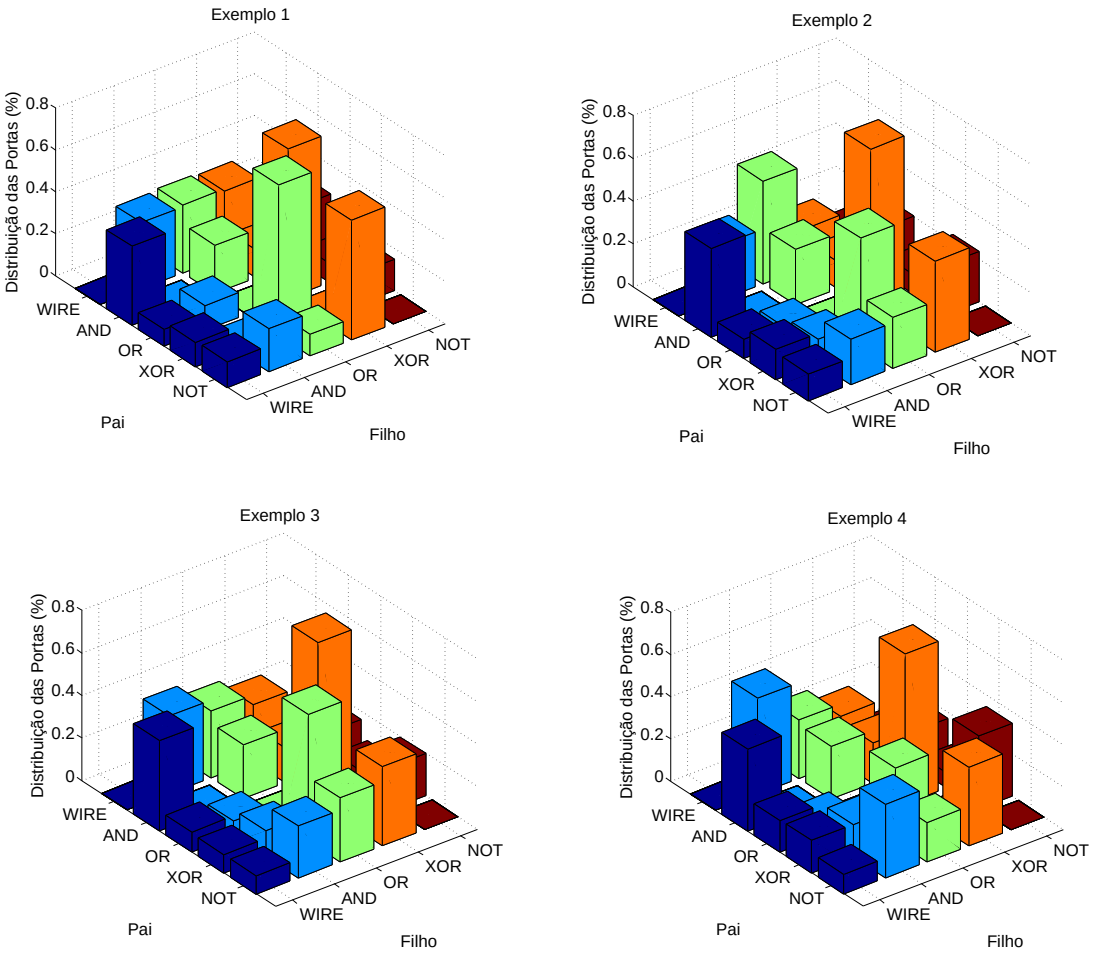


Figura 40 – Matriz Probabilidade de Transição - região infactível. Mostra a frequência de ocorrência das mutações benéficas tipo porta entre os indivíduos Pai e Filho.

5.2 HEURÍSTICA PARA A SEMEADURA DA POPULAÇÃO INICIAL

A tendência do aumento do número de nós ativos durante a busca por uma solução factível, apresentados nos resultados da Seção 5.1.3.2, motivou o desenvolvimento de uma nova heurística para semeadura da população inicial (MANFRINI; BERNARDINO; BARBOSA, 2016b).

5.2.1 **Tendência do crescimento de número de nós ativos**

Nos experimentos da Seção 5.1.3.2, em especial nos gráficos da Figura 35, os quais mostram a variação do número de nós ativos na região infactível, percebe-se que o processo evolutivo atuou de tal forma que o número de nós ativos aumentasse gradualmente ao longo da gerações. Esse comportamento pode ser justificado em razão do número de nós ativos da população inicial ser insuficiente para representar uma solução factível; diante disso o aumento da aptidão dos indivíduos estava correlacionada a um aumento no número de nós ativos.

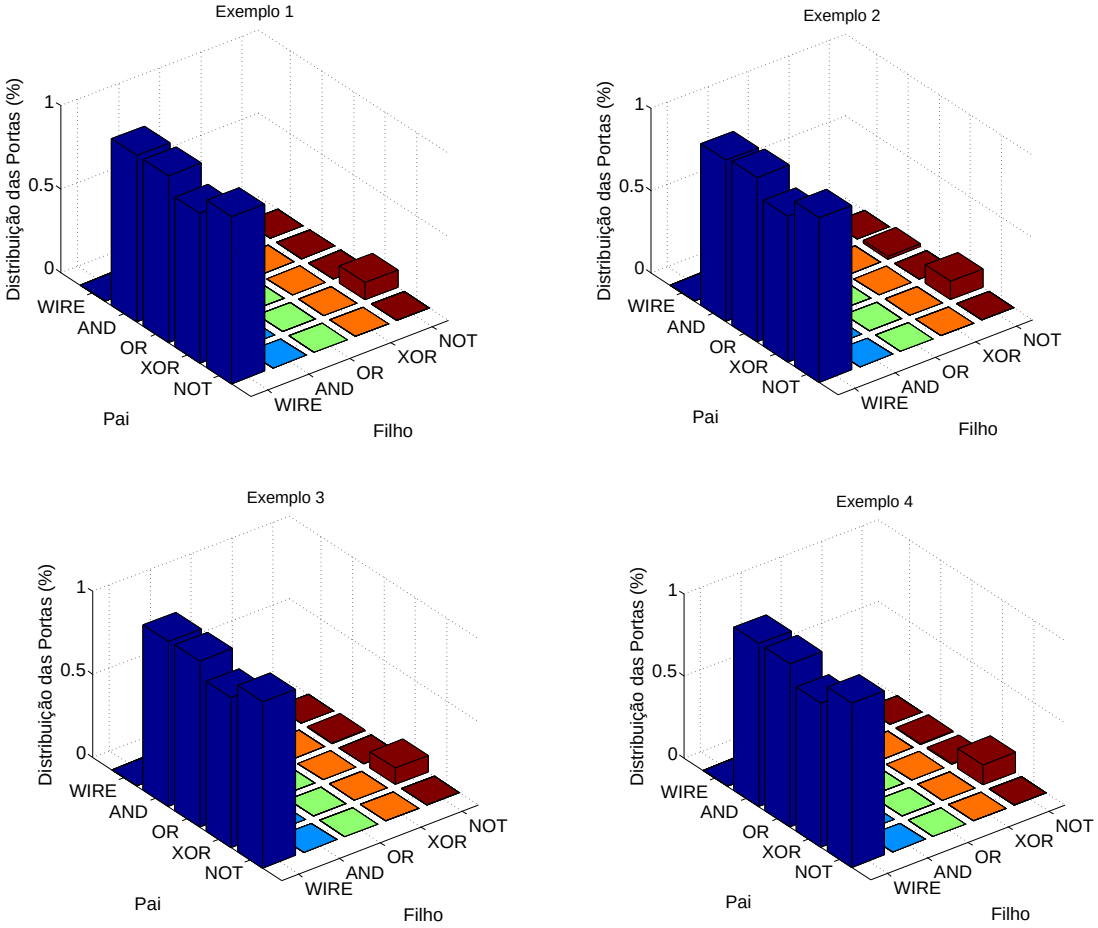


Figura 41 – Matriz Probabilidade de Transição - região factível. Mostra a frequência de ocorrência das mutações benéficas tipo porta entre os indivíduos Pai e Filho.

5.2.2 Influência do *levels-back*

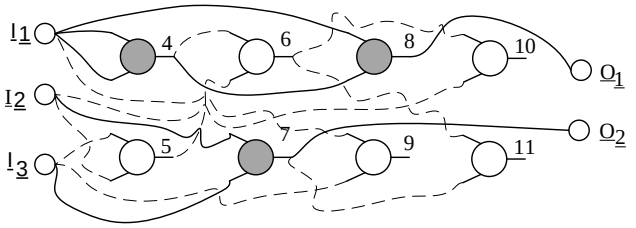
O *levels-back* influencia diretamente no número de nós ativos da semeadura. Comparando o simples exemplo da Figura 42, que representa um grafo onde os nós estão dispostos em uma matriz 2 x 4 tendo três entradas (I_1, I_2, I_3) e duas saídas (O_1, O_2), nota-se que é possível controlar a conectividade do grafo variando apenas lb.

Na Figura 42(a), o grafo foi gerado aleatoriamente com $lb = nc$, ocasionando um grafo com apenas três nós ligantes, modificando a taxa de retorno para $lb = 1$ tem-se um grafo com cinco nós ligantes, como mostra a Figura 42(b).

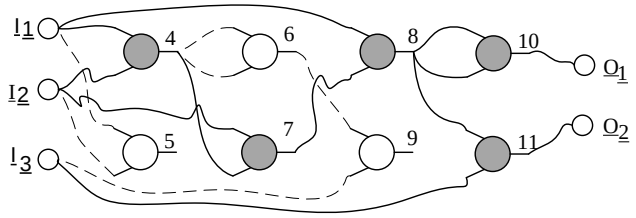
5.2.3 Descrição da heurística proposta

Apesar da literatura apontar que a população inicial deve ser gerada de forma mais genérica com, $n_r = 1$, $lb = n_c$, a análise da seção anterior indica que a semeadura da população inicial pode influenciar o desempenho do processo evolutivo.

A ideia principal da abordagem proposta consiste em inserir restrições na formação da população inicial. Através do ajuste dos três parâmetros que definem a representação



(a) Exemplo de grafo com *level-back = número de colunas*



(b) Exemplo de grafo com *level-back = 1*

Figura 42 – Variação da conectividade do grafo através do ajuste do *levels-back*

.

do indivíduo: numero de linhas n_r, número colunas n_c e *levels-back* lb é possível obter diversas conformações topológicas para a semeadura. Esses parâmetros devem ser ajustados para cada problema.

Ressalta-se que as modificações serão apenas na população inicial, e ao longo da evolução o indivíduo terá a mesma representação proposta por Miller (MILLER, 2011), ou seja: $lb = n_c$, dessa forma o espaço de busca não é modificado pela heurística proposta.

5.3 UM NOVO OPERADOR DE MUTAÇÃO APLICADO AO PROJETO DE CLCs

A abordagem proposta utiliza o conceito de aprendizagem supervisionado, no qual o procedimento usual para soluções de problemas, começa com uma fase denominada aprendizagem, em que a partir da análise de um conjunto de exemplos, é possível extrair características importantes, que posteriormente são utilizadas para gerar respostas para o problema. O aprendizado por reforço pode ser considerado um caso particular do aprendizado supervisionado, e funciona da seguinte forma: se uma determinada ação tomada pelo sistema é seguida de estados satisfatórios, então a tendência do sistema de produzir a ação em questão é reforçada (BRAGA; FERREIRA; LUDERMIR, 2007).

De uma maneira geral, os algoritmos de aprendizagem possuem um viés (*bias*) indutivo que está associado a uma preferência de uma hipótese sobre a outra, que não são igualmente prováveis (CUNHA; TAKAHASHI; ANTUNES, 2012).

A ideia da mutação proposta, consiste em analisar o comportamento do operador de mutação em um conjunto de problemas e construir a Matriz Probabilidade de Transi-

ção, conforme apresentada na seção 5.1.3.5. Essa matriz é então utilizada para direcionar a atuação do operador de mutação em outros problemas.

5.3.1 **Descrição do novo operador de mutação proposto**

Tradicionalmente o operador de mutação utilizado na CGP é um operador de mutação pontual onde as mudanças no cromossomo ocorrem de forma aleatória e a variável *pm* determina o número de genes que serão submetidos à mutação. Goldman e Punch (GOLDMAN; PUNCH, 2015) propuseram uma estratégia para evitar desperdício de avaliações, denominada *Single Active Mutation* SAM. Nessa estratégia os genes são mutados até que um nó ativo seja trocado; dessa forma garante-se que todos os filhos tenham fenótipo diferente dos pais (detalhes na Seção 4.3.1.1).

Quando a CGP representa um CLCs a mutação pode ser de dois tipos: *Tipo porta* quando um alelo que define a função lógica (porta) que o gene executa é selecionado ou *Tipo conexão* quando um alelo que fornece a ligação do gene é selecionado, conforme descrito na Seção 5.1.3.3. O operador proposto utiliza a mutação SAM com a inclusão de um viés capaz de guiar a mutação. A ideia é analisar o comportamento do genótipo durante o processo evolucionário, para um dado conjunto de problemas, e construir a Matriz Probabilidade de Transição (ver Seção 5.1.3.5). Essa Matriz é utilizada para direcionar o operador de mutação em outros problemas. Toda vez que ocorre uma mutação tipo porta em um nó ativo, o novo valor da porta é escolhido de acordo com a Matriz Probabilidade de Transição previamente determinada. Esse novo operador de mutação proposto é denominado *Biased SAM* (MANFRINI; BERNARDINO; BARBOSA, 2016a).

A Figura 43 ilustra um exemplo da atuação do operador *Biased SAM*, onde o alelo que determina a função do nó 6 (AND) foi selecionado aleatoriamente para ser submetido à mutação. O novo valor do alelo (NOR) foi escolhido em uma roleta onde a probabilidade de cada porta é determinada pela Matriz Probabilidade de Transição.

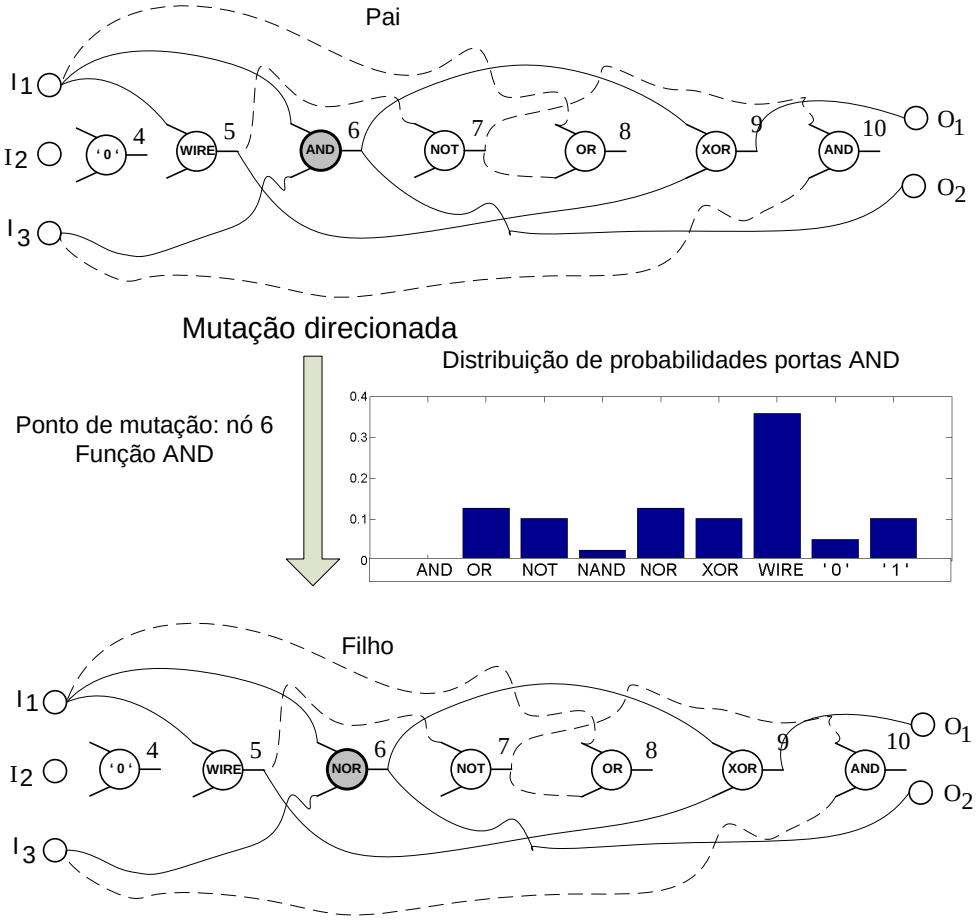


Figura 43 – Exemplo de aplicação do operador *Biased SAM*. Quando o nó 6 é selecionado para ser submetido à mutação, a troca da função AND ocorre através de uma roleta definida pela Matriz de Probabilidade de Transição.

6 EXPERIMENTOS COMPUTACIONAIS

Os experimentos computacionais que visam analisar o desempenho da Heurística da semeadura, bem como do operador de mutação denominado *Biased SAM*, são descritos neste capítulo.

6.1 EXPERIMENTOS - HEURÍSTICA PARA A SEMEADURA DA POPULAÇÃO

Para verificar a eficiência da heurística para a semeadura da população (MANFRINI; BERNARDINO; BARBOSA, 2016b), descrita em detalhes na Seção 5.2, quatro exemplos *benchmark* estudados por (GOLDMAN; PUNCH, 2015) e extensamente utilizados na literatura de circuitos eletrônicos (TOCCI, 2011; ERCEGOVAC; MORENO; LANG, 1998) foram escolhidos. De acordo com Goldman e Punch (GOLDMAN; PUNCH, 2013), esses problemas são úteis porque podem ser resolvidos com frequência, ou seja uma solução factível é encontrada na maioria das execuções. Isso permite uma análise estatística dos resultados. Posteriormente um circuito considerado complexo pela literatura (HRBACEK; SEKANINA, 2014) é apresentado no Exemplo 5.

Inicialmente, o conjunto de operadores lógicos permitidos foram Γ_1 ={AND, OR, NAND, NOR}. De acordo com Goldman e Punch, a exclusão da porta XOR aumenta a complexidade do problema (GOLDMAN; PUNCH, 2013; WALKER; MILLER, 2008). Em seguida outro cenário foi analisado com a inclusão de mais uma restrição: a população foi gerada utilizando apenas portas NAND, Γ_2 ={NAND}. As portas NAND's são consideradas portas universais e podem ser utilizadas para gerar as expressões contendo OR, AND e NOT. Essa característica é muito utilizada no projeto de circuitos lógicos (TOCCI, 2011) e por serem portas internamente simples, os circuitos gerados possuem melhores características (ERCEGOVAC; MORENO; LANG, 1998; KARAKATIC; PODGORELEC; HERICKO, 2013).

Os experimentos foram implementados em MATLAB[1], e para as análises estatísticas foi utilizado o *software* SPSS[2]. Para a comparação dos diversos cenários foram medidas as seguintes grandezas:

- **Hits**: Porcentagem do número de vezes que cada cenário encontrou uma solução factível.
- **MES** *(Median Evaluations to Success)*: Mediana do número de avaliações necessárias para encontrar uma solução factível.
- **Nós Ativos da Semeadura**: Mediana do número de nós ativos da semeadura.

[1] http://www.mathworks.com
[2] http://www-03.ibm.com/software/products/en/spss-statistics

- **Nós Ativos da Solução**: Mediana do número de nós ativos quando uma solução factível é encontrada.

Para cada um dos exemplos analisados foram utilizados $\mu = 1$, $\lambda = 4$ e a mutação *SAM* conforme utilizado por Goldman (GOLDMAN; PUNCH, 2015), sendo realizadas 51 execuções independentes. O algoritmo termina quando o máximo número de avaliações é atingido ou quando uma solução factível é encontrada. Essa abordagem tem como objetivo encontrar um circuito que atenda as restrições da tabela verdade, conforme descrito na Seção 2.3.1.1, sendo o valor da aptidão proporcional às restrições atendidas, ou seja, o número de linhas da tabela verdade que a solução candidata satisfaz.

Para todos os experimentos as diferentes semeaduras foram obtidas variando os parâmetros da seguinte forma: $\Gamma_{nr} = \{1, 2, 5\}$; $\Gamma_{lb} = \{n_c, \frac{n_c}{2}, \frac{n_c}{4}, \}$ ou simplesmente $\Gamma_{lb} = \{100\%, 50\%, 25\%\}$; onde n_c é o número de colunas e os conjuntos Γ_{nr} e Γ_{lb} representam os valores atribuídos às variáveis *número de linhas* (n_r) e *levels-back* (lb), respectivamente. O conjunto de operadores permitidos foram Γ_1 ={OR, AND, NAND, NOR} ou Γ_2 ={NAND}.

Desta forma, foram criados dezoito cenários diferentes para a criação da população inicial e o p-valor compara os resultados encontrados com o controle via Mann-Whitney U teste. Ressalta-se que o controle possui os seguintes parâmetros: $n_r = 1$, $lb = n_c$. Significância estatística ($p < 0.05$) é denotada por $*$.

6.1.1 Análise da evolução

A análise do comportamento da topologia ao longo da evolução será feita de acordo com o número de nós ativos associado a cada *fitness*. A Figura 44 mostra um simples exemplo dessa abordagem: Considerando que a CGP tem 6 rodadas independentes para resolver um problema de maximização, onde a função objetivo pode assumir valores inteiros entre $[1, 5]$. O algoritmo termina quando uma solução factível é atingida, isto é, quando *fitness* $= 5$.

Para cada rodada, toda vez que o *fitness* do melhor filho é maior que o *fitness* do progenitor, ou seja, quando ocorre uma mutação benéfica, então o número de nós ativos do filho é armazenado em um vetor, conforme ilustra a Figura 44.

Analisando os valores da primeira rodada, o indivíduo começou com *fitness* $= 1$ e 12 nós ativos, após uma mutação benéfica esse indivíduo foi substituído por outro com *fitness* $= 3$ e 23 nós ativos. Nota-se que na primeira rodada não houve indivíduos com *fitness* $= 2$, isso explica o motivo de várias células em branco na Figura 44.

Os dados de todas as rodadas foram agrupados de acordo com o valor do *fitness*. Por exemplo, o vetor $F_3 = [23\ 13\ 10\ 14\ 20]$, cuja mediana é 14, representa o número de nós ativos quando *fitness* $= 3$, conforme ilustra a Figura 44.

Em todos os exemplos a mediana desse vetor foi escolhida como parâmetro para analisar a variação do número de nós ativos dos indivíduos durante a busca, fornecendo assim informações sobre o comportamento da topologia.

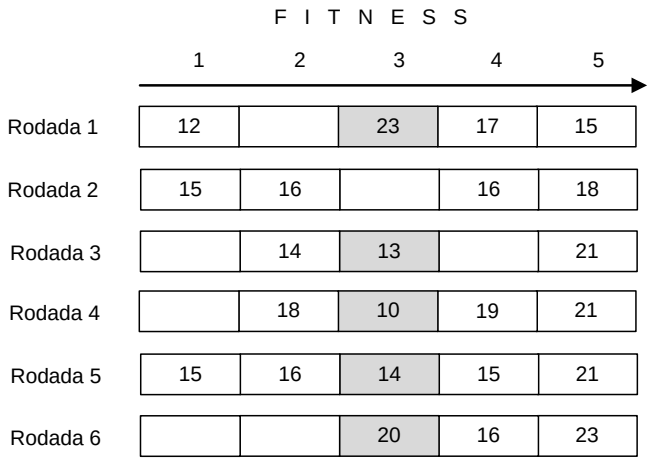


Figura 44 – Diagrama ilustrando a metodologia adotada para analizar o comportamento da topologia. Cada célula representa o número de nós ativos relacionado ao *fitness*.

6.1.2 **Exemplo 1**

O primeiro exemplo é um circuito detector de paridade de 3-Bits, considerado simples, mas que representa uma aplicação muito comum da CGP (MILLER; SMITH, 2006; WALKER; MILLER, 2008; YU; MILLER, 2001; GOLDMAN; PUNCH, 2015) e pode contribuir para o entendimento de como a semeadura influencia no processo de busca. Todas as execuções foram limitadas em 5000 avaliações. As configurações da semeadura, assim como os resultados de todos os cenários analisados estão apresentados na Tabela 3. A primeira linha é o controle de acordo com Goldman (GOLDMAN; PUNCH, 2015).

Pode-se observar que a topologia formada por portas NAND e *levels-back* = 50% ou *levels-back* = 25% converge para uma solução factível com um menor número de avaliações. O melhor resultado foi encontrado com MES = 341 enquanto o controle tem MES = 1421.

Analisando a variação do número de nós ativos conforme a metodologia apresentada na Seção 6.1.1 foi plotada a mediana do número de nós ativos em função da aptidão para cada semeadura. Neste exemplo, uma solução factível ocorre quando $Fitness = 8$. Os vários cenários mostrados na Tabela 3 foram separados de acordo com a topologia dando origem a vários sub-gráficos, conforme a Figura 45.

Nota-se que para cada topologia, o comportamento do número de nós ativos foi diferente, de acordo com o valor do *levels-back*. Verifica-se, ainda, que o comportamento dos vários sub-gráficos apresentam uma similaridade entre si, conforme o valor do *levels-back*.

Em cada sub-grafo, os símbolos $\square, \circ, *$, denotam a mediana do número de nós ativos da semeadura da população para os valores de *levels-back* 25%, 50% e 100% respectivamente.

6.1.3 **Exemplo 2**

O segundo exemplo é um codificador 16-4 bit proposto em (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015). Todas as execuções são limitadas a 150000 avaliações. As configurações da semeadura, assim como os resultados de todos os cenários analisados estão apresentados na Tabela 4. A primeira linha é o controle de acordo com Goldman (GOLDMAN; PUNCH, 2015).

Pode-se observar que a topologia formada com portas NAND e *levels-back* = 25% converge para uma solução factível com um menor número de avaliações. O melhor resultado foi encontrado com MES = 16341 enquanto o controle tem MES = 26445.

Analisando a variação do número de nós ativos conforme a metodologia apresentada na Seção 6.1.1 foi plotada a mediana do número nós de ativos em função da aptidão para cada semeadura. Neste exemplo, uma solução factível ocorre quando $Fitness = 64$.

Os vários cenários mostrados na Tabela 4 foram separados de acordo com a topologia e são mostrados na Figura 46. Nota-se, em cada sub-gráfico, que o comportamento do número de nós ativos foi diferente de acordo com *levels-back* e o comportamento dos vários sub-gráficos são similares.

Em cada sub-grafo, os símbolos $\square, \circ, *$, denotam a mediana do número de nós ativos da semeadura da população para os valores de *levels-back* 25%, 50% e 100% respectivamente.

6.1.4 **Exemplo 3**

O terceiro exemplo é um decodificador 4-16 bit proposto em (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015). Todas as execuções são limitadas a 150000 avaliações. As configurações da semeadura, assim como os resultados de todos os cenários analisados estão apresentados na Tabela 5. A primeira linha é o controle de acordo com Goldman (GOLDMAN; PUNCH, 2015).

Pode-se observar que a topologia 5 x 200 formada pelo conjunto Γ_1 e *levels-back* = 50% e a topologia 5 x 200 formada por Γ_2 e *levels-back* = 100% convergem para uma solução factível com um menor número de avaliações. O melhor resultado foi encontrado com MES = 55973 enquanto o controle tem MES = 69851.

Analisando a variação do número de nós ativos conforme a metodologia apresentada na Seção 6.1.1 foi plotada a mediana do número de nós ativos em função da aptidão para cada semeadura. Neste exemplo uma solução factível ocorre quando $Fitness = 256$.

Os vários cenários mostrados na Tabela 5 foram separados de acordo com o número de linhas da topologia e são mostrados na Figura 47. Nota-se que o comportamento número de nós ativos foi diferente para cada valor de *levels-back* e o comportamento dos vários sub-gráficos são similares para cada valor de *levels-back*.

Em cada sub-grafo, os símbolos $\square, \circ, *$, denotam a mediana do número de nós ativos da semeadura da população para os valores de *levels-back* 25%, 50% e 100% respectivamente.

6.1.5 **Exemplo 4**

O quarto exemplo é um multiplicador de 3-Bits proposto em (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015). Todas as execuções são limitadas a 1100000 avaliações. As configurações da semeadura, assim como os resultados de todos os cenários analisados estão apresentados na Tabela 6. A primeira linha é o controle de acordo com Goldman (GOLDMAN; PUNCH, 2015).

Pode-se observar que a topologia 1 x 5000 formada pelo conjunto Γ_1 e *levels-back* = 50% e a topologia 2 x 2500 Γ_1 e *levels-back* = 25% e a topologia 5 x 1000 Γ_1 e *levels-back* = 50% e todas as topologias formadas por portas NAND, Γ_2 convergem para uma solução factível com um menor número de avaliações. O melhor resultado foi encontrado com MES = 118147 enquanto o controle tem MES = 458697.

Analisando a variação do número de nós ativos conforme a metodologia apresentada na Seção 6.1.1 foi plotada a mediana do número de nós ativos em função da aptidão para cada semeadura. Neste exemplo uma solução factível ocorre quando $Fitness = 384$. Os vários cenários mostrados na Tabela 6 foram separados de acordo com o número de linhas da topologia e são mostrados na Figura 48. Nota-se que o comportamento do número de nós ativos foi diferente para cada valor de *levels-back* e o comportamento dos vários sub-gráficos são similares para cada valor de *levels-back*.

Em cada sub-grafo, os símbolos $\square, \circ, *$, denotam a mediana do número de nós ativos da semeadura da população para os valores de *levels-back* 25%, 50% e 100% respectivamente.

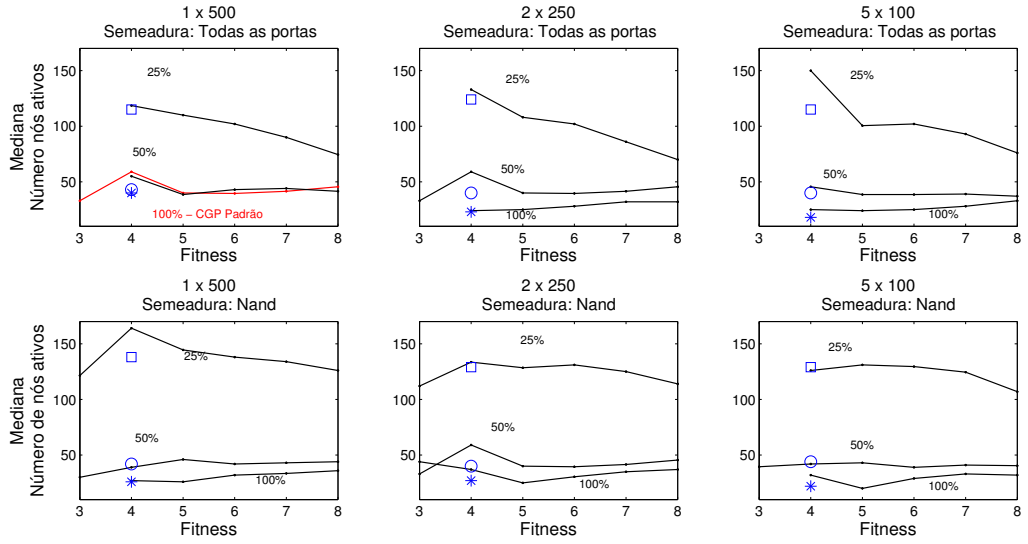


Figura 45 – Variação do número de nós ativos em função da aptidão para o exemplo 1. Cada sub-grafo representa uma topologia com os *levels-back* de 25%, 50% e 100%. Os símbolos □ ○ ∗ denotam a mediana do número de nós ativos da semeadura para cada *levels-back*.

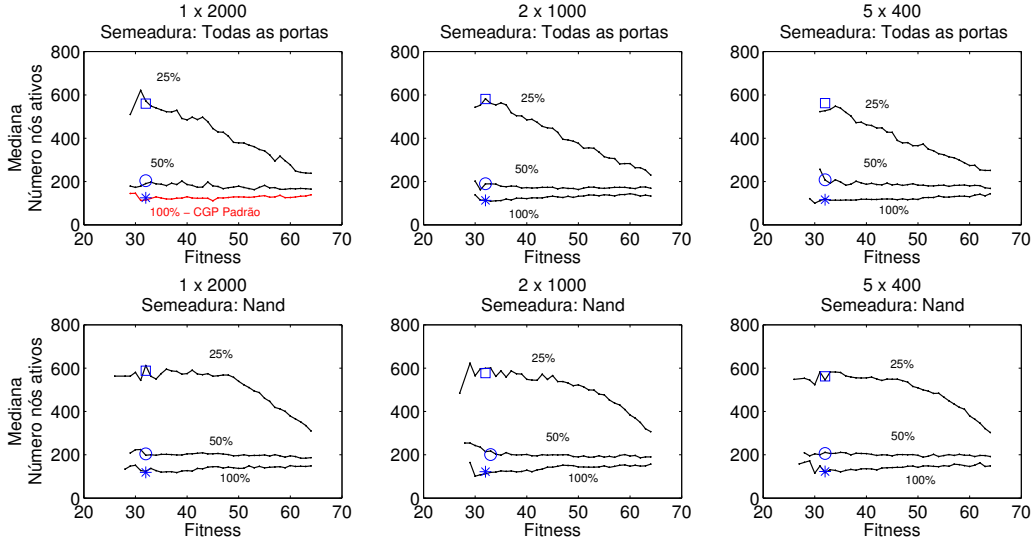


Figura 46 – Variação do número de nós ativos em função da aptidão para o exemplo 2. Cada sub-grafo representa uma topologia com os *levels-back* de 25%, 50% e 100%. Os símbolos □ ○ ∗ denotam a mediana do número de nós ativos da semeadura para cada *levels-back*.

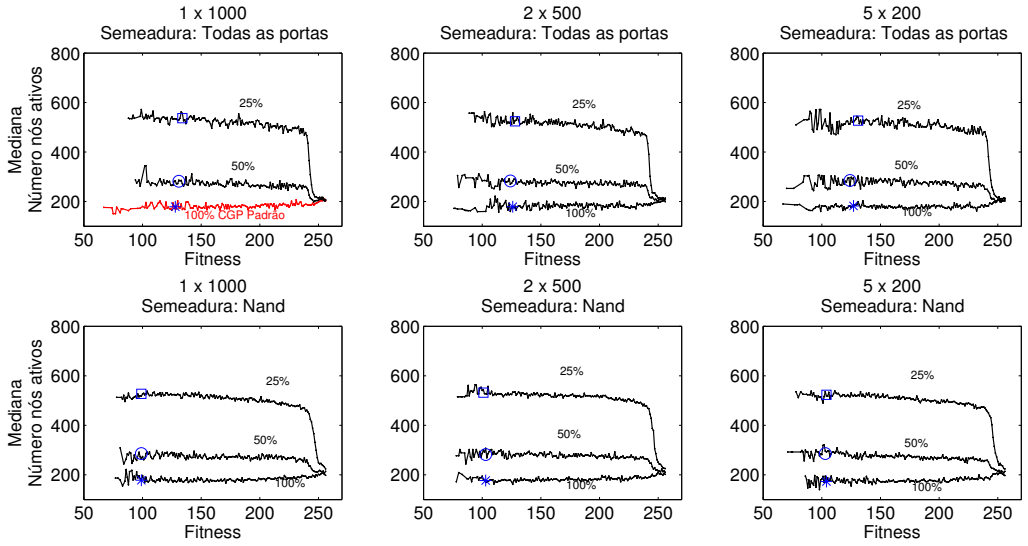


Figura 47 – Variação do número de nós ativos em função da aptidão para o exemplo 3. Cada sub-grafo representa uma topologia com os *levels-back* de 25%, 50% e 100%. Os símbolos □ ○ ∗ denotam a mediana do número de nós ativos da semeadura para cada *levels-back*.

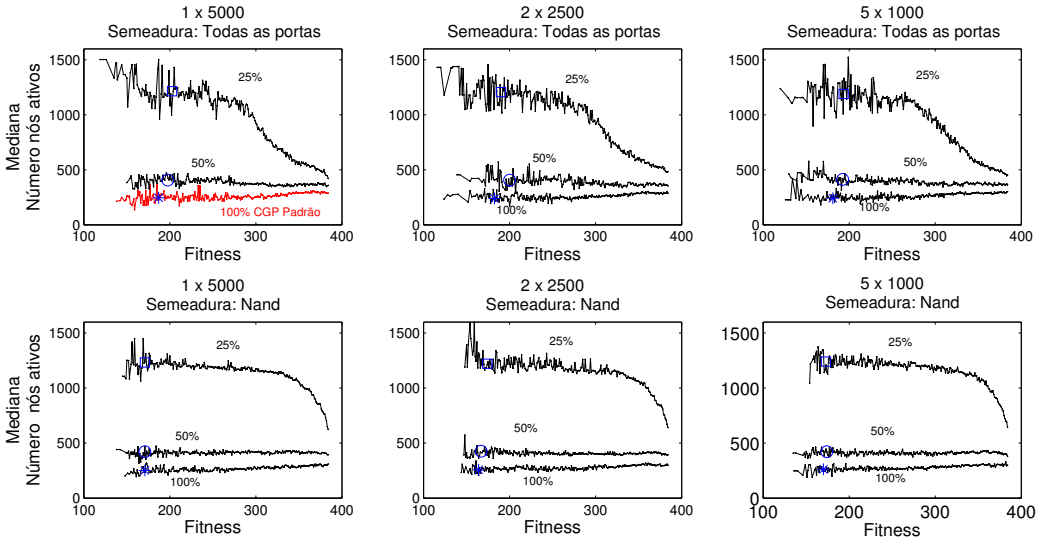


Figura 48 – Variação do número de nós ativos em função da aptidão para o exemplo 4. Cada sub-grafo representa uma topologia com os *levels-back* de 25%, 50% e 100%. Os símbolos □ ○ ∗ denotam a mediana do número de nós ativos da semeadura para cada *levels-back*.

Tabela 3 – Resultados para as diversas conformações topológicas do Exemplo 1, onde Γ_1 = {OR, AND, NAND, NOR} e Γ_2 = {NAND}. **Hits** é a porcentagem do número de vezes que cada cenário encontrou uma solução factível. **MES** é a mediana do número de avaliações necessárias para encontrar uma solução factível. O p-valor compara o controle (linha destacada) com as diversas semeaduras através do teste Mann-Whitney (U). Os resultados com significância estatística ($p < 0.05$) são denotados por $*$.

	Topologia	**Levels Back(%)**	**Portas**	**Hits (%)**	**MES**	**Intervalo Confiança**	**Nós Ativos Semeadura**	**Nós Ativos Solução**	***p* Valor**
1		100		86	1421	953 .. 1689	26	36	NA
2	1 x 500	50	Γ_1	94	1143	793 .. 1549	43	40	0.272
3		25		98	1391	1117 .. 1873	115	74	0.928
4		100		96	1473	1269 .. 1825	23	32	0.899
5	2 x 250	50	Γ_1	100	897	749 .. 1369	39	41	0.140
6		25		96	1049	821 ..1485	124	70	0.501
7		100		92	1157	793 .. 1401	18	33	0.227
8	5 x 100	50	Γ_1	98	1107	791 .. 1357	40	37	0.180
9		25		100	1077	785 .. 1437	115	76	0.286
10		100		100	1081	913 .. 1493	27	36	0.180
11	1 x 500	50	Γ_2	92	545	409 .. 857	42	43	0 $*$
12		25		98	**341**	217 .. 469	138	126	0 $*$
13		100		98	907	689 .. 1321	27	37	0.090
14	2 x 250	50	Γ_2	98	639	373 .. 1213	40	45	0.003 $*$
15		25		100	557	321 .. 673	129	114	0 $*$
16		100		98	957	673 .. 1301	22	32	0.130
17	5 x 100	50	Γ_2	98	665	453 .. 1145	44	41	0.001 $*$
18		25		100	389	285 .. 473	129	107	0 $*$

Tabela 4 – Resultados para as diversas conformações topológicas do Exemplo 2, onde Γ_1 = {OR, AND, NAND, NOR} e Γ_2 = {NAND}. **Hits** é a porcentagem do número de vezes que cada cenário encontrou uma solução factível. **MES** é a mediana do número de avaliações necessárias para encontrar uma solução factível. O p-valor compara o controle (linha destacada) com as diversas semeaduras através do teste Mann-Whitney (U). Os resultados com significância estatística ($p < 0.05$) são denotados por $*$.

	Topologia	**Levels Back(%)**	**Porta**	**Hits (%)**	**MES**	**Intervalo Confiança**	**Nós Ativos Semeadura**	**Nós Ativos Solução**	***p* Valor**
1		100		100	26445	18525 .. 34677	125	139	NA
2	1 x 2000	50	Γ_1	100	24721	19613 .. 27445	204	187	0.311
3		25		96	31361	21505 .. 33581	560	233	0.801
4		100		100	25629	21921 .. 31937	113	133	0.971
5	2 x 1000	50	Γ_1	100	20801	17497 .. 24009	190	169	0.155
6		25		98	22873	17733 .. 30413	581	229	0.411
7		100		100	23733	19253 .. 30689	116	144	0.337
8	5 x 400	50	Γ_1	100	25153	21229 .. 33669	208	168	0.595
9		25		98	23695	18109 .. 32429	562	249	0.407
10		100		100	28865	22329 ..37673	119	149	0.647
11	1 x 2000	50	Γ_2	100	24721	19613 ..27445	204	187	0.311
12		25		100	**16341**	13361 ..19677	588	310	0 $*$
13		100		100	34457	23621 ..39513	122	157	0.156
14	2 x 1000	50	Γ_2	98	25831	21361 ..35853	201	190	0.739
15		25		100	19153	14537 ..22445	577	307	0.015 $*$
16		100		100	24817	21361 ..30893	123	149	0.981
17	5 x 400	50	Γ_2	100	23761	18469 .. 30093	206	192	0.283
18		25		100	18849	14921 .. 21033	562	302	0.006 $*$

Tabela 5 – Resultados para as diversas conformações topológicas do Exemplo 3, onde Γ_1 = {OR, AND, NAND, NOR} e Γ_2 = {NAND}. **Hits** é a porcentagem do número de vezes que cada cenário encontrou uma solução factível. **MES** é a mediana do número de avaliações necessárias para encontrar uma solução factível. O p-valor compara o controle (linha destacada) com as diversas semeaduras através do teste Mann-Whitney (U). Os resultados com significância estatística ($p < 0.05$) são denotados por $*$

	Topologia	**Levels Back(%)**	**Portas**	**Hits (%)**	**MES**	**Intervalo Confiança**	**Nós Ativos Semeadura**	**Nós Ativos Solução**	***p* Valor**
1		100		98	69851	60573 .. 76537	179	204	NA
2	1 x 1000	50	Γ_1	98	66993	57221 .. 78217	282	205	0.833
3		25		100	63537	57485 ..73097	537	208	0.478
4		100		98	73175	67097..81869	179	200	0.356
5	2 x 500	50	Γ_1	98	61905	53065 .. 81901	283	214	0.667
6		25		100	65025	53745 ..68829	524	208	0.363
7		100		98	59957	51577 .. 66865	182	202	0.095
8	5 x 200	50	Γ_1	98	56635	49329 .. 68373	285	210	0.042$*$
9		25		100	64549	59485 ..69697	527	213	0.439
10		100		94	68621	58637 .. 80053	178	198	0.749
11	1 x 1000	50	Γ_2	96	70549	58813 .. 84393	286	214	0.580
12		25		100	64681	60921 ..77545	527	224	0.823
13		100		98	67575	61509 .. 74589	177	202	0.899
14	2 x 500	50	Γ_2	100	64573	56661 .. 78897	282	210	0.555
15		25		98	70905	61517 .. 80461	532	226	0.664
16		100		98	**55973**	47773 .. 68709	173	197	0.047 $*$
17	5 x 200	50	Γ_2	98	65791	56973 .. 80733	286	214	0.945
18		25		100	72269	58133 ..81445	523	224	0.649

Tabela 6 – Resultados para as diversas conformações topológicas do Exemplo 4, onde Γ_1 = {OR, AND, NAND, NOR} e Γ_2 = {NAND}. **Hits** é a porcentagem do número de vezes que cada cenário encontrou uma solução factível. **MES** é a mediana do número de avaliações necessárias para encontrar uma solução factível. O p-valor compara o controle (linha destacada) com as diversas semeaduras através do teste Mann-Whitney (U). Os resultados com significância estatística ($p < 0.05$) são denotados por $*$.

	Topologia	**Levels Back(%)**	**Portas**	**Hits (%)**	**MES**	**Intervalo Confiança**	**Nós Ativos Semeadura**	**Nós Ativos Solução**	***p* Valor**
1		100		86	458697	333029 .. 555413	246	292	NA
2	1 x 5000	50	Γ_1	90	288481	231209 .. 335893	407	358	0.016$*$
3		25		86	392473	289353 .. 465717	1214	414	0.212
4		100		92	372781	285765 .. 479721	239	296	0.166
5	2 x 2500	50	Γ_1	92	338975	303385 .. 423281	407	361	0.097
6		25		88	276645	219425 .. 372065	1207	478	0.01$*$
7		100		88	395071	268437 .. 474533	242	294	0.269
8	5 x 1000	50	Γ_1	92	262407	211513 .. 389801	413	352	0.002$*$
9		25		90	389089	324265 .. 438153	1189	446	0.103
10		100		86	356599	230897 .. 427333	252	308	0.035$*$
11	1 x 5000	50	Γ_2	98	191545	160433 .. 293529	422	394	0$*$
12		25		96	148241	119389 .. 232745	1233	624	0$*$
13		100		88	281465	225017 .. 448297	257	289	0.015$*$
14	2 x 2500	50	Γ_2	96	177849	139421 .. 221721	426	393	0$*$
15		25		98	**118147**	88965 .. 155857	1221	642	0$*$
16		100		88	238005	158869 .. 401913	262	280	0.002$*$
17	5 x 1000	50	Γ_2	100	168005	137137 .. 257173	422	379	0$*$
18		25		94	127449	93561 .. 156653	1241	643	0$*$

6.1.6 **Exemplo 5**

Para verificar a eficiência da heurística da semeadura em problemas maiores, foram projetados multiplicadores 4x4. Os circuitos multiplicadores nxn possuem $2n$ saídas, gerando assim $2 \times n \times 2^{2 \times n}$ restrições. No caso particular para $n = 4$ tem-se 2048 restrições a serem atendidas.

O projeto de circuitos multiplicadores 4x4 é apontado pela literatura como uma tarefa muito complexa (VASSILEV; JOB; MILLER, 2000; STOMEO; KALGANOVA; LAMBERT, 2006; HRBACEK; SEKANINA, 2014). Até 2014 esse problema havia sido resolvido somente por Miller *et al.* (VASSILEV; JOB; MILLER, 2000; MILLER, 2011), onde a CGP foi executada apenas uma vez com 700 milhões de chamadas da função de avaliação. Hrbacek e Sekanina utilizaram computação massivamente paralela e conseguiram projetar multiplicadores 4x4, bem como multiplicadores 5x5 (HRBACEK; SEKANINA, 2014).

Para os testes comparativos da heurística da semeadura, utilizou-se a mesma quantidade de nós e o mesmo conjunto de portas lógicas $\Gamma_1 = \{\text{OR, AND, NAND, XOR}\}$ que Hrbacek e Sekanina (HRBACEK; SEKANINA, 2014).

Foram feitas 5 execuções independentes inicializadas conforme o padrão da literatura, ou seja, $n_r = 1$, $lb = n_c$, e 5 execuções inicializadas de acordo com a heurística da semeadura, onde adotou-se $n_r = 5$, $lb = 25\%$, $\Gamma_2 = \{\text{NAND}\}$. Ressalta-se que as modificações ocorrem apenas na população inicial, conforme descrito na seção 5.2. Em todos as execuções foram permitidas até 50000000 avaliações.

Os experimentos computacionais foram feitos no *software* MATLAB 2014 utilizando um computador *desktop* com sistema operacional Windows 7 processador AMD Phenom(tm) IIX4 B95 de 3GHz e com 8GB de memória RAM. Durante os experimentos, verificou-se que 1000000 avaliações da função objetivo demoram cerca de 63 minutos para a população semeada da forma tradicional e 70 minutos quando se utiliza a heurística da semeadura. A Tabela 7 apresenta os resultados obtidos.

Analisando a Tabela 7 verifica-se que quando a Heurística da semeadura foi aplicada para projetar um circuito multiplicador 4x4 um resultado factível foi obtido em 4 das 5 execuções e a média das avaliações necessárias para se obter o sucesso foi 27319065. Quando a inicialização da população ocorreu conforme a literatura, apenas uma execução entre as 5 realizadas foi factível, sendo que nessa execução foram necessárias 36415029 avaliações da função objetivo.

6.1.7 **Discussão dos Resultados**

A análise da heurística proposta pode ser feita sob diferentes pontos de vista. Inicialmente será considerado os dados presentes nas respectivas Tabelas 3, 4, 5, 6 e

Tabela 7 – Resultado obtido para o projeto do circuito multiplicador 4x4. As 5 primeiras execuções foram feitas de acordo com o padrão da literatura, as demais representam o desempenho do heurístca da semeadura.

Execução	**Topologia**	**Levels Back(%)**	**Portas**	**Número de avaliações**	**Resultado**
1	1x800	100	Γ_1	36415029	Factível
2				50000000	Infactível
3				50000000	Infactível
4				50000000	Infactível
5				50000000	Infactível
1	5x160	25	Γ_2	20907341	Factível
2				13561401	Factível
3				35915425	Factível
4				50000000	Infactível
5				38892093	Factível

7. Em seguida a observação dos gráficos presentes nas Figuras 45, 46, 47 e 48, dará informações sobre o desempenho de cada semeadura no processo evolutivo.

Analisando as Tabelas 3, 4, 5 , 6 e 7 observa-se que:

- A variação apenas do *levels-back*, representados pelos cenários 2 e 3, foi benéfica para o processo evolutivo do exemplo 4, conforme pode ser verificado na Tabela 6 linha 2.
- A variação apenas da topologia, representados pelos cenários 4 e 7 não ocasionou melhora de desempenho, conforme verifica-se nas Tabelas 3 4, 5 e 6 linhas 4 e 7.
- A variação da topologia e *levels-back*, representados pelos cenários 5, 6 , 8 e 9 foi benéfica ao processo evolucionário no exemplo 3 e 4 conforme Tabelas 5 linha 8 e Tabela 6, linhas 6 e 8.
- A inicialização da população utilizando somente portas NAND foi benéfica para o processo evolucionário no exemplo 4 conforme mostra a linha 10 da Tabela 6.
- A inicialização utilizando somente portas NAND simultaneamente com a variação do *levels-back*, representados pelos cenários 11 e 12, foi benéfica ao processo evolucionário nos exemplos 1, 2 e 4 conforme mostra a Tabela 3 linhas 11 e 12, Tabela 4 linha 12 e Tabela 6 linhas 11 e 12.
- A inicialização utilizando somente portas NAND simultaneamente com a variação topológica, representados pelos cenários 13 e 16, foi benéfica ao processo evolucionário nos exemplos 3 e 4 conforme mostram a Tabela 5 linha 16 e Tabela 6 linha 13 e 16.

- A inicialização utilizando somente portas NAND simultaneamente com a variação da topologia e do *levels-back*, representados pelos cenários 14, 15, 17 e 18 foi benéfica ao processo evolutivo nos exemplos 1, 2 e 4 conforme mostra a Tabela 3 linhas 14, 15, 17 e 18, Tabela 4 linhas 15 e 18 e Tabela 6 linhas 14, 15, 17 e 18.

- De acordo com os resultados apresentados na Tabela 7, a heurística da semeadura foi benéfica ao resolver o problema do multiplicador 4x4, encontrando circuitos factíveis em 4 das 5 execuções realizadas. Ressalta-se que quando a inicialização da população foi feita de acordo com o padrão da literatura, apenas uma das 5 execuções realizadas foi bem sucedida.

Comparando os gráficos das Figuras 45, 46, 47 e 48 nota-se:

- Em todos os cenários quando *levels-back* = 25 % houve um decréscimo no número de nós ativos da população ao longo da busca. Em vários esse comportamento foi benéfico para o processo evolutivo.

- Na população que começou com um alto grau de conectividade foi possível diminuir o número de nós ativos e explorar regiões do espaço de busca com diferentes graus de conectividade.

- Quando o *levels-back* foi 100% ou 50% o número de nós ativos da solução ficou na mesma ordem de grandeza do número de nós ativos da população inicial, e o espaço de busca não foi bem explorado.

- *Levels-back* define não somente a conectividade da população inicial, mas também determina o comportamento da topologia ao longo da evolução.

6.2 EXPERIMENTOS - OPERADOR DE MUTAÇÃO ENVIESADO

Esses experimentos utilizam um operador de mutação enviesado, denominado *Biased - SAM* (MANFRINI; BERNARDINO; BARBOSA, 2016a), o qual é capaz de guiar a mutação. Inicialmente é construída a Matriz Probabilidade de Transição para um dado conjunto de problemas, em seguida essa matriz é utilizada para direcionar o operador de mutação em outros problemas, conforme metodologia detalhada na Seção 5.3.1.

6.2.1 **Análise da Evolução**

Quatro problemas *benchmark* (ALBA et al., 2007) foram escolhidos para análise do processo evolutivo e criação da Matriz Probabilidade de Transição. O conjunto de operadores lógicos foi expandido conforme (GAJDA; SEKANINA, 2010): Γ = {AND, OR, NOT, NAND, NOR, XOR, WIRE, c_0, c_1}, onde c_k representa uma constante de

Tabela 8 – Tabela Probabilidade de Transição. A cada ocorrência de uma mutação benéfica tipo porta, a célula correspondente PAI → FILHO foi incrementada.

PAI \ FILHO	AND	OR	NOT	NAND	NOR	XOR	WIRE	C_1	C_0
AND	0	5	4	1	5	4	14	2	4
OR	7	0	4	5	2	34	8	3	1
NOT	6	4	0	23	17	3	2	4	3
NAND	1	6	16	0	7	33	0	5	4
NOR	4	2	15	6	0	6	0	3	1
XOR	0	10	7	12	3	0	4	5	3
WIRE	17	18	2	3	3	7	0	4	5
C_1	9	14	8	14	7	7	10	0	3
C_0	16	8	11	5	17	10	10	6	0

valor k. Na E.E. foi utilizado $\mu = 1$, $\lambda = 4$, número de linhas $n_r = 1$, número de colunas $n_c = 100$, levels-back, $l = n_c$, assim como em (MILLER, 2011).

Os problemas testes são definidos como:

Circuito 1: O primeiro problema tem quatro entradas e uma saída. O conjunto F indica os mintermos:
$F = \{0, 1, 3, 6, 7, 8, 10, 13\}$.

Circuito 2: O segundo problema tem cinco entradas e uma saída, e
$F= \{2, 3, 6, 7, 10, 11, 13, 15, 18, 19, 21, 23, 25, 27, 29, 31\}$.

Circuito 3: O terceiro problema tem quatro entradas e três saídas, e
$F_1 = \{0, 5, 10, 15\}$; $F_2 = \{1, 2, 3, 6, 7, 11\}$; $F_3 = \{4, 8, 9, 12, 13, 14\}$.

Circuito 4: O quarto exemplo tem cinco entradas e três saídas, e
$F_1 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 28, 29, 30, 31\}$;
$F_2 = \{0, 2, 4, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20, 22, 23 24, 26, 28, 30, 31\}$;
$F_3 = \{4, 5, 12, 13, 20, 21, 28, 29\}$.

Foram feitas 100 execuções independentes onde o algoritmo terminou quando uma solução factível foi encontrada ou quando o número máximo de avaliações permitidas foi atingido (100000 avaliações).

Analisando as mutações benéficas do conjunto de problemas citados foi construída a Matriz Probabilidade de Transição, representada na Tabela 8 e ilustrada na Figura 49 em valores percentuais. Toda vez que um alelo que represente a função que o nó executa (alelo tipo porta), selecionado para ser mutado, o novo valor desse alelo será escolhido em uma roleta onde a probabilidade de cada função é determinada pela Matriz Probabilidade

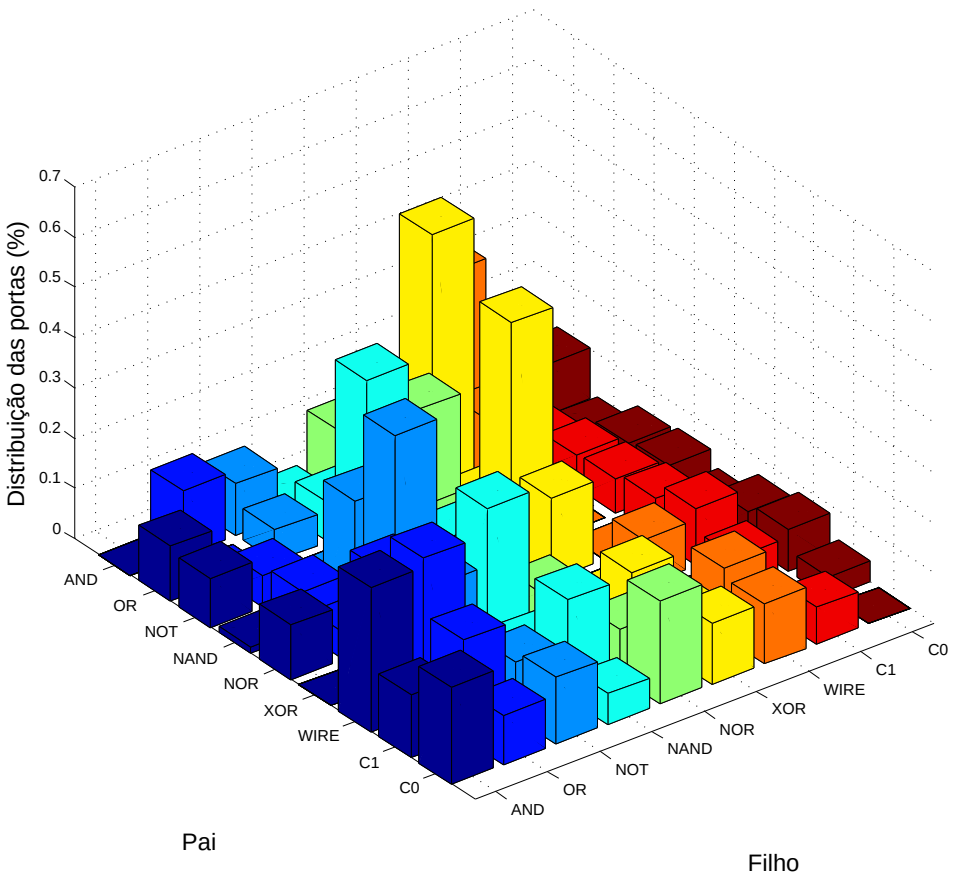


Figura 49 – Matriz Probabilidade de Transição dos problemas teste, onde pode-se encontrar as frequências das ocorrências de mutações benéficas do tipo porta.

de Transição.

A Matriz Probabilidade de Transição será utilizada para guiar a mutação *biased SAM* em outros problemas como será visto na Seção 6.2.2.

6.2.2 **Projeto de circuitos lógicos combinacionais**

Para o estudo comparativo quatro problemas *benchmark* estudados por Goldman e Punch (GOLDMAN; PUNCH, 2015) e amplamente utilizados na literatura de circuitos eletrônicos (TOCCI, 2011; ERCEGOVAC; MORENO; LANG, 1998) foram escolhidos para verificar a eficiência do operador *biased SAM*. De acordo com Goldman e Punch (GOLDMAN; PUNCH, 2013), esses problemas são úteis porque podem ser resolvidos com frequência, ou seja uma solução factível é encontrada na maioria das execuções. Isso permite uma análise estatística dos resultados. Posteriormente um circuito considerado complexo pela literatura (HRBACEK; SEKANINA, 2014) é apresentado no Exemplo 5.

Todos os experimentos foram implementados em MATLAB e as análises estatísticas foram feitas usando o *software* SPSS. Para a comparação dos diversos cenários foram

medidas as seguintes grandezas:

- **Hits**: Porcentagem do número de vezes que cada cenário encontrou uma solução factível.
- **MES** (*Median Evaluations to Success*): Mediana do número de avaliações necessárias para encontrar uma solução factível.
- **Mutação benéfica/1000 avaliações**: Taxa de ocorrência de mutações benéficas. Nota-se que um valor alto indica um melhor desempenho da CGP, pois há menos desperdício de chamadas da avaliação.

Para todos os problemas testes utilizou-se a CGP com estratégia evolutiva ($\mu + \lambda$), com $\mu = 1$ e $\lambda = 4$, e foram feitas 51 execuções independentes com o conjunto de funções $\Gamma = \{$AND, OR, NOT, NAND, NOR, XOR, WIRE, C_0, $C_1\}$. Para todos os problemas foram empregados os mesmos número de nós utilizados por Goldman e Punch (GOLDMAN; PUNCH, 2015).

Para garantir que uma solução factível seja encontrada na maioria das execuções, possibilitando assim uma análise estatística dos resultados, foi permitido um grande número de avaliações: 5000, 500000, 500000 e 1000000, para os problemas 1, 2, 3 e 4 respectivamente. Goldman e Punch (GOLDMAN; PUNCH, 2015) conseguiram resolver estes problemas em 95% das execuções com 1487, 42278, 74939 e 516034 avaliações respectivamente. Portanto o máximo número de avaliações permitidas foi pelo menos 60% maior que o requisitado em (GOLDMAN; PUNCH, 2015) para resolver os problemas.

Os resultados para cada tipo de mutação são mostrados na Tabela 10. A primeira linha de cada problema é o controle da configuração, correspondente à mutação SAM.

6.2.3 Exemplo 1

O primeiro problema é um detector de paridade de 3-Bits considerado simples, mas é um problema teste comum na literatura de CGP (YU; MILLER, 2001; MILLER; SMITH, 2006; WALKER; MILLER, 2008; GOLDMAN; PUNCH, 2015) e pode contribuir na compreensão de como o operador de mutação proposto afeta o resultado. A configuração da topologia foi: *número de linhas* $n_r = 1$, *número de colunas* $n_c = 500$, assim como escolhido em (GOLDMAN; PUNCH, 2015).

Na Tabela 10 verifica-se que *biased SAM* converge para uma solução factível com um menor número de avaliações. O método proposto obteve MES = 269 enquanto o controle da configuração MES = 413.

6.2.4 **Exemplo 2**

O segundo problema é um codificador 16-4 bit que pode ser encontrado em (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015). A topologia da configuração foi: número de linhas $n_r = 1$, número de colunas $n_c = 2000$, como escolhido por (GOLDMAN; PUNCH, 2015). A técnica proposta obteve MES = 16153 enquanto no controle MES = 19473.

6.2.5 **Exemplo 3**

O terceiro problema é um decodificador 4-16 bits proposto em (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015). A configuração da topologia foi número de linhas $n_r = 1$, número de colunas $n_c = 1000$, assim como escolhido em (GOLDMAN; PUNCH, 2015). A técnica proposta obteve MES = 165665 enquanto na configuração controle MES = 332813.

6.2.6 **Exemplo 4**

O quarto problema é um multiplicador de 3 bits, proposto em (GOLDMAN; PUNCH, 2013; GOLDMAN; PUNCH, 2015). A topologia da configuração foi número de linhas $n_r = 1$, número de colunas $n_c = 5000$, assim como (GOLDMAN; PUNCH, 2015). Este problema é muito difícil comparado com os anteriores e a técnica proposta obteve MES = 435781 enquanto o controle teve MES = 559385.

6.2.7 **Exemplo 5**

Para verificar a eficiência da heurística do operador de mutação *Biased - SAM* em problemas maiores, foram projetados multiplicadores 4x4. sendo utilizado a mesma quantidade de nós que Sekanina (HRBACEK; SEKANINA, 2014) com a topologia 1x800. Foram feitas 5 execuções independentes utilizando a mutação SAM e 5 execuções com o operador *Biased - SAM*. Em todos as execuções foram permitidas até 50000000 avaliações.

Os experimentos computacionais foram feitos no *software* MATLAB 2014 utilizando um computador *desktop* com sistema operacional Windows 7 processador AMD Phenom(tm) IIX4 B95 de 3GHz e com 8GB de memória RAM. Observou-se durante os experimentos que 1000000 avaliações da função objetivo demoram cerca de 70 minutos quando se utiliza o operador SAM padrão e 68 minutos quando o operador *Biased - SAM* é utilizado. A Tabela 9 apresenta os resultados obtidos.

Nos experimentos comparativos entre o operador de mutação SAM e *Biased* SAM enviesado apresentados na Tabela 9 verifica-se que quando a evolução levou em consideração a Matriz Probabilidade de Transição a solução factível foi encontrada em 3 as 5 execuções realizadas, sendo necessárias 23575726 avaliações da função objetivo em mé-

Tabela 9 – Resultado obtido para o projeto do circuito multiplicador 4x4. Foram feitas 5 execuções com o operador padrão e 5 execuções utilizando operador de mutação enviesado.

Execução	**Single Active Mutation**	**Número de Avaliações**	**Resultado**
1	Padrão	24533553	Factível
2		17073001	Factível
3		27503532	Factível
4		19061333	Factível
5		16133009	Factível
1	Enviesado	17966005	Factível
2		23021481	Factível
3		29739693	Factível
4		50000000	Infactível
5		50000000	Infactível

dia. Quando o processo evolutivo utilizou o operador de mutação SAM foi encontrado o resultado factível em todas as 5 execuções sendo necessárias 20860885 avaliações em média.

6.2.8 **Discussão dos Resultados**

Analisando a Matriz Probabilidade de Transição extraída dos quatro problemas *benchmark* da Seção 6.2.1, verifica-se que uma mutação importante é NAND→XOR. Por outro lado, a mutação NOT→OR acontece com baixa frequência. Dessa forma quando *biased SAM* prioriza a troca NAND→XOR e diminui a probabilidade de ocorrência NOT→ OR a CGP melhora o desempenho.

Analisando a saída da tabela verdade da porta XOR denotada por $F_{XOR} = \{0110\}$ e comparando com a saída da tabela verdade da porta NAND denotada por $F_{NAND} = \{1110\}$ nota-se que o comportamento desses operadores divergem em apenas em um caso, essa diferença mostra que a mutação suave é naturalmente priorizada pela evolução.

É interessante notar na Figura 49 que não existe um operador que seja preferencial para todos os casos. A probabilidade de transição varia de acordo com o operador a ser modificado. Por exemplo, a porta XOR tem um valor alto de probabilidade quando portas NAND ou OR são modificadas enquanto apresenta baixa probabilidade de melhoria para substituir WIRE ou NOT.

De acordo com os resultados apresentados na Seção 6.2.2 quando a busca pela solução factível levou em consideração a Matriz Probabilidade de Transição, houve uma melhoria do desempenho da CGP em relação ao operador de mutação SAM em 4 dos 5 problemas apresentados. Nota-se que além do decréscimo no número de avaliações necessárias para encontrar uma solução factível, o parâmetro *(Mutações Beneficas)*/1000

Tabela 10 – Comparação entre mutação padrão SAM e *biased SAM* para os quatro problemas testados. "Hits" representa o número de vezes que a abordagem encontra a solução factível. O número de vezes que a mutação benéfica ocorre a cada 1000 avaliações é denotada por "Mutação benéfica/1000 *evals*". "MES" é a mediana de avaliações para o sucesso. O p-valor calculado com o teste Mann-Whitney em relação a MES também são apresentados.

Circuito	Single Active Mutation	Hits (%)	Mutações Benéficas /1000 *evals*	MES	Intervalo Confiança	p-valor
Detector	Padrão	98	15.7	413	285 .. 469	–
Paridade	Enviesado	100	15.9	269	189 .. 393	0.049
Codificador	Padrão	100	1.6	19473	16673 .. 22953	–
16-4 bits	Enviesado	100	1.9	16153	14525 .. 20837	0.103
Decodificador	Padrão	90	1.0	332813	293501 .. 360637	–
16-4 bits	Enviesado	100	2.1	165665	145681 .. 184161	0
Multiplicador	Padrão	76	1.3	559385	464745 .. 744909	–
3 bits	Enviesado	88	1.6	435781	382125 .. 550645	0

evals aumentou, mostrando que a eficiência da busca melhorou em 4 dos 5 problemas testados.

Nos experimentos comparativos entre o operador de mutação SAM e *Biased* SAM enviesado apresentados na Tabela 9 verifica-se que quando a evolução levou em consideração a Matriz Probabilidade de Transição a solução factível foi encontrada em 3 as 5 execuções, sendo necessárias 23575726 avaliações da função objetivo em média. Quando o processo evolutivo utilizou o operador de mutação SAM foi encontrado o resultado factível em todas as 5 execuções sendo necessárias 20860885 avaliações em média.

7 CONCLUSÕES

Neste trabalho foram desenvolvidas novas estratégias para projetar circuitos lógicos combinacionais através de algoritmos evolutivos. A partir dos estudos apresentados e dos experimentos computacionais realizados (MANFRINI; BERNARDINO; BARBOSA, 2016b; MANFRINI; BERNARDINO; BARBOSA, 2016a), as conclusões podem ser descritas sob os seguintes aspectos: (i) investigação do processo evolutivo; (ii) heurística para semeadura da população; (iii) um novo operador de mutação eficiente; e (iv) trabalhos futuros.

(i) Investigação do processo evolutivo

A metodologia criada para investigar os detalhes envolvidos no mecanismo evolutivo, em particular as análises das mutações benéficas, confirmou alguns resultados prévios da literatura, como a importância da presença dos nós neutros na evolução.

Além disso, tal metodologia contribuiu para a compreensão das causas fundamentais da evolução, possibilitando a criação da heurística para a semeadura da população inicial e de um operador de mutação eficiente.

Ressalta-se que a investigação do processo evolutivo apresentada abre caminho para que outras vertentes sejam exploradas, possibilitando o desenvolvimento de novas heurísticas e operadores aplicados em diferentes áreas do conhecimento.

(ii) Heurística para semeadura da população.

Os resultados experimentais confirmaram a superioridade da heurística proposta para a semeadura da população inicial em relação à inicialização aleatória da população, reduzindo o número de avaliações para encontrar uma solução factível.

Adicionalmente, utilizando somente portas NAND na inicialização da população o desempenho da CGP melhorou na maioria dos casos estudados.

Ao contrário do que vem sendo apontado pela literatura, mostrou-se que o parâmetro *levels-back* usado para gerar a população inicial deve ser restringido ao invés de permitir a conexão entre qualquer par de nós. O valor inicial do *levels-back* é muito importante, pois determina a região do espaço de busca que será explorada durante o processo evolutivo.

(iii) Um novo operador de mutação eficiente

Através da análise do processo evolucionário para um dado conjunto de problemas, foi possível extrair o conhecimento e então guiar a busca em outros problemas. Ressalta-se que algumas informações obtidas não estavam no campo de conhecimento dos especialistas.

A incorporação do conhecimento sobre o desempenho do operador de mutação

constitui um importante passo para aumentar a eficácia da CGP como ferramenta de projeto.

(iv) Trabalhos futuros

A CGP não tem utilizado o operador de recombinação no processo de busca. A principal razão para isso foi apresentada por Miller (MILLER, 2011) que realizou experimentos e constatou que a recombinação foi prejudicial para a evolução. Uma proposta a ser testada é a criação de um operador de recombinação que atue somente em nós inativos.

Um operador de mutação exclusivo para os nós inativos é capaz de cruzar indivíduos sem degradar a aptidão, além de diminuir a perda de material genético dos indivíduos com alta aptidão.

O operador de mutação apresentado *biased SAM* poderá ser construído pelo próprio algoritmo ao longo da evolução. Dessa forma, além do algoritmo ficar mais autônomo com a eliminação dos problemas testes, o operador *biased SAM* passa a ser adaptativo, podendo direcionar a busca de acordo com as modificações do próprio processo evolutivo.

REFERÊNCIAS

ABD-EL-BARR, M. et al. A modified ant colony algorithm for evolutionary design of digital circuits. In: IEEE. *Evolutionary Computation CEC'03. The 2003 Congress on*. [S.l.], 2003. v. 1, p. 708–715.

AGUIRRE, A. H.; BUCKLES, B. P.; COELLO, C. C. Ga-based learning of KNDF boolean formulas. In: *Evolvable Systems: From Biology to Hardware*. [S.l.]: Springer, 2001. p. 279–290.

AGUIRRE, A. H.; COELLO, C. A. C. Using genetic programing and multiplexers for the synthesis of logic circuits. *Engineering Optimization*, Taylor & Francis, v. 36, n. 4, p. 491–511, 2004.

AHMAD, A. M. et al. Breast cancer detection using cartesian genetic programming evolved artificial neural networks. In: ACM. *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. [S.l.], 2012. p. 1031–1038.

ALBA, E. et al. Comparative study of serial and parallel heuristics used to design combinational logic circuits. *Optimisation Methods and Software*, Taylor & Francis, v. 22, n. 3, p. 485–509, 2007.

ANJOMSHOA, M.; MAHANI, A. A novel evolutionary method for designing optimized multifunctional logic modules. *Intl. Journal of Reconfigurable and Embedded Systems (IJRES)*, v. 2, n. 2, p. 55–63, 2013.

ANJOMSHOA, M.; MAHANI, A.; SADEGHIFARD, S. A new automated design and optimization method of CMOS logic circuits based on modified imperialistic competitive algorithm. *Applied Soft Computing*, Elsevier, v. 21, p. 423–432, 2014.

BOOLE, G. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. [S.l.]: Walton and Maberly, 1854. v. 2.

BRAGA, A. de P.; FERREIRA, A. C. P. de L.; LUDERMIR, T. B. *Redes neurais artificiais: teoria e aplicações*. [S.l.]: LTC Editora, 2007.

BRAYTON, R. K. et al. *Logic minimization algorithms for VLSI synthesis*. [S.l.]: Springer Science & Business Media, 1984. v. 2.

CAMPOS, T. J. de. *Hardware Evolutivo Aplicado a Geração Automática de Controladores para Servo-Mecanismos*. Tese (Doutorado) — Universidade Estadual de Campinas, 2007.

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

COELLO, C. A. C.; AGUIRRE, A. H. Design of combinational logic circuits through an evolutionary multiobjective optimization approach. *AI EDAM*, Cambridge Univ Press, v. 16, n. 01, 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. *Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on*. [S.l.], 2003. p. 3–12.

COELLO, C. A. C.; CHRISTIANSEN, A. D.; AGUIRRE, A. H. Automated design of combinational logic circuits using genetic algorithms. In: *in Proc. of the Int. Conf. on Artificial Neural Nets and Genetic Algorithm (ICANNGA'97) Eds.* [S.l.]: Publisher: Springer-Verlag, 1997. p. 335–338.

COELLO, C. A. C.; CHRISTIANSEN, A. D.; AGUIRRE, A. H. Towards automated evolutionary design of combinational circuits. *Computers & Electrical Engineering*, Elsevier, v. 27, n. 1, p. 1–28, 2000.

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

COELLO, C. A. C.; LUNA, E. H.; AGUIRRE, A. H. Use of particle swarm optimization to design combinational logic circuits. In: *Evolvable Systems: From Biology to Hardware.* [S.l.]: Springer, 2003. p. 398–409.

COELLO, C. A. C. et al. Ant colony system for the design of combinational logic circuits. In: *Evolvable Systems: From Biology to Hardware.* [S.l.]: Springer, 2000. p. 21–30.

CUNHA, A. G.; TAKAHASHI, R.; ANTUNES, C. H. *Manual de computação evolutiva e metaheurística.* Coimbra: Imprensa da Universidade de Coimbra e Editora da Universidade Federal de Minas Gerais, 2012. ISBN 978-989-26-0583-8 (PDF).

CUSTÓDIO, F. *Algoritmos genéticos para predição Ab Initio de Estruturas de Proteínas.* Tese (Doutorado) — PhD thesis, LNCC, Pretrópolis, RJ.[Links], 2008.

DARWIN, C. *On the Origen of Species by Means of Natural Selection.* [S.l.]: Culture et Civilisation, 1969.

DEJONG, K. *An analysis of the behavior of a class of genetic adaptive systems.* Tese (Doutorado), 1975.

EBERHART, R. C.; SHI, Y. Particle swarm optimization: developments, applications and resources. In: IEEE. *Evolutionary Computation, 2001. Proc. of the 2001 Congress on.* [S.l.], 2001. v. 1, p. 81–86.

EIBEN, A. E.; SMITH, J. E. *Introduction to evolutionary computing.* [S.l.]: Springer, 2003.

EIGEN, M. *Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution.* [S.l.]: mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.

ERCEGOVAC, M. D.; MORENO, J. H.; LANG, T. *Introduction to digital systems.* [S.l.]: John Wiley & Sons, Inc., 1998.

FOGEL, L. J. *Artificial Intelligence Through Simulated Evolution.[By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh.* [S.l.]: John Wiley and Sons, 1966.

FRIEDMAN, G. J. *Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy.* Dissertação (Mestrado) — University of California, Los Angeles, 1956.

GAJDA, Z.; SEKANINA, L. An efficient selection strategy for digital circuit evolution. In: SPRINGER. *International Conference on Evolvable Systems*. [S.l.], 2010. p. 13–24.

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

GOLDMAN, B. W.; PUNCH, W. F. *Reducing wasted evaluations in cartesian genetic programming*. [S.l.]: Springer, 2013.

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

GUIMARÃES, A. L. P. *Estruturas CMOS Programáveis para Aplicação em Eletrônica Evolucionária*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2006.

HAN, J.; ORSHANSKY, M. Approximate computing: An emerging paradigm for energy-efficient design. In: IEEE. *2013 18th IEEE European Test Symposium (ETS)*. [S.l.], 2013. p. 1–6.

HAN, K.-H.; KIM, J.-H. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *Evolutionary Computation, IEEE Transactions on*, IEEE, v. 6, n. 6, p. 580–593, 2002.

HARDING, S.; MILLER, J. F. Evolution of robot controller using cartesian genetic programming. In: SPRINGER. *European Conference on Genetic Programming*. [S.l.], 2005. p. 62–73.

HARVEY, I. *The Artificial Evolution of Adaptive Behaviour*. Tese (Doutorado) — University of Sussex, School of Cognitive and Computing Sciences, 1993.

HILDER, J.; WALKER, J. A.; TYRRELL, A. Use of a multi-objective fitness function to improve cartesian genetic programming circuits. In: IEEE. *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*. [S.l.], 2010. p. 179–185.

HOLLAND, J. H. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. [S.l.]: U. Michigan Press, 1975.

HORNBY, G. et al. Automated antenna design with evolutionary algorithms. In: *Space 2006*. [S.l.: s.n.], 2015. p. 7242.

HRBACEK, R.; MRAZEK, V.; VASICEK, Z. Automatic design of approximate circuits by means of multi-objective evolutionary algorithms. In: IEEE. *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. [S.l.], 2016. p. 1–6.

HRBACEK, R.; SEKANINA, L. Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In: ACM. *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2014. p. 1015–1022.

IGWE, K.; PILLAY, N. Automatic programming using genetic programming. In: IEEE. *Information and Communication Technologies (WICT), 2013 Third World Congress on*. [S.l.], 2013. p. 337–342.

JAN, A.; KHURSHID, N.; KHATTAK, M. I. Developing resource efficient heart arrhythmia classifier. *International Journal of Computer Applications*, Foundation of Computer Science, v. 109, n. 16, 2015.

JR, G. G. L. A decomposition chart technique to aid in realizations with multiplexers. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 2, p. 157–159, 1978.

KARAKATIC, S.; PODGORELEC, V.; HERICKO, M. Optimization of combinational logic circuits with genetic programming. *Electronics & Electrical Engineering*, v. 19, n. 7, 2013.

KHAN, G. M.; MILLER, J. F.; HALLIDAY, D. M. Evolution of cartesian genetic programs for development of learning neural architecture. *Evolutionary Computation*, MIT Press, v. 19, n. 3, p. 469–523, 2011.

KOZA, J. R. *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*. [S.l.]: MIT press, 1992. v. 1.

KOZA, J. R. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, Springer, v. 11, n. 3-4, p. 251–284, 2010.

LOUIS, S. J. *Genetic Algorithms As a Computational Tool for Design*. Tese (Doutorado), Bloomington, IN, USA, 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. *Evolutionary Computation (CEC), 2014 IEEE Congress on*. [S.l.], 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 (PPSN)*. [S.l.], 2016. p. 665–674.

MANFRINI, F. A.; BERNARDINO, H. S.; BARBOSA, H. J. On heuristics for seeding the initial population of cartesian genetic programming applied to combinational logic circuits. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: ACM, 2016. (GECCO '16 Companion), p. 105–106. ISBN 978-1-4503-4323-7. Disponível em: <http://doi.acm.org/10.1145-/2908961.2909031>.

MCCLUSKEY, E. J. Minimization of boolean functions. *Bell Systems Technical Journal*, IEEE, v. 35, n. 5, p. 1417–1444, 1956.

MCCLUSKEY, E. J. Minimal sums for boolean functions having many unspecified fundamental products. In: IEEE. *Switching Circuit Theory and Logical Design, SWCT 1961. Proc. of the Second Annual Symposium on*. [S.l.], 1961. p. 10–17.

MILLER, J. *http://www.cartesiangp.co.uk*. [S.l.], 2017 (accessed February 14, 2017).

MILLER, J.; TURNER, A. Cartesian genetic programming. In: ACM. *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. [S.l.], 2015. p. 179–198.

MILLER, J. F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. [S.l.: s.n.], 1999. v. 2, p. 1135–1142.

MILLER, J. F. Evolution of digital filters using a gate array model. In: SPRINGER. *Workshops on Applications of Evolutionary Computation*. [S.l.], 1999. p. 17–30.

MILLER, J. F. *Cartesian genetic programming*. [S.l.]: Springer, 2011.

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

MILLER, J. F.; THOMSON, P.; FOGARTY, T. *Designing electronic circuits using evolutionary algorithms. Arithmetic Circuits: A case study*. [S.l.]: Wiley, 1997.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT Press, 1996.

MOORE, P.; VENAYAGAMOORTHY, G. K. Evolving combinational logic circuits using a hybrid quantum evolution and particle swarm inspired algorithm. In: IEEE. *Evolvable Hardware, Proceedings. 2005 NASA/DoD Conference on*. [S.l.], 2005. p. 97–102.

PEDRONI, V. A. *Digital electronics and design with VHDL*. [S.l.]: Morgan Kaufmann, 2008.

RECH, F. *Fundamentos de Circuitos Digitais*. [S.l.]: Sagra Luzzatto, 2006.

REIS, R. A. d. L. et al. Concepção de circuitos integrados. *Instituto de Informática da UFRGS. Editora Sagra Luzzatto*, 2000.

REN, A. et al. Implement of evolvable hardware based improved genetic algorithm. In: IEEE. *Natural Computation (ICNC), 2011 Seventh International Conference on*. [S.l.], 2011. v. 4, p. 2112–2115.

SASAO, T. *Logic synthesis and optimization*. [S.l.]: Springer, 1993. v. 212.

SASAO, T. A design method for and-or-exor three-level networks. In: CITESEER. *Proc. int. Workshop on Logic Synthesis*. [S.l.], 1995. p. 8.

SCHWEFEL, H.-P. P. *Evolution and optimum seeking: the sixth generation*. [S.l.]: John Wiley & Sons, Inc., 1993.

SHANNON, C. E. The synthesis of two-terminal switching circuits. *BSTJ*, v. 28, n. 1, p. 459–466, 1949.

STOMEO, E.; KALGANOVA, T.; LAMBERT, C. Generalized disjunction decomposition for evolvable hardware. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, IEEE, v. 36, n. 5, p. 1024–1043, 2006.

TOCCI, R. J. *Digital Systems: principles and applications*. [S.l.]: Pearson Education India, 2011.

TURNER, A. J.; MILLER, J. F. Neutral genetic drift: an investigation using cartesian genetic programming. *Genetic Programming and Evolvable Machines*, Springer, p. 1–28, 2015.

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.

UYEMURA, J. P. *First Course in Digital Systems Design: An Integrated Approach*. [S.l.]: International Thomson Publishing, 1999.

VAHID, F.; WILEY, J. *Digital design*. [S.l.]: Wiley, 2006.

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

VASICEK, Z.; SEKANINA, L. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, Springer, v. 12, n. 3, p. 305–327, 2011.

VASICEK, Z.; SEKANINA, L. On area minimization of complex combinational circuits using cartesian genetic programming. In: IEEE. *2012 IEEE Congress on Evolutionary Computation*. [S.l.], 2012. p. 1–8.

VASICEK, Z.; SEKANINA, L. Evolutionary design of approximate multipliers under different error metrics. In: IEEE. *Design and Diagnostics of Electronic Circuits & Systems, 17th International Symposium on*. [S.l.], 2014. p. 135–140.

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

VASICEK, Z.; SEKANINA, L. Circuit approximation using single-and multi-objective cartesian gp. In: SPRINGER. *European Conference on Genetic Programming*. [S.l.], 2015. p. 217–229.

VASICEK, Z.; SEKANINA, L. Evolutionary design of complex approximate combinational circuits. *Genetic Programming and Evolvable Machines*, Springer, p. 1–24, 2016.

VASSILEV, V. K.; JOB, D.; MILLER, J. F. Towards the automatic design of more efficient digital circuits. In: IEEE. *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*. [S.l.], 2000. p. 151–160.

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

WHITLEY, L. D. et al. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In: *ICGA*. [S.l.: s.n.], 1989. v. 89, p. 116–123.

YU, T.; MILLER, J. Neutrality and the evolvability of boolean function landscape. In: *Genetic programming*. [S.l.]: Springer, 2001. p. 204–217.

ZEBULUM, R. S. Síntese de circuitos eletrônicos por computação evolutiva. *PhD These, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil*, 1999.

ZOMAYA, A. Y. *Handbook of nature-inspired and innovative computing: integrating classical models with emerging technologies*. [S.l.]: Springer Science & Business Media, 2006.

APÊNDICE A – Métodos Tradicionais para Projeto de Circuitos Combinacionais

A.0.1 Soma de Produtos SOP – Rede OR-AND

É uma rede de dois níveis do tipo OR - AND [1]. Para analisar esse tipo de rede são necessárias algumas definições:

Literal: É uma variável complementada ou não complementada.

Exemplos: A, B, $\overline{B}$, $\overline{C}$

Termo de Produto: É uma literal, ou a função AND (produto) de literais.

Exemplos: $(A \cdot \overline{B} \cdot C)$, $(\overline{A} \cdot C)$

Soma de Produtos - SDP: É um expressão que consiste em um termo de produto ou na função OR (soma) desses termos.

Exemplo: $(A \cdot B \cdot \overline{C}) + (\overline{A} \cdot C)$

Mintermo: Um mintermo de n variáveis é um termo de produto de n literais, em que cada variável aparece exatamente uma vez, (complementada ou não complementada). Por exemplo, para $n = 3$ e dada a equação

$$F(A, B, C) = A \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B},$$

os dois primeiros termos são mintermos enquanto A $\cdot\overline{B}$ não é mintermo, pois a variável C não está presente. Cada mintermo tem o valor 1 para uma única atribuição das variáveis. Por exemplo, o mintermo $A \cdot B \cdot \overline{C}$ será igual a 1, se e somente se: $A = 1$, $B = 1$, e $C = 0$. Como consequência, cada mintermo está associado de forma unívoca à uma determinada linha da tabela verdade, que pode ser identificada fazendo a conversão de binário para decimal. No caso acima, isso equivale a: $(110)_2 = 6_{10}$; portanto, $A \cdot B \cdot \overline{C}$ representa o mintermo m_6.

A Figura 50 mostra o circuito equivalente à equação booleana:

$$F = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot B \cdot \overline{C},$$

onde cada termo de produto é um mintermo. Analisando a tabela verdade da Figura 50, verifica-se que cada linha que possui a saída $F(A, B, C) = 1$ é representada por um termo de produto na equação booleana. Portanto, cada mintermo corresponde a uma

[1] Por definição, na rede de 2 níveis não se considera os inversores conectados às variáveis de entrada como um nível .

porta AND no circuito lógico equivalente. Dessa forma, a saída F também pode ser representada como uma soma de mintermos:

$$F(A, B, C) = m_1 + m_2 + m_6$$

ou, de forma compacta, como

$$F = \sum m(1, 2, 6).$$

	A B C	F
0	0 0 0	0
1	0 0 1	1
2	0 1 0	1
3	0 1 1	0
4	1 0 0	0
5	1 0 1	0
6	1 1 0	1
7	1 1 1	0

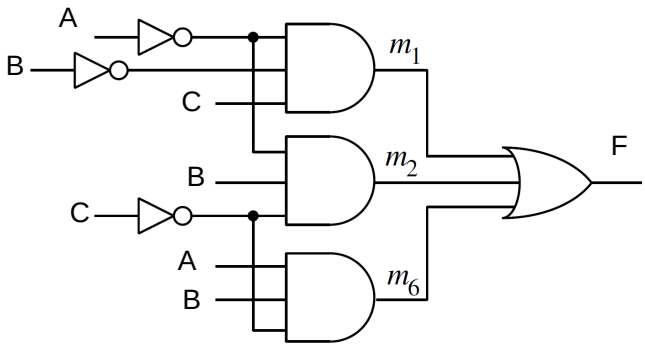


Figura 50 – Soma de Produtos

A.0.2 **Produto das Somas – Redes AND-OR**

É uma rede de dois níveis, do tipo AND -OR. Para analisar esse tipo de rede, são necessárias algumas definições:

Termo da Soma: É um literal ou a função OR (soma) de literais.

Exemplos: $(A + \overline{B} + C)$, $(\overline{A} + C)$

Produto das Somas: É uma expressão que consiste em uma função OR ou no AND (produto) desses termos.

Exemplo: $(A + B + \overline{C}) \cdot (\overline{A} + C)$

Maxtermo: Um Maxtermo de n variáveis é uma função OR de n literais em que cada variável aparece exatamente uma vez (na forma complementada ou não complementada). Por exemplo, para $n = 3$ e seja a equação

$$F(A, B, C) = (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \cdot (A + \overline{B}),$$

os dois primeiros termos são Maxtermos, enquanto $(A + \overline{B})$ não é Maxtermo, pois a variável C não está presente. Cada Maxtermo tem o valor 0 para uma única atribuição das variáveis. Por exemplo, o Maxtermo $(\overline{A} + B + \overline{C})$ será igual a 0,

se e somente se: $A = 1$, $B = 0$, e $C = 1$. Como consequência, cada Maxtermo está associado de forma unívoca a uma determinada linha da tabela verdade, que pode ser identificada fazendo a conversão de *binário para decimal*. No exemplo apresentado, isso equivale a: $(101)_2 = 5_{10}$; portanto, $(\overline{A} + B + \overline{C})$ representa o $Maxtermo\ M_5$.

Exemplo:

A Figura 51 mostra o circuito equivalente à equação booleana

$$F = (A + B + C) \cdot (\overline{A} + B + \overline{C}) \cdot (\overline{A} + \overline{B} + C),$$

onde cada termo de soma é um Maxtermo. Analisando a tabela verdade da Figura 51, verifica-se que cada linha que possui a saída $F(A, B, C) = 0$ é representada por um termo de soma na equação booleana. Portanto, cada Maxtermo corresponde a uma porta OR no circuito lógico equivalente. Dessa forma, a saída F também pode ser representada como um produto de Maxtermos:

$$F(A, B, C) = M_0 \cdot M_5 \cdot M_6$$

ou de forma compacta

$$F = \prod M(0, 5, 6)$$

	A B C	F
0	0 0 0	0
1	0 0 1	1
2	0 1 0	1
3	0 1 1	1
4	1 0 0	1
5	1 0 1	0
6	1 1 0	0
7	1 1 1	1

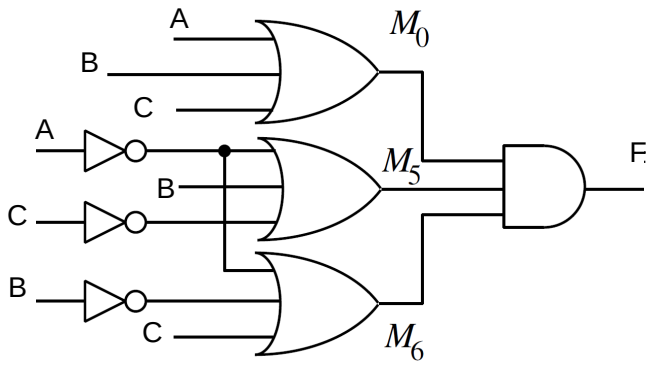


Figura 51 – Produto de Somas

A.0.3 **Redes Mínimas de 2 níveis**

Várias equações lógicas podem corresponder à uma mesma tabela verdade. Através de manipulações algébricas é possível obter expressões da forma SOP e POS que sejam menores que as formas canônicas. Uma identidade simples permitirá o desenvolvimento de métodos automáticos de redução:

$$A + \overline{A} = 1$$

Como exemplo, pode-se verificar que os dois primeiros termos da equação

$$F(A,B,C) = A \cdot B \cdot C + A \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot C + A \cdot B \cdot C \tag{A.1}$$

podem ser combinados e simplificados da seguinte forma:

$$A \cdot B \cdot C + A \cdot B \cdot \overline{C} = A \cdot B \cdot (C + \overline{C}) = A \cdot B.$$

O agrupamento desses termos é possível porque eles divergem em apenas uma variável(C), a qual é eliminada quando são colocados os termos em comum em evidência. Da mesma forma, o terceiro e o quarto termo também podem ser simplificados fazendo:

$$A \cdot \overline{B} \cdot C + A \cdot B \cdot C = A \cdot (B + \overline{B}) \cdot C = A \cdot C$$

Portanto, a expressão $F(A,B,C)$ apresentada na Equação (A.1) pode ser escrita como:

$$F(A,B,C) = A \cdot B + A \cdot C$$

Verifica-se que a expressão final gera a mesma tabela verdade, porém está em uma forma simplificada. A álgebra booleana possui diversas propriedades que permitem a simplificação de uma equação booleana. Entretanto um método denominado mapa Karnaugh é amplamente utilizado para essa finalidade.

A.1 MAPA DE KARNAUGH

Dois mintermos que divergem de apenas uma variável são denominados vizinhos (ERCEGOVAC; MORENO; LANG, 1998). É o caso, por exemplo dos mintermos m_6 e m_7, que representam respectivamente as expressões $A \cdot B \cdot \overline{C}$ e $A \cdot B \cdot C$. Esse fato também pode ser identificado pela representação binária de 6 $(110)_2$ e 7 $(111)_2$. Entretanto, não são todos os mintermos consecutivos que são vizinhos, como por exemplo m_3 $(011)_2$ e m_4 $(100)_2$. Além disso, alguns mintermos não consecutivos são vizinhos como m_0 $(000)_0$ e m_2 $(010)_2$.

Para que os vizinhos sejam visualmente identificados e agrupados é necessário colocá-los próximos uns dos outros. Para isso utiliza-se uma matriz denominada mapa de Karnaugh (TOCCI, 2011) em que as células que representam a saída da tabela verdade estão alocadas em outra ordem, de forma que todos os vizinhos fiquem próximos. Na Figura 52 são mostrados diversos mapas de Karnaugh e seus respectivos agrupamentos. Além de todas as informações provenientes de uma tabela verdade, algumas barras identificando a região em que uma determinada variável está associada ao bit 1 são colocadas no mapa. Por exemplo, a barra denominada A demarca os mintermos $m_8 \cdots m_{15}$, região em que os mintermos apresentam a variável A não complementada. Por sua vez, na região não demarcada pela barra $(m_0 \cdots m_7)$ os mintermos estão associados à variável $\overline{A}$. Com

as demarcações de todas as variáveis é possível identificar os agrupamentos e simplificar a expressão booleana de modo visual, sem a necessidade de aplicar as propriedades da álgebra booleana.

Na Figura 52(a), referente à função $F = \sum m(9, 13)$, os mintermos m_9 e m_{13} podem ser agrupados em uma *dupla*, sendo que a expressão simplificada pode ser obtida analisando o posicionamento do agrupamento em relação a cada uma das variáveis, da seguinte maneira:

- **Variável A:** O agrupamento está totalmente na região A;
- **Variável B:** O agrupamento está parcialmente na região B e parcialmente na região $\overline{B}$, o que acarreta a exclusão da variável B na expressão final;
- **Variável C:** O agrupamento está totalmente na região $\overline{C}$;
- **Variável D:** O agrupamento está totalmente na região D.

A expressão final, portanto, é: $F(A, B, C, D) = A \cdot \overline{C} \cdot D$.

Na Figura 52(b), referente à função $F = \sum m(5, 7, 13, 15)$, os mintermos podem ser agrupados em uma *quadra* e a expressão simplificada pode ser obtida analisando o posicionamento do agrupamento em relação a cada uma das variáveis:

- **Variável A:** O agrupamento está parcialmente na região A e parcialmente na região $\overline{A}$, o que acarreta a exclusão da variável A na expressão final;
- **Variável B:** O agrupamento está totalmente na região B;
- **Variável C:** O agrupamento está parcialmente na região C e parcialmente na região $\overline{C}$, o que acarreta a exclusão da variável C na expressão final; e
- **Variável D:** O agrupamento está totalmente na região D.

A expressão final, portanto, é: $F(A, B, C, D) = B \cdot D$.

Na Figura 52(c) é possível identificar que o agrupamento pertence somente à região $\overline{C}$; portanto, a expressão final é: $F(A, B, C, D) = \overline{C}$.

A Figura 52(d) possui 2 agrupamentos e a expressão final é a soma desses agrupamentos: $F(A, B, C, D) = B \cdot \overline{C} + \overline{A} \cdot D$.

Analisando os agrupamentos presentes na Figura 52, percebe-se que pode-se eliminar uma variável do termo com uma *dupla*, é possível retirar duas variáveis com uma *quadra* e finalmente, a eliminação de três variáveis é possível com um *octeto*. Portanto, para a simplificação da expressão booleana através do mapa de Karnaugh, buscam-se

sempre os maiores agrupamentos, sendo importante agrupar todos os mintermos. Em seguida, verificam-se quais são os agrupamentos essenciais, ou seja, quais os agrupamentos que possuem um termo exclusivo[2].

Toda a análise apresentada para a soma de produtos com o mapa de Karnaugh pode ser feita de modo similar para o produto de somas, mas neste caso os agrupamentos são feitos com as células *zero*.

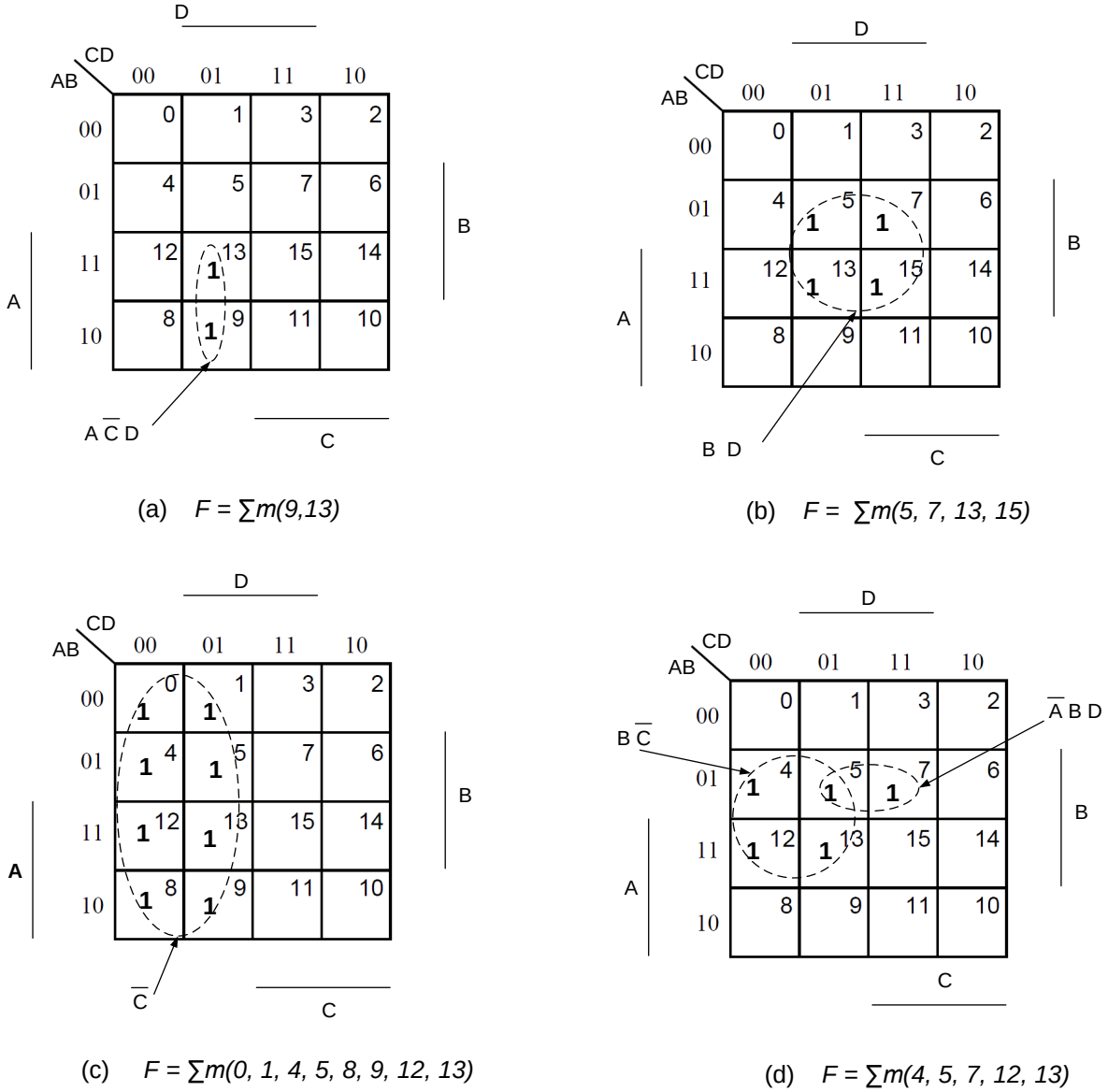


Figura 52 – Exemplos de agrupamentos do mapa Karnaugh para diversas funções

[2] Nos exemplos da Figura 52, todos os agrupamentos são essenciais. Entretanto, existem diversas situações de agrupamentos não essenciais mas que fogem do escopo desse texto. Estes podem ser encontrados em diversas referências, tais como: (TOCCI, 2011; UYEMURA, 1999; VAHID; WILEY, 2006).

A.2 REDES XOR - OR - AND

Sasao (SASAO, 1993; SASAO, 1995) desenvolveu uma metologia que permite melhorar os projetos obtidos com o mapa de Karnaugh através da inclusão de uma porta XOR na saída do circuito.

A saída de uma porta XOR de duas entradas tem o valor 1 quando as duas entradas forem diferentes e 0, caso contrário. Uma outra interpretação para as portas XOR é representada na Figura 53, onde uma das entrada é denominada controle e a outra é uma variável A. Nesse caso, pode-se interpretar a operação XOR como:

- quando $controle = 0$, se $A = 0$ tem-se $saida = 0$. Se $A = 1$, tem-se a $saida = 1$, ou seja: $saida = A$.
- quando $controle = 1$, se $A = 0$ tem-se $saida = 1$. Se $A = 1$, tem-se a $saida = 0$, ou seja: $saida = \overline{A}$.

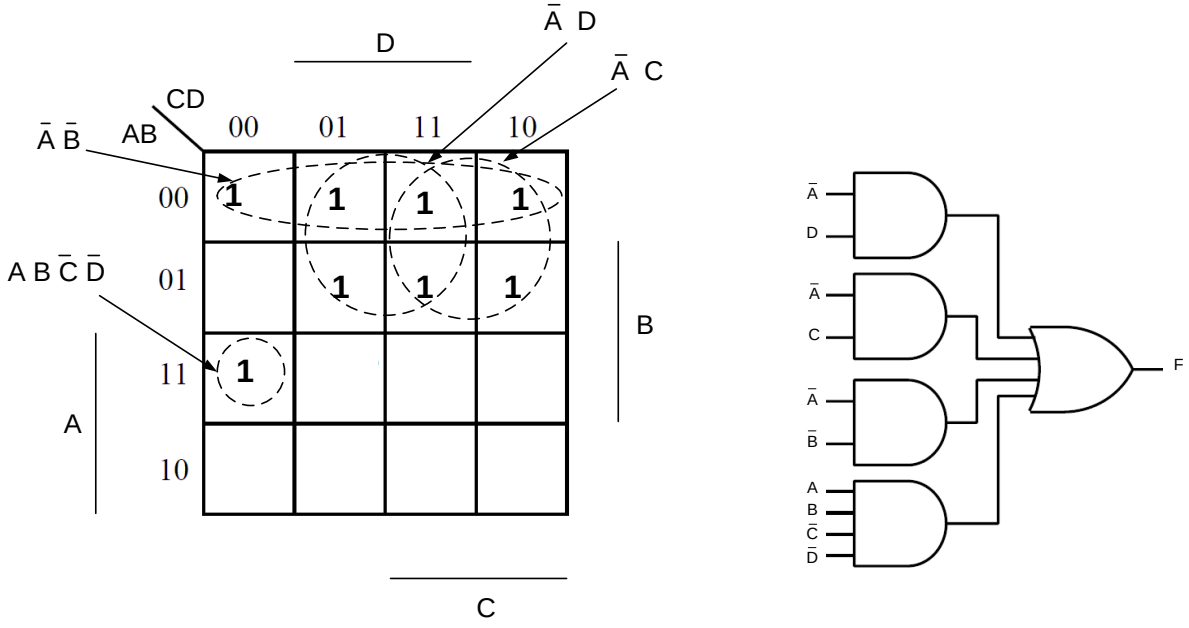
Figura 53 – Porta XOR como um inversor controlado

Portanto, a porta XOR pode ser considerada um inversor controlado. Quando $controle = 0$ o inversor está "desligado" e a entrada A aparece na saída. Quando $controle = 1$, o inversor está "ligado" e a saída é $\overline{A}$.

Essa interpretação pode ser utilizada para compreender o método de Sasao, que propõe um relaxamento nos agrupamentos e uma compensação. Caso o mapa de Karnaugh tenha uma quadra ou um octeto, por exemplo, faltando um dos termos, é possível efetuar o agrupamento mesmo assim (relaxamento), sendo que os erros cometidos também são agrupados. Quando se conectam os agrupamentos em uma XOR, ela fornece a saída desejada. Como exemplo, é considerado o projeto da função booleana $F(A, B, C, D) = \sum m(0, 1, 2, 3, 5, 6, 7, 12)$. A análise pelo mapa Karnaugh e o circuito projetado estão representados na Figura 54.

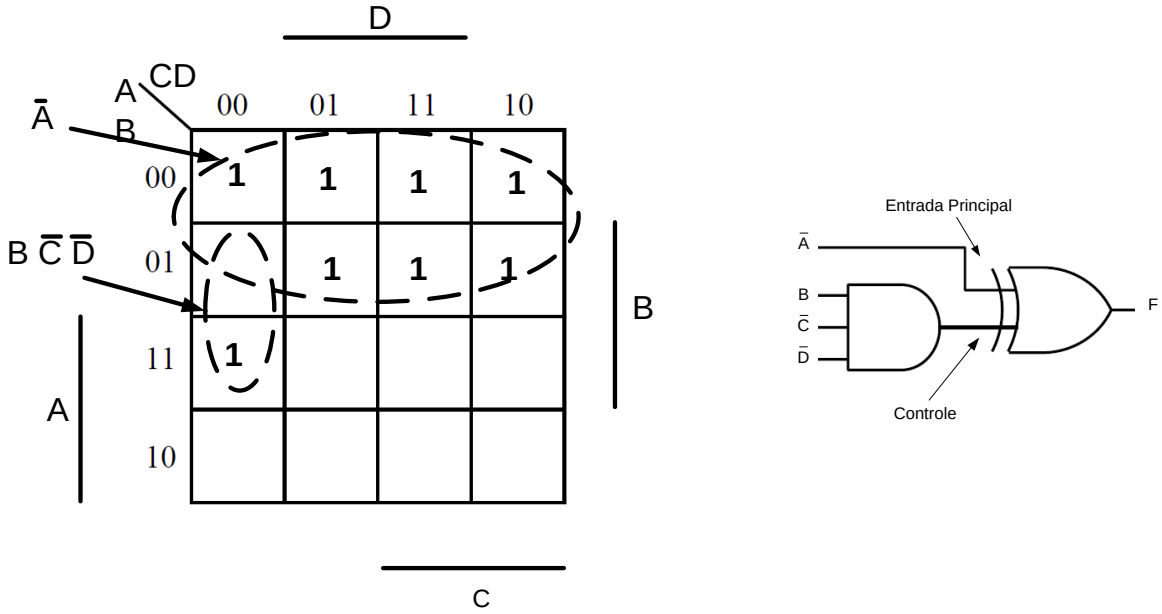
A Figura 55(a) ilustra o método de Sasao, onde observa-se que mesmo faltando um mintermo o octeto é formado e, dessa forma, o mapa fica com 2 mintermos que são denominados aqui de "incorretos"[3], os quais também são agrupados. Os dois agrupamentos são conectados através de uma porta XOR, conforme mostra a Figura 55(b).

[3] Considera-se como incorretos o termo agrupado indevidamente e o termo não agrupado, ambos no exemplo descrito formam uma dupla.



(a) $F = \sum m(0, 1, 2, 3, 5, 6, 7, 12)$ (b) Circuito projetado

Figura 54 – Agrupamentos do mapa Karnaugh e circuito relacionado



(a) Mapa Karnaugh modificado (b) Circuito projetado

Figura 55 – Implementação proposta por Sasao (SASAO, 1993)

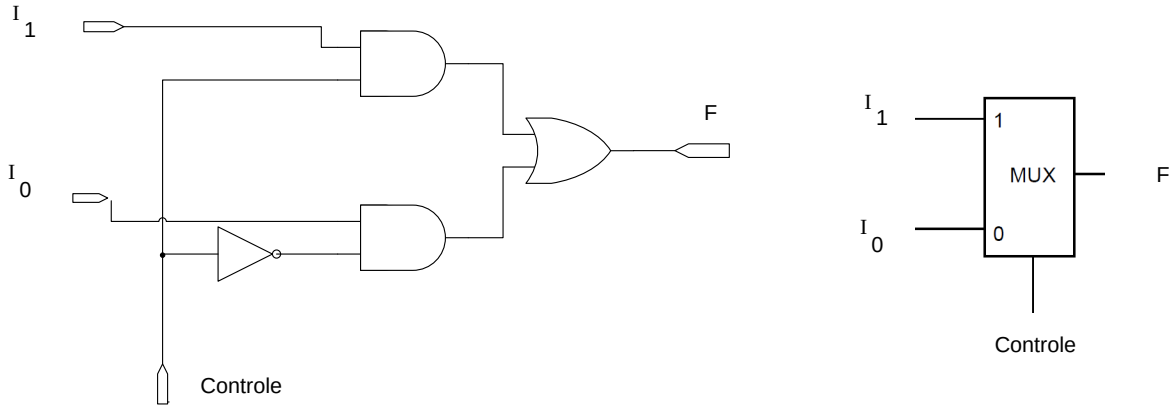
Quando se conecta através de uma porta XOR o agrupamento formado pelo octeto e pelos termos “incorretos”, o circuito pode ser analisado em função dos mintermos e do controle:

- $m_0, m_1, m_2, m_3, m_5, m_6, m_7$: a entrada de controle está desativada ($controle = 0$) e a saída assume o valor da entrada principal, ou seja, $\overline{A}$.
- m_4: o controle fica ativo e, com isso, a saída assume o valor da entrada principal complementada, ou seja, A. Observa-se que foi feita a correção.
- m_{12}: o controle fica ativo e a saída assume o valor da entrada principal complementada, ou seja, A. Observa-se que foi feita a correção.

A.3 PROJETO DE CLCs COM MULTIPLEXADORES

Outra forma de projetar CLCs é construindo uma rede de multiplexadores (árvore) através da decomposição de Shannon (SHANNON, 1949).

Um multiplexador de duas entradas é um bloco funcional com duas entradas de dados binários I_0, I_1, uma entrada de *controle* e uma saída denominada F, conforme Figura 56(b). Esse módulo implementa a função $F = I_1 \cdot controle + I_0 \cdot \overline{controle}$. O multiplexador pode ser considerado uma chave seletora, pois quando $controle = 1$ a entrada I_1 se conecta na saída; quando $controle = 0$, a entrada I_0 se conecta na saída. O circuito interno de um multiplexador de 2 entradas é mostrado na Figura 56(a). É possível implementar um multiplexador de 2^n entradas de dados; para isso são necessárias n entradas de controle.



(a) Circuito interno (b) Símbolo do multiplexador

Figura 56 – Multiplexador de duas entradas de dados

Qualquer projeto de CLC pode ser realizado utilizando uma rede de multiplexadores. Esse tipo de implementação baseia-se na aplicação repetida da decomposição de Shannon (SHANNON, 1949; JR, 1978) descrita como:

$$F(A, B, C, D) = F(A, B, C, 1) \cdot D + F(A, B, C, 0) \cdot \overline{D}.$$

Essa decomposição pode ser interpretada como uma divisão da expressão em relação à variável D em duas partes: $D = 1$, correspondente ao primeiro termo da equação e $D = 0$, correspondendo ao segundo termo. Colocando-se a variável D no controle pode-se implementar essa decomposição utilizando um multiplexador, conforme Figura 57.

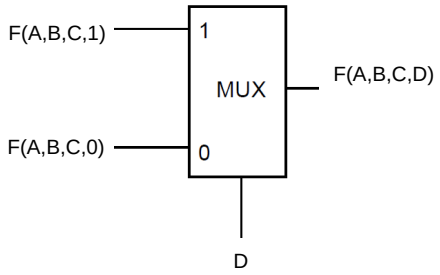


Figura 57 – Decomposição de Shannon (ERCEGOVAC; MORENO; LANG, 1998).

A decomposição de Shannon pode ser aplicada sucessivamente até que se tenha como entrada do multiplexador uma variável ou uma constante 0 ou 1, como exemplifica a Figura 58.

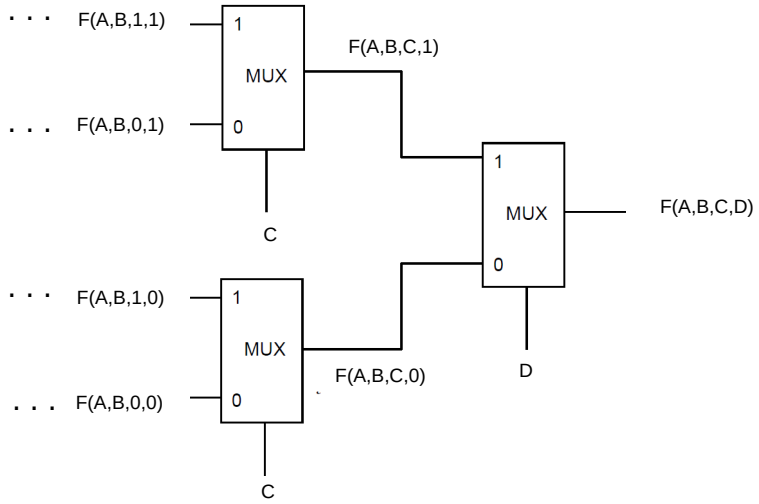
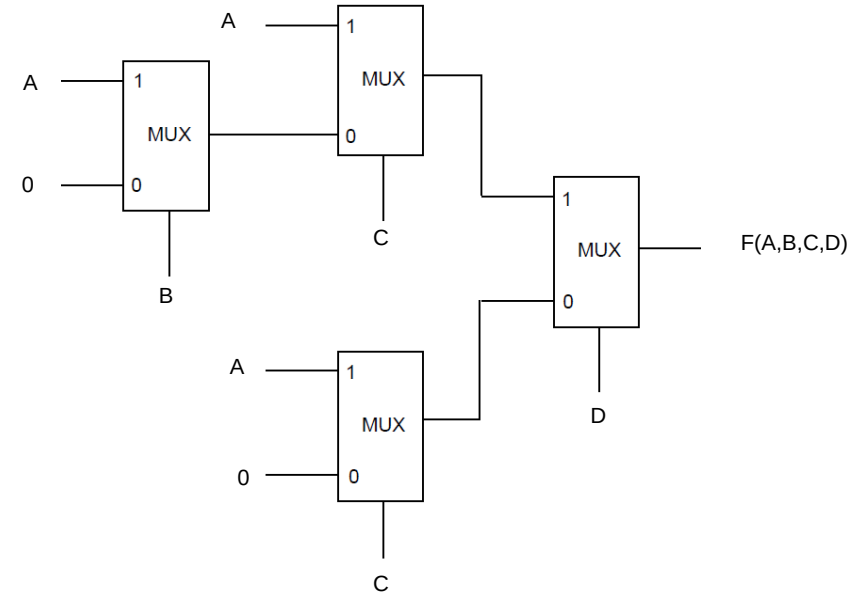


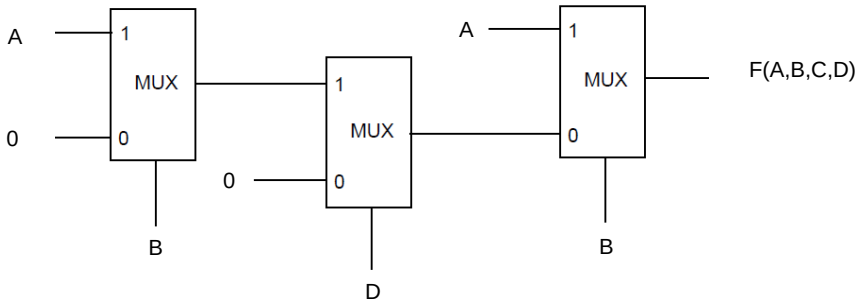
Figura 58 – Decomposição de Shannon em vários níveis (ERCEGOVAC; MORENO; LANG, 1998)

De acordo com a ordem em que é feita a decomposição, diferentes árvores são obtidas, não sendo possível determinar previamente qual a ordem das variáveis na decomposição que gera o circuito com menor número de multiplexadores (ERCEGOVAC;

MORENO; LANG, 1998). Como exemplo tem-se na Figura 59, duas árvores distintas que implementam a função: $F(A, B, C, D) = D \cdot (B + CA)$.



(a) Decomposição de Shannon - Solução 1



(b) Decomposição de Shannon - Solução 2

Figura 59 – Duas decomposições de Shannon para a mesma função (ERCEGOVAC; MORENO; LANG, 1998).

APÊNDICE B – Algoritmo Genético AGMUX

B.1 ALGORITMO GENÉTICO PROPOSTO - AGMUX

Um algoritmo genético denominado AGMUX (MANFRINI; BARBOSA; BERNARDINO, 2014) é proposto. O AGMUX tem como objetivo minimizar o número de elementos lógicos de um circuito combinacional. A diferença principal desse algoritmo para as diversas metaheurísticas relatadas pela literatura é a codificação utilizada. Desde a publicação de Louis (LOUIS, 1993) em 1993, onde foi proposta uma representação matricial, diversos trabalhos utilizam a mesma representação para a codificação do circuito (COELLO; CHRISTIANSEN; AGUIRRE, 2000b; KARAKATIC; PODGORELEC; HERICKO, 2013; COELLO; LUNA; AGUIRRE, 2003; MILLER; THOMSON; FOGARTY, 1997; COELLO; CHRISTIANSEN; AGUIRRE, 2000a; GARCÍA; COELLO, 2002; COELLO et al., 2000; COELLO; CHRISTIANSEN; AGUIRRE, 1997; COELLO; AGUIRRE, 2002; ALBA et al., 2007; MILLER; JOB; VASSILEV, 2000; MOORE; VENAYAGAMOORTHY, 2005; ANJOMSHOA; MAHANI, 2013; ANJOMSHOA; MAHANI; SADEGHIFARD, 2014).

A codificação proposta é baseada na inclusão de um multiplexador na saída do circuito. Ao invés de utilizar apenas uma matriz conectou-se um multiplexador e este por sua vez, fornece a saída do circuito. Foram utilizados multiplexadores com duas entradas de dados e multiplexadores com quatro entradas de dados, que serão denominados MUX2 e MUX4, respectivamente.
A Figura 60 mostra os 3 sub-circuitos, denominados circuito 0, circuito 1 e circuito 2, conectados às duas entradas de dados e à entrada de controle do MUX2, respectivamente.

O acoplamento do MUX2 na saída do circuito possibilita, através da entrada de controle (circuito 2), a divisão do circuito em duas partes independentes, e o sistema

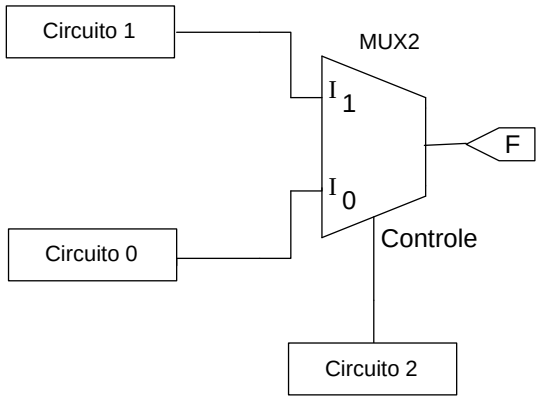
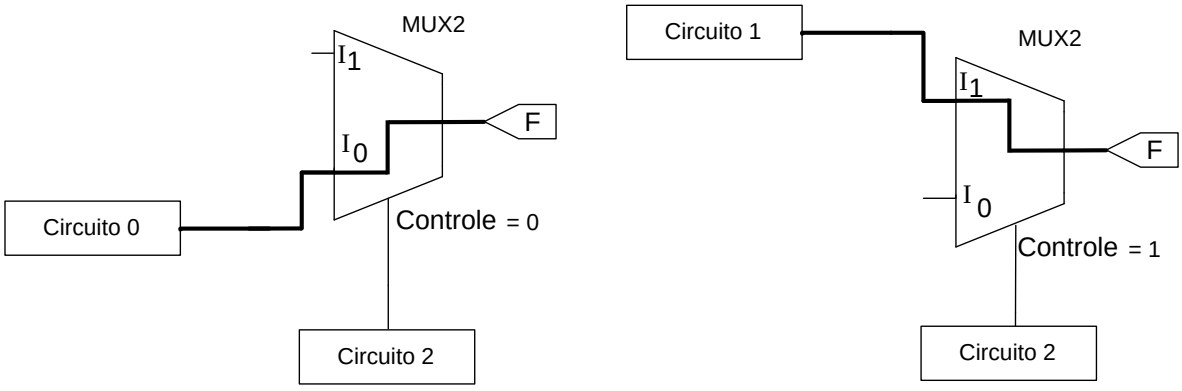


Figura 60 – Representação do circuito, composto de três sub-circuitos ligados ao MUX2. Os circuitos 0 e 1 são conectados nas entradas de dados I_0 e I_1 respectivamente, o circuito 2 é conectado na entrada de controle e F fornece a saída final do circuito.

funciona da seguinte forma:

- Quando a saída do circuito de controle for "zero", a entrada de dados I_0 do MUX2 é selecionada e o circuito 0 é ativado, conforme mostra a Figura 61(a).
- Quando a saída do circuito de controle for "um", a entrada de dados I_1 do MUX2 é selecionada e o circuito 1 é ativado, conforme mostra a Figura 61(b).



(a) Sub-circuito 0 conectado a saída F

(b) Sub-circuito 1 conectado a saída F

Figura 61 – Funcionamento do multiplexador como uma chave seletora, quando Controle= 0 a saída F é conectada no circuito 0; quando Controle = 1 a saída F é conectada ao circuito 1.

O fato do multiplexador está fixo na saída do circuito permite que a evolução ocorra em 3 sub-circuitos, explorando dessa forma uma região do espaço não considerada pela codificação de Louis (LOUIS, 1993).

A análise da influência do MUX2 pode ser feita através da tabela verdade considerando cada linha da referida tabela como uma restrição a ser atendida, conforme mostra a Tabela 62. Essa configuração pode ser interpretada como uma divisão das restrições, onde o circuito de controle seleciona quais restrições deverão ser atendidas pelo circuito 0 e quais restrições serão satisfeitas pelo circuito 1. Com isso, dois circuitos que atendam parcialmente as restrições impostas pela tabela verdade podem se auto-complementarem através do MUX2 e atender totalmente a todas as restrições. Em termos de computação evolucionista, esse raciocínio pode ser interpretado como dois indivíduos com uma "baixa" aptidão podem originar um indivíduo com uma alta aptidão, quando se considera o MUX2 acoplado na saída do circuito.

Uma representação mais detalhada, onde os sub-circuitos representados por uma matriz 3x3 são conectados ao MUX2, pode ser vista no exemplo da Figura 63. Cada

A	B	C	D	Control	I_0	I_1	F	
0	0	0	0	0	1	-	1	I_0
0	0	0	1	0	0	-	0	
0	0	1	0	1	-	0	0	I_1
0	0	1	1	1	-	1	1	
⋮				⋮		⋮		

Figura 62 – Exemplo de decomposição da tabela verdade. Quando a entrada de controle é 0 a saída do circuito I_0 é conectada na saída F. Quando a entrada de controle tem o valor 1 a saída do circuito I_1 é conectado na saída F.

elemento da matriz pode ser uma porta lógica ou um fio (WIRE), cuja posição está identificada pelos números de 5 a 13 (camada intermediária). As saídas das portas lógicas 11, 12 e 13 (saídas dos sub-circuitos) estão ligadas e fixadas diretamente às entradas do MUX2, enquanto as demais (5 a 8), podem ser conectadas às entradas das portas das colunas subsequentes, como por exemplo, a ligação da saída 5 com as entradas 8 e 12.

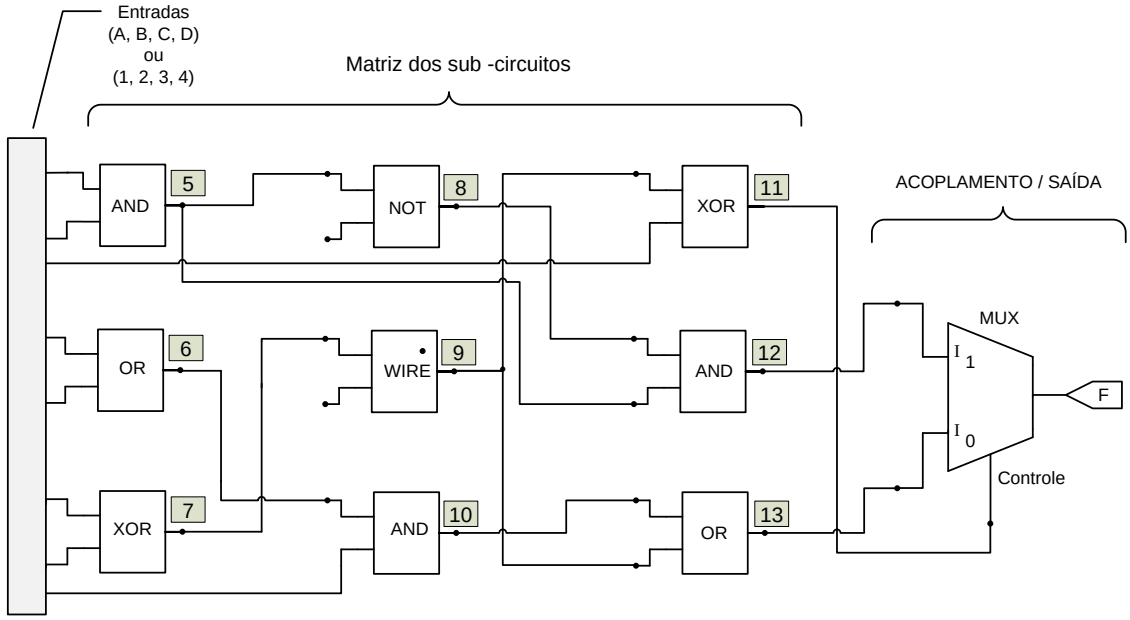


Figura 63 – Exemplo dos sub-circuitos (matriz 3x3) acoplado a um multiplexador. A célula 11 é conectada na entrada de controle do MUX, as células 12 e 13 são conectadas nas entradas de dados I_1 e $I0$ respectivamente. A saída F do MUX fornece a saída final do circuito.

Para a implementação do algoritmo genético, faz-se necessário a representação do circuito como um cromossomo, para isso as portas lógicas foram codificadas conforme a Figura 64. As entradas das portas lógicas podem ser as próprias variáveis de entradas, bem como a saída de outras portas lógicas. Essas conexões foram codificadas conforme a Figura 65, onde os números de 1 a 4 representam as variáveis de entradas (A, B, C, D) e

os números de 5 a 10 representam as saídas das portas lógicas de acordo com a posição mostrada na Figura 63.

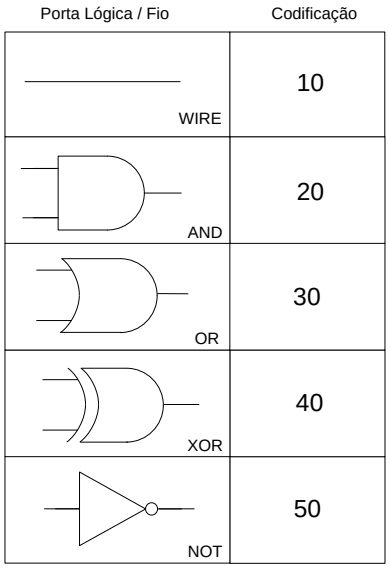


Figura 64 – Codificação utilizada para as portas lógicas

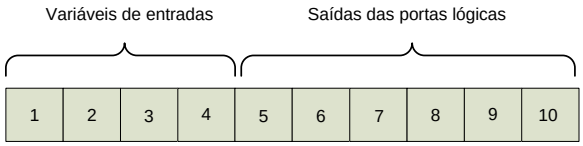


Figura 65 – Codificação utilizada para as entradas das portas lógicas

Considerando a codificação apresentada nas Figuras 64 e 65 pode-se definir um gene (cada elemento da matriz) como um vetor composto por três valores representando as entradas das portas lógicas, bem como a própria porta lógica. A Figura 66(a) mostra o exemplo de um gene com as duas entradas e a porta associada; na parte b tem-se a mesma informação, representada na forma de um circuito e na parte c tem-se o circuito associado ao gene (fenótipo).

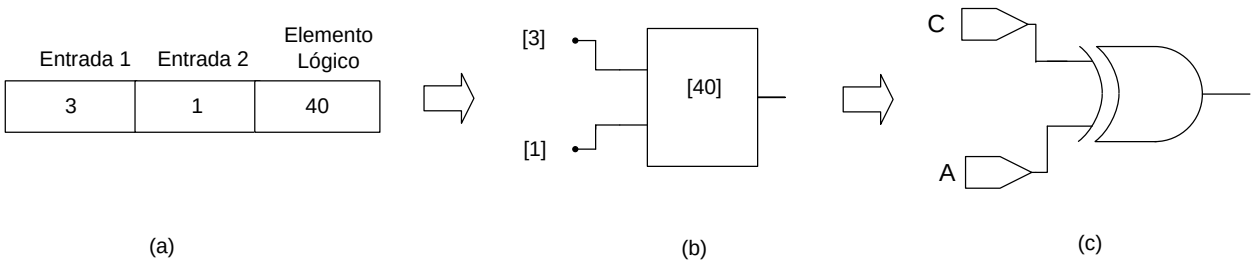


Figura 66 – genótipo e fenótipo

Em um gene relacionado a uma porta NOT ou um WIRE (que possuem apenas uma entrada), será desconsiderada a entrada 2 do gene, conforme mostra a Figura 67.

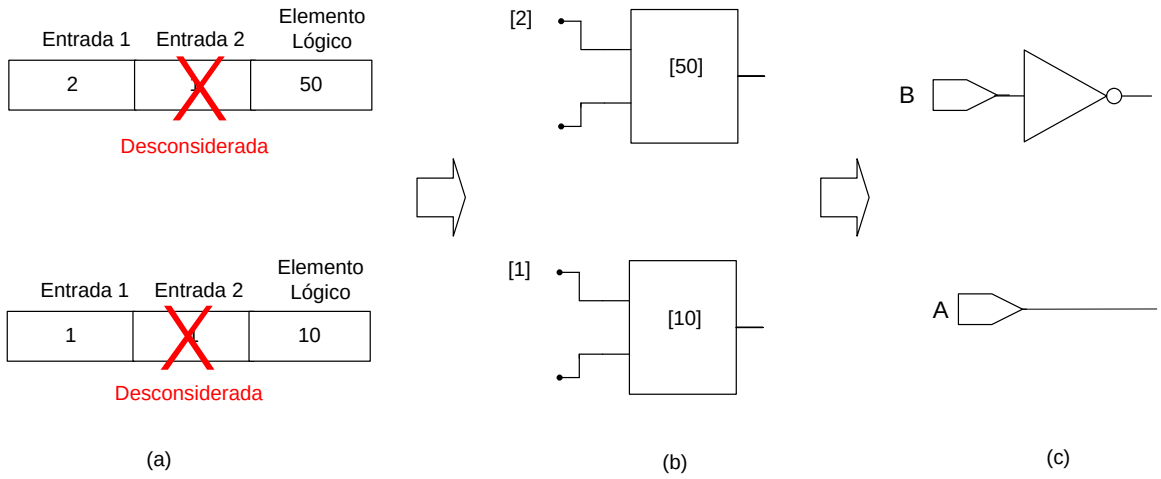


Figura 67 – gene de uma porta not e de um fio

Para garantir que todos os indivíduos gerados sejam factíveis, é necessário que cada elemento do gene tenha uma faixa restrita de valores, também denominada alelos. Essa faixa de valores possíveis será indicada conforme a Figura 68, ou de uma maneira compacta conforme a Figura 69.

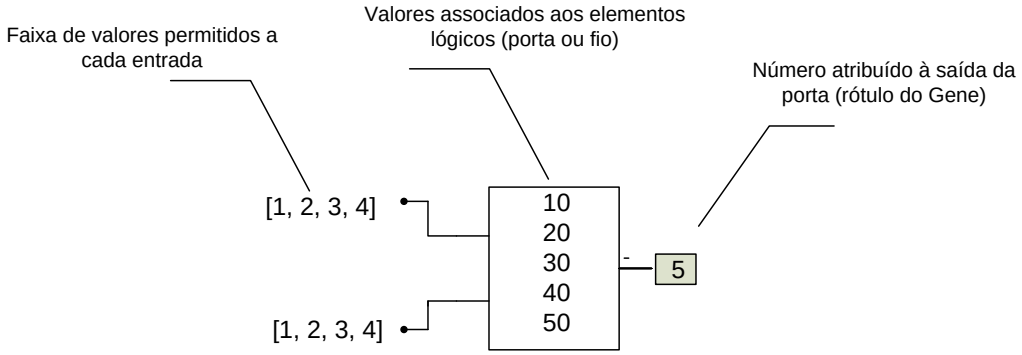


Figura 68 – faixa de valores permitido a cada entrada de porta

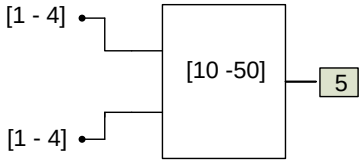


Figura 69 – Notação compacta para faixa de valores do gene

A representação detalhada da codificação proposta, incluindo os valores possíveis para cada elemento do gene, é mostrada na Figura 70. Cada gene pode ser classificado como pertencente aos níveis 1, 2 ou 3, e para que todos os indivíduos gerados sejam factíveis são necessárias as seguintes condições:

- As entradas de cada porta do nível 1 só poderão assumir os valores relacionados às variáveis de entrada da tabela verdade.
- A $entrada1$ de cada porta do nível i ($i > 1$) deve estar conectada às saídas das portas do nível $(i - 1)$, sendo que todas as saídas do nível $(i - 1)$ têm de estar conectadas a alguma entrada de porta do nível i. Isso pode ser feito através de uma permutação dos valores, ou seja, uma das entradas dos genes do nível i receberão uma permutação dos valores das saídas do nível $(i - 1)$. Como exemplo, pode-se identificar na Figura 70 que as $entradas1$ dos genes do nível 2 recebem a permutação de 5, 6 e 7.
- A $entrada2$ e cada porta do nível i ($i > 1$) pode assumir qualquer valor relacionado às variáveis de entradas ou às saídas do nível $(i - 1, i - 2, ...1)$. Como exemplo, pode-se identificar na Figura 70 que as $entrada2$ dos genes do nível 2 receberão qualquer valor inteiro entre 1 e 7.

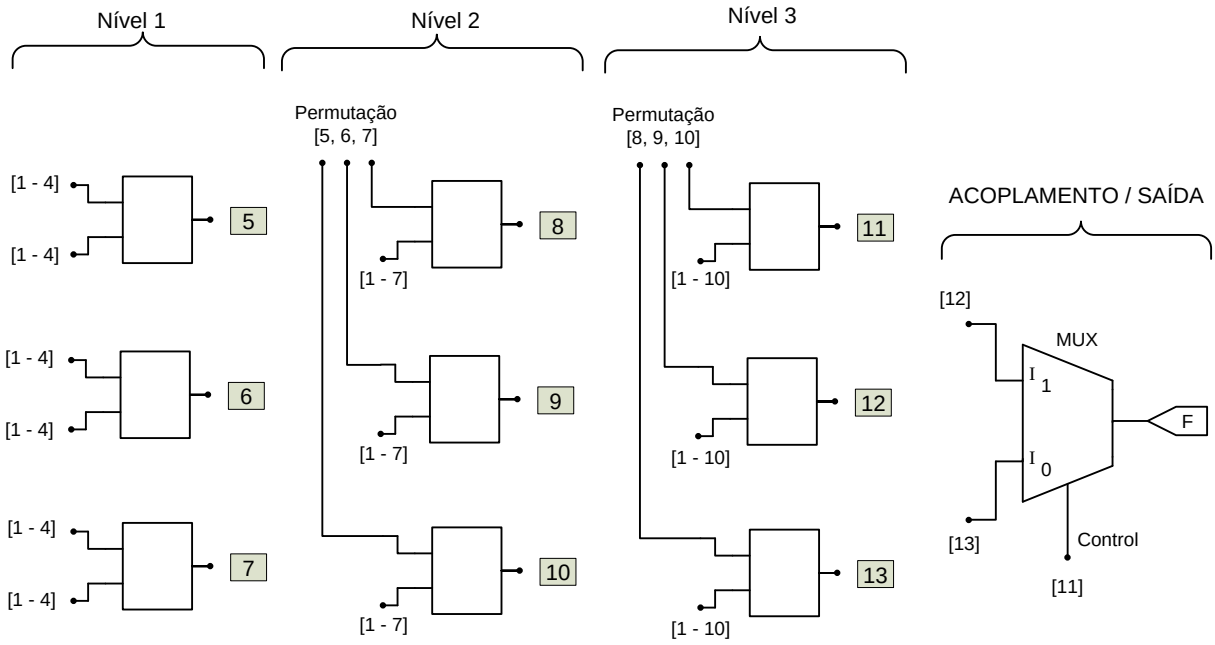
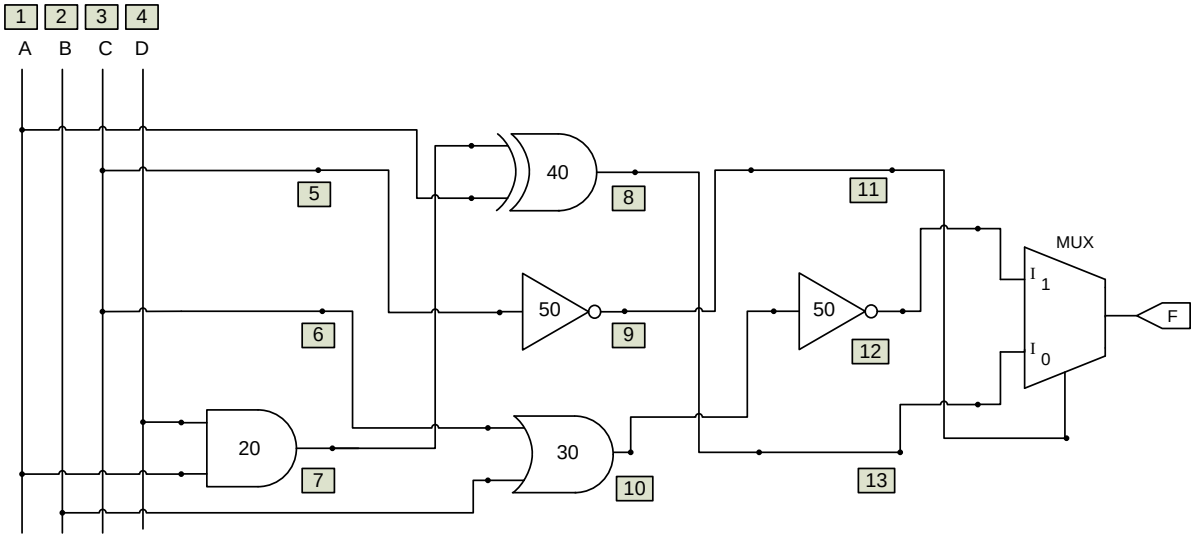


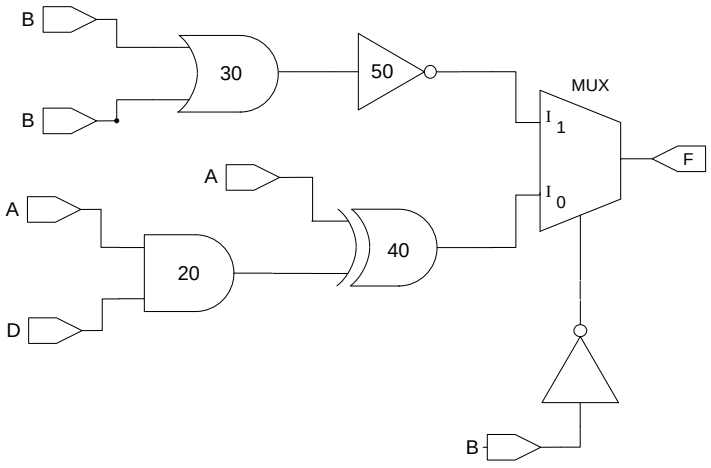
Figura 70 – Representação do circuito. Uma matriz 3 x 3 onde cada elemento é uma porta lógica é acoplada a um multiplexador, o qual fornece a saída final do circuito denominada F.

```
3    2    10    7    1    40    9    7    10
3    3    10    5    2    50    10   2    50
4    1    20    6    2    30    8    2    10
```

(a) Exemplo de Cromossomo



(b) Circuito obtido através da decodificação do cromossomo (a)



(c) Circuito (b) com a topologia ajustada (redesenhado)

Figura 71 – Processo de decodificação do croossomo (genótipo - fenótipo)

B.1.1 **Processo de decodificação do cromossomo**

Utilizando o software Matlab e com base nos critérios definidos anteriormente, um cromossomo gerado com valores randômicos é mostrado na Figura 71 (a). A obtenção do circuito relacionado a um cromossomo pode ser vista detalhadamente na Figura 71(b). Na Figura 71(c) tem-se o mesmo circuito redesenhado de forma mais simples.

B.1.2 **Codificação usando multiplexador com 4 entradas de dados**

A mesma codificação utilizada para um MUX2 pode ser extrapolada para um MUX4. Nesse caso, o tamanho da matriz acoplada ao MUX4 passa a ser 6x6, gerando seis sub-circuitos ligados ao MUX4, sendo dois deles ligados às entradas de seleção e 4 sub-circuitos conectados nas entradas de dados, conforme Figura 72.

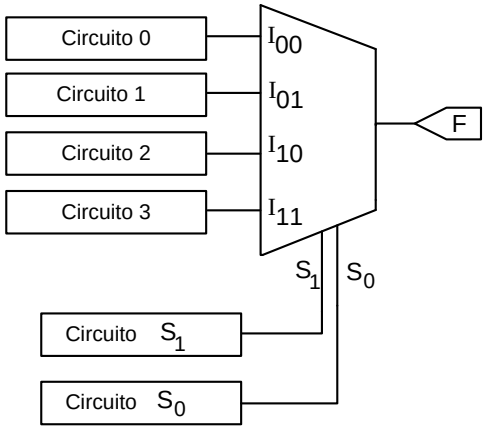


Figura 72 – Representação do circuito, composto por 6 sub-circuitos ligados ao MUX4. As entradas I_{00}, I_{01}, I_{10}, I_{11} são as entradas de dados, S_1, S_0 são as entradas de controle e F é a saída do MUX4.

B.1.3 **Função Aptidão**

O objetivo do algoritmo proposto é projetar um CLC que atenda a todas as restrições impostas pela tabela verdade, com o menor número de elementos lógicos. Considerando a matriz acoplada a um MUX2, o melhor circuito também pode ser encontrado quando maximizamos a quantidade de *wires* (representa a ausência de portas lógicas) presente na matriz e maximizamos a quantidade de acertos da tabela verdade. Portanto, a qualidade de cada indivíduo pode ser determinada pela função aptidão, conforme a equação (B.1).

$$f(acertos, wires) \quad = P_1 \cdot acertos + P_2 \cdot wires \tag{B.1}$$

onde:

acertos : a quantidade de restrições que o indivíduo satisfaz.

$wires$: número de fios.

P_1 e P_2 : são os pesos dos parâmetros *acertos* e $wires$, respectivamente.

Para que o mecanismo de busca priorize que sejam encontrados circuitos viáveis (representem com fidelidade a tabela verdade) e, posteriormente, otimize a quantidade de elementos lógicos nos circuitos, é preciso que P_1 seja maior que P_2 na seguinte proporção:

$$P_1 > cel.P_2 \tag{B.2}$$

onde *cel* é número de elementos da matriz.

De acordo com as equações B.1 e B.2, a pontuação referente ao acerto de uma linha da tabela verdade é maior do que a pontuação se todas as células da matriz forem formadas por WIRES, dessa forma o algoritmo sempre irá priorizar que o indivíduo acerte todas as linhas da tabela verdade. Somente após atingir a região factível o processo evolutivo irá buscar priorizar o número de WIRES.

B.1.4 **Seleção**

Os indivíduos que participarão da reprodução são escolhidos através da Seleção por Torneio, com $k = 2$. A cada geração são feitos 4 torneios onde são selecionados 4 indivíduos (modelo *Steady State*).

B.1.5 **Mutação**

Após o processo de seleção, os indivíduos são submetidos ao processo reprodutivo. Nesse algoritmo optou-se por aplicar apenas o operador de mutação com uma taxa de 0.06.

B.1.6 **Inclusão e Exclusão dos indivíduos**

Os 4 filhos gerados pela mutação entram na população. Para que o tamanho da população mantenha-se constante, é feita a ordenação dos indivíduos baseada na aptidão, sendo que os 4 piores indivíduos são excluídos.

B.1.7 **Algoritmo AGMUX**

O processo do AGMUX pode ser visto no Algoritmo 2.

B.2 EXPERIMENTOS - AGMUX

Para validar o funcionamento do AGMUX (MANFRINI; BARBOSA; BERNARDINO, 2014), descrito na Seção B.1, o mesmo foi aplicado para resolver os problemas

Algoritmo 2: **AGMUX** Algoritmo Genético Implementado

```
1  Entrada
2      Função objetivo descrita na Equação (B.1);
3  Saída
4      Circuito Otimizado;
5  início
6      Gera a população inicial de indivíduos
7      Avalia a população inicial
8      repita
9          Selecionar através de torneio os indivíduos que irão participar da mutação;
10         para cada individuo selecionado faça
11             Aplicar o operador de mutação
12             Avaliar sua aptidão
13         fim
14         Inclui os novos indivíduos na população
15         Ordena a população
16         Elimina os piores indivíduos da população
17     até número máximo de gerações atingido;
18     Devolver o melhor indivíduo da ultima geração (circuito otimizado);
19 fim
```

propostos pela literatura que possuem 4 e 5 variáveis de entradas e uma saída. Os resultados foram comparados com outras meta-heurísticas e técnicas tradicionais de projeto. No exemplo 1, a verificação do circuito projetado será feita de forma detalhada e o mesmo raciocínio pode ser estendido aos demais exemplos.

B.2.1 **Exemplo 1**

O objetivo desse exemplo, proposto inicialmente por (COELLO; CHRISTIANSEN; AGUIRRE, 2000b), é encontrar um circuito otimizado, com o menor número de portas lógicas, e que tenha os seguintes mintermos: [1]

$$F(A, B, C, D) = \sum m(0, 1, 3, 6, 7, 8, 10, 13)$$

O resultado obtido com o AGMUX é apresentado na Figura 73.

[1] Os mintermos estão definidos na Seção A.0.1.

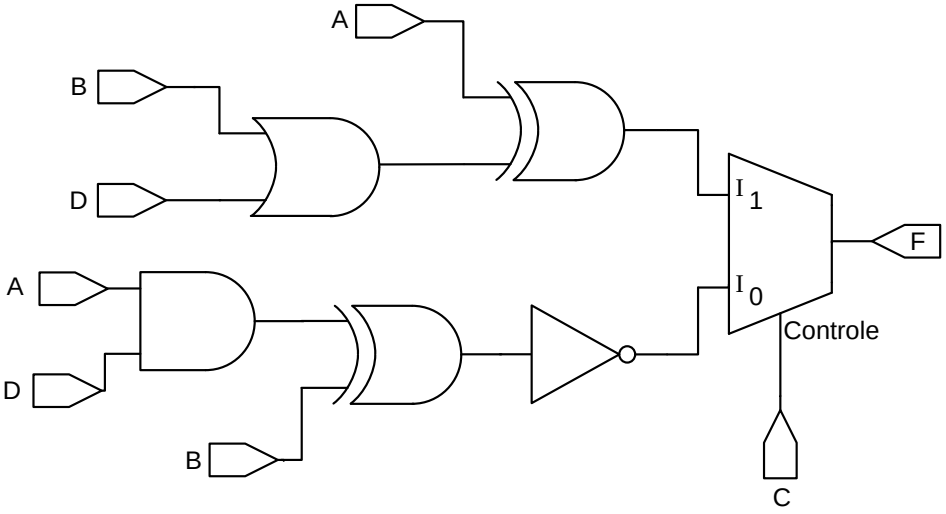


Figura 73 – Melhor solução encontrada pelo AGMUX para o exemplo 1. Controle, I_0 e I_1 são as entradas do MUX as quais estão conectadas nos respectivos sub-circuitos. A saída do MUX é denotada por F.

A verificação se o funcionamento do circuito projetado condiz com a saída desejada pode ser feita analisando detalhadamente a Tabela 11. As colunas mostram na sequência as variáveis de entrada (A, B, C, D), as saídas dos circuitos ligados ao MUX (*controle*, I_0 e I_1) e a saída do próprio MUX (F). A coluna da tabela referente ao *controle* seleciona qual circuito será conectado na saída F e se comporta da seguinte forma.

- $Controle = 0 \begin{cases} I_0 \text{ se conecta na saida F,} \\ I_1 \text{ é irrelevante (-)} \end{cases}$
- $Controle = 1 \begin{cases} I_1 \text{ se conecta na saida F} \\ I_0 \text{ é irrelevante (-)} \end{cases}$

Dessa forma, analisando as entradas do MUX, verifica-se que a saída F referente ao circuito mostrado na Figura 73 corresponde aos mintermos especificados.

A comparação dos melhores circuitos encontrados na literatura, que atendem as especificações desse projeto com o método proposto, está mostrada na Tabela 12. Verifica-se que a codificação proposta adicionou um MUX2 ao circuito e diminuiu a quantidade de portas lógicas. Ao considerar um MUX2 e as portas lógicas como "um elemento lógico", assim como em Miller (MILLER; THOMSON; FOGARTY, 1997), constata-se que o AGMUX diminuiu a quantidade de elementos lógicos.

Tabela 11 – Tabela verdade para o Exemplo 1. As colunas A B C D representam as diversas possibilidades das variáveis de entradas. Controle, I_0 e I_1 fornecem as saídas dos sub-circuitos ligados ao MUX, o qual fornece a saída final denotada por F.

Mintermo	A	B	C	D	Controle	I_0	I_1	F
0	0	0	0	0	0	1	-	1
1	0	0	0	1	0	1	-	1
2	0	0	1	0	1	-	0	0
3	0	0	1	1	1	-	1	1
4	0	1	0	0	0	0	-	0
5	0	1	0	1	0	0	-	0
6	0	1	1	0	1	-	1	1
7	0	1	1	1	1	-	1	1
8	1	0	0	0	0	1	-	1
9	1	0	0	1	0	0	-	0
10	1	0	1	0	1	-	1	1
11	1	0	1	1	1	-	0	0
12	1	1	0	0	0	0	-	0
13	1	1	0	1	0	1	-	1
14	1	1	1	0	1	-	0	0
15	1	1	1	1	1	-	0	0

Tabela 12 – Comparação dos melhores resultados encontrados na literatura para o exemplo 1. Número de componentes lógicos do melhor circuito projetado por diversos algoritmos.

Algoritmo	Tamanho
Algoritmo Proposto (AGMUX)	5 portas , 1MUX
Programação Genética (KARAKATIC; PODGORELEC; HERICKO, 2013)	7 portas
Híbrido (GASA2) (COELLO; ALBA; LUQUE, 2003) Algoritmo Genetico e Recozimento Simulado	7 portas
Colônia de formigas (GARCÍA; COELLO, 2002)	7 portas
Algoritmo Genético (MGA) (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	8 portas
Algoritmo Genético (NGA) (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	10 portas
Projetista Humano 1 (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	11 portas
Projetista Humano 2 (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	12 portas
SASAO (SASAO, 1993) Técnica baseada em ANDs, XOR	12 portas

B.2.2 **Exemplo 2**

O objetivo desse exemplo, proposto inicialmente por (COELLO; CHRISTIANSEN; AGUIRRE, 2000b), é encontrar um circuito otimizado que tenha os seguintes mintermos:

$$F(A, B, C, D) = \sum m(0, 4, 5, 6, 7, 8, 9, 10, 13, 15)$$

O resultado obtido com o AGMUX é apresentado na Figura 74 e nas Tabelas 13 e 14.

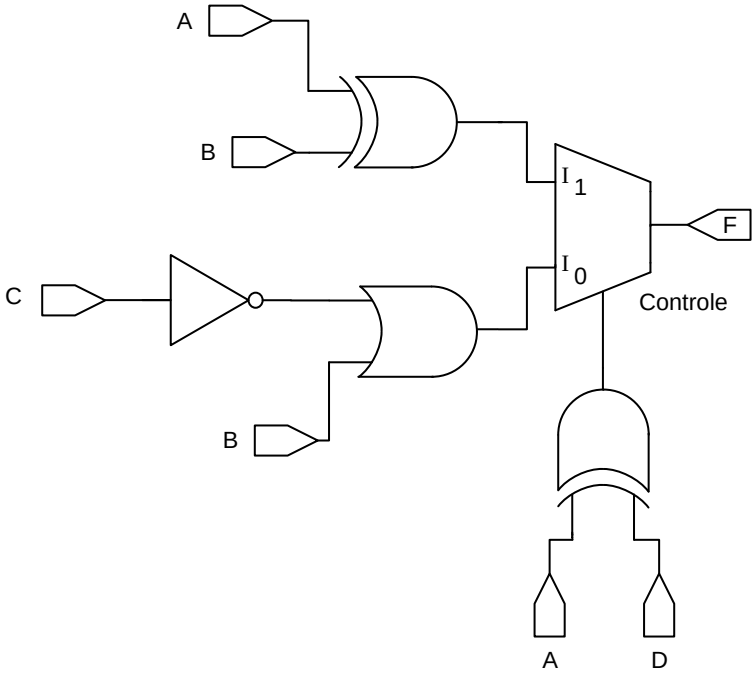


Figura 74 – Melhor solução encontrada pelo AGMUX para o exemplo 2

Tabela 13 – Tabela verdade para o Exemplo 2

Mintermo	A	B	C	D	Controle	I_0	I_1	F
0	0	0	0	0	0	1	-	1
1	0	0	0	1	1	-	0	0
2	0	0	1	0	0	0	-	0
3	0	0	1	1	1	-	0	0
4	0	1	0	0	0	1	-	1
5	0	1	0	1	1	-	1	1
6	0	1	1	0	0	1	-	1
7	0	1	1	1	1	-	1	1
8	1	0	0	0	1	-	1	1
9	1	0	0	1	0	1	-	1
10	1	0	1	0	1	-	1	1
11	1	0	1	1	0	0	-	0
12	1	1	0	0	1	-	0	0
13	1	1	0	1	0	1	-	1
14	1	1	1	0	1	-	0	0
15	1	1	1	1	0	1	-	1

Tabela 14 – Comparação dos melhores resultados encontrados na literatura para o exemplo 2

Algoritmo	Tamanho
Algoritmo Proposto (AGMUX) (MANFRINI; BARBOSA; BERNARDINO, 2014)	4 portas , 1MUX
Colônia de Formigas (GARCÍA; COELLO, 2002)	7 portas
Algoritmo Genético (MGA) (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	7 portas
Algoritmo Genético (NGA) (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	7 portas
Projetista Humano 1 (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	9 portas
Projetista Humano (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	10 portas

B.2.3 **Exemplo 3**

O objetivo desse exemplo, proposto inicialmente por (MOORE; VENAYAGAMOORTHY, 2005); é encontrar um circuito otimizado que tenha os seguintes mintermos:

$$F(A, B, C, D) = \sum m(2, 3, 8, 9, 11, 12, 13, 14)$$

O resultado obtido com o AGMUX é apresentado na Figura 75. Para o Exemplo 3 a

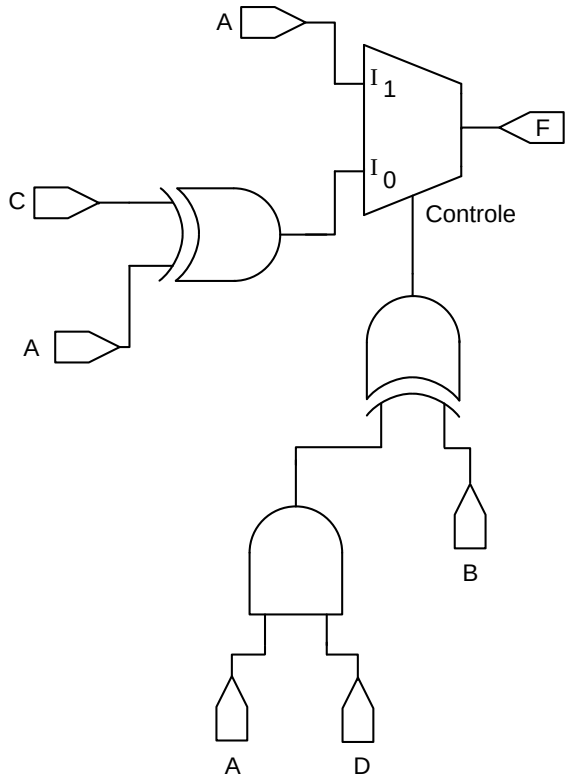


Figura 75 – Melhor solução encontrada pelo AGMUX para o exemplo 3

tabela verdade está mostrada na Tabela 15. Analisando as entradas do MUX, verifica-se que a saída F referente ao circuito apresentado na Figura 75 corresponde aos mintermos especificados. A comparação com os resultados da literatura encontra-se na Tabela 16

onde percebe-se que o método proposto foi capaz de reduzir o número de elementos lógicos.

Tabela 15 – Tabela Verdade para o exemplo 3

Mintermo	A	B	C	D	Controle	I_0	I_1	F
0	0	0	0	0	0	0	-	0
1	0	0	0	1	0	0	-	0
2	0	0	1	0	0	1	-	1
3	0	0	1	1	0	1	-	1
4	0	1	0	0	1	-	0	0
5	0	1	0	1	1	-	0	0
6	0	1	1	0	1	-	0	0
7	0	1	1	1	1	-	0	0
8	1	0	0	0	0	1	-	1
9	1	0	0	1	1	-	1	1
10	1	0	1	0	0	0	-	0
11	1	0	1	1	1	-	1	1
12	1	1	0	0	1	-	1	1
13	1	1	0	1	0	1	-	1
14	1	1	1	0	1	-	1	1
15	1	1	1	1	0	0	-	0

B.2.4 **Exemplo 4**

O objetivo desse exemplo, proposto inicialmente por (MOORE; VENAYAGAMOORTHY, 2005) é encontrar um circuito otimizado que tenha os seguintes mintermos:

$$F(A, B, C, D) = \sum m(0, 1, 4, 5, 6, 7, 10, 15)$$

O resultado obtido com o AGMUX é apresentado na Figura 76. Para o exemplo 4 a tabela verdade está mostrada na Tabela 17. Analisando as entradas do MUX, verifica-se

Tabela 16 – Comparação dos melhores resultados para o exemplo 3, encontrados na literatura utilizando diferentes meta-heurísticas

Algoritmo	Tamanho
Algoritmo Proposto (AGMUX)	3 portas, 1MUX
Híbrido (QEPSO) (MOORE; VENAYAGAMOORTHY, 2005) Evolução Quântica e PSO	5 portas
Enxame de Partículas (PSO) (MOORE; VENAYAGAMOORTHY, 2005)	5 portas
Algoritmo Genético (MGA) (MOORE; VENAYAGAMOORTHY, 2005)	5 portas

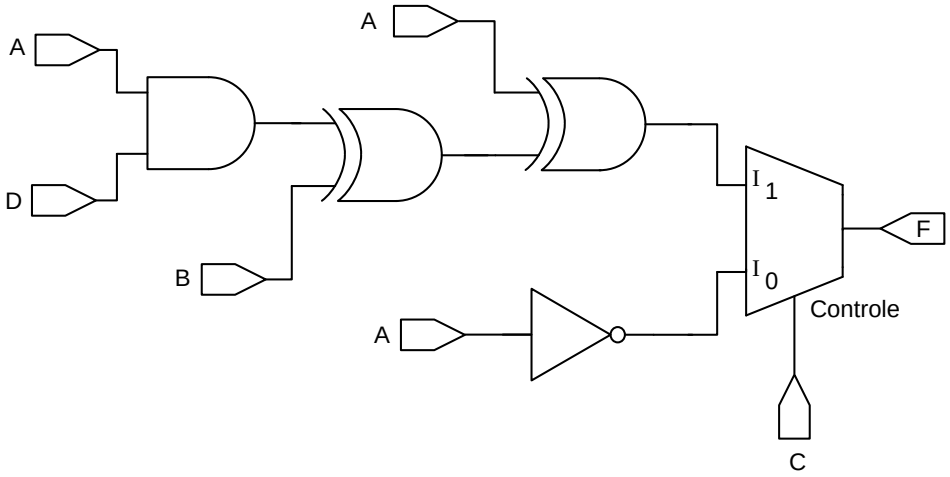


Figura 76 – Melhor solução encontrada pelo AGMUX para o exemplo 4

Tabela 17 – Tabela verdade para o exemplo 4

Minterm0	A	B	C	D	Controle	I_0	I_1	F
0	0	0	0	0	0	1	-	1
1	0	0	0	1	0	1	-	1
2	0	0	1	0	1	-	0	0
3	0	0	1	1	1	-	0	0
4	0	1	0	0	0	1	-	1
5	0	1	0	1	0	1	-	1
6	0	1	1	0	1	-	1	1
7	0	1	1	1	1	-	1	1
8	1	0	0	0	0	0	-	0
9	1	0	0	1	0	0	-	0
10	1	0	1	0	1	-	1	1
11	1	0	1	1	1	-	0	0
12	1	1	0	0	0	0	-	0
13	1	1	0	1	0	0	-	0
14	1	1	1	0	1	-	0	0
15	1	1	1	1	1	-	1	1

que a saída F referente ao circuito apresentado na Figura 76 corresponde aos mintermos especificados. A comparação com os resultados da literatura encontra-se na Tabela 18 onde percebe-se que o método proposto foi capaz de reduzir o número de elementos lógicos.

Tabela 18 – Comparação dos melhores resultados para o Exemplo 4, encontrados na literatura utilizando diferentes meta-heurísticas

Algoritmo	Tamanho
Algoritmo Proposto (AGMUX)	4 portas , 1MUX
Híbrido (QEPSO) (MOORE; VENAYAGAMOORTHY, 2005) Evolução Quântica e PSO	6 portas
Enxame de partículas(PSO) (MOORE; VENAYAGAMOORTHY, 2005)	6 portas
Projetista Humano (MOORE; VENAYAGAMOORTHY, 2005)	12 portas

B.2.5 **Exemplo 5**

O objetivo desse exemplo, proposto inicialmente por (COELLO; ALBA; LUQUE, 2003) é encontrar um circuito otimizado cuja saída F corresponda aos seguintes mintermos:

$$F(A, B, C, D, E) = \sum m(2, 3, 6, 7, 10, 11, 13, 15, 18, 19, 21, 23, 25, 27, 29, 31)$$

Para este exemplo o AGMUX utilizou MUX2 e MUX4 e os resultados obtidos são apresentados nas Figuras 77 e 78, respectivamente.

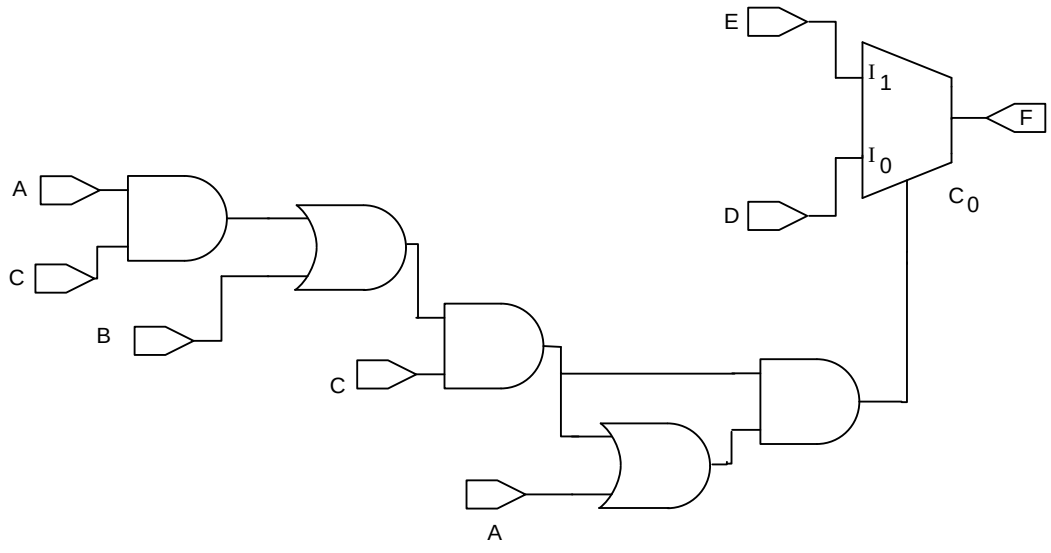


Figura 77 – Melhor solução encontrada pelo AGMUX utilizando MUX2 para o exemplo 5

A verificação se o funcionamento do circuito projetado condiz com a saída desejada, pode ser feita analisando detalhadamente a Tabela 19.

As colunas mostram as variáveis de entrada (A, B, C, D, E), as saídas dos circuitos ligados ao MUX2 (C_0, I_0 e I_1) e a saída do próprio MUX2 (F). Na sequência estão

Tabela 19 – Tabela Verdade referente ao Exemplo 5

mintermo	A	B	C	D	E	C_0	I_0	I_1	F	C_1	C_0	I_{00}	I_{01}	I_{10}	I_{11}	F
0	0	0	0	0	0	0	0	-	0	0	0	0	-	-	-	0
1	0	0	0	0	1	0	0	-	0	0	0	0	-	-	-	0
2	0	0	0	1	0	0	1	-	1	0	0	1	-	-	-	1
3	0	0	0	1	1	0	1	-	1	0	0	1	-	-	-	1
4	0	0	1	0	0	0	0	-	0	1	0	-	-	0	-	0
5	0	0	1	0	1	0	0	-	0	1	0	-	-	0	-	0
6	0	0	1	1	0	0	1	-	1	1	0	-	-	1	-	1
7	0	0	1	1	1	0	1	-	1	1	0	-	-	1	-	1
8	0	1	0	0	0	0	0	-	0	0	1	-	0	-	-	0
9	0	1	0	0	1	0	0	-	0	0	1	-	0	-	-	0
10	0	1	0	1	0	0	1	-	1	0	1	-	1	-	-	1
11	0	1	0	1	1	0	1	-	1	0	1	-	1	-	-	1
12	0	1	1	0	0	1	-	0	0	1	1	-	-	-	0	0
13	0	1	1	0	1	1	-	1	1	1	1	-	-	-	1	1
14	0	1	1	1	0	1	-	0	0	1	1	-	-	-	0	0
15	0	1	1	1	1	1	-	1	1	1	1	-	-	-	1	1
16	1	0	0	0	0	0	0	-	0	0	1	-	0	-	-	0
17	1	0	0	0	1	0	0	-	0	0	1	-	0	-	-	0
18	1	0	0	1	0	0	1	-	1	0	1	-	1	-	-	1
19	1	0	0	1	1	0	1	-	1	0	1	-	1	-	-	1
20	1	0	1	0	0	1	-	0	0	1	1	-	-	-	0	0
21	1	0	1	0	1	1	-	1	1	1	1	-	-	-	1	1
22	1	0	1	1	0	1	-	0	0	1	1	-	-	-	0	0
23	1	0	1	1	1	1	-	1	1	1	1	-	-	-	1	1
24	1	1	0	0	0	1	-	0	0	1	1	-	-	-	0	0
25	1	1	0	0	1	1	-	1	1	1	1	-	-	-	1	1
26	1	1	0	1	0	1	-	0	0	1	1	-	-	-	0	0
27	1	1	0	1	1	1	-	1	1	1	1	-	-	-	1	1
28	1	1	1	0	0	1	-	0	0	1	1	-	-	-	0	0
29	1	1	1	0	1	1	-	1	1	1	1	-	-	-	1	1
30	1	1	1	1	0	1	-	0	0	1	1	-	-	-	0	0
31	1	1	1	1	1	1	-	1	1	1	1	-	-	-	1	1

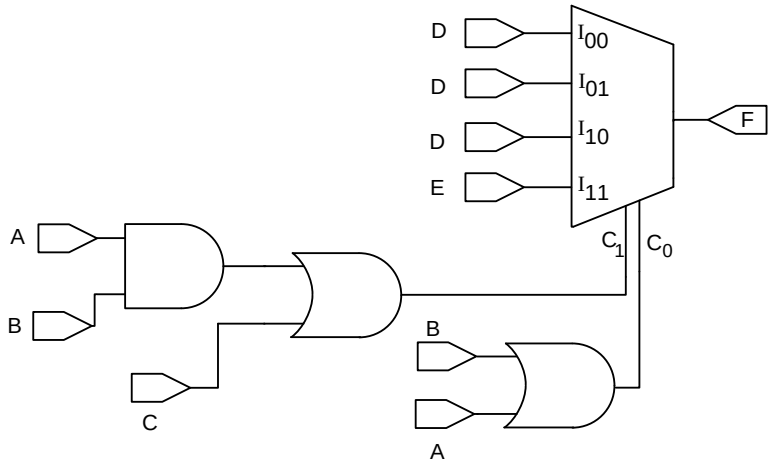


Figura 78 – Melhor solução encontrada pelo AGMUX utilizando MUX4 para o exemplo 5

colocadas as saídas dos circuitos conectados ao MUX4 (C_1, C_0, I_{00}, I_{01}, I_{10}, I_{11}) e a saída final do circuito, produzida pelo MUX4.

O resultado do circuito projetado com MUX2 pode ser confirmado, conforme os exemplos anteriores. A análise da tabela verdade e a verificação do circuito projetado com o MUX4 é feita da seguinte forma:

- $Controle : C_1 = 0$ e $C_0 = 0 \begin{cases} I_{00} \text{ se conecta na saida F,} \\ I_{01}, I_{10}, I_{11} \text{ são irrelevantes (-)} \end{cases}$
- $Controle : C_1 = 0$ e $C_0 = 1 \begin{cases} I_{01} \text{ se conecta na saida F,} \\ I_{00}, I_{10}, I_{11} \text{ são irrelevantes (-)} \end{cases}$
- $Controle : C_1 = 1$ e $C_0 = 0 \begin{cases} I_{10} \text{ se conecta na saida F,} \\ I_{00}, I_{01}, I_{11} \text{ são irrelevantes (-)} \end{cases}$
- $Controle : C_1 = 1$ e $C_0 = 1 \begin{cases} I_{11} \text{ se conecta na saida F,} \\ I_{00}, I_{01}, I_{10} \text{ são irrelevantes (-)} \end{cases}$

Dessa forma, analisando as entradas do MUX4, verifica-se que a saída F referente ao circuito mostrado nas Figuras 77 e 78 correspondem aos mintermos especificados.

A comparação dos melhores circuitos encontrados na literatura que atendem às especificações desse projeto com o método proposto está mostrada na Tabela 20. Verifica-se que a codificação proposta diminuiu a quantidade de elementos lógicos.

Tabela 20 – Comparação dos melhores resultados do exemplo 5 encontrados na literatura utilizando diferentes algoritmos

Algoritmo	Tamanho
Algoritmo Proposto 1 (AGMUX)	5 portas 1 MUX2
Algoritmo Proposto 2 (AGMUX)	3 portas 1 MUX4
Algoritmo Híbrido (GASA2) (COELLO; ALBA; LUQUE, 2003) Algoritmo Genético e Recozimento Simulado	9 portas
Algoritmo Genético (NGA) (COELLO; CHRISTIANSEN; AGUIRRE, 2000b)	10 portas
Método de Quine-McCluskey (MCCLUSKEY, 1956)	9

B.2.6 **Exemplo 6**

O objetivo desse exemplo é encontrar um circuito otimizado que tenha os seguintes mintermos:

$$F(A,B,C,D,E) = \sum m(0,1,2,4,7,8,11,13,14,15,16,19,21,22,23,25,26,27,28,29,30,31)$$

Para esse exemplo, o AGMUX utilizou MUX2 e MUX4 e os melhores resultados obtidos são apresentados nas Figuras 79 e 80, respectivamente.

Para o exemplo 6, a tabela verdade está mostrada na Tabela 21. Analisando as entradas do MUX2 e do MUX4, verifica-se que a saída F referente aos circuitos mostrados nas Figuras 79 e 80 correspondem aos mintermos especificados. A comparação com o método de Quine-McCluskey está mostrado na Tabela 22. Observa-se que o método proposto foi capaz de reduzir o número de elementos lógicos.

Tabela 22 – Comparação dos melhores resultados do exemplo 6 com o método de Quine McCluskey

Algoritmo	Tamanho
Algoritmo Proposto 1 (AGMUX)	10 portas 1 MUX2
Algoritmo Proposto 2 (AGMUX)	8 portas 1 MUX4
Método de Quine-McCluskey (MCCLUSKEY, 1956)	19 portas

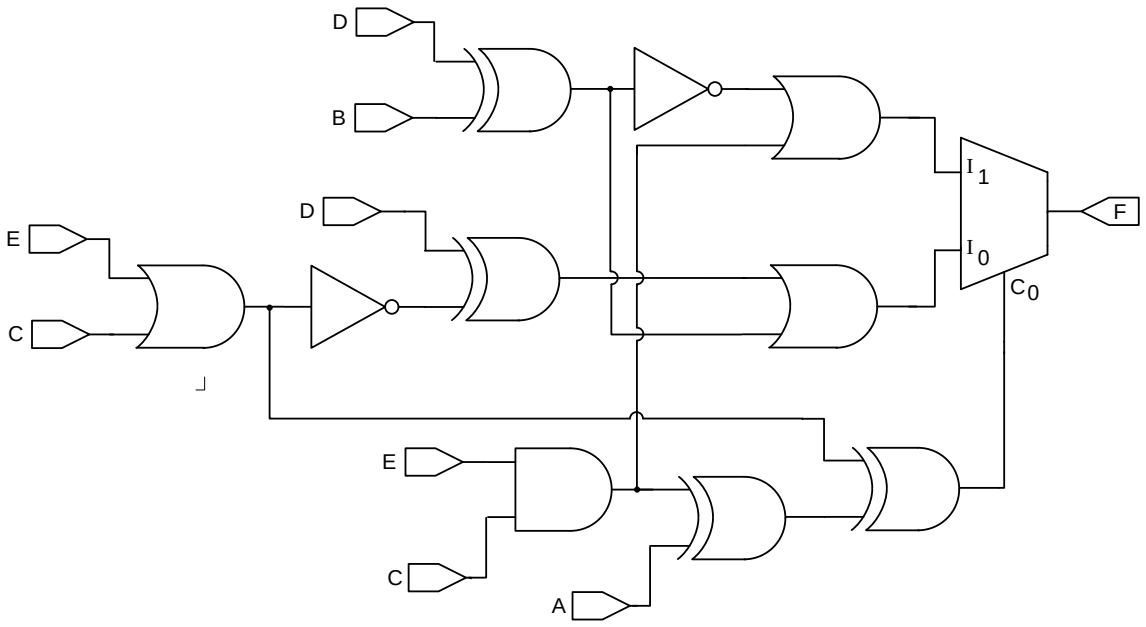


Figura 79 – Circuito projetado pelo AGMUX para o exemplo 6 utilizando MUX2

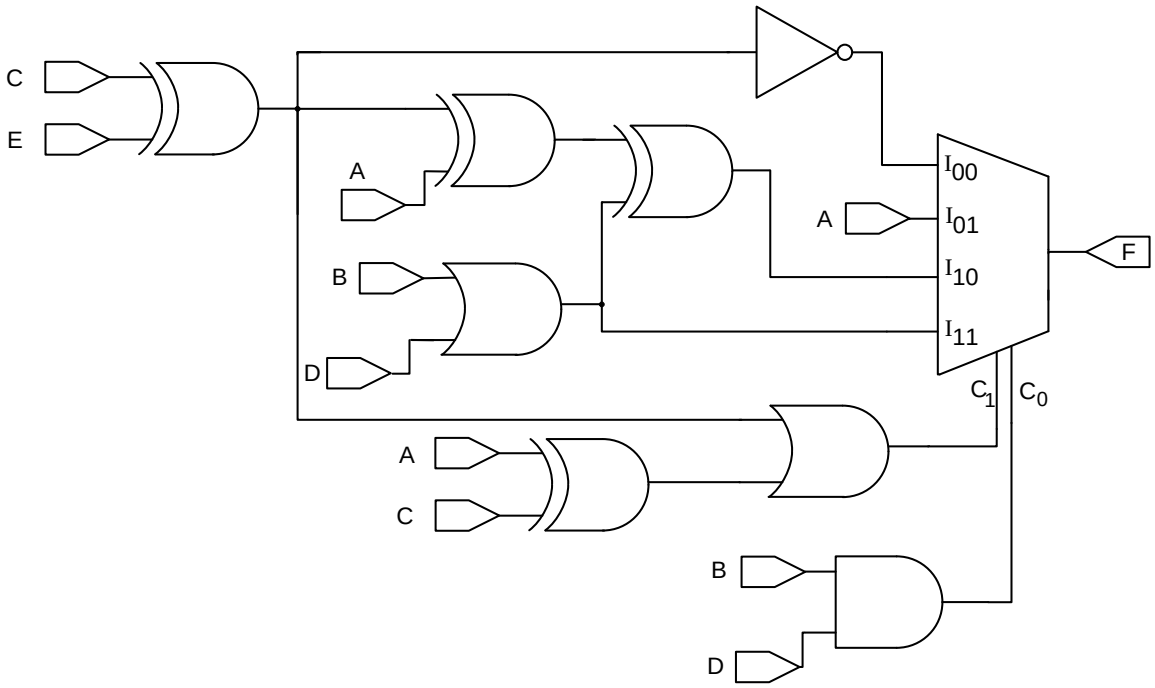


Figura 80 – Circuito projetado pelo AGMUX para o exemplo 6 utilizando MUX4

Tabela 21 – Tabela Verdade referente ao exemplo 6

mintermo	A	B	C	D	E	C_0	I_0	I_1	F	C_1	C_0	I_{00}	I_{01}	I_{10}	I_{11}	F
0	0	0	0	0	0	0	-	1	1	0	0	1	-	-	-	1
1	0	0	0	0	1	1	1	-	1	1	0	-	-	1	-	1
2	0	0	0	1	0	0	-	1	1	0	0	1	-	-	-	1
3	0	0	0	1	1	1	0	-	0	1	0	-	0	-	-	0
4	0	0	1	0	0	1	1	-	1	1	0	-	1	-	-	1
5	0	0	1	0	1	0	-	0	0	1	0	-	0	-	-	0
6	0	0	1	1	0	1	0	-	0	1	0	-	0	-	-	0
7	0	0	1	1	1	0	-	1	1	1	0	-	1	-	-	1
8	0	1	0	0	0	0	-	1	1	0	0	1	-	-	-	1
9	0	1	0	0	1	1	0	-	0	1	0	-	-	0	-	0
10	0	1	0	1	0	0	-	0	0	0	1	-	0	-	-	0
11	0	1	0	1	1	1	1	-	1	1	1	-	-	-	1	1
12	0	1	1	0	0	1	0	-	0	1	0	-	-	0	-	0
13	0	1	1	0	1	0	-	1	1	1	0	-	-	1	-	1
14	0	1	1	1	0	1	1	-	1	1	1	-	-	-	1	1
15	0	1	1	1	1	0	-	1	1	1	1	-	-	-	1	1
16	1	0	0	0	0	1	1	-	1	1	0	-	-	1	-	1
17	1	0	0	0	1	0	-	0	0	1	0	-	-	0	-	0
18	1	0	0	1	0	1	0	-	0	1	0	-	-	0	-	0
19	1	0	0	1	1	0	-	1	1	1	0	-	-	1	-	1
20	1	0	1	0	0	0	-	0	0	1	0	-	-	0	-	0
21	1	0	1	0	1	1	1	-	1	0	0	1	-	-	-	1
22	1	0	1	1	0	0	-	1	1	1	0	-	-	1	-	1
23	1	0	1	1	1	1	1	-	1	0	0	1	-	-	-	1
24	1	1	0	0	0	1	0	-	0	1	0	-	-	0	-	0
25	1	1	0	0	1	0	-	1	1	1	0	-	-	1	-	1
26	1	1	0	1	0	1	1	-	1	1	1	-	-	-	1	1
27	1	1	0	1	1	0	-	1	1	1	1	-	-	-	1	1
28	1	1	1	0	0	0	-	1	1	1	0	-	-	1	-	1
29	1	1	1	0	1	1	1	-	1	0	0	1	-	-	-	1
30	1	1	1	1	0	0	-	1	1	1	1	-	-	-	1	1
31	1	1	1	1	1	1	1	-	1	0	1	-	1	-	-	1

B.2.7 **Discussão do AGMUX**

Nos problemas *benchmark* apresentados nesta seção com 4 variáveis de entradas, Coello utiliza uma codificação matricial e recomenda aumentar gradativamente a dimensão da matriz até que uma solução viável seja encontrada. Nos algoritmos da literatura esses resultados aparecem quando a matriz atingiu a dimensão 5x5, o que corresponde a um espaço de busca da ordem de 10^{52}. Quando aplicado o AGMUX, uma matriz 3x3 correspondente a um espaço de busca 10^{16} foi suficiente. Dessa forma, a codificação proposta possibilitou a pesquisa em um espaço de busca antes desconsiderado pela codificação utilizada pela literatura.

Em relação ao número de componentes do circuito, a solução encontrada pelo AGMUX foi melhor quando comparada com a programação genética e outros algoritmos evolucionários, NGA, MGA, QESPO, GASA2. É importante ressaltar que nos problemas *benchmark* usados nos testes, as melhores soluções possuíam entre 5 e 7 elementos lógicos, dependendo do problema, o que torna a redução de um componente uma tarefa complexa.

Desde a dissertação de Louis em 1993 (LOUIS, 1993), onde foi proposta uma representação matricial para evoluir CLCs, foram publicados diversos trabalhos que utilizaram a mesma codificação (COELLO; CHRISTIANSEN; AGUIRRE, 2000b; COELLO; CHRISTIANSEN; AGUIRRE, 2000a; GARCÍA; COELLO, 2002; COELLO et al., 2000; COELLO; CHRISTIANSEN; AGUIRRE, 1997; COELLO; AGUIRRE, 2002; ALBA et al., 2007; COELLO; LUNA; AGUIRRE, 2003; KARAKATIC; PODGORELEC; HERICKO, 2013; MILLER; THOMSON; FOGARTY, 1997; MILLER; JOB; VASSILEV, 2000; MOORE; VENAYAGAMOORTHY, 2005; ANJOMSHOA; MAHANI, 2013; ANJOMSHOA; MAHANI; SADEGHIFARD, 2014), os quais propõem novos algoritmos, ressaltando-se que em todos eles o espaço de busca sempre fica limitado à codificação de Louis.

Tanto a programação genética, que utiliza uma representação em árvore, quanto os demais algoritmos que utilizam a codificação de Louis (LOUIS, 1993), geram circuitos com uma similaridade na topologia, qual seja: o número de portas lógicas tende a crescer exponencialmente da saída (nível 1) para os demais níveis, acompanhando o crescimento das ramificações do circuito. A codificação proposta neste trabalho não apresenta esse viés, o que fica evidenciado quando se compara os circuitos da Figura 5, apresentadas na introdução desse trabalho, com os circuitos referentes aos resultados obtidos com o AGMUX (por exemplo as Figuras 79 e 80). Dessa forma, a inclusão de multiplexadores de n entradas de dados gera a possibilidade de topologias diversificadas, que não apresentam viés no posicionamento dos componentes.

Analisando os circuitos projetados com o AGMUX, constata-se que as suas configurações estão fora do domínio das técnicas tradicionais, não sendo possível obter tais circuitos por métodos sistemáticos e projetistas experientes.