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

Frederico José Dias Möller

Mutações Adaptativas via Aprendizado por Reforço para Programação Genética Cartesiana Aplicada ao Projeto e Otimização de Circuitos Lógicos Combinacionais

Juiz de Fora
2022

Frederico José Dias Möller

Mutações Adaptativas via Aprendizado por Reforço para Programação Genética Cartesiana Aplicada ao Projeto e Otimização de Circuitos Lógicos Combinacionais

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Heder Soares Bernardino

Coorientador: Prof. Dr. Stênio Sã Rosário Furtado Soares

Juiz de Fora

2022

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Möller, Frederico José Dias.

Mutações Adaptativas via Aprendizado por Reforço para Programação Genética Cartesiana Aplicada ao Projeto e Otimização de Circuitos Lógicos Combinacionais / Frederico José Dias Möller. – 2022.

69 f. : il.

Orientador: Heder Soares Bernardino

Coorientador: Stênio Sã Rosário Furtado Soares

Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2022.

1. Programação Genética Cartesiana. 2. Otimização. 3. Circuitos Lógicos. I. Bernardino, Heder Soares, orient. II. Soares, Stênio Sã Rosário Furtado, coorient. III. Título.

Frederico José Dias Möller

Mutações Adaptativas via Aprendizado por Reforço para Programação Genética Cartesiana Aplicada ao Projeto e Otimização de Circuitos Lógicos Combinacionais

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação
da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 12 de abril de 2022.

BANCA EXAMINADORA

Prof. Dr. Heder Soares Bernardino - Orientador

Universidade Federal de Juiz de Fora

Prof. Dr. Stênio Sã Rosário Furtado Soares - Coorientador

Universidade Federal de Juiz de Fora

Prof. Dr. Helio José Corrêa Barbosa

Universidade Federal de Juiz de Fora

Prof. Dr. Francisco Augusto Lima Manfrini

Instituto Federal do Sudeste de Minas Gerais

Juiz de Fora, 21/03/2022.

Documento assinado eletronicamente por **Heder Soares Bernardino**, **Professor(a)**, em 12/04/2022, às 18:14, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Stenio Sa Rosario Furtado Soares**, **Professor(a)**, em 11/07/2022, às 13:59, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Helio Jose Correa Barbosa**, **Professor(a)**, em 11/07/2022, às 14:05, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Francisco Augusto Lima Manfrini**, **Usuário Externo**, em 11/07/2022, às 15:51, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **0716816** e o código CRC **8010814F**.

AGRADECIMENTOS

Agradeço primeiramente a Deus. Agradeço aos meus pais, por sempre terem priorizado minha educação e estimulado meus estudos. Meu pai por ter me mostrado diversos experimentos sobre eletricidade e mecânica, quando era pequeno e até ter comprado um livro de BASIC para mim, muito antes de eu ter qualquer base para compreender tal conteúdo. Minha mãe, que me fez ter gosto pela matemática, de quem eu acho que eu herdei qualquer talento que eu tenho para as ciências exatas. Agradeço às minhas irmãs e ao meu cunhado, por todo o suporte e inspiração. Agradeço à Letícia, por ter me ajudado a superar o meu desligamento da AMAN e redescobrir minha identidade no meio científico. Agradeço à professora Eugênia e ao professor Jabour, pelos anos de trabalho no PET, onde tive oportunidade de ter melhor contato com o método científico, com a experimentação e ao desenvolvimento acadêmico. Agradeço aos professores Luiz Oscar e Francisco Manfrini, bem como ao meu amigo Felipe Amaral, pois foram eles que me motivaram a tentar o mestrado. Agradeço aos professores Helder, Stênio e Luciana por toda orientação, ajuda e paciência durante esses 3 anos de trabalho. Agradeço ao Lucas Miller pelo suporte com a versão do código da CGP com o qual eu trabalhei. Agradeço à CAPES pela bolsa de mestrado. Agradeço à banca avaliadora: professores Hélio, Manfrini, Helder e Stênio, pela presença, paciência e pelos comentários que contribuíram tanto com a revisão deste texto, como para sustentação de trabalhos futuros.

RESUMO

A otimização de circuitos lógicos combinacionais pode levar a dispositivos eletrônicos mais rápidos e baratos, mas é um problema NP-completo. Os algoritmos determinísticos para esta tarefa são limitados a pequenos problemas e a lógica de dois níveis. Para resolver esse tipo de problema, recorre-se ao uso de meta-heurísticas e a Programação Genética Cartesiana (CGP) é amplamente adotada na literatura, dentre as alternativas de computação evolutiva. Na CGP, a mutação é tradicionalmente o único operador para gerar novas soluções candidatas. Assim, o desempenho dessa abordagem depende da eficiência de tal operador. Diferentes operadores de mutação foram propostos para a CGP, mas a maioria deles se diferencia apenas na escolha dos nós que serão modificados. Porém, na modificação dos nós a mutação atua de forma não enviesada. Existem trabalhos na literatura que propõem matrizes estáticas de enviesamento nas mutações do tipo de porta lógica para a CGP e tais trabalhos obtiveram bons resultados. Entretanto, o enviesamento adotado nesses trabalhos não varia ao longo da busca, de modo que não acompanham as mudanças que podem ocorrer no processo de otimização a depender da região do espaço de busca que a solução candidata se encontra. Portanto, uma mutação adaptativa via Aprendizado por Reforço (RL) é proposta aqui para a CGP. A chamada CGP-RL foi avaliada num conjunto de problemas de otimização de portas lógicas e em outro para a minimização do número de transistores de circuitos digitais combinacionais. A CGP-RL proposta conseguiu melhores soluções em 3 dos 5 problemas de otimização de portas lógicas quando comparada com uma CGP com mutação enviesada da literatura e melhores médias em quase 60% dos problemas testados nesse trabalho para a minimização do número de transistores quando comparada com uma CGP tradicional. Em seguida à CGP-RL, outra técnica foi desenvolvida focando na escolha do elemento do nó a ser modificado pelo operador de mutação: entradas e operação lógica. A *Node* CGP-RL conseguiu melhores médias em cerca de 75% dos problemas abordados alcançando também as melhores soluções, quando comparada com a CGP, em 13 dos 23 problemas analisados, apresentando soluções finais com até 10% menos transistores que a CGP.

Palavras-chave: Programação Genética. Otimização. Circuitos Lógicos.

ABSTRACT

Optimizing combinational logic circuits can lead to faster and cheaper electronic devices, but it is an NP-complete problem. The deterministic algorithms for this task are limited to small problems and two-level logic. To solve this type of problem, meta-heuristics are used and Cartesian Genetic Programming (CGP) is widely adopted in the literature, among the alternatives to evolutionary computation. In CGP, the mutation is traditionally the only operator to generate new candidate solutions. Thus, the performance of this approach depends on the efficiency of such an operator. Different mutation operators have been proposed for the CGP, but most of them differ only in the choice of nodes to be modified. However, when modifying nodes, the mutation acts in an unbiased way. There are works in the literature that propose static bias matrices in the mutations of the logic gate type for the CGP and such works obtained good results. However, the bias adopted in these works does not vary along the search, so they do not follow the changes that may occur in the optimization process depending on the region of the search space where the candidate solution is. Therefore, an adaptive mutation via Reinforcement Learning (RL) is proposed here for PGC. The so-called CGP-RL was evaluated in a set of logic gate optimization problems and another for minimizing the number of transistors in combinational digital circuits. The proposed CGP-RL achieved better solutions in 3 out of 5 logic gate optimization problems when compared to a skewed mutation CGP in the literature and better averages in almost 60% of the problems tested in this work for the minimization of the number of transistors when compared with a traditional PGC. After CGP-RL, another technique was developed, focusing on choosing the node element to be modified by the mutation operator: inputs and logic operation. The *Node* CGP-RL achieved better averages in about 75% of the problems addressed, also reaching the best solutions, when compared to the CGP, in 13 of the 23 problems analyzed, presenting final solutions with up to 10% fewer transistors than CGP.

Keywords: Genetic Programming. Optimization. Logical Circuits.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

(CGP) *Cartesian Genetic Programming* - Programação Genética Cartesiana
(CLC) *Combinationla Logic Circuits* - Circuitos Lógicos combinacionais
(PM) *Point Mutation* - Operador de mutação pontual
(RL) *Reiforced Learning* - Aprendizado por reforço
(SAM) *Single Active Mutation* - Operador de única mutação ativa
(TTL) *Transistor-Transistor Logic* - Lógica Tansistor-Transistor

SUMÁRIO

1 INTRODUÇÃO

Circuitos lógicos Combinacionais (CLCs) são sistemas onde as saídas são resultados de funções lógica temporalmente independentes de suas entradas (36, 34). Tais circuitos são elementos fundamentais na eletrônica digital, campo de estudos que trata do projeto e manutenção de dispositivos que utilizam sinais discretos para seu funcionamento (36). A eletrônica digital é um campo tecnológico bastante demandado pela humanidade. Portanto, o bom projeto de CLCs implica em economia material e, portanto, barateamento de preços e aumento da acessibilidade a equipamentos eletrônicos digitais, como computadores, celulares, relógios entre outros. Tal otimização também pode implicar em redução de tamanho, de calor dissipado e tempo de resposta. Todavia, a minimização destes é um problema de otimização lógica, que é um problema NP completo e os métodos determinísticos existentes não são práticos para a maioria das aplicações (38).

A abordagem comum é usar heurísticas como o algoritmo ESPRESSO (1) que é bastante adotado para a minimização de CLCs. Este método opera manipulando "cubos" que representam o produto dos mintermos da expressão lógica do circuito a ser minimizado. O EXPRESSO consegue chegar em soluções de boa qualidade com baixo custo de tempo e memória na maioria dos problemas.

Uma alternativa ao ESPRESSO são os métodos evolutivos, que em geral conduzem à melhores soluções (8). Dentre estes, destaca-se a Programação Genética Cartesiana (CGP) (27). Na CGP, os circuitos são representados como dígrafos acíclicos e cada nó possui uma função lógica associada. A CGP geralmente gera novos indivíduos por meio de mutação que, neste caso, consiste na alteração de uma das entradas de um nó, ou de sua função lógica. A CGP é capaz não apenas de projetar um circuito lógico combinacional que atenda a uma tabela verdade alvo, mas também de minimizar alguns atributos desejados, como o número de transistores.

Há diversos trabalhos na literatura propondo alterações na CGP para melhorar seu desempenho. Propostas como o *crossover* sobre o fenótipo descrito em (6) aumentam significantemente o custo em tempo da CGP. Outras, como o método semanticamente orientado visto em (15) dependem da avaliação usando a tabela verdade, o que torna o custo computacional do método muito alto para circuitos com muitas entradas.

Outra abordagem que tem sido bastante usada na literatura, não só relativa à CGP, mas em Computação Evolutiva em geral é o enviesamento de operadores. Trabalhos como (24) usaram a estratégia ϵ-greedy para selecionar os melhores operadores evolutivos para gerar novos indivíduos ao longo da execução de um algoritmo genético. Em (7), os autores propuseram uma matriz de enviesamento para a mutação de portas lógicas na CGP com valores estáticos.

Uma extrapolação da técnica apresentada em (7) é a construção da matriz de

enviesamento usando o aprendizado de máquina. A estratégia ϵ-*greedy*, uma técnica de balanço de estratégias exploratórias e gulosas, foi aplicada nas mutações do tipo de porta lógica, gerando a primeira técnica proposta aqui: a CGP-RL (33). Outra técnica proposta foi a *Node* CGP-RL (32), que aplica o mesmo princípio, mas em outro momento da mutação de um nó: a escolha de qual elemento vai sofrer mutação: entradas ou função lógica. Ambas as técnicas, usando uma estratégia de baixa complexidade, apresentaram melhoras no resultado da CGP aplicada ao projeto e otimização de CLCs.

1.1 OBJETIVOS

O objetivo principal dos trabalhos realizados é melhorar o desempenho da CGP quando aplicada ao projeto e otimização da quantidade de transistores de CLCs, propondo dois operadores adaptativos, CGP-RL e *Node* CGP-RL,baseado em aprendizado. Os trabalhos têm também como objetivos secundários:

- Analisar o processo de mutação da CGP, buscando entender como cada elemento impacta na geração de melhores soluções ao longo do processo evolutivo;
- Propor um operador de aprendizado por reforço para enviesar a mutação de portas lógicas da CGP; e
- Propor um operador de aprendizado por reforço para enviesar a mutação de nós da CGP;

2 REVISÃO BIBLIOGRÁFICA

2.1 HISTÓRICO

Apesar da existência de dispositivos que utilizavam sinais eletrônicos discretizados para realizar tarefas, como computadores de relés e valvulados. O marco que consolidou a eletrônica digital foi a invenção do transistor em 1947, por William Shockley, John Bardeen e Walter Brattian no Bell Laboratories. Tal invenção lhes rendeu o prêmio Nobel de Física em 1956. De tal outras tecnologias surgiram, como o Circuito Integrado, que rendeu à Jack Kilby em 1958 e o primeiro microprocessador integrado, o intel 4004, em 1971 (39).

A complexidade dos circuitos lógicos tem crescido com o tempo, ainda que de forma linear (13). A demanda e produção de circuitos digitais também tem crescido, de forma geral exponencialmente (22). Portanto a é justificada a necessidade de métodos automáticos de projeto e otimização de circuitos lógicos.

Tradicionalmente, se adotam técnicas *top down*, que consistem em decompor os circuitos em sub-circuitos, de menor complexidade e portanto mais fáceis de projetar e otimizar (29). Abordagens *bottom up* consistem em resolver problemas pontuais no sistema e então ir integrando essas soluções em circuitos de maior complexidade. Tal abordagem tende a gerar circuitos com menor redundância e mais simplificados (29), porém exige técnicas de maior complexidade, como as abordagens evolutivas. As abordagens evolutivas não são as únicas técnicas de inteligência computacional propostas para projetar circuitos em uma abordagem *bottom up*, porém são as que apresentam melhores resultados (19).

2.2 ABORDAGENS EVOLUTIVAS

Abordagens evolutivas para o projeto de circuitos lógicos podem ser traçadas desde 1956, com o trabalho de Friedman (11). Que utilizou analogias com o desenvolvimento do sistema nervoso para projetar hardware,

Todavia, foi o algoritmo genético de Loius, em 1993 que expandiu os trabalhos feitos nessa área. A codificação do genoma adotada em seu trabalho foi uma matriz onde cada elemento representa uma porta lógica e suas entradas (23). Tal algoritmo pesquisa primeiramente por soluções que atendam todas as linhas da tabela verdade do circuito desejado e depois as classifica pelo número de portas WIRE, ou seja, portas que significam uma ligação direta entre duas outras portas. Ele então entrega o circuito com maior número de portas WIRE que cumpre a tabela verdade desejada.

Tal codificação de genoma também foi usada em (27). A diferença da abordagem de Miller et al para a de Loius é que Miller aumentou o espaço de busca adicionando a possibilidade de cada elemento da matriz, além de ser uma porta lógica, ser também um

multiplexador 2x1.

Outra proposta, criada a partir da codificação de Louis foi o algoritmo NGA (*N-cardinality Genetic Algorithm*), que consiste em inicializar o algoritmo de Louis com uma matriz pequena e ir aumentando o tamanho da mesma até que uma solução seja encontrada (5). Os mesmos autores de (5), também propuseram uma variação multiobjetivo, que não só projetava o circuito, mas também buscava maximizar o número de portas WIRE (4).

2.3 PROGRAMAÇÃO GENÉTICA CARTESIANA

A Programação genética cartesiana (CGP) é uma abordagem evolutiva cujo a codificação do genoma é feita na forma de dígrafos acíclicos (27, 28, 30). Os nós de tal dígrafo são organizados de forma matricial em linhas e colunas e cada nó possui informação de suas entradas e tipo de porta lógica, em codificação similar à de Louis. A CGP será explicada de forma mais detalhada no Capítulo 5.

A maior parte do genótipo na CGP não é mapeada no fenótipo (37). No entanto, o genótipo não mapeado carrega informações evolutivas de forma análoga ao que acontece com o DNA não codificante na biologia. Tal informação, ao longo do processo evolutivo, pode voltar a compor a parte do genótipo mapeada para o fenótipo, contribuindo para gerar novas soluções e sendo uma das características que ajudam a CGP a evitar ótimos locais (37).

A CGP tem melhor desempenho no projeto e otimização de circuitos lógicos que os demais métodos de Programação Genética (12). A CGP não apresenta o problema de aumento desordenado da solução, uma vez que o tamanho de cada individuo é limitado durante todo o processo evolutivo. Além disso, o indivíduo na CGP apresenta um formato otimizado para a representação de circuitos combinacionais lógicos, com cada individuo representando um circuito por completo, com todas as suas saídas (28).

Diversas alterações foram propostas para a CGP desde sua criação. O operador de mutação do tipo SAM *(Single Active Mutation)* que será detalhado no Capítulo 5, modifica a forma como a CGP escolhe quais nós serão modificados. Um único nó ativo é modificado por geração (12). O operador SAM tem apresentado melhor desempenho na otimização e projeto de CLC que o operador tradicional da CGP e outros operadores propostos, como o GAM (*Guided Active Mutation*) (12, 8), que é uma versão baseada no SAM que atua em subgrafos que o pior resultado com relação à tabela verdade desejada.

A CGP não usa tradicionalmente um operador de *crossover* (28). Tal operador chegou a ser utilizado inicialmente na CGP, mas mas se mostrou disruptivo e portanto abandonado (28, 6). Operadores *crossover* foram propostos e investigados em (3, 18, 40, 2). (6) propôs um operador que faz a recombinação com base no fenótipo do individuo, apresentando resultados promissores.

Um dos maiores desafios da CGP é a escalabilidade (28, 15). Na literatura, os circuitos multiplicador 5x5 e somador 9+9 são os de maior complexidade que a CGP conseguiu gerar um individuo factível à partir de uma população aleatória (16). Propostas como o Operador Semanticamente Orientado (15) e o operador de mutação enviesado proposto em (7) tem dentre seus objetivos superar tal problema.

O Operador Semanticamente Orientado (SOMO) mapeia a saída esperada de um nó para um conjunto de entradas e então modifica uma de suas entradas para algum nó em colunas anteriores que torne a saída do nó que sofrendo mutação se aproxime da saída desejada. O SOMO consegue gerar um Somador 9+9 em cerca de pouco menos de três horas, um somador 10+10 em cerca de vinte de duas horas e um multiplicador 5x5 em cerca de quarenta e duas horas (15). Tais resultados não são apenas melhorias no desempenho da CGP como também um indicativo de que o problema de escalabilidade pode vir a ser resolvido com modificações que direcionem as decisões do operador de mutação (15).

O operador enviesado proposto em (7) possui uma matriz que guia a mutação de portas lógicas em um nó. As linhas de tal matriz representam a porta lógica do nó que vai sofrer mutação e as colunas representam para qual tipo de porta esta vai ser modificada. Os valores de cada célula são um peso que influencia na probabilidade de tal troca acontecer. Dessa forma, trocas que são consideradas vantajosas tem maior chance de ocorrer.

2.4 USO DO APRENDIZADO POR REFORÇO NA COMPUTAÇÃO EVOLUTIVA

De maneira geral, abordagens evolutivas envolvem uma população inicial de soluções, a criação de novas soluções a partir desta população e a seleção das melhores soluções para compor a próxima geração. Há uma grande variação na forma como executar cada uma das etapas. Existem métodos que usam a recombinação junto com a mutação para gerar novas soluções, outros que usam apenas a mutação (9). . Para determinadas abordagens a melhor forma de selecionar as melhores soluções é o ranqueamento, que é o ordenamento das soluções de acordo com sua aptidão para então selecionar as melhores, para outras é a estratégia de roleta, onde cada solução tem uma probabilidade de ser selecionada para a próxima geração de acordo com suas aptidão. Existem outros métodos de seleção de soluções. Cada uma dessas formas possuem variações ou parâmetros que precisam ser ajustados de acordo com a abordagem evolutiva, contexto, ou problema (9).

Abordagens usando aprendizado de máquina vem otimizando esses ajustes. Em (25), os autores desenvolveram uma técnica que usa Aprendizado por Reforço (RL) para ajuste de parâmetros em um Algoritmo Genético na resolução de problemas de Satisfatibilidade Booleana, ou seja, determinar se existe um conjunto de valores para variáveis de uma expressão lógica que torne a expressão verdadeira, um problema complexo da classe NP-

completo e computacionalmente custoso. Em (24), os autores usaram o RL para escolher quais operadores seriam usados para gerar um novo indivíduo. O trabalho (17) propõe uma abordagem dinâmica do problema dos *k-armed bandits* também para a seleção de operadores, apresentando bons resultados.

A CGP, como método de computação evolutiva, está sujeita aos problemas de escolha de operadores e ajustes de parâmetros. Na mutação do tipo de nó, o operador tem que decidir qual elemento do nó vai sofrer alteração e como essa mutação vai acontecer. De forma tradicional essa decisão é feita de forma não enviesada, mas em (7) foi apontado que um enviesamento na mutação de portas lógicas pode levar a melhores resultados que uma abordagem neutra. Por outro lado em (15), um direcionamento para a mutação de entradas ajuda na convergência da CGP para uma solução factível. Os problemas de decisão de qual elemento do nó vai sofrer mutação, bem como qual mutação de porta lógica vai acontecer se encaixam no modelo do problema dos *k-armed bandits* e portanto uma estratégia RL, inspirada nos trabalhos (24, 17) pode melhorar o desempenho do operador de mutação de nós e de portas lógicas.

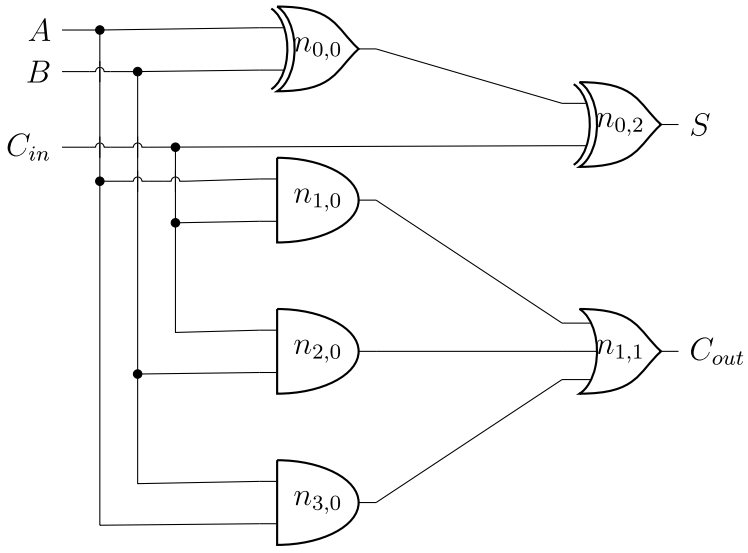
3 FUNDAMENTOS DE CIRCUITOS DIGITAIS

As técnicas de Programação Genética Cartesiana abordadas nesse trabalho são aplicadas ao projeto e otimização de circuitos lógicos combinacionais. Tais circuitos trabalham com sinais digitais, ou seja, sinais representados por valores que variam de forma discreta (36). Neste capítulo serão abordados conceitos fundamentais de eletrônica e lógica digital.

3.1 CIRCUITOS LÓGICOS

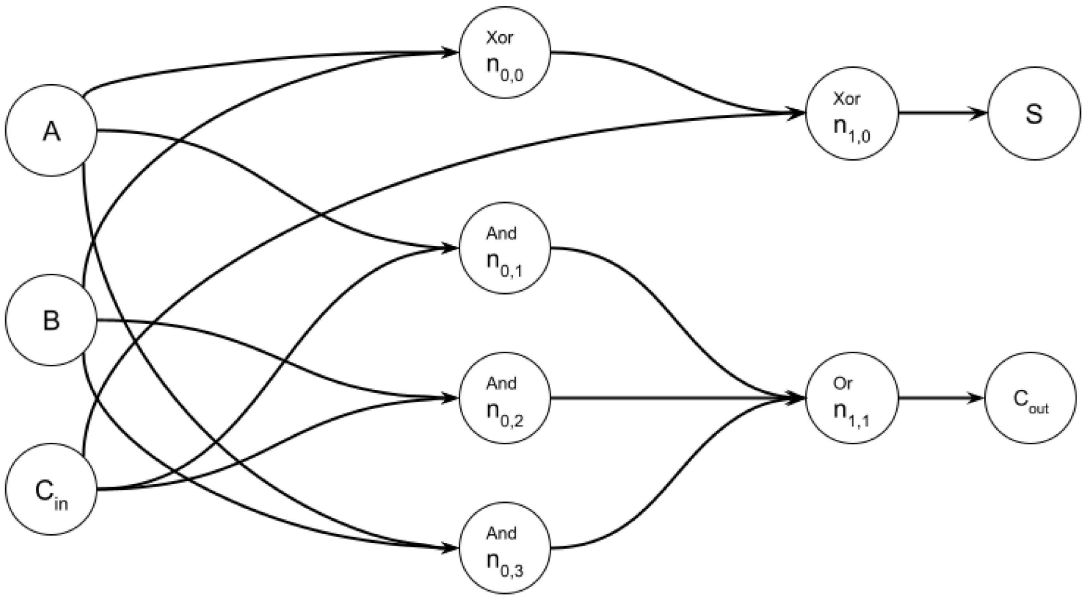
Circuitos lógicos são sistemas que processam informações usando funções Booleanas (36, 34). Tais são constituídos de variáveis de entrada e de saída, e portas lógicas. A Figura 1 representa um circuito Somador Completo. Tal circuito realiza a soma de dois bits (entradas A e B) e do bit de transporte de outro Somador Completo C_{in}. O resultado é expresso na saída S e se o valor da soma for maior que 1, no bit de transporte de saída C_{out}.

Figura 1 – Circuito Somador Completo, que realiza a soma dos bits de entrada A e B e do bit de transporte C_{in} expressando o resultado nas saídas S e no bit de transporte de saída C_{out}. O somador completo tem esse nome, pois pode-se aumentar a quantidade de bits a serem somados ligando a saída C_{out} de um somador na entrada C_{in} de outro.



Fonte: Figura produzida pelo autor.

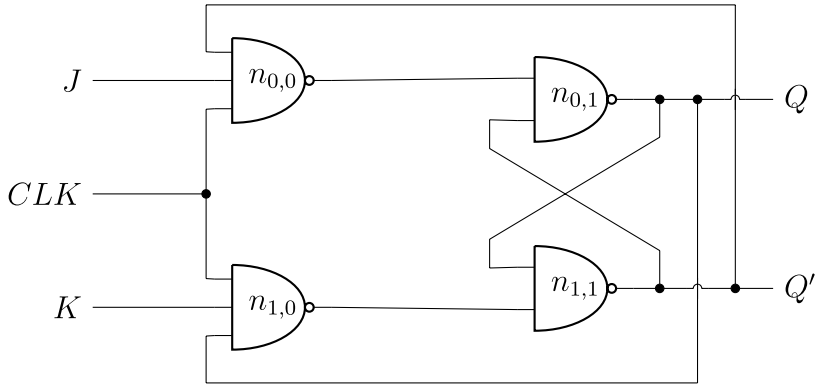
Figura 2 – Circuito Somador Completo representado como um digrafo.



Fonte: Figura produzida pelo autor

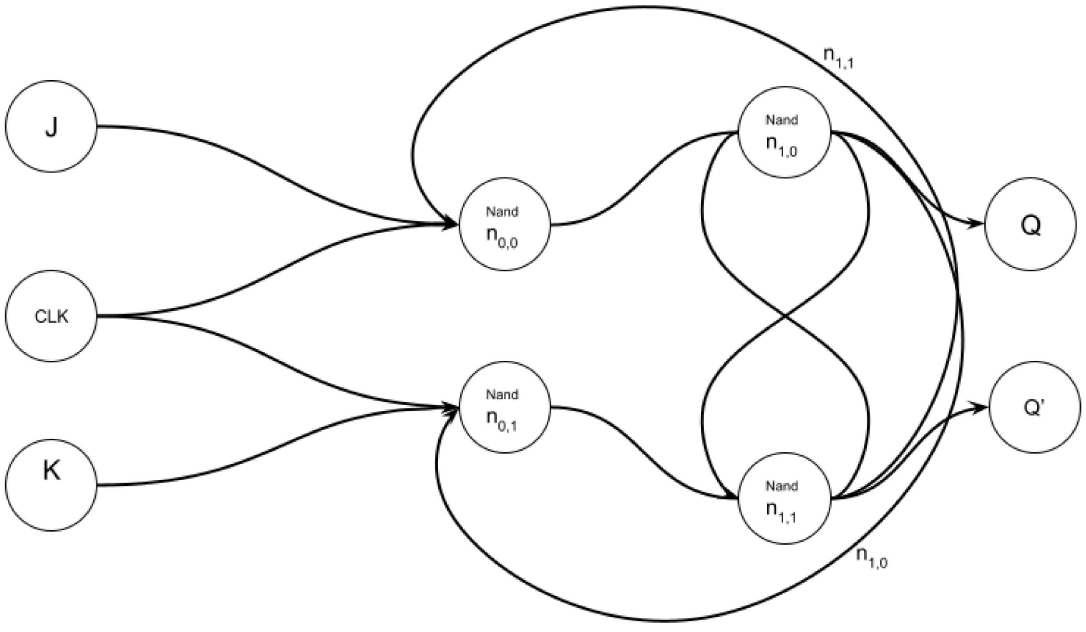
No caso do circuito Somador Completo, é possível observar que cada saída depende exclusivamente dos valores das entradas. O valor das saídas anteriores não interfere no valor da saída atual, ou seja as saídas são independentes do tempo. Para esse tipo de circuito lógico é dado o nome de circuitos lógicos combinacionais (CLC). Outro tipo de circuitos lógicos são os sequenciais A Figura 3 representa um *Flip Flop* JK. A Figura 2 e a Figura 4 são a representação em grafos dos circuitos Somador Completo e *Flip Flop* JK. Observando a representação em grafos é possível notar que a diferença fundamental é a ausência de ciclos no CLC. Por conveniência, será adotada a representação dos circuitos pela forma de grafos a partir deste ponto do trabalho.

Figura 3 – Circuito *Flip Flop* JK, um circuito lógico sequêncial. Sua saída não depende apenas das entradas, mas também do estado anterior de suas saídas.



Fonte: Figura produzida pelo autor

Figura 4 – Circuito *Flip Flop* JK representado como um digrafo. É possível observar o ciclo formado pelas saídas dos nós n_n. Essa característica dos Circuitos Sequenciais é que faz com que suas saídas dependam de seu estado anterior.



Fonte: Figura produzida pelo autor.

3.2 TABELA VERDADE

Como quantidade de variáveis de entrada e de saída de um circuito lógico são finitas e os valores que cada uma pode assumir é apenas Verdadeiro ou Falso, é possível mapear cada entrada possível com sua saída e assim gerar o artefato conhecido como tabela verdade. A Tabela 1 é a tabela verdade de um Somador Completo. Dois circuitos lógicos são logicamente idênticos, isto é, desempenham a mesma função lógica, se suas

tabelas verdades forem idênticas. A tabela possui $n_i + n_o$ colunas, onde n_i é o número de variáveis de entrada do circuito e n_0 é o número de variáveis de saída. Portanto, as n_i primeiras colunas da tabela representam cada uma das variáveis de entrada e seus estados, enquanto as últimas colunas representam os estados das variáveis de saída. Desta forma, cada linha da tabela verdade representa um possível valor de entrada do circuito e seu valor de saída associado. Várias saídas podem ter o mesmo valor, mas cada valor de entrada é único.

Tabela 1 – Tabela verdade do circuito Somador Completo.

A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fonte: Elaborada pelo autor.

O método tradicional de comparação de Circuitos Lógicos envolve medir a distância de Hamming de acordo as saídas das tabelas verdade dos circuitos. A distância de Hamming é uma métrica de comparação entre duas cadeias de valores de mesmo tamanho e contabiliza quantas posições tem valores diferentes. Na comparação de circuitos, pode-se obter tal distância comparando as saídas das tabelas verdade de acordo com a Equação 3.1, onde i é a linha da tabela verdade, o a saída, n_i é a quantidade de variáveis de entrada, n_o a quantidade de variáveis de saída e $a_{i,o}$ e $b_{i,o}$ são os valores das saídas o dos circuitos a e b, na linha i. Porém, o tamanho da tabela verdade é exponencialmente proporcional à quantidade de variáveis de entrada de um circuito e isso torna esse cálculo computacionalmente custoso para circuitos com maiores quantidades de variáveis de entrada.

$$h(a,b) = \sum_{i=1}^{i=2^{n_i}} \sum_{o=1}^{o=n_o} |a_{i,o} - b_{i,o}| \tag{3.1}$$

Uma alternativa para verificar a similaridade lógica de circuitos é fazer sua comparação por diagramas de decisão binários (BDD). BDDs são digrafos acíclicos que representam de forma comprimida a relação das variáveis de entrada de um circuito lógico com seu valor de uma de suas saídas. O uso de BDDs para a comparação de circuitos lógicos é reconhecido na literatura como um método menos custoso que a comparação de tabelas verdade (8) e é o método de comparação de circuitos adotado neste trabalho.

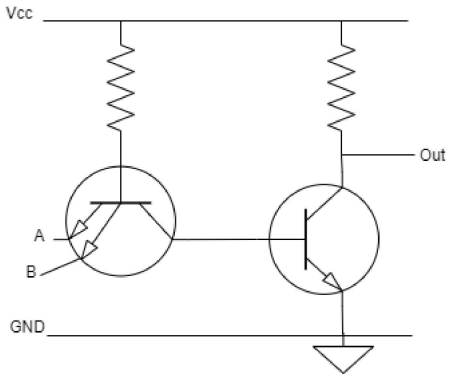
3.3 PORTAS LÓGICAS

Portas lógicas são os blocos fundamentais de construção de um circuito lógico e representam as funções lógicas básicas (36). Comumente, as portas lógicas estão relacionadas às funções lógicas mais simples, como AND, OR, ou NOT. Porém, existem portas mais complexas, como as chamadas superportas. Não existe uma porta lógica fundamental, ou seja, que não pode ser construída a partir de outras. Qualquer tabela verdade de uma porta lógica pode ser construída a partir de portas NOR e NAND e essas podem ser construídas a partir de outras portas lógicas (34).

Portas lógicas ideais realizam a operação lógica de forma instantânea e sua saída é sempre um valor booleano. Isso não acontece nas portas lógicas aplicadas na eletrônica digital. Elas possuem um tempo de atraso de resposta e suas saídas são um valor de tensão não inteiro. De acordo com a tecnologia de construção, elas são sujeitas à ruído, perda de potência e interferências do meio.

A tecnologia de construção de portas lógicas utilizada como referência neste trabalho é a *Transistor-Transistor Logic*, ou Logica Transistor-Transistor (TTL). Tal tecnologia utiliza transistores de junção bipolar tanto para executar a operação lógica, quanto para amplificar o sinal dentro do circuito da porta. (10). Portas lógicas TTL operam com tensão entre 0V e 5V, sendo o sinal baixo (falso) entre 0V e 0,5V e o alto (verdadeiro) entre 2,7V e 5V, porém com as tolerâncias ao ruído sendo consideradas o sinal baixo passa a ser entre 0V e 0,8V e o alto entre 2V e 5V. A Tabela 2 relaciona a quantidade de transistores por porta lógica TTL e a Figura 5 apresenta o diagrama de uma porta NAND construída em tal tecnologia. Todas as portas possuem duas entradas exceto a NOT.

Figura 5 – Diagrama de construção de uma porta lógica do tipo NAND de acordo com a tecnologia TTL.



Fonte: Figura produzida pelo autor.

Tabela 2 – Número de transistores por porta lógica de acordo com a tecnologia TTL.

Porta Lógica	Número de transistores
NOT	1
AND	2
OR	2
NAND	2
NOR	1
XOR	3
XNOR	4

Fonte: EREN, 2003.

Uma vez que o aspecto físico de um CLC é dado pelos elementos de construção de suas portas lógicas, é natural que estratégias de otimização foquem nesses elementos ao invés de focar nas portas lógicas. Tal abordagem foi evidenciada em (8), que vinculou o valor da quantidade de transistores para cada porta lógica e fez a minimização dos CLCs comparando a quantidade de transistores. A mesma abordagem é utilizada aqui. Apesar de existirem tecnologias mais modernas, como a CMOS, o trabalho de referência (8) utiliza a tecnologia TTL, portanto tal abordagem foi mantida neste estudo.

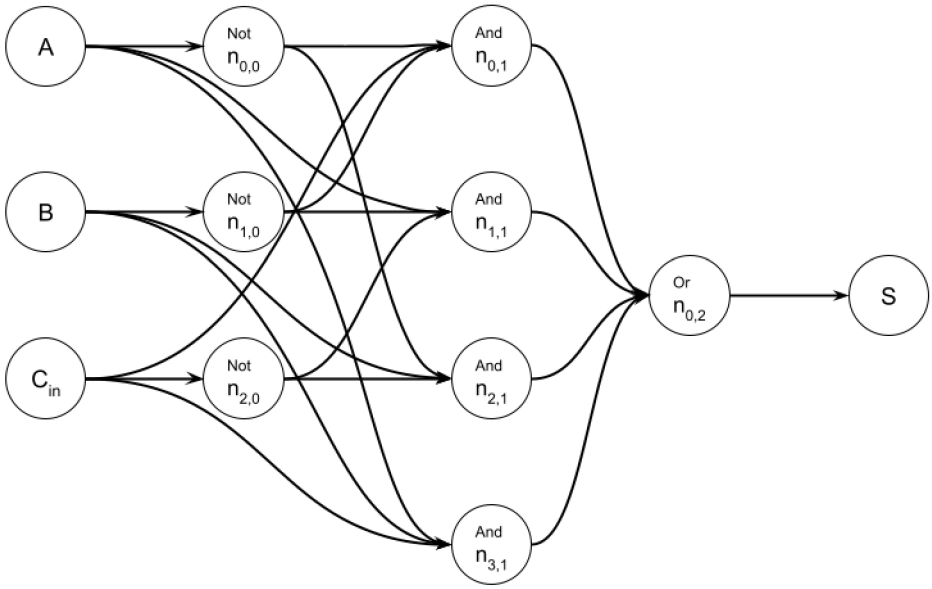
3.4 PROJETO DE CIRCUITOS LÓGICOS COMBINACIONAIS

É possível projetar um CLC a partir de sua tabela verdade. Um método simples de se fazer isso é usar a soma dos produtos: para cada linha da tabela verdade onde uma determinada saída *o* tem valor "1", lê-se os valores das variáveis de entrada dessa linha e cria-se um elemento contendo o produto das variáveis que estão com valor "1" com a negação das variáveis com valor "0". No fim, cada elemento é somado e essa soma de produtos corresponde à expressão lógica daquela saída. A soma dos produtos pode ser traduzida em um circuito lógico fazendo-se com que cada elemento negado se transforme em uma porta NOT, os conjuntos de produtos em uma sequência de portas AND e a soma em uma sequência de portas OR. É possível fazer processo similar e projetar o circuito usando produtos da soma. A Figura 6 mostra a saída S do Somador Completo como uma soma de produtos enquanto a Figura 7 mostra a saída S como um produto de somas.

Esses métodos, no entanto, não são ideais, pois acaba gerando circuitos muito grandes, o que aumenta seu custo e pode gerar diversas características negativas, como atraso no tempo de resposta, consumo energético e potência dissipada em forma de calor. Existem métodos determinísticos como o mapa de Karnaugh (20) e o método de Quine - McCluskey (26) conseguem entregar circuitos otimizados dentro do escopo de lógica de dois níveis, utilizando apenas as portas NOT, AND e OR. Porém tais métodos não são práticos para problemas maiores, devido a sua complexidade e nem se aplicam à lógica

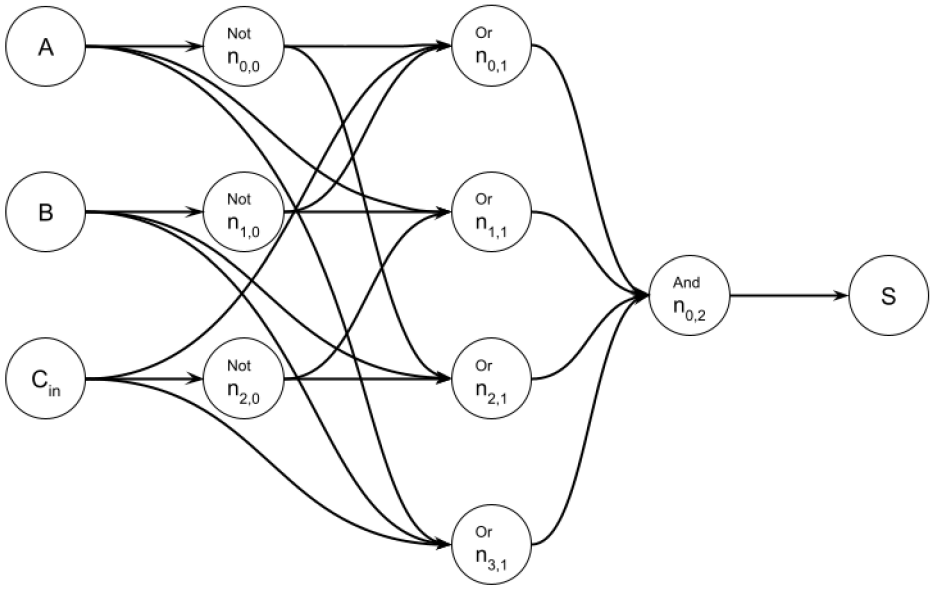
multi-nível que utiliza um conjunto maior de tipos de portas lógicas. Sendo assim, são necessários métodos heurísticos de otimização de CLCs.

Figura 6 – Saída S do Circuito Somador Completo projetada pela soma de produtos.



Fonte: Figura produzida pelo autor.

Figura 7 – Saída S do Circuito Somador Completo projetada pelo produto das somas.



Fonte: Figura produzida pelo autor.

4 APRENDIZADO POR REFORÇO

Os três paradigmas do aprendizado de máquina são o aprendizado supervisionado, o não supervisionado e o aprendizado por reforço (31). No aprendizado supervisionado, tenta-se predizer o valor de uma variável dependente a partir de uma lista de variáveis independentes. A base de dados nesse caso é rotulada. Com isso, é possível saber o erro do sistema para um conjunto de entradas e assim ajustar os valores de seus parâmetros de acordo com esse erro. Já no aprendizado não supervisionado, tenta-se estabelecer padrões dentro da base de dados. Esta por sua vez não possui rótulos e o aprendizado se dá estabelecendo relações e similaridades entre os parâmetros dos objetos ali contidos (31).

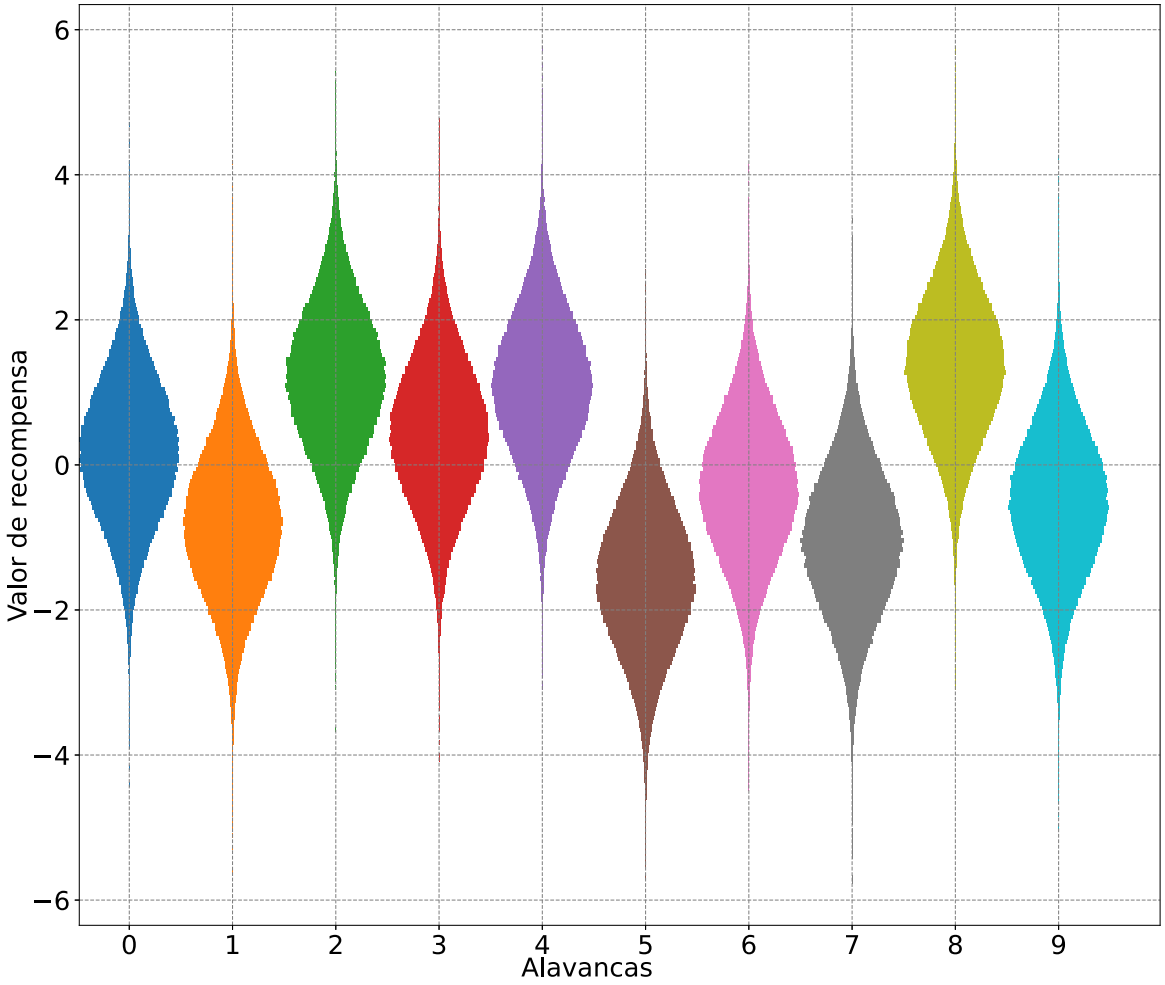
Já no aprendizado por reforço, tenta-se guiar um agente inteligente na tomada de decisões. As decisões são chamadas de políticas. O aprendizado ocorre à medida que o agente utiliza as políticas disponíveis: quando a política dá um bom resultado o agente é recompensado positivamente e quando a política dá um resultado ruim ele é punido, ou seja, é recompensado negativamente (35). Como o agente tende a escolher políticas de acordo com sua experiência, as políticas boas são reforçadas e passam a ser tomadas com cada vez mais frequência e, com isso, a eficácia do agente para aquela situação melhora com o passar do tempo.

4.1 O PROBLEMA DO *K-ARMED BANDITS* E A ESTRATÉGIA ϵ-*GREEDY*

O problema do *K-armed Bandits*, ou caça níqueis de K-alavancas, é um problema alegórico que visa descrever uma situação onde um agente inteligente tem um conjunto definido de políticas para adotar. Cada política tendo uma recompensa média distinta, porém desconhecidas para o agente. Este deseja maximizar seus ganhos no final de um número finito, porém relativamente longo, de iterações (21).

No *K-armed Bandits* um apostador se vê frente a uma máquina caça niqueis com diversas alavancas. Cada alavanca retorna valores de recompensa aleatórios, porém distribuídos em torno de uma média m_k, que varia de alavanca para alavanca. A Figura 8 é um exemplo da distribuição de recompensas de um caça níqueis de 10 alavancas, nela o zero representa a situação onde o apostador conseguiu o mesmo valor da ficha que colocou no caça-niqueis. Valores negativos ocorrem quando ele conseguiu prêmios menores do que o preço da ficha. Valores positivos, quando conseguiu valores maiores que o preço da ficha.

Figura 8 – Distribuição de recompensas das alavancas em um caça-níqueis em um problema do tipo *k-armed bandits* de 10 alavancas. Cada posição representa uma alavanca e sua distribuição de recompensas.



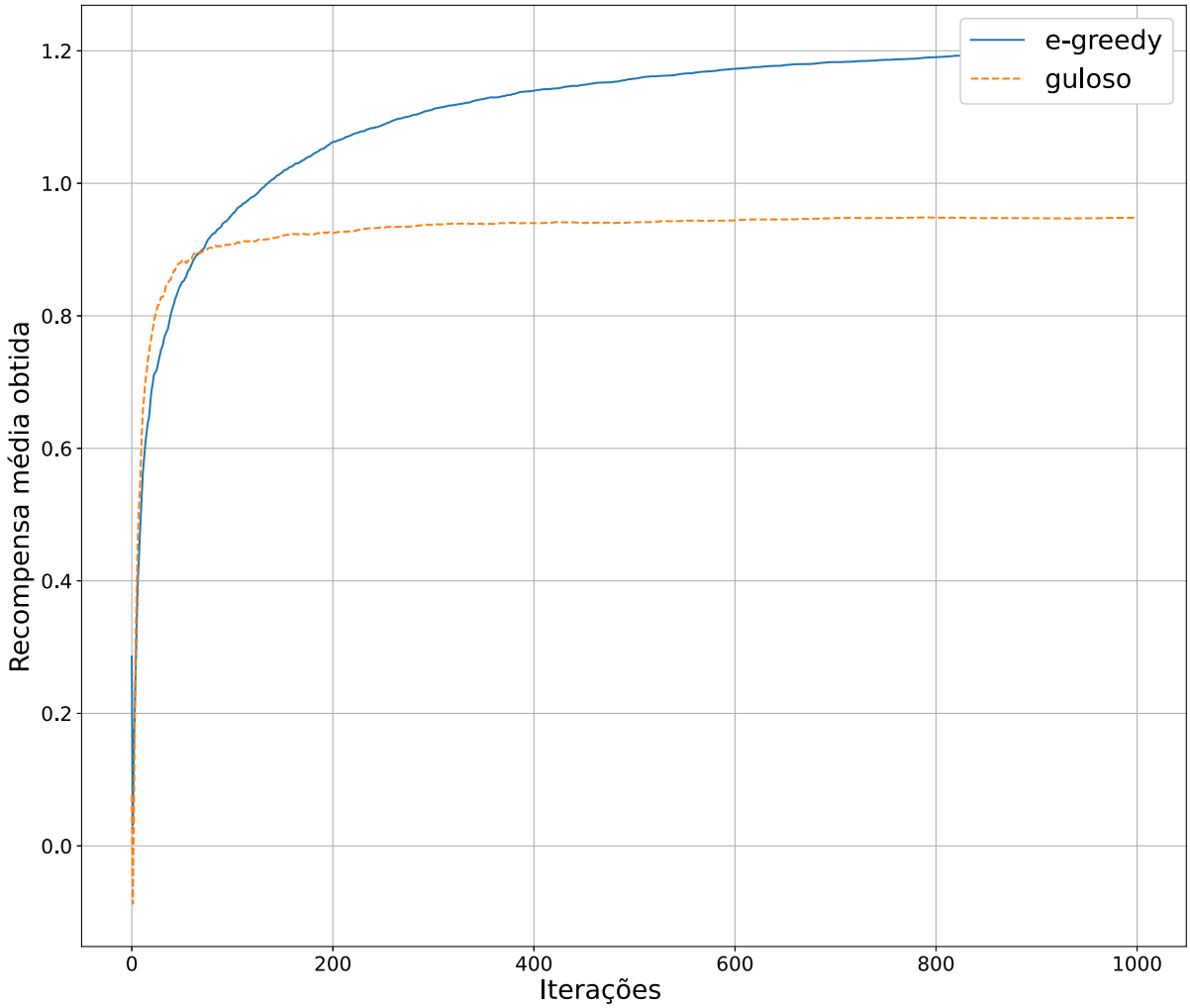
Fonte: Figura produzida pelo autor.

O apostador não sabe qual alavanca é a melhor e ele quer maximizar seus ganhos. À medida que ele usa as alavancas, ele calcula a média de recompensa que obteve por cada uma. Ele pode, portanto, adotar uma estratégia gulosa, puxando a alavanca na qual ele obteve melhor recompensa média até o momento. Apesar de ser uma estratégia aparentemente boa, é provável que o apostador gaste mais tentativas do que o necessário em uma alavanca que não é a melhor e, portanto, ele vai terminar suas jogadas com um ganho total menor do que poderia ter se tivesse localizado a melhor alavanca mais cedo.

Uma forma de se contornar esse problema é com a estratégia ϵ-*greedy* ou ϵ-gulosa. Por tal estratégia, existe uma probabilidade ϵ do apostador ignorar a estratégia gulosa naquela jogada e puxar uma alavanca aleatória. Quando ele faz isso, ele está adotando um movimento exploratório e, portanto, aumentando as chances de encontrar a melhor

alavanca mais cedo e, com isso, ter um ganho final maior do que se tivesse adotado uma estratégia puramente gulosa. A Figura 9 mostra a comparação da eficácia dos apostadores usando a estratégia gulosa e com a estratégia ϵ-*greedy*, com $\epsilon = 0.1$. Onde o eixo das abscissas representa quantas vezes o apostador puxou uma alavanca e o das ordenadas o valor que ele obteve ao acionar a alavanca escolhida. Os valores representados no gráfico são a média de 200 jogos. É possível observar que a estratégia gulosa normalmente se estabiliza em uma alavanca com um valor bom de recompensa antes da estratégia ϵ-*greedy*. No entanto, a estratégia ϵ-*greedy* consegue encontrar um valor melhor de alavanca na maioria das vezes, conseguindo uma melhor média de recompensa por ativação do que a estratégia gulosa.

Figura 9 – Comparação dos valores médios de recompensa, por acionamento de alavanca, das estratégias gulosa e a ϵ-*greedy* com $\epsilon = 0.1$.



Fonte: Figura produzida pelo autor.

Para se manter o registro da média de recompensa obtida por cada política, utiliza-se

a Equação da média dinâmica, definida por (35):

$$R_{i,t+1} = R_{i,t} + \frac{Q_{i,t} - R_{i,t}}{O_{i,t}} \tag{4.1}$$

onde $R_{i,t}$ é a recompensa média obtida pela política i na iteração t, $Q_{i,t}$ é o valor de recompensa obtido pela política i na iteração t, e $O_{i,t}$ é o número de vezes que a política i foi tomada.

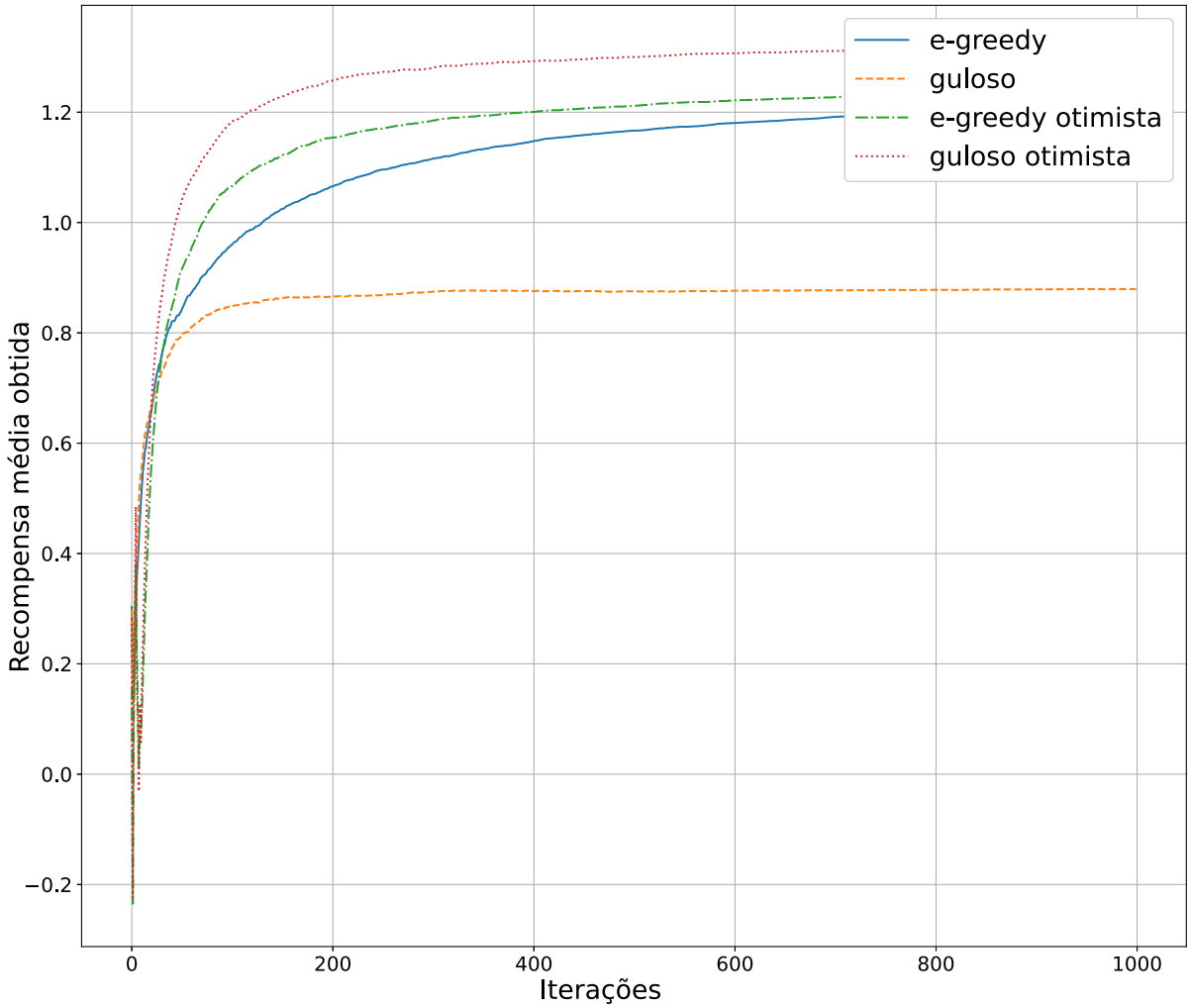
4.1.1 Inicialização com valores otimistas

No problema clássico do *K-armed Bandits*, o apostador assume inicialmente que cada alavanca dá recompensa 0. Isto é, valor arbitrário que não é o pior nem o melhor, já que valores positivos e negativos de recompensa são possíveis. Independente do valor inicial assumido, a Equação 4.1 garante que a estimativa do apostador vai convergir para a recompensa média de cada alavanca (35), desde que haja iterações o suficiente.

O valor inicial influencia na velocidade de convergência. Valores iniciais muito ruins, como por exemplo as recompensas negativas no problema dos k-*armed bandits*, retardam a convergência, pois nas iterações iniciais, a tendência vai ser o agente tomar repetidamente as politicas com que teve contato. Cada política que ele não assumiu vai ter um valor de recompensa muito pior que as demais e só terá chance de ser tomada durante a fase exploratória.

O oposto acontece quando se assume valores iniciais otimistas, como por exemplo recompensas altas no problema do k-*armed bandits*. Nesse caso, nas iterações iniciais, cada política que for tomada vai ter sua recompensa estimada como sendo pior que as demais. Isso força com que o agente alterne entre as políticas nas iterações iniciais, enquanto age com atitude gulosa. Esse comportamento vai se repetir até que as melhores políticas, que têm valores médios de recompensa mais próximos dos valores otimistas, se sobressaiam. A Figura 10 mostra a diferença entre a adoção de valores iniciais otimistas e não-otimistas para o problema com as alavancas da Figura 8.

Figura 10 – Comparação entre valor médio de recompensa média obtida por iteração entre a estratégia gulosa e a ϵ-*greedy* e suas variações com inicialização de valores otimistas. $\epsilon = 0.1$



Fonte: Figura produzida pelo autor.

4.1.2 Estratégia para problemas não estacionários

No caso de problemas de recompensa não estacionária, ou seja, naquele em que as recompensas das políticas se desviam de suas distribuições iniciais ao longo do tempo, a convergência do sistema devido à Equação 4.1 acaba sendo prejudicial ao sistema de aprendizagem. Nesta equação, não há diferença de peso entre as recompensas das ações tomadas mais recentemente das demais. Porém, as recompensas obtidas no passado referem-se às distribuições de recompensas que não são condizentes com as distribuições mais recentes. Ou seja, não ponderar os valores de recompensa de acordo com quão atual é a iteração favorece com que informação obsoleta influencie no sistema.

A solução para esse problema é manter um peso fixo para a diferença entre o valor

da recompensa ganha e a recompensa estimada (35). Essa mudança resulta na seguinte função de recompensa:

$$R_{i,t+1} = R_{i,t} + \alpha(Q_{i,t} - R_{i,t}) \tag{4.2}$$

onde $\alpha \in [0, 1]$ determina o peso que o valor de recompensa obtido ($Q_{i,t}$) pela política i na iteração t tem no cálculo de estimativa de valor médio da recompensa da mesma. Um $\alpha = 0$ inibe o aprendizado, uma vez que o valor médio estimado da recompensa não é atualizado, já $\alpha = 1$ implica que o valor médio estimado da recompensa vai ser sempre igual o último valor de recompensa ($Q_{i,t}$) obtido pela própria política i. Essa função de estimativa de recompensa média é não convergente e é isso que possibilita que o operador de aprendizado se adapte à mudança de valores médios das políticas (35).

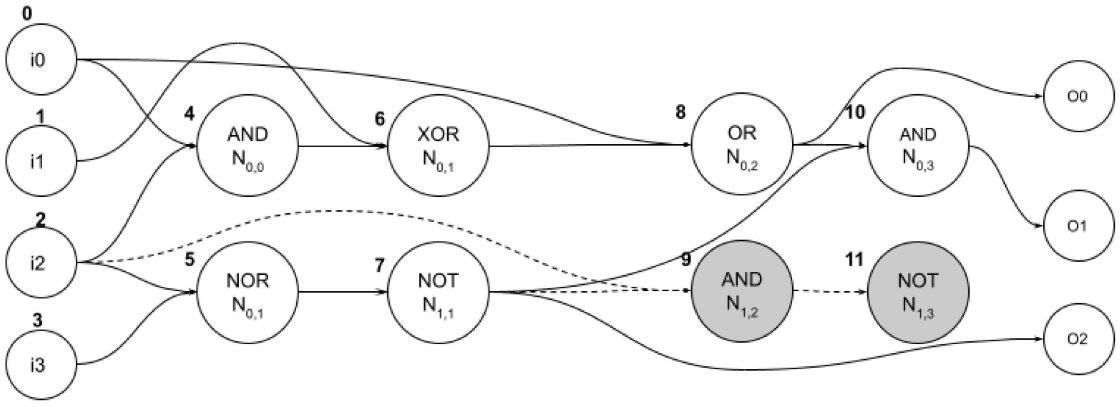
5 PROGRAMAÇÃO GENÉTICA CARTESIANA

Proposta por Miller (27, 28), a Programação Genética Cartesiana é um método de programação genética onde os indivíduos são representados como um conjunto de nós dispostos em uma matriz. Cada nó tem uma função e entradas. Aplicadas ao projeto e otimização de CLCs, as funções são portas lógicas e o número de entradas varia de acordo com o tipo de porta, sendo geralmente duas.

Embora Miller assuma que a CGP possa trabalhar com ciclos, tradicionalmente um nó de uma determinada coluna só pode receber como entradas nós das colunas anteriores. Assim, o indivíduo da CGP pode ser visto como um digrafo acíclico. Outro atributo da CGP é o *levelsback*. Dado um nó na posição k, este pode ter os nós entre as posições $k-$ *levelsback* e $k-1$ como suas entradas. Além disso, qualquer saída ou nó pode receber as entradas do sistema como sua entrada.

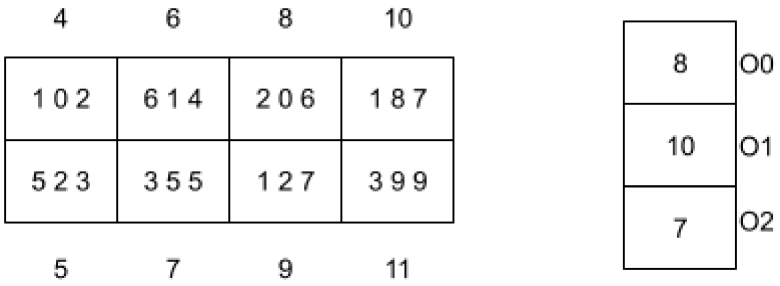
A Figura 11 ilustra a representação de uma solução da CGP. É possível observar que alguns nós não possuem caminho para uma das saídas do circuito. Esses nós, em cinza na figura, são chamados de nós inativos e não afetam o fenótipo, mas carregam informações evolutivas por deriva genética. A Figura 12 mostra o genótipo da solução candidata apresentada na Figura 11. Existem três valores em cada nó: os dois últimos são os nós de entrada e o primeiro valor representa uma função lógica. isto é mapeado como 1 sendo a função AND, 2 OR, 3 NOT, 5 NOR e 6 XOR.

Figura 11 – Representação de uma solução da CGP.



Fonte: Figura produzida pelo autor.

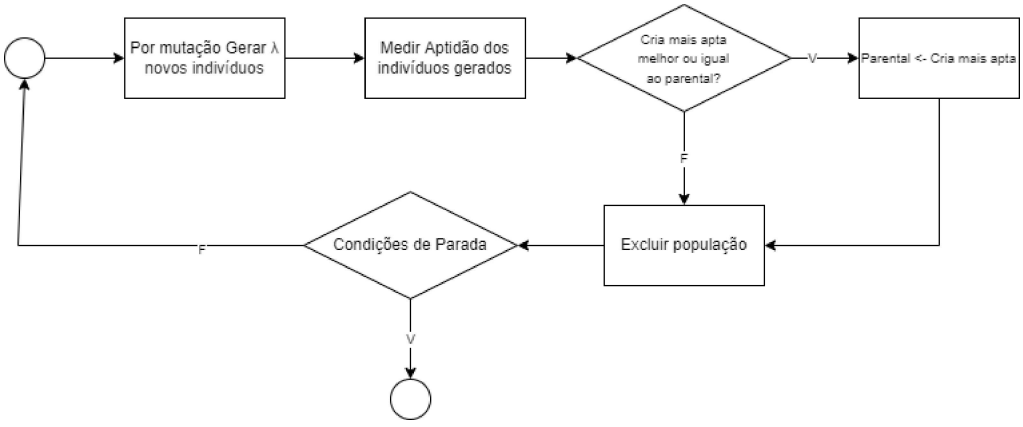
Figura 12 – Genótipo de uma solução da CGP.



Fonte: Figura produzida pelo autor.

O Algoritmo 1 e a Figura 13 descrevem o processo de otimização de CLCs pela CGP. Neste a CGP é inicializada com uma população de $\mu + \lambda$ indivíduos onde μ é o número de progenitores e λ o número de filhos. É comum utilizar a configuração $1 + 4$ na otimização de CLCs (8, 28) e, portanto, tal configuração é adotada neste trabalho. Dentro dessa população, o indivíduo mais apto é selecionado para ser o progenitor no começo do processo evolutivo. Durante o processo evolutivo o progenitor gera λ indivíduos por mutação e em seguida a função de aptidão é aplicada. Caso o individuo com melhor aptidão tenha a mesma aptidão que seu progenitor, o progenitor é sempre descartado. Esse processo se repete até que uma condição de fim seja alcançada.

Figura 13 – Diagrama que demonstra o processo evolutivo da CGP tradicional.



Fonte: Figura produzida pelo autor.

5.1 OPERADORES DE MUTAÇÃO

A CGP possui diferentes tipos de operadores de mutação. Será abordado nessa seção o operador Mutação Pontual, que é o operador inicialmente proposto para a CGP, e o *Single Active Mutation* (SAM), considerado o melhor operador dentre os comparados em (12).

Algorithm 1 Programação Genética Cartesiana

```
1: procedure CGP(μ, λ, população = [individuo 0, individuo 1, ..., individuo μ + λ − 1 ])
2:     Seleciona os melhores μ indivíduos como progenitores e descarta os demais
3:     while Condições de parada não são alcançadas do
4:         for i de 0 até λ do
5:             Realizar mutação em um progenitor P dentre os μ progenitores para gerar novo individuo c_i
6:         Seleciona os melhores μ indivíduos como possíveis progenitores CP
7:         for i de 0 até μ do
8:             Seleciona os λ+μ melhores indivíduos da população como novos progenitores.
9:             Elimine os indivíduos da população, preservando apenas os progenitores P
10:    Retorna o progenitor mais apto
```

A diferença entre esses operadores está em como estes selecionam quais nós sofrerão mutação. No tradicional Mutação Pontual (Algoritmo 2), uma quantidade pré-definida de nós é sorteada e então passam pelo processo de mutação. Não há garantias de que dentre os nós sorteados pela Mutação Pontual exista um nó ativo e portanto que o individuo criado por essa mutação seja fenotipicamente diferente de seu parental. Por esse motivo, circuitos com muitos nós inativos podem ser problemáticos para o operador de Mutação Pontual. Quanto mais nós inativos, maior a chance de apenas estes serem selecionados para a mutação e portanto gerar um indivíduo fenotipicamente idêntico ao progenitor. Isso diminui as chances de melhoria durante o processo evolutivo, uma vez que isso afeta negativamente a taxa de convergência para uma solução melhor na CGP.

Algorithm 2 Operador de mutação pontual (PM)

```
1: procedure OPERADOR PM(Progenitor P, Quota de nós A)
2:     CT = 0 ← Contador de nós que sofreram mutação
3:     while CT < A do
4:         Escolha um nó aleatório em P
5:         Escolha um elemento aleatório (entrada ou tipo de porta lógica)
6:         Realize a mutação no elemento do nó escolhido
7:         CT++
```

O operador SAM (Algoritmo 3), por outro lado, repetidamente escolhe um nó aleatório, realiza a mutação neste e verifica se tal nó é ativo. Após um nó ativo ser modificado, o operador finaliza o processo de mutação no individuo. Isso faz com que, na maioria das vezes[1]. , o individuo gerado por tal tipo de mutação seja fenotipicamente

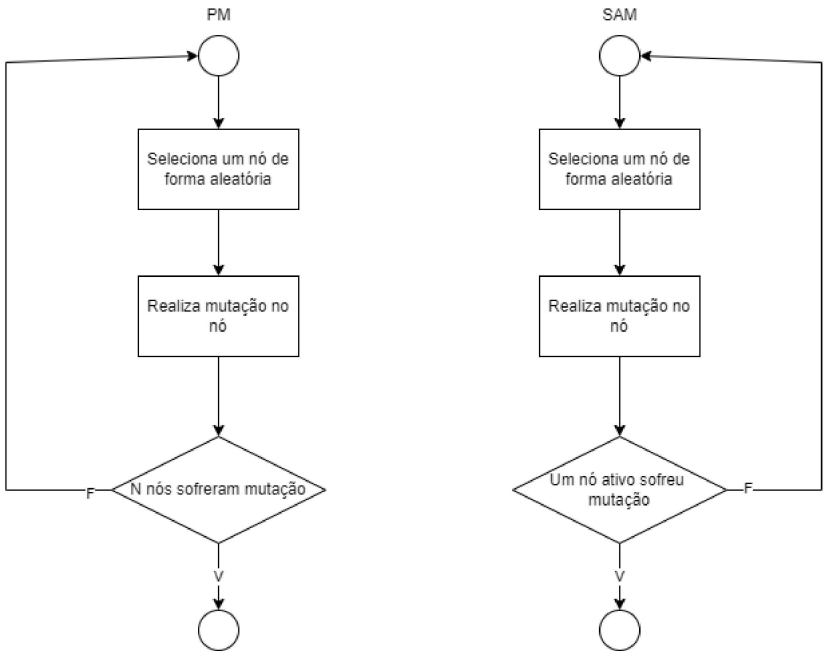
[1] Do ponto de vista de estrutura do circuito, a SAM pode gerar um individuo fenotipicamente idêntico ao seu progenitor fazendo uma mutação redundante, como por exemplo trocar uma porta lógica por ela mesma. Do ponto de vista da tabela verdade/BDD do circuito, a SAM pode gerar indivíduos logicamente idênticos por diversos meios, como por exemplo quando uma das entradas do nó ativo modificado for uma tautologismo, ou uma contradição nesse caso algumas trocas de porta lógica podem resultar na mesma saída.

diferente do seu pai, o que favorece a taxa de convergência para uma solução melhor. A Figura 14 mostra o fluxo do operador PM e do operador SAM. É possível observar que a única diferença é a forma como cada operador define quando encerrar o processo de mutação nos nós de um individuo.

Algorithm 3 Operador SAM.

```
1: procedure OPERADOR SAM(Parental P)
2:     while Nenhum nó ativo sofreu mutação do
3:         Escolha um nó aleatório em P
4:         Escolha um elemento aleatório (entrada ou tipo de porta lógica)
5:         Realize a mutação no elemento do nó escolhido
```

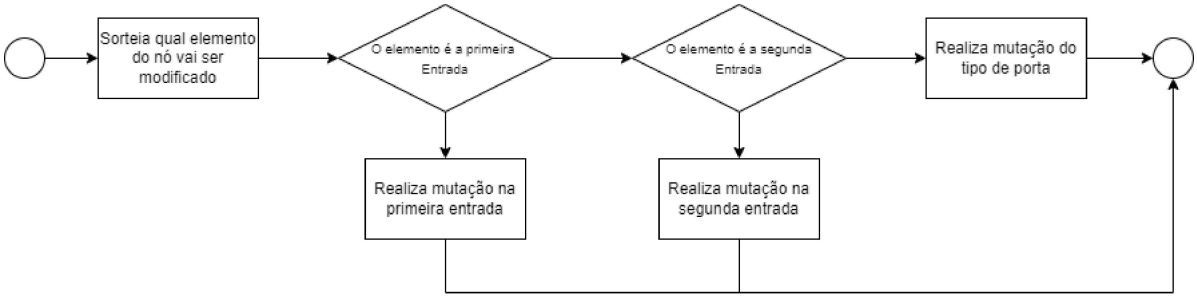
Figura 14 – Diagrama demonstrando o operador de mutação pontual (PM) em comparação com o operador *Single Active Mutation* (SAM). Apenas a forma de decidir quando parar de realizar as mutações muda.



Fonte: Figura produzida pelo autor.

A mutação nos nós, começa com o operador escolhendo aleatoriamente qual elemento do nó será modificado. No caso da CGP aplicada à CLCs, os elementos são, entradas do nó e sua porta lógica. Caso uma das entradas seja selecionada, ela pode ser modificada para ser qualquer outro nó dentro do limite estabelecido pelo *levelsback* configurado no início da execução da CGP. Caso seja a porta lógica, o operador vai trocar a porta atual por qualquer porta dentro do conjunto de portas possíveis do problema. Todas essas escolhas são feitas de forma não enviesada e com distribuição de probabilidade constante entre os elementos possíveis. A Figura 15 representa esse processo.

Figura 15 – Diagrama demonstrando o processo de mutação em um nó da CGP



Fonte: Figura produzida pelo autor.

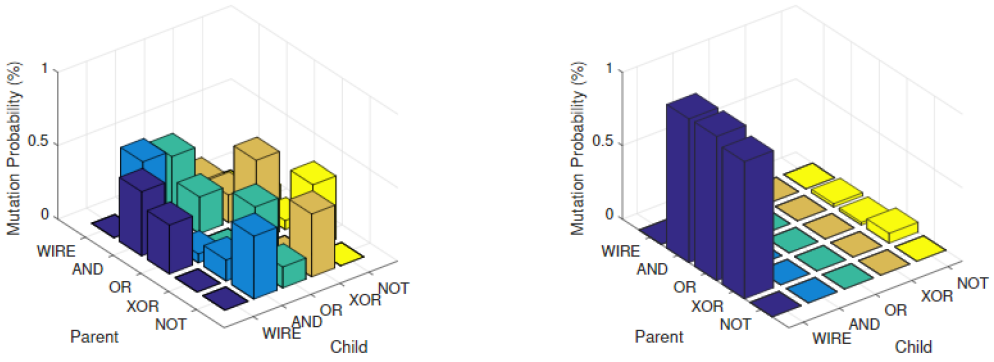
6 MÉTODO PROPOSTO

Ao longo desse estudo, dois métodos foram propostos: a CGP-RL (33) e a Node CGP-RL (32). Ambos os métodos envolvem um operador de mutação adaptativo que usa a estratégia ϵ-*greedy* para enviesar a mutação das portas lógicas (CGP-RL) ou dos elementos do nó (Node CGP-RL). Neste capítulo, ambos os métodos são explicados detalhadamente.

6.1 CGP-RL

No trabalho (7), foi sugerida uma matriz de prioridade de trocas de portas lógicas pelo operador de mutação da CGP. Tal operador funciona da seguinte forma: Ao realizar a mutação no tipo de porta lógica de um nó, o operador consulta uma matriz, tal qual na Figura 16 com um peso de enviesamento para trocas que tem maior chance de resultarem em um melhor individuo. O operador então acessa a linha dessa matriz correspondente ao tipo de porta lógica atual do nó. Cada coluna corresponde a um tipo de porta lógica e os valores de cada célula na linha acessada correspondem as chances da troca da porta lógica atual do nó pela porta correspondente daquela coluna gerar um melhor individuo. Esses valores são convertidos em um aumento ou diminuição da probabilidade daquela troca ocorrer. Em (7) as matrizes tem valores fixos aos quais os autores chegaram após observação do comportamento da CGP em diversos problemas.

Figura 16 – Matrizes de enviesamento de mutação propostas em (7).



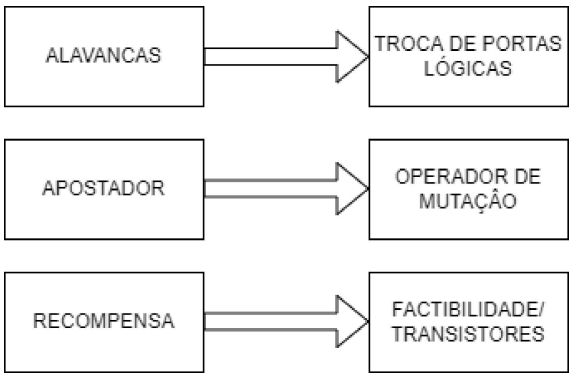
Fonte: SILVA et al., 2018.

O enviesamento proposto pelos autores conseguiu melhorar a convergência para uma solução factível, bem como minimizar a quantidade de portas lógicas nos problemas investigados. Se um operador de mutação com viés estático tem um bom desempenho, é razoável imaginar que um operador adaptativo pode ter um desempenho melhor.

A CGP-RL, como pode ser visto na Figura 17, trata a escolha das portas lógicas como um problema do tipo *k-armed bandits*, onde as alavancas são o tipo de troca a

ser executada, o operador de mutação é o apostador e a aptidão do individuo, neste caso factibilidade para o projeto e número de transistores para a otimização é o valor de recompensa obtida pela aposta (33). O método proposto usa a estratégia ϵ-*greedy* para montar adaptativamente uma tabela de preferência de troca de portas lógicas e, assim, guiar o operador de mutação para as trocas que tem sido mais bem sucedidas e desestimular as piores trocas. Como pode ser visto no Algoritmo 4 e na Figura 18, esta começa com a inicialização de duas matrizes quadradas de tamanho N, onde N é o total de diferentes tipos de portas lógicas que estão sendo utilizadas para resolver aquele problema. Uma das matrizes é a matriz de recompensas médias estimadas (R), que é onde o valor estimado de recompensa das políticas vai estar armazenado. A outra matriz é a matriz de ocorrências (OM), que é onde fica o registro de quantas vezes cada política foi tomada. Em seguida, a população é inicializada da mesma forma que na CGP tradicional, como descrito no Capítulo 5. O algoritmo também é inicializado com uma taxa de exploração ϵ que determina a probabilidade do operador de mutação escolher uma alternativa gulosa, ou adotar uma movimento exploratório.

Figura 17 – Relação da abordagem do problema de escolha de troca de portas lógicas pelo operador de mutação da CPG com o problema alegórico dos *k-armed bandits*.



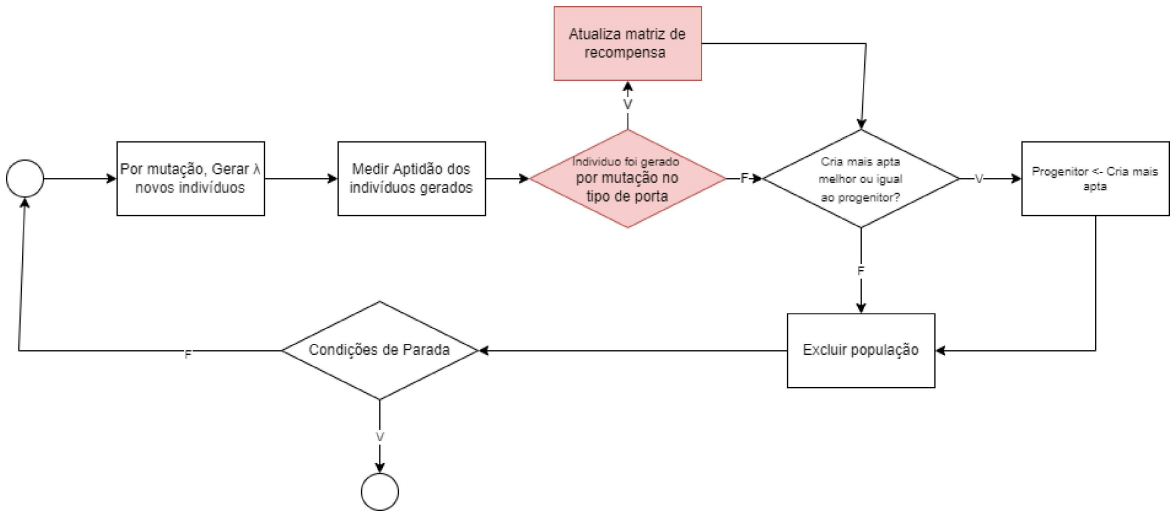
Fonte: Figura produzida pelo autor.

A CGP-RL foi projetada para operar com a mutação SAM. Sendo assim, cada individuo é gerado por uma quantidade variável de mutações em nós inativos e apenas uma mutação, a última, em um nó ativo. Na CGP-RL, cada indivíduo carrega informações sobre essa mutação, ou seja, a mutação no nó ativo que o gerou. Depois que o parental gera λ filhos, a aptidão de cada individuo é medida. A aptidão depende do objetivo do processo evolutivo. No trabalho onde foi proposta, a distância de Hamming com relação ao circuito desejado foi usada para aptidão no processo de design de CLCs enquanto o total de transistores foi usada no processo de otimização de CLCs. Em ambos os casos o individuo mais apto é o que pontuava menor nas métricas utilizadas. A aptidão, junto com a informação sobre qual tipo de troca de porta ocorreu são usadas para atualizar a matriz de recompensas médias estimadas usando a Equação 4.1.

Algorithm 4 CGP-RL

procedure CGP-RL(λ,ϵ)
    $OM \leftarrow$ Matriz de ocorrências
    $R \leftarrow$ Matriz de recompensas estimadas
    while Condições de parada não alcançadas **do**
        for i de 0 até λ **do**
            Realizar mutação CGP-RL em P para gerar c_i, com taxa de exploração ϵ
        for i de 0 até λ **do**
            if Indivíduo c_i foi criado por mutação no tipo de porta lógica **then**
                $g \leftarrow$ tipo de porta lógica no nó ativo de P que sofreu mutação
                $f \leftarrow$ tipo de porta lógica resultante da mutação no nó de c_i
                $OM_{p,f} + +$
                Atualiza $R_{p,f}$, considerando a aptidão de c_i de acordo com a Equação 4.1
        $C_i \leftarrow$ cria mais apta do progenitor P
        if $Aptidão(C_i)$ é melhor ou igual a $Aptidão(P)$ **then**
            $P \leftarrow C_i$
        Elimine os indivíduos da população, preservando apenas P
Retorna P

Figura 18 – Diagrama demonstrando o processo evolutivo da CGP-RL. A diferença com relação à CGP tradicional são os passos em vermelho, correspondentes à atualização da matriz de recompensas de troca de portas.



Fonte: Figura produzida pelo autor.

A atualização das matrizes ocorre da seguinte maneira: primeiramente é obtida a informação sobre a mutação que gerou o individuo e, apenas se essa mutação foi do tipo de porta, as matrizes são atualizadas. Então, usa-se o valor correspondente ao tipo da porta antes da mutação para acessar as linhas das matrizes e o valor depois da mutação para acessar as colunas. OM é atualizada primeiro, com um incremento na célula acessada. Cada troca de tipo de porta é uma política na CGP-RL, portanto, cada célula da OM

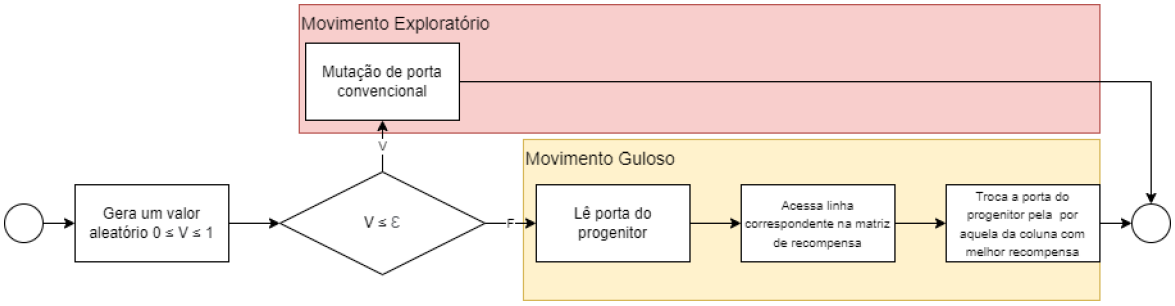
representa a quantidade de vezes que cada política foi tomada. A célula atualizada de OM entra na Equação 4.1 como $O_{i,j}$ onde i é o valor do tipo de porta do parental, e j o valor do tipo de porta do filho. A aptidão do indivíduo entra como a recompensa obtida $Q_{i,j}$. R armazena os valores médios estimados de recompensa e, portanto, $R_{i,j}$ vai ser a recompensa estimada para a política que gerou o individuo.

O operador de mutação funciona em conjunto com o SAM. Na mutação dos nós, primeiramente o operador sorteia qual elemento do nó vai ser modificado (primeira entrada, segunda entrada e porta lógica) de forma não enviesada. Caso a mutação seja nas entradas, ela é feita da mesma forma que na CGP tradicional, sem qualquer viés. Caso a mutação ocorra nas portas lógicas, existe uma probabilidade ϵ do mesmo realizar um movimento de exploração e uma probabilidade de $1 - \epsilon$ dele realizar um movimento guloso, realizando sempre a troca onde obteve a melhor recompensa. No movimento exploratório a mutação da porta ocorre tal qual a CGP tradicional: a porta atual é trocada por uma outra porta sorteada de forma não enviesada. No movimento guloso, o operador de mutação acessa a linha da matriz R correspondente ao valor da porta a ser modificada e procura pelo menor valor da melhor recompensa naquela linha. Então, o operador troca a porta atual pela porta correspondente à coluna onde a melhor recompensa está. O operador ignora a diagonal principal da matriz de recompensas. Após a mutação do tipo de porta, independente do movimento realizado (exploratório ou guloso), o operador registra qual troca de porta foi realizada na criação do individuo. O Algoritmo 5 e descreve o operador de mutação da CGP-RL enquanto a Figura 19 representa a mutação do tipo de porta lógica em um nó.

Algorithm 5 Operador SAM CGP-RL

```
1: procedure Operador SAM CGP-RL(Parental P,ϵ)
2:     while Nenhum nó ativo sofreu mutação do
3:         Escolha um nó aleatório em P
4:         Escolha um elemento aleatório (entradas ou tipo de porta lógica)
5:         if O elemento escolhido foi o tipo de porta lógica then
6:             g ← O tipo de porta lógica no nó sorteado
7:             V ← aleatório(0, 1)
8:             if V ≤ ϵ then
9:                 f ← porta lógica escolhida aleatoriamente
10:            else
11:                R_g ← Linha da matriz de R referente à g
12:                f ← Porta lógica correspondente à coluna da primeira ocorrência do
   menor valor de R_g exceto coluna referente à g
13:            Troque g por f
14:            Registre a troca de portas ocorrida
15:        else
16:            Realize a mutação da entrada sorteada de forma não enviesada
```

Figura 19 – Mutação do tipo de porta lógica pela CGP-RL. O movimento exploratório se refere à mutação não enviesada da CGP tradicional. No movimento guloso a CGP-RL vai sempre executar a troca na qual há a melhor recompensa média obtida



Fonte: Figura produzida pelo autor.

6.2 *Node* CGP-RL

A CGP-RL possui $N^2 - N$ políticas, onde N é o total de tipos de portas disponíveis para a solução do problema. A estimativa de recompensa das políticas é atualizada sempre que uma mutação do tipo de porta lógica acontece, o que corresponde a cerca de um terço das mutações durante o processo evolutivo. A escolha de qual elemento do nó sofrerá mutação, por outro lado, é uma tomada de decisão entre três políticas (realizar mutação na primeira entrada, segunda entrada e porta lógica) e a atualização da estimativa de recompensas ocorre sempre que um nó sofrer mutação. Tal tipo de mutação ocorre sempre.

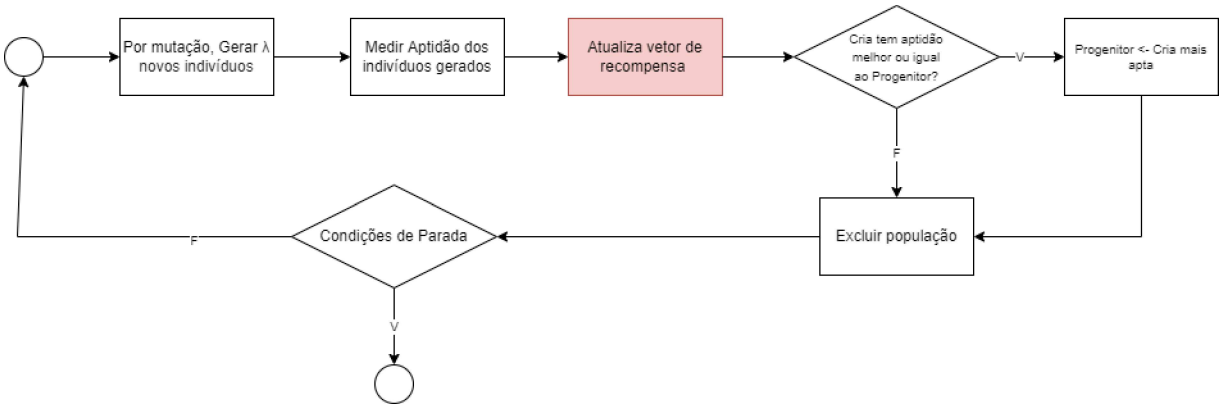
Com menos politicas e uma frequência de aprendizado maior, um operador RL tem maiores chances de convergir mais rápido para a melhor política, desde que haja de fato alguma (35). Para determinar se há uma diferença entre as políticas de mutação de um nó que justifiquem o uso de um operador adaptativo, um estudo foi feito com base nos problemas classificados como pequenos em (8). A discussão sobre os experimentos realizados nesse estudo serão discutidos na Seção 7.1 do Capítulo 7.

Os estudos levaram ao desenvolvimento da Node CGP-RL, uma versão da CGP cujo o operador de mutação tem viés adaptativo sobre qual elemento do nó vai sofrer mutação. A Node CGP-RL funciona de forma parecida com a CGP-RL. O Algoritmo 6 e a Figura 20 descrevem seu funcionamento. Inicialmente o vetor de recompensa média estimada (r) é inicializado. De acordo com os resultados de experimentos que serão vistos na seção 7.1, as políticas de mutação de um nó são de recompensa não estacionária.

Algorithm 6 *Node* CGP-RL

procedure NODE CGP-RL(λ,ϵ)
    $r \leftarrow$ Vetor de recompensas estimadas
    while Condições de parada não alcançadas **do**
        for i de 0 até λ **do**
            Realizar mutação CGP-RL em P para gerar a c_i, tendo ϵ como taxa de exploração
        for i de 0 até λ **do**
            $k \leftarrow$ tipo de mutação no nó ativo que P sofreu para gerar c_i
            Atualiza r_k, considerando a aptidão de c_i de acordo com a Equação 4.2
        $C_i \leftarrow$ cria mais apta do parental P
        if *Aptidão*(C_i) É melhor ou igual a *Aptidão*(P) **then**
            $P \leftarrow C_i$
        Elimine os indivíduos da população, preservando apenas P
Retorna o parental P

Figura 20 – Diagrama demonstrando o processo evolutivo da *Node* CGP-RL. A diferença com relação à CGP tradicional é o passo em vermelho, correspondente à atualização do vetor de recompensas de troca de portas. A diferença com relação À CGP-RL é a ausência do desvio condicional para mutações do tipo de porta: Qualquer individuo que foi gerado por uma mutação em um nó ativo contribui para a atualização do vetor de recompensas.



Fonte: Figura produzida pelo autor.

A inicialização da população ocorre tal qual na CGP tradicional e então o processo evolutivo é iniciado. Cada individuo gerado tem registrado o tipo de mutação que o gerou. Da mesma forma que na CGP-RL, a aptidão de cada individuo e sua informação de mutação são usadas para atualizar o valor recompensa média estimada nas políticas contidos no vetor r, conforme a Equação 4.2, onde $Q_{i,t}$ é o valor de aptidão do individuo, gerado pela política de mutação i na iteração t, $R_{i,t}$ é o valor estimado de recompensa média da política e $0 \leq \alpha \leq 1$ é o peso dado para as informações mais recentes.

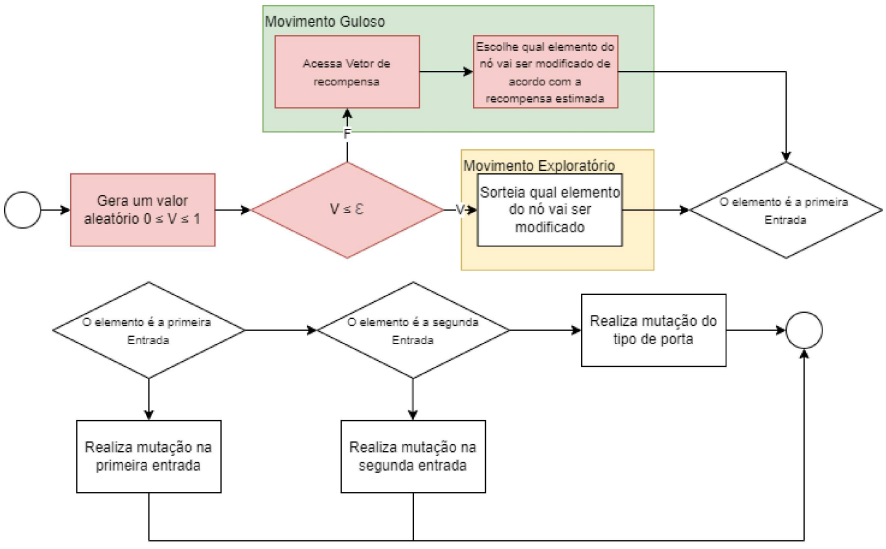
Durante o processo de mutação, para cada nó, o operador tem uma probabilidade ϵ de executar um movimento exploratório e realizar a mutação de forma não enviesada.

Caso contrário, ele acessa o vetor r, busca a política com melhor recompensa e realiza o tipo de mutação correspondente a tal política. Independente da ação, exploratória ou gulosa, o individuo carrega consigo a informação do tipo de mutação que o gerou. Como a CGP-RL foi projetada para trabalhar com o operador SAM, apenas a mutação no nó ativo é considerada pelo operador adaptativo. O Algoritmo 6 e a Figura 21 descrevem o processo do operador de mutação de nó da *Node* CGP-RL.

Algorithm 7 Operador SAM *Node* CGP-RL

procedure OPERADOR SAM NODE CGP-RL(Progenitor P_i,ϵ)
  while Nenhum nó ativo sofreu mutação **do**
    Escolha um nó aleatório em P
    $V \leftarrow aleatório(0, 1)$
    if $V \leq \epsilon$ **then**
      $k \leftarrow$ Elemento a sofrer mutação no nó, escolhido de maneira não enviesada
    else
      $k \leftarrow$ Elemento a sofrer mutação correspondente à primeira ocorrência do menor valor de r
    Realizar mutação do tipo k

Figura 21 – Operador de mutação de nó da *Node* CGP-RL. A diferença para o operador tradicional de nós da CGP são os passos em vermelho e representam a tomada de decisão de o operador vai realizar um movimento exploratório, em amarelo, ou guloso, em verde, bem como a decisão de qual elemento do nó vai sofrer mutação no movimento guloso.



Fonte: Figura produzida pelo autor.

7 EXPERIMENTOS COMPUTACIONAIS

Experimentos computacionais foram realizados utilizando os problemas em (8), exceto para o problema vda devido ao alto tempo de processamento necessário para a resolução do problema. As características dos problemas estão apresentadas na Tabela 3 e são: **Grupo**, que são a forma como os problemas foram classificados em (8), onde 1 é pequeno, com até 8 entradas, 2 é médio com até 16 entradas e 3 é grande, com mais de 16 entradas. **Nome**, que é o nome do problema e o orçamento total, em gerações é dado por **Orçamento**. **Baseline** é a quantidade de transistores da solução para o problema gerada pelo ESPRESSO.**Funcionalidade** é tipo de uso do circuito em questão.

As características **Entradas** e **Saídas**, representam respectivamente a quantidade de variáveis de entrada e saída de um circuito e portanto, no tamanho de sua tabela verdade, impactando também em sua complexidade.

Balanceamento é proporção entre o total de ocorrências do valor de saída mais frequente e o total de valores de saída na tabela verdade. Por exemplo se um circuito tem 4 entradas, tal vai ter 16 linhas de tabela verdade. Se este mesmo circuito tem 2 saídas, terá $2 \times 16 = 32$ valores de saída. Se seu valor de saída mais frequente é 1, tendo 24 ocorrências, seu balanceamento será de 0,75. Dessa forma, nunca haverá circuitos com balanceamento menor que 0,5 ou maior que 1.

A taxa de **Simplificação**, é a proporção entre a quantidade de portas lógicas necessárias para construir o problema em (41) e a quantidade de portas lógicas necessárias para construir a solução pela soma de produtos. De acordo com (14), quanto que apresentam melhores taxas de simplificação tendem a serem mais facilmente otimizados por métodos evolucionários.

Os experimentos da CGP-RL, ao contrário dos demais, foram realizados em um computador com processador Intel i5 3230m @ 2,60 GHz dual-core, 6GB DDR3 DIMM e sistema operacional Ubuntu 18.04.4 LTS. Os demais experimentos foram realizados em um computador com 16 CPUs de 2,4 GHz - 64 bits, 32 GB de RAM e sistema operacional Ubuntu 18. A maioria dos experimentos foi inicializada com uma população gerada de maneira aleatória, os demais experimentos, que foram inicializados com uma população factível gerada por ESPRESSO. O algoritmo ESPRESSO foi implementado em (8) e o código disponível em [1], são discutidos nas Seções 7.2 e **7.3.2**.

[1] https://github.com/ciml/ciml-lib/tree/applied-soft-computing-2019/src foi usado como base para as o código implementado neste trabalho

Tabela 3 – Problemas usados para comparar os algoritmos.

Grupo	Nome	Entradas	Saídas	Balanceamento	Simplificação	Orçamento	*Baseline*	Funcionalidade
1	C17	5	2	0.93750	3.000E-1	1.20E+7	26	Lógico
	cm42a	4	10	0.93750	1.921E-2	1.28E+7	156	Lógico
	cm82a	5	3	0.50000	7.872E-2	8.00E+6	159	Lógico
	cm138a	6	8	0.98437	3.758E-3	1.92E+7	148	Lógico
	decod	5	16	0.96875	2.292E-1	1.20E+7	132	Decoder
	f51m	8	8	0.50000	3.510E-3	1.92E+7	638	Aritimético
	majority	5	1	0.65625	6.207E-2	8.00E+6	24	Voter
	z4ml	7	4	0.50000	7.622E-3	1.68E+7	503	Somador 2-bit
2	9symml	9	1	0.82031	7.585E-3	2.16E+7	1039	Contador
	alu2	10	6	0.63363	9.658E-3	1.60E+7	1790	ALU
	alu4	14	8	0.56482	5.250E-4	3.36E+7	10336	ALU
	cm85a	11	3	0.73437	9.110E-4	1.76E+7	610	Lógico
	cm151a	12	2	0.75000	4.480E-4	2.88E+7	154	Lógico
	cm162a	14	5	0.78125	3.900E-5	4.48E+7	200	Lógico
	cu	14	11	0.92436	8.800E-5	4.48E+7	261	Lógico
	x2	10	7	0.80915	5.520E-4	2.40E+7	174	Lógico
3	cc	21	20	0.70703	1.137E-7	5.04E+7	256	Lógico
	cmb	16	4	0.99976	1.300E-5	3.84E+7	144	Lógico
	cordic	23	2	0.91595	3.423E-7	7.36E+7	27669	Lógico
	frg1	28	3	0.73570	9.255E-9	6.72E+7	1605	Lógico
	pm1	16	13	0.80183	4.000E-6	3.84E+7	2084	Lógico
	sct	19	15	0.74722	8.180E-7	4.56E+7	466	Lógico
	t481	16	1	0.64111	2.055E-3	2.56E+7	9518	Lógico
	tcon	17	16	0.50000	1.564E-6	4.08E+7	49	Lógico

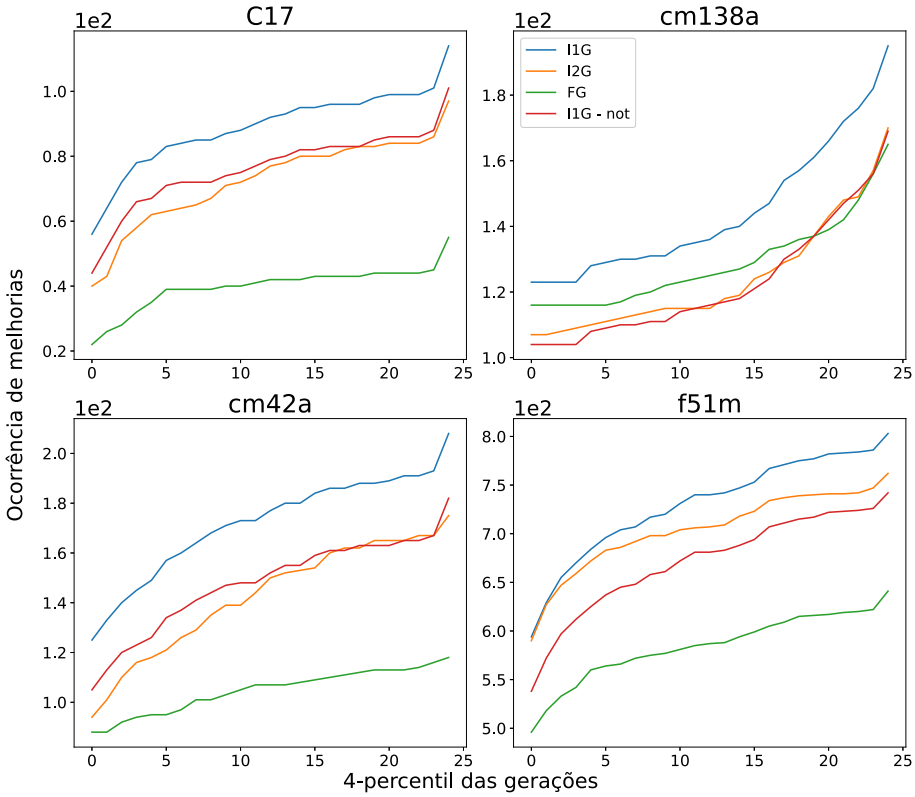
Fonte: Souza et al, 2020.

Na Seção 7.1 é feita uma análise do comportamento evolutivo da CGP usando SAM: quais tipos de mutação geram melhores indivíduos com maior frequência e com qual frequência indivíduos factíveis surgem durante o processo de otimização. Tal estudo serviu de base para a proposta da Node CGP-RL. Na Seção 7.2 são analisados os experimentos da CGP-RL e na Seção 7.3 serão analisados os da *Node* CGP-RL

7.1 ANÁLISES SOBRE A CGP

Para determinar se algum elemento do nó tem maior probabilidade de gerar melhores indivíduos, foi analisada a frequência acumulada com que a mutação em cada elemento aumenta a qualidade da solução atual ao longo do processo evolutivo. O estudo foi feito nos problemas identificados como pequenos em (8), usando a mutação SAM e com 25 execuções independentes por problema. Cada ocorrência de melhoria foi agrupada por um intervalo escolhido arbitrariamente que representa 4-percentil da quantidade de gerações utilizada cada etapa: projeto, que é quando o objetivo é chegar a um indivíduo factível e otimização, que é quando se tenta reduzir a quantidade de transistores de tal indivíduo. Os dados são mostrados em gráficos nas Figuras 22, 23, 24 e 25.

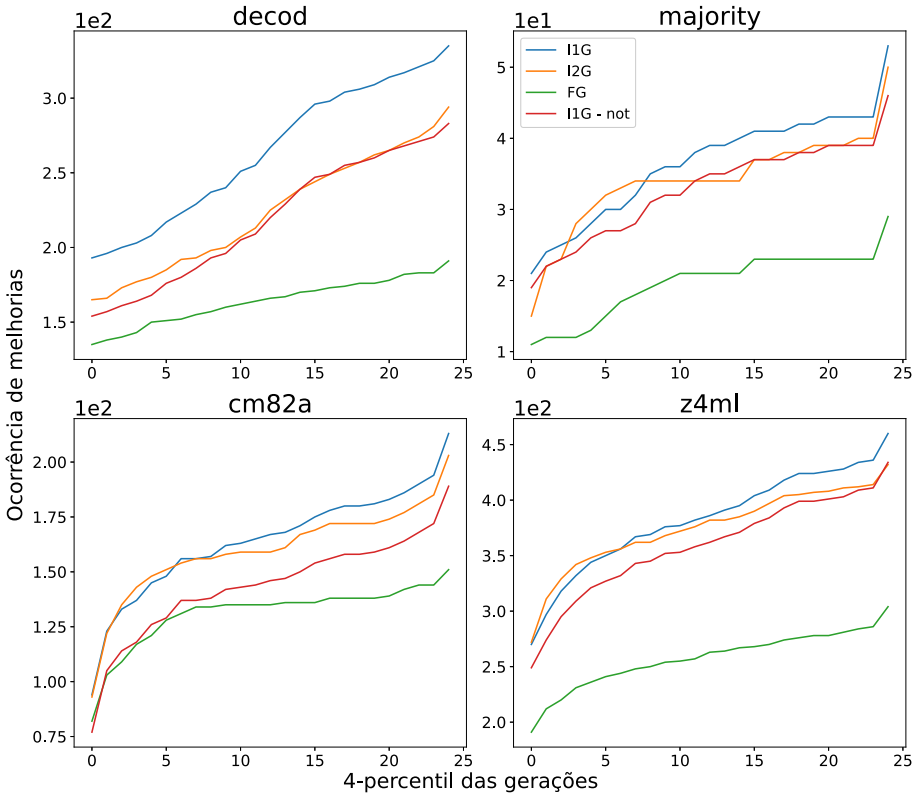
Figura 22 – Ocorrência de melhorias nos quatro primeiros problemas de tamanho pequeno, durante a fase de projeto, onde o objetivo é gerar um indivíduo factível. I1G representa melhorias geradas por mutação na primeira entrada, I2G na segunda entrada e FG mutações no tipo de porta lógica. I1G - not representa o valor de I1G desconsiderando as mutações em nós cujo a porta lógica é do tipo NOT.



Fonte: Figura produzida pelo autor.

As Figuras 22, 23 apresentam o valor cumulativo de cada ocorrência de melhoria na solução atual gerada por mutações na 1ª entrada (I1G, Azul), segunda entrada (I2G, Laranja) e no tipo de porta lógica (FG, em verde) na fase de projeto. Como a porta NOT só utiliza a 1ª entrada, foi registrado também a quantidade de melhorias causadas por mutação na 1ª entrada desconsiderando aquelas em que a porta lógica é do tipo NOT (I1G - not, Vermelho). Pode-se notar que todos os problemas mostraram padrões semelhantes para a fase de projeto, com mutações nas entradas sendo responsáveis por gerar indivíduos melhores com mais frequência do que mutações no tipo de porta. As diferenças entre mutações na primeira entrada e segunda entrada podem ser explicadas por uma particularidade na implementação de (8), que foi utilizada como base para a CGP utilizada neste trabalho. Nesta implementação as portas do tipo NOT usam sempre o primeiro endereço de entrada. Quando desconsiderada as mutações nas portas NOT, a frequência com que as mutações do tipo I1G geram melhores individuos são próximas da I2G.

Figura 23 – Ocorrência de melhorias nos quatro últimos problemas de tamanho pequeno, durante a fase de projeto, onde o objetivo é gerar um indivíduo factível. I1G representa melhorias geradas por mutação na primeira entrada, I2G na segunda entrada e FG mutações no tipo de porta lógica. I1G - not representa o valor de I1G desconsiderando as mutações em nós cujo a porta lógica é do tipo NOT.

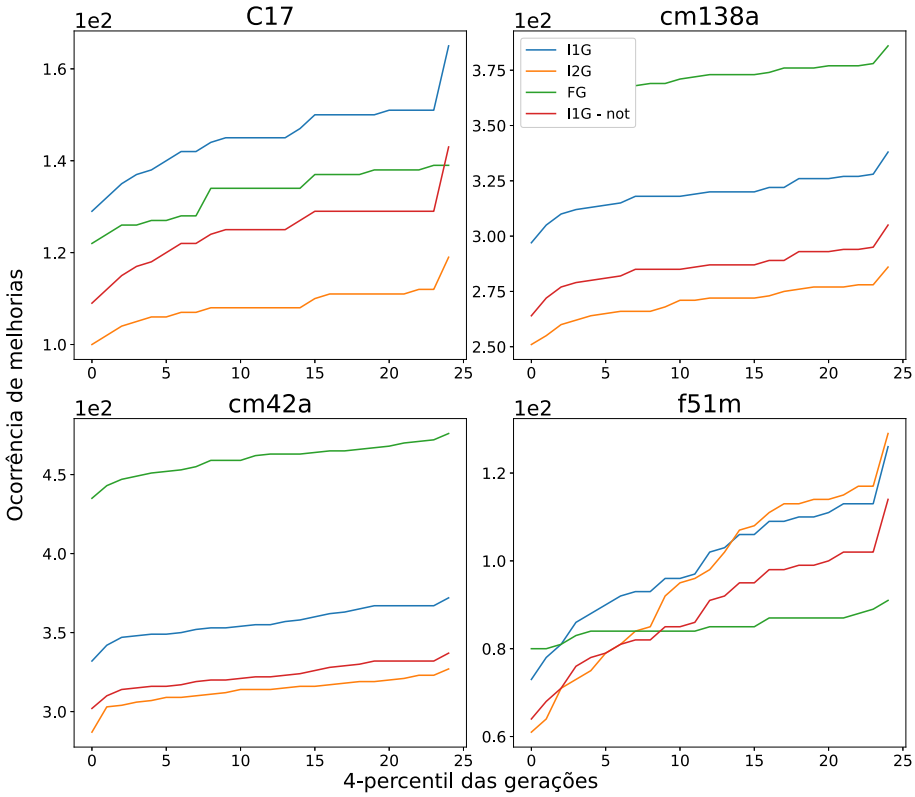


Fonte: Figura produzida pelo autor.

De forma geral as curvas mostram que as mutações no tipo de porta geram indivíduos melhores em cerca de 25% a 30% da frequência com que as mutações nas entradas geram. Algumas diferenças na forma como a curva evolui e as proporções desses impactos sugerem que um operador de mutação adaptativo é mais apropriado do que um viés estático, e espera-se que tal operador seja capaz de melhorar o desempenho de otimização de CLCs da CGP.

Na fase de otimização (Figuras 24 e 25) cada problema mostra um padrão particular. As diferenças entre a frequência de melhora para cada tipo de mutação são menos evidentes. No entanto, estas ainda são distintas para apontar que certas políticas de mutação podem ser mais bem sucedidas do que outras e, portanto, que um operador adaptativo pode tirar vantagem positiva dessa característica.

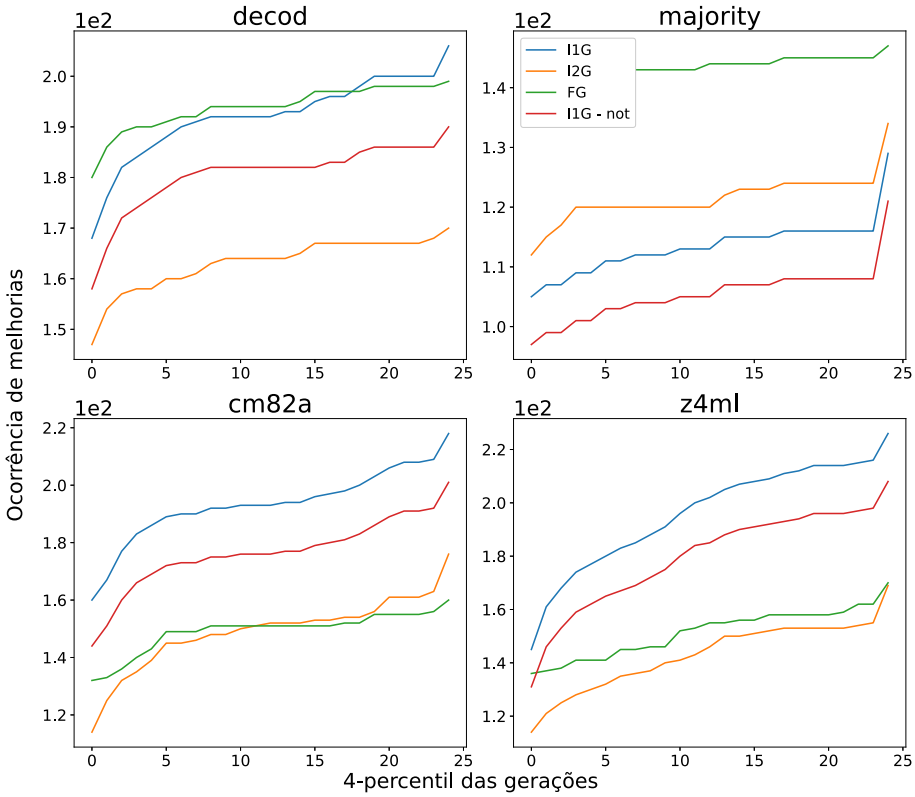
Figura 24 – Ocorrência de melhorias nos quatro primeiros problemas de tamanho pequeno, durante a fase de otimização, onde o objetivo é reduzir o total do circuito factível.I1G representa melhorias geradas por mutação na primeira entrada, I2G na segunda entrada e FG mutações no tipo de porta lógica. I1G - not representa o valor de I1G desconsiderando as mutações em nós cujo a porta lógica é do tipo NOT.



Fonte: Figura produzida pelo autor.

Para problemas em que a CGP tem a maior taxa de sucesso, a maioria das avaliações de função objetivo e restrições são usadas durante a fase de otimização. Este é o caso para todos os problemas pequenos de (8). No entanto, as Figuras 22, 23, 24 e 25 mostram que o número total de vezes que um indivíduo melhor aparece é similar para as fases de projeto e otimização. Espera-se uma maior dificuldade em gerar melhores indivíduos na fase de otimização, já que na otimização existe a restrição de factibilidade.

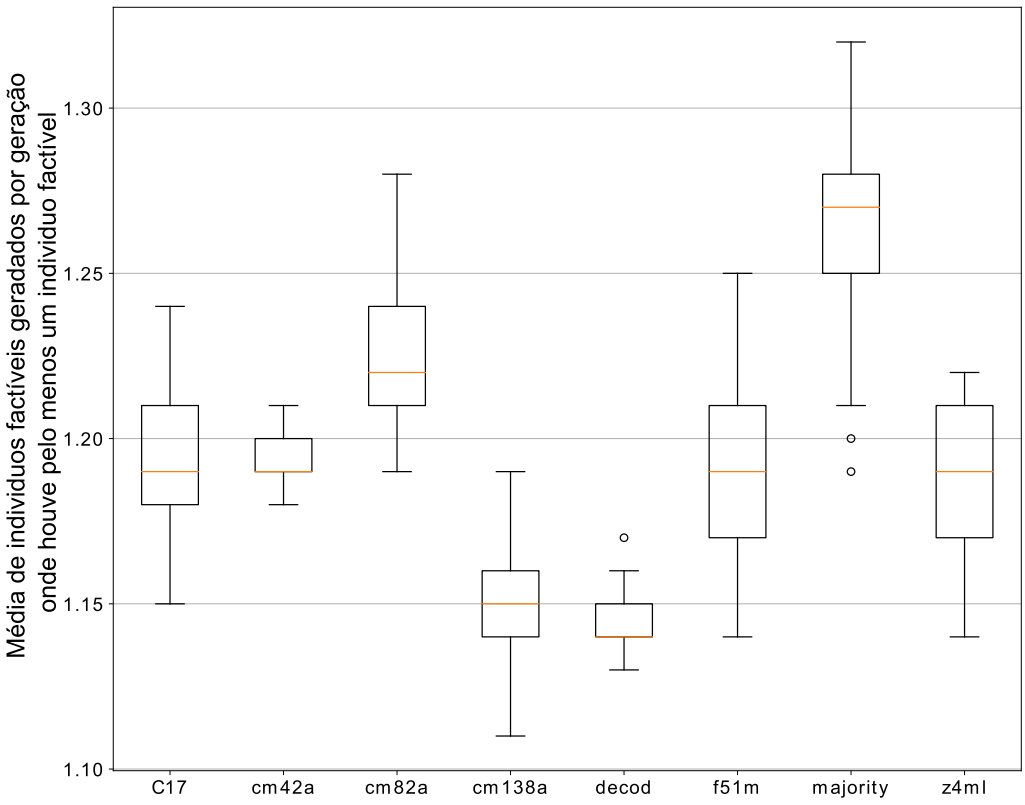
Figura 25 – Ocorrência de melhorias nos quatro últimos problemas de tamanho pequeno, durante a fase de otimização, onde o objetivo é reduzir o total do circuito factível.I1G representa melhorias geradas por mutação na primeira entrada, I2G na segunda entrada e FG mutações no tipo de porta lógica. I1G - not representa o valor de I1G desconsiderando as mutações em nós cujo a porta lógica é do tipo NOT.



Fonte: Figura produzida pelo autor.

Em problemas pequenos, a CGP gerou indivíduos viáveis em cerca de metade das gerações do processo de otimização. Para essas gerações, a média de indivíduos viáveis foi de 1,2. Ambos os valores podem ser observados na Tabela 4 e nos *boxplots* das Figuras 26 e 27. Os gráficos vistos nessa seção mostram que cada tipo de mutação tem um impacto diferente na geração de indivíduos viáveis, com mutações no tipo de função lógica parecendo ter uma maior capacidade para oferecer indivíduos viáveis do que mutações na entrada.

Figura 26 – Ocorrência média de indivíduos factíveis, em gerações onde ao menos um dos indivíduos gerados é factível.



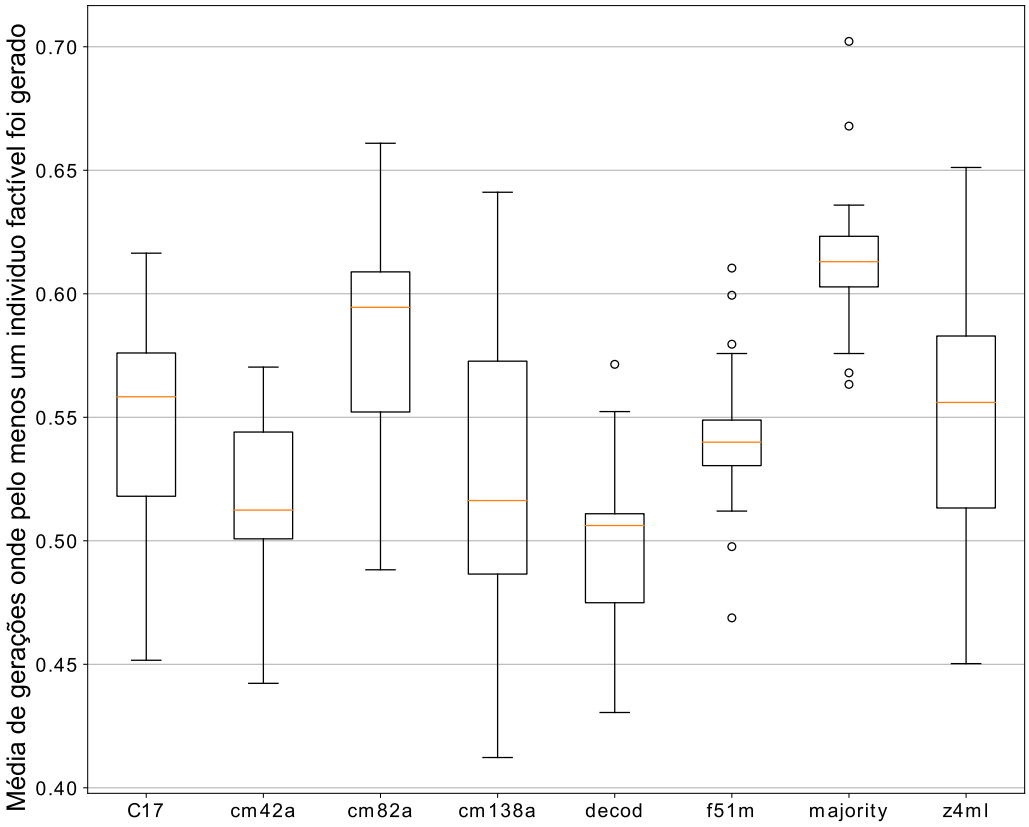
Fonte: Figura produzida pelo autor.

Tabela 4 – Média de gerações nas quais houve pelo menos um individuo factível gerado durante a fase de otimização, na CGP e na *Node* CGP-RL.

Problema	CGP	*Node* CGP-RL
C17	54%	33%
cm42a	52%	30%
cm82a	58%	35%
cm138a	53%	28%
decod	50%	28%
f51m	54%	32%
majority	61%	36%
z4ml	55%	31%

Fonte: Elaborada pelo autor.

Figura 27 – Gerações onde pelo menos um indivíduo factível foi gerado



Fonte: Figura produzida pelo autor.

7.2 AVALIAÇÃO COMPARATIVA DA CGP-RL

A abordagem proposta foi implementada na linguagem de programação C e está disponível publicamente[2].

A Tabela 5 mostra os dados de execuções de cada problema para cada algoritmo. Foram registrados os resultados com a menor quantidade de transistores obtida por cada algoritmo (Melhor), sua taxa de sucesso, relacionada à quantidade de execuções onde foi gerada pelo menos uma solução factível (TS), o número médio de transistores obtidos pelas soluções (Med) e o desvio padrão (DP). Nas coluna com terminação R (CGP-RL-R e SAM-R) a população inicial foi gerada de forma aleatória, sendo não factível. As colunas com terminação E (CGP-RL-E e SAM-E) a população inicial era factível e foi gerada pelo algoritmo ESPRESSO. Os melhores resultados estão expressos em negrito.

[2] `https://github.com/FMoller/cgp-rl`

Tabela 5 – Resultados comparativos entre a CGP-RL e CGP usando apenas SAM.

Problema	CGP-RL R				SAM-R				CGP-RL E				SAM-E			
	Melhor	TS	Med	DP	Melhor	TS	Med	DP	Melhor	TS	Med	DP	Melhor	TS	Med	DP
C17	**9**	**100%**	10.32	1.95E+00	**9**	**100%**	9.80	**4.90E-01**	**9**	**100%**	10.48	1.17E+00	**9**	**100%**	**9.60**	8.00E-01
cm42a	**29**	**100%**	34.84	3.70E+00	**29**	**100%**	**33.48**	2.65E+00	30	**100%**	35.96	**2.41E+00**	31	**100%**	35.56	2.65E+00
cm82a	**19**	**100%**	30.00	6.66E+00	20	**100%**	27.36	6.67E+00	20	**100%**	24.76	3.85E+00	**19**	**100%**	**22.40**	**3.36E+00**
cm138a	28	**100%**	32.48	3.44E+00	**27**	**100%**	**32.36**	2.94E+00	34	**100%**	39.00	3.41E+00	32	**100%**	38.64	**2.91E+00**
decod	**36**	**100%**	**48.04**	6.83E+00	41	**100%**	53.12	**5.52E+00**	54	**100%**	70.44	6.84E+00	56	**100%**	67.80	8.31E+00
f51m	**69**	88%	**89.41**	1.46E+01	74	**100%**	89.60	**1.22E+00**	363	**100%**	414.92	2.78E+01	344	**100%**	412.32	2.49E+01
majority	**11**	**100%**	13.76	2.64E+00	**11**	**100%**	**12.92**	2.56E+00	**11**	**100%**	16.52	2.53E+00	**11**	**100%**	14.00	1.67E+00
z4ml	35	**100%**	**46.08**	9.12E+00	35	**100%**	48.08	**8.86E+00**	38	**100%**	83.00	2.84E+01	**33**	**100%**	65.32	2.56E+01
9symml	83	**100%**	**128.64**	4.69E+01	**77**	**100%**	128.84	**3.95E+01**	171	**100%**	309.40	6.56E+01	197	**100%**	262.92	5.03E+01
alu2	-	0%	-	-	**306**	8%	**327.50**	**2.15E+01**	508	**100%**	600.16	4.77E+01	497	**100%**	595.76	4.20E+01
alu4	**149**	80%	**207.00**	4.20E+01	158	92%	209.70	**3.40E+01**	-	-	-	-	8333	**100%**	8685.25	2.33E+02
cm85a	**38**	**100%**	**51.56**	8.01E+00	42	**100%**	51.84	**7.14E+00**	40	**100%**	87.80	3.96E+01	43	**100%**	69.12	2.07E+01
cm151a	**32**	**100%**	**45.52**	8.40E+00	36	**100%**	46.00	8.11E+00	42	**100%**	51.48	6.05E+00	45	**100%**	51.36	**3.45E+00**
cm162a	**54**	**100%**	66.60	8.48E+00	**54**	**100%**	66.00	5.97E+00	59	**100%**	65.96	5.85E+00	59	**100%**	**64.84**	**3.58E+00**
cu	**51**	**100%**	**62.16**	5.84E+00	58	**100%**	66.28	5.12E+00	58	**100%**	71.44	5.16E+00	63	**100%**	71.88	**4.48E+00**
x2	55	**100%**	63.16	5.69E+00	**51**	**100%**	**62.80**	9.51E+00	57	**100%**	64.36	**3.21E+00**	55	**100%**	63.76	3.39E+00
cc	**62**	**100%**	**77.20**	1.18E+01	74	**100%**	86.76	**6.27E+00**	77	**100%**	87.80	7.34E+00	75	**100%**	86.56	7.26E+00
cmb	-	0%	-	-	-	0%	-	-	**45**	**100%**	**57.64**	**4.69E+00**	46	**100%**	57.76	5.57E+00
cordic	-	0%	-	-	-	0%	-	-	-	-	-	-	190	**100%**	13156.16	8.44E+03
frg1	**77**	88%	**96.14**	8.90E+00	84	**100%**	96.20	**7.27E+00**	307	**100%**	387.40	4.43E+01	238	**100%**	354.32	5.34E+01
pm1	**52**	92%	60.04	4.35E+00	53	**100%**	**57.84**	**2.09E+00**	67	**100%**	82.00	5.29E+00	74	**100%**	84.52	5.22E+00
sct	86	84%	**90.90**	9.78E+00	**83**	72%	97.72	**6.93E+00**	94	**100%**	115.92	1.37E+01	92	**100%**	105.60	8.47E+00
t481	**42**	**100%**	**60.72**	**1.49E+01**	43	**100%**	63.60	2.40E+01	59	**100%**	92.16	2.44E+01	60	**100%**	91.71	2.06E+01
tcon	29	**100%**	44.04	7.17E+00	36	**100%**	**43.80**	6.31E+00	**25**	**100%**	46.04	5.37E+00	43	**100%**	46.60	**1.70E+00**
vda	-	0%	-	-	-	0%	-	-	-	-	-	-	7394	100%	8218.46	3.81E+02

Fonte: Elaborada pelo autor.

Em (8), apenas 24 execuções independentes dos os problemas alu4, cordic, t481 e vda foram utilizadas no algoritmo SAM-E, pois sofreram uma interrupção devido ao tempo de execução, superior ao de 72 horas imposto pelos autores. Exceto para t481, tais problemas não foram executados com a CGP-RL-E, devido a restrições na capacidade de processamento do *hardware* e do tempo disponível para a realização dos trabalhos.

Como pode ser visto na Tabela 5, a CGP-RL-R conseguiu obter as melhores soluções em seis dos oito problemas pequenos, cinco dos oito problemas médios e quatro dos nove problemas grandes, sendo o método que obteve as melhores soluções na maioria dos problemas. A CGP-RL-R também obteve melhores médias gerais: três melhores médias nos problemas pequenos, cinco nos médios e quatro nos grandes.

Os orçamentos dados em (8) são proporcionais ao tamanho e complexidade dos problemas. A Figura 28 mostra uma tendência de melhora no desempenho da CGP-RL-R sobre a SAM-R conforme o orçamento do problema cresce. Para avaliar essa melhora relativa no desempenho quando o número de chamadas da função objetivo aumenta, foi calculada a correlação entre $P_{RRL,R}$, definida pela Equação 7.1, que se trata da razão entre a média de transistores obtidas pelos algoritmos para um dado problema e o orçamento para cada problema. e foi obtido o valor de -0,46. Isso indica tendências de que o algoritmo proposto apresenta melhor desempenho à medida que o orçamento do problema aumenta e, consequentemente, sua dificuldade. Também é visto na Figura 28 que os problemas decod e frg1 são diferentes da correlação observada. O problema decod é um problema de orçamento relativamente baixo e onde a CGP-RL-R conseguiu um resultado significativamente melhor que a referência e frg1 é um problema com grande orçamento, mas o desempenho da CGP-RL-RL foi apenas ligeiramente melhor do que a referência. Retirando-os do cálculo da correção, passa-se para o valor de -0,75, o que pode ser considerado uma correlação forte, reforçando a indicação de que o algoritmo proposto é uma boa escolha para problemas grandes e complexos.

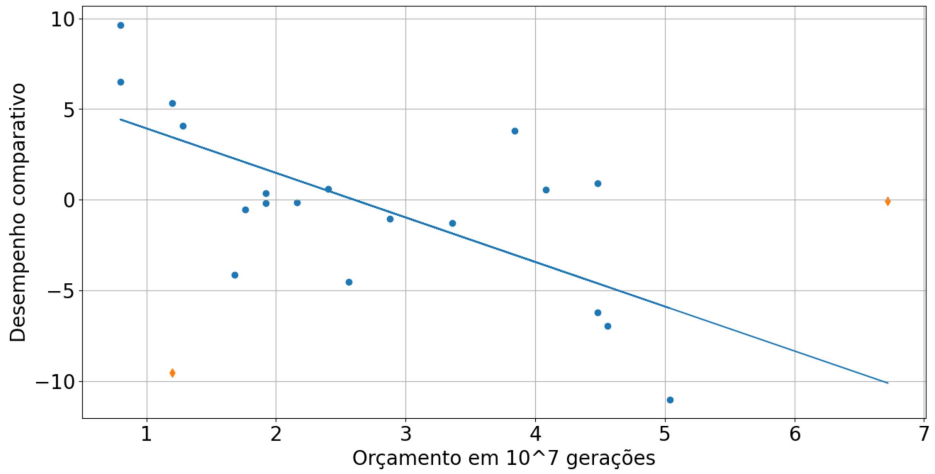
$$P_{RRL,R} = \frac{\overline{x}_{RRL}}{\overline{x}_R} - 1 \tag{7.1}$$

7.2.1 Comparação da CGP-RL com a CGP usando viés estático

Em (7), foram sugeridas matrizes de enviesamento estático para trocas de porta lógica na CGP. Como tanto, a CGP-RL quanto o método de (7) funcionam na da mutação de portas lógicas é natural que sejam comparados.

Os experimentos apresentados aqui foram modificados para facilitar a comparação dos resultados com os apresentados em (7). O foco da otimização passa a ser reduzir a quantidade de portas lógicas ao invés do número de transistores. O conjunto de tipo de portas passa a ser {wire, not, and, or, xor} ao invés de {not, and, nand, or, nor, xor, xnor}. A CGP-RL será comparada com a variante SAM+G+2P de (7), pois é a versão que é inicializada com população aleatória e tem fase de projeto e fase de otimização distintas,

Figura 28 – Índices de desempenho $P_{RRL,R}$ e orçamentos computacionais dos problemas aqui resolvidos. Os problemas `decode` e `frg1` são marcados como losangos laranja.



Fonte: Figura produzida pelo autor.

que é o que mais se aproxima do método proposto. A variante SAM+G+2P também foi apontada por (7) como a melhor método proposto no trabalho para a otimização de CLCs, baseado no número de portas lógicas.

Os problemas são os mesmos cinco apresentados em (7):

- **Problema 1:** De 4 entradas e 1 saída com os mintermos $F = \sum m(0, 1, 3, 6, 7, 8, 10, 13)$[3]
- **Problema 2:** De 4 entradas e 3 saídas com os mintermos $F_0 = \sum m(7, 10, 11, 13, 14, 15)$, $F_1 = \sum m(2, 3, 5, 6, 8, 9, 12, 15)$ e $F_2 = \sum m(1, 3, 4, 6, 9, 11, 12, 14)$
- **Problema 3:** Um multiplicador de 2 bits por 2 bits, contendo 4 entradas e 4 saídas.
- **Problema 4:** De 4 entradas e 2 saídas com os mintermos $F_0 = \sum m(0, 1, 2, 4, 5, 8)$ e $F_1 = \sum m(11, 14, 15)$
- **Problema 5:** Um comparador de 4 bits com 8 entradas e 3 saídas.

Tais problemas podem ser classificados como pequenos pelos critérios apresentados em (8), ou seja, são problemas 8 ou menos variáveis de entrada. Nos Problemas de 1 a 4, foram realizadas 30 execuções independentes e no Problema 5 apenas 10, conforme procedimento descrito em (7).

[3] $F_a = \sum m(b, c, d, e)$ é uma notação simplificada para descrever a tabela verdade da saída de um circuito. Significa que para a saída a, as linhas b, c, d, e da tabela verdade resultam em $a = 1$, enquanto nas demais linhas tem-se que $a = 0$

Como pode ser visto na Tabela 6, a CGP-RL teve melhores resultados nos Problemas 1, 2 e 5, com uma melhoria em cerca de 10% da média de portas obtidas. No Problema 5, a CGP-RL conseguiu taxa de sucesso de 100%, ou seja, conseguiu gerar um individuo factível em todas as execuções independentes, enquanto o método proposto em (7) teve taxa de sucesso de 80%.

Tabela 6 – Comparação dos resultados da CGP-RL com o melhor método proposto em (7). Melhor significa a menor quantidade de portas que o algoritmo conseguiu para o relativo problema. TS é a taxa de sucesso do algoritmo, Média é a média de portas que o algoritmo conseguiu e DP é o desvio padrão.

Problema	CGP-RL R				SAM+G+2P			
	Melhor	TS	Média	DP	Melhor	TS	Média	DP
Problema 1	**7**	**1.00**	**9.08**	**1.52**	**7**	**1.00**	10.33	1.77
Problema 2	**7**	**1.00**	**8.20**	**1.94**	**7**	**1.00**	9.13	2.01
Problema 3	8	**1.00**	9.16	**2.05**	**7**	**1.00**	**9.10**	2.54
Problema 4	8	**1.00**	9.00	1.30	**7**	**1.00**	**8.10**	**1.16**
Problema 5	**19**	**1.00**	**23.48**	3.71	22	0.80	26.17	**3.6**

Fonte: Elaborada pelo autor.

No Problema 3, os resultados entre a CGP-RL e o método de (7) foram próximos, com a média da CGP-RL sendo apenas 0,6% maior que a média obtida em (7). Os resultados obtidos pela CGP-RL estavam no mesmo intervalo dos resultados obtidos pela SAM+G+2P e o desvio padrão da CGP-RL foi menor.

Os resultados comparativos foram considerados favoráveis para a CGP-RL, que não só obteve melhores médias em mais da metade dos problemas, como teve melhor desempenho no problema considerado mais difícil em (7). Além disso como foi apontado em (33) a CGP-RL tem seu melhor desempenho em problemas maiores. Os problemas abordados em (7) são do tipo pequeno. Mesmo assim o método proposto conseguiu melhores resultados mesmo trabalhando num conjunto de problemas nos quais seu desempenho é relativamente pior.

7.3 ANÁLISES SOBRE NODE CGP-RL

A Node CGP-RL precisou passar por diversos ajustes de parâmetros. Para facilitar a avaliação de diferentes configurações, as seguintes métricas foram utilizadas: **Melhor**, **Mdiff**, **Nscore** e **Taxa de Sucesso**.

A métrica **Melhor** se refere à soma da quantidade de execuções independentes que o algoritmo conseguiu a melhor solução para cada problema. A Melhor solução, usada como referência é a melhor solução dentro do grupo de algoritmos e configurações sendo avaliadas e não a melhor solução conhecida na literatura. Quanto maior o valor dessa métrica, melhor o algoritmo foi em encontrar a melhor solução.

A métrica diff mede a diferença relativa do resultado de cada execução independente com a melhor solução encontrada para cada problema. **Mdiff** é a média da diff e quanto menor seu valor melhor, pois isso indica que o algoritmo está gerando soluções com valores mais próximos da melhor solução relativa ao grupo de algoritmos analisados.

A métrica *score* indica quantos algoritmos obtiveram uma média de transistores menor do que o algoritmo em questão, para um determinado problema. **Nscore** é a soma de todos os scores. Quanto menor o Nscore, melhor é o desempenho médio do algoritmo em questão em relação ao grupo avaliado.

A **Taxa de Sucesso** é a única métrica que é independente dos demais algoritmos do grupo. Ela indica o percentual de execuções independentes onde o algoritmo conseguiu chegar em uma solução factível e só é aplicada nos casos onde a Node CGP-RL, ou qualquer outro algoritmo que estiver sendo comparado, foi inicializado com uma população randomizada.

O código fonte da *Node* CGP-RL está disponível publicamente[4].

7.3.1 Estudo da variação do valor α

O parâmetro α, conforme indicado pela Equação 4.2, pondera a diferença entre a recompensa obtida pela política e a recompensa estimada para ela. Para analisar o impacto da variação de α para a Node CGP-RL, foram testados valores de α de 0,1 a 1,0, com incrementos de 0,1. Não foi considerado $\alpha = 0$ porque as informações mais recentes não são usadas neste caso e, portanto, o vetor de aprendizagem não é atualizado. Quando $\alpha = 1,0$, o vetor de aprendizagem sempre contém a última recompensa para a política dada, ignorando as recompensas anteriores. De acordo com (35), α não pode ser maior que 1,0.

Em geral, α =0,6 apresentou os melhores resultados, seguido por α = 1,0; 0,5 e 0,3, todos estes com resultados semelhantes, como pode ser visto na Tabela 7. Agrupando os problemas por tamanho, pode-se observar na Tabela 8 que $\alpha = 0,6$ apresenta melhores resultados nos problemas pequenos, $\alpha = 0,1$ para os problemas médios, $\alpha = 0,3$ e $\alpha = 1,0$ são os melhores para os problemas grandes. Mesmo nas categorias em que não se destaca, $\alpha = 0,6$ ainda aparece com bons resultados. Assim, $\alpha = 0,6$ foi escolhido para comparar o *Node* CGP-RL com outros algoritmos.

[4] https://github.com/FMoller/TCC/tree/master/cgprlnodes

Tabela 7 – Análise de performace para $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$.

α	*Melhor*	*Mdiff*	*Nscore*	*TS*
0.1	15	0.356	98	0.865
0.2	9	0.361	111	0.870
0.3	19	0.346	95	0.868
0.4	9	0.355	114	0.868
0.5	18	0.348	94	0.865
0.6	19	**0.336**	**72**	0.868
0.7	17	0.357	94	0.867
0.8	16	0.365	112	**0.878**
0.9	15	0.361	108	0.872
1.0	**25**	0.342	91	0.870

Fonte: Elaborada pelo autor.

Tabela 8 – Análise de performance para $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$ por tamanho de problema. *M* significa Melhor and *Ns* significa Nscore.

α	Pequenos				Médios				Grandes			
	M	*Mdiff*	*Ns*	*TS*	*M*	*Mdiff*	*Ns*	*TS*	*M*	*Mdiff*	*Ns*	*TS*
0.1	13	0.315	34	1.0	2	**0.367**	**27**	0.865	0	0.399	37	0.649
0.2	8	0.336	45	1.0	0	0.378	35	0.875	1	0.374	31	0.653
0.3	16	0.316	34	1.0	1	0.396	38	0.870	**2**	**0.330**	23	0.653
0.4	8	0.320	40	1.0	0	0.390	42	0.880	1	0.359	32	0.644
0.5	15	0.309	26	1.0	2	0.379	36	0.870	1	0.363	32	0.644
0.6	18	**0.287**	**16**	1.0	1	0.375	31	0.870	0	0.357	25	0.653
0.7	14	0.318	33	1.0	1	0.397	36	0.875	**2**	0.364	25	0.644
0.8	16	0.348	48	1.0	0	0.395	41	**0.890**	0	0.352	23	**0.662**
0.9	15	0.342	46	1.0	0	0.389	34	0.885	0	0.354	28	0.649
1.0	**20**	0.303	33	1.0	**3**	0.386	36	0.885	**2**	0.342	**22**	0.644

Fonte: Elaborada pelo autor.

7.3.2 Análise comparativa dos resultados da *Node* CGP-RL com a CGP e CGP-RL

A *Node* CGP-RL obteve melhores resultados que a CGP e a CGP-RL, como pode ser visto na Tabela 9. Também é possível observar que a *Node* CGP-RL é melhor ao lidar com o problema de forma não estacionária, já que a versão da CGP-RL que utilizou a estratégia estacionária (*Node* CGP-RL s) obteve piores valores em todas as métricas que a *Node* CGP-RL utilizando a estratégia não estacionária.

Tabela 9 – Comparação entre CGP com apenas SAM, CGP-RL (33), Node CGP-RL não estacionária (ns) com $\alpha = 0.6$ e Node CGP-RL estacionária (s).

alg	*Nbest*	*Mdiff*	*Nscore*	*SR*
SAM-R	14	0.34	29	0.86
CGP-RL	17	0.37	36	0.85
node CGP-RL ns	**24**	**0.31**	**13**	**0.87**
node CGP-RL s	14	0.39	50	0.86

Fonte: Elaborada pelo autor.

Fazendo uma comparação problema a problema, é possível observar que a *Node* CGP-RL obteve melhores resultados do que os encontrados pela CGP. Na Tabela 10, a *Node* CGP-RL obteve melhores resultados médios em 5 dos 8 problemas pequenos, 6 dos 8 problemas médios e 3 dos 7 problemas grandes, enquanto o algoritmo de controle, a CGP, obteve 1, 2 e 3 melhores médias para o respectivos grupos. A *Node* CGP-RL conseguiu, em pelo menos uma execução independente, atingir os melhores valores em 5, 4 e 5 problemas dos grupos pequeno, médio e grande, respectivamente. O algoritmo de controle conseguiu tais resultados em 4, 4 e 3 problemas para os mesmos grupos.

Esperava-se que o operador RL, enviesasse a mutação de forma a diminuir a violação das restrições durante a otimização e, assim, com indivíduos mais viáveis, maior a chance de obter melhorias durante a busca. No entanto, embora a redução na quantidade de transistores tenha sido observada, a hipótese levantada não se mostrou verdadeira. A *Node* CGP-RL reduziu, ao invés de aumentar, a ocorrência de indivíduos factíveis, como pode se ver na Tabela 4 da Seção 7.1.

Tabela 10 – Resultados obtidos pela Node CGP-RL e pela CGP usando apenas o operador SAM, inicializadas com população aleatória (R) e factível (E). Nos algoritmos das colunas R, os problemas `alu2` e `cmb` aparecem sem dados onde estes falharam em geral alguma solução factível ao final da fase de projeto. O problema `alu4` não foi executado para o algoritmo Node CGP-RL devido à restrições na disponibilidade de recursos computacionais.

Problema	Node-CGP-RL R				SAM-R				Node-CGP-RL E				SAM-E			
	Melhor	*TS*	Med	DP	*Melhor*	*TS*	Med	DP	*Melhor*	*TS*	Med	DP	*Melhor*	*TS*	Med	DP
C17	**9**	**1.00**	9.72	1.00E+00	**9**	**1.00**	9.80	**4.90E-01**	**9**	**1.00**	10.68	7.33E-01	**9**	**1.00**	**9.60**	8.00E-01
cm42a	30	**1.00**	34.40	2.38E+00	**29**	**1.00**	**33.48**	2.66E+00	31	**1.00**	35.72	**2.22E+00**	31	**1.00**	35.56	2.65E+00
cm82a	**19**	**1.00**	26.96	3.85E+00	20	**1.00**	27.36	6.67E+00	**19**	**1.00**	**21.72**	**3.01E+00**	**19**	**1.00**	22.40	3.36E+00
cm138a	**26**	**1.00**	**31.60**	3.19E+00	27	**1.00**	32.36	**2.49E+00**	30	**1.00**	36.16	2.59E+00	32	**1.00**	38.64	2.91E+00
decod	42	**1.00**	**51.32**	4.92E+00	41	**1.00**	53.12	5.52E+00	55	**1.00**	64.08	**4.51E+00**	56	**1.00**	67.80	8.31E+00
f51m	**67**	**1.00**	**86.92**	**7.94E+00**	74	**1.00**	89.60	1.22E+01	322	**1.00**	404.44	3.04E+01	344	**1.00**	412.32	2.49E+01
majority	**11**	**1.00**	**12.28**	**1.18E+00**	**11**	**1.00**	12.92	2.56E+00	**11**	**1.00**	14.52	2.28E+00	**11**	**1.00**	14.00	1.67E+00
z4ml	34	**1.00**	**45.24**	**8.65E+00**	35	**1.00**	48.08	8.86E+00	**33**	**1.00**	84.28	4.18E+01	**33**	**1.00**	65.32	2.56E+01
9symml	83	**1.00**	**115.28**	**2.25E+01**	**77**	**1.00**	128.84	3.95E+01	201	**1.00**	260.96	3.22E+01	197	**1.00**	262.92	5.03E+01
alu2	-	0.00	-	-	**306**	0.08	**327.50**	**2.15E+01**	516	**1.00**	578.20	3.57E+01	497	**1.00**	595.76	4.21E+01
alu4	**136**	**0.96**	**204.00**	4.45E+01	158	0.92	209.70	**3.40E+01**	-	-	-	-	8333	**0.96**	8685.25	2.33E+02
cm85a	**40**	**1.00**	**49.24**	**4.86E+00**	42	**1.00**	51.84	7.14E+00	39	**1.00**	81.16	2.75E+01	43	**1.00**	69.12	2.08E+01
cm151a	**35**	**1.00**	**45.28**	8.60E+00	36	**1.00**	46.00	8.11E+00	43	**1.00**	50.76	3.65E+00	45	**1.00**	51.36	**3.45E+00**
cm162a	**51**	**1.00**	**64.68**	6.75E+00	54	**1.00**	66.00	5.97E+00	61	**1.00**	65.48	4.16E+00	59	**1.00**	64.84	**3.59E+00**
cu	61	**1.00**	67.04	**4.19E+00**	**58**	**1.00**	**66.28**	5.12E+00	63	**1.00**	71.40	4.34E+00	63	**1.00**	71.88	4.48E+00
x2	52	**1.00**	**61.76**	5.87E+00	**51**	**1.00**	62.80	9.51E+00	58	**1.00**	62.72	**2.79E+00**	55	**1.00**	63.76	3.39E+00
cc	**70**	**1.00**	**84.12**	7.51E+00	74	**1.00**	86.76	**6.27E+00**	74	**1.00**	88.48	9.79E+00	75	**1.00**	86.56	7.26E+00
cmb	-	0.00	-	-	-	0.00	-	-	**45**	**1.00**	**55.16**	**5.44E+00**	46	**1.00**	57.76	5.57E+00
frg1	**82**	**1.00**	97.04	**6.17E+00**	84	**1.00**	**96.20**	7.27E+00	271	**1.00**	339.68	3.52E+01	238	0.96	354.42	5.45E+01
pm1	**53**	**1.00**	58.24	2.67E+00	**53**	**1.00**	**57.84**	**2.09E+00**	71	**1.00**	80.96	5.62E+00	74	**1.00**	84.52	5.22E+00
sct	88	0.88	**96.77**	**5.84E+00**	**83**	0.72	97.72	6.93E+00	96	**1.00**	110.28	9.42E+00	92	**1.00**	105.60	8.47E+00
t481	**41**	**1.00**	**57.84**	**1.28E+01**	43	**1.00**	63.60	2.40E+01	50	**1.00**	72.96	1.45E+01	60	0.96	91.71	2.06E+01
tcon	37	**1.00**	46.08	6.74E+00	**36**	**1.00**	**43.80**	6.31E+00	43	**1.00**	47.80	2.08E+00	43	**1.00**	46.60	**1.70E+00**

7.3.3 Análise do enviesamento da primeira Entrada

Os métodos propostos neste trabalho foram feitos à partir da implementação da CGP vista de (8). Em tal implementação, todos os nós possuem os genes de primeira entrada, segunda entrada e porta lógica. Quando a porta lógica é do tipo NOT, só a primeira entrada é considerada: o nó continua tendo o gene de segunda entrada mas esta informação não é mapeada para o fenótipo. Isso significa que, nesta implementação, uma mutação na segunda entrada tem menor chance de gerar um indivíduo fenotipicamente diferente de seu progenitor, como pode ser visto nos experimentos abordados na Seção 7.1.

Enquanto tal característica não impacta na CGP-RL, pois o sistema adaptativo desta só atua sobre as mutações em tipo de porta lógica, o mesmo não vale para a *Node* CGP-RL. A *Node* CGP-RL atua sobre os genes de um nó e usa um sistema de recompensa baseado no sucesso, ou fracasso, que a mutação de cada gene tem gerado. Portanto essa característica da porta NOT na implementação utilizada pode impactar negativamente o operador proposto.

Por este motivo, um ajuste foi feito na *Node* CGP-RL de forma que mutações em entradas de nós cujo a porta lógica é do tipo NOT não são consideradas para a atualização do vetor de recompensas estimadas. Isso diminui a frequência com o qual o vetor é atualizado, porém elimina o enviesamento que a porta NOT causa nas mutações nas entradas.

Para avaliar o impacto do enviesamento da porta NOT no operador,a *Node* CGP-RL ajustada foi comparada com a *Node* CGP-RL e com a CGP utilizando o operador SAM. Por limitações no tempo, dado o alto custo computacional dos experimentos, tal comparação envolveu apenas os experimentos inicializados com população aleatória. Além disso, foi utilizado $\alpha = 0.6$ para o operador ajustado. Para os problemas pequenos, foram executadas 25 execuções independentes. Para os problemas médios e grandes foram executadas apenas 13 execuções independentes. Os resultados dos experimentos podem ser vistos na Tabela 11.

É possível observar que de fato o enviesamento da porta NOT estava contribuindo negativamente para o desempenho da *Node* CGP-RL. Antes do ajuste, a *Node* CGP-RL obtinha pelo menos um individuo com a menor quantidade de transistores, quando comparada com a CGP, em 13 dos 23 problemas avaliados. Após os ajustes o método proposto passou a obter a melhor solução em 21 dos 23 problemas. Em relação às médias de transistores na solução final, antes dos ajustes a *Node* CGP-RL obtinha melhores médias em 15 dos 23 problemas e após os ajustes passou a obter melhores médias em 21 dos 23 problemas. A *Node* CGP-RL sem ajustes conseguiu melhores médias que sua versão ajustada em apenas 3 problemas.

Outro aspecto notável da *Node* CGP-RL ajustada é que a capacidade do algoritmo gerar uma solução factível aumentou. No problema alu2, a *Node* CGP-RL não conseguiu

gerar nenhum indivíduo factível nos testes realizados. Todavia, após os ajustes, conseguiu gerar um individuo factível em cerca de metade das execuções independentes, passando a CGP com SAM, que gerou um individuo factível em apenas 8% das execuções. Um aumento da taxa de sucesso foi verificado também no problema cmb: a *Node* CGP-RL chegou a uma solução factível em uma das 13 execuções independentes enquanto o algoritmo de referência não conseguiu gerar nenhum. Também foram verificadas melhorias na taxa de sucesso nos problemas alu4 e sct. Porém, essas melhorias são pequenas e podem ser apenas fruto da quantidade reduzida de execuções independentes na *Node* CGP-RL.

Houve significância estatística entre os resultados da *Node* CGP-RL ajustada e a CGP em 12 de 22 problemas: sct, pm1, C17, cm162a, z4ml, cc, decod, f51m, cm151a, cu, x2 e alu4. Os valores p foram obtidos pelo teste de Kruskal-Wallis, sendo o menor valor p de 1.27E-04 para o problema sct e 3.94E-02 para o alu4, sendo este o maior valor p considerado significante. O maior valor p obtido foi para o problema 8.66E-01 para o problema 9symml. Em todos os problemas onde há diferença estatística significativa, a *Node* CGP-RL ajustada obteve melhor média de transistores que a referência. Tais resultados indicam que o algoritmo proposto tem melhor desempenho para a minimização de CLCs que a CGP tradicional usando apenas o operador de mutação do tipo SAM.

Tabela 11 – Resultados obtidos pela *Node* CGP-RL com o ajuste para evitar o enviesamento da porta NOT, a *Node* CGP-RL sem o ajuste e a CGP usando a mutação SAM. Problemas onde os dados estão preenchidos com -"significam que o algoritmo não conseguiu gerar uma solução factível me nenhuma das execuções independentes.

Problema	*Node*-CGP-RL Ajustada				*Node*-CGP-RL				CGP			
	Melhor	*TS*	Med	DP	*Melhor*	*TS*	Med	DP	*Melhor*	*TS*	Med	DP
C17	**9**	**1**	**9.4**	1.02E+00	**9**	**1**	9.72	1.00E+00	**9**	**1**	9.8	**4.90E-01**
cm42a	**29**	**1**	**33.4**	2.39E+00	30	**1**	34.4	**2.38E+00**	**29**	**1**	33.48	2.66E+00
cm82a	**19**	**1**	27.96	4.99E+00	**19**	**1**	**26.96**	**3.85E+00**	20	**1**	27.36	6.67E+00
cm138a	27	**1**	**31.6**	2.76E+00	**26**	**1**	**31.6**	3.16E+00	27	**1**	32.36	**2.49E+00**
decod	**36**	**1**	**48.72**	6.24E+00	42	**1**	51.32	**4.92E+00**	41	**1**	53.12	5.52E+00
majority	**11**	**1**	12.48	2.52E+00	**11**	**1**	**12.28**	**1.18E+00**	**11**	**1**	12.92	2.56E+00
f51m	68	**1**	**82**	9.64E+00	**67**	**1**	86.92	**7.94E+00**	74	**1**	89.6	1.22E+01
z4ml	**32**	**1**	**42.2**	**5.94E+00**	34	**1**	45.24	8.65E+00	35	**1**	48.8	8.86E+00
9symml	88	**1**	124.46	2.73E+01	**83**	**1**	**115.28**	**2.25E+01**	77	**1**	128.84	3.95E+01
alu2	**238**	**0.53**	387.43	8.79E+01	N	0	N	N	306	0.08	**327.5**	**2.15E+01**
alu4	**129**	**1**	**196.53**	7.46E+01	136	0.96	204	4.45E+01	158	0.92	209.7	**3.40E+01**
cm85a	41	**1**	49.53	6.40E+00	**40**	**1**	**49.24**	**4.86E+00**	42	**1**	51.84	7.14E+00
cm151a	**30**	**1**	**40.61**	**5.21E+00**	35	**1**	45.28	8.60E+00	42	**1**	51.84	8.11E+00
cm162a	**51**	**1**	**60.15**	**4.64E+00**	**51**	**1**	64.68	6.75E+00	54	**1**	66	5.97E+00
cu	**58**	**1**	**62.61**	**3.39E+00**	61	**1**	67.04	4.19E+00	**58**	**1**	66.28	5.12E+00
x2	53	**1**	**57.38**	**2.90E+00**	52	**1**	61.76	5.87E+00	**51**	**1**	62.8	9.15E+00
cc	**69**	**1**	**80.92**	**6.17E+00**	70	**1**	84.12	7.51E+00	74	**1**	86.76	6.27E+00
cmb	**47**	**0.08**	**47**	-	-	0	-	-	-	0	-	-
frg1	**80**	**1**	**92.08**	8.18E+00	**82**	**1**	97	**6.16E+00**	84	**1**	96.2	7.27E+00
pm1	**49**	**1**	**53.92**	3.05E+00	53	**1**	58.24	2.67E+00	53	**1**	57.84	**2.09E+00**
sct	**83**	**1**	**87.38**	**3.15E+00**	88	0.88	96.77	5.84E+00	**83**	0.72	97.72	6.93E+00
t481	**40**	**1**	**52.69**	**1.11E+01**	41	**1**	57.84	1.28E+01	43	**1**	63.6	2.40E+01
tcon	**28**	**1**	**39.53**	**4.97E+00**	37	**1**	46.08	6.74E+00	36	**1**	43.8	6.32E+00

8 CONCLUSÃO

Com o objetivo de melhorar o desempenho da CGP na otimização de CLCs quanto à quantidade de transistores, foram propostos dois algoritmos de mutação adaptativos, CGP-RL e Node *CGP*-RL, ao longo dos estudos que culminaram nesta dissertação. Ambas técnicas propostas utilizam o paradigma de aprendizado por reforço e a estratégia *ϵ-greedy* para enviesar a mutação em um nó. Elas são utilizadas em conjunto com o operador de mutação SAM e não modificam a ordem de complexidade da CGP em relação ao tamanho da entrada.

A CGP-RL enviesa a mutação do tipo de porta lógica em um nó. Tal método, quando iniciado com uma população aleatória, obteve as melhores soluções em 15 dos 25 problemas adotados aqui e os melhores resultados médios em 12. Além disso, há uma correlação entre seu desempenho em relação à CGP usando apenas a SAM e o orçamento calculado para os problemas. Tal orçamento é influenciado pelo tamanho do problema (8) havendo portanto uma tendência da CGP-RL trabalhar melhor com problemas maiores. Quando iniciada com uma população factível, gerada pelo ESPRESSO, a CGP-RL teve um resultado geral pior do que a CGP.

Nas comparações com a SAM+G+2P, método de mutação do tipo de porta da CGP que utiliza uma matriz de enviesamento estática, a CGP-RL conseguiu melhor resultado em 3 dos 5 problemas, sendo que todos os problemas são categorizados aqui como pequenos, tipo de problema no qual a técnica proposta tem, relativamente, seu pior desempenho.

Os resultados da CGP-RL motivaram um estudo de como a mutação dos elementos de um nó impactam no processo evolutivo. Tal estudo culminou a proposta da técnica *Node* CGP-RL que enviesa a decisão de qual elemento de um nó sofrerá mutação.

A *Node* CGP-RL conseguiu chegar em melhores circuitos em 14 dos 23 problemas inicializados com população aleatória e em 16 dos 23 problemas no contexto de inicialização com população factível gerada pelo ESPRESSO. Em relação às médias, a *Node* CGP-RL, inicializada com população aleatória, teve resultado melhor ou igual à CGP em 16 problemas e, no contexto de população criada pelo ESPRESSO, em 14 problemas.

Nas comparações entre a CGP-RL e *Node* CGP-RL, esta última teve desempenho melhor que a primeira. No entanto não foi verificada nenhuma correlação entre o desempenho da *Node* CGP-RL e o tamanho do problema ou seu orçamento calculado, como foi observado na CGP-RL.

A Node CGP-RL se mostrou sensível à particularidades da implementação da CGP utilizada neste trabalho. A escolha das portas NOT utilizarem apenas a primeira entrada impactou seu resultado de forma negativa. Quando tal particularidade foi ignorada pela *Node* CGP-RL seus resultados melhoraram: o método proposto encontrou a melhor solução

em 21 dos 23 problemas e obteve a melhor média 15 dos 23, sendo que em 12 dessas melhoras houve diferença estatística significativa de acordo com o teste de Kruskal-Wallis. Além disso a *Node* CGP-RL demonstrou maior capacidade de gerar um individuo factível à partir de uma população inicial aleatória.

A adoção das técnicas não apenas impactam positivamente na otimização de CLCs como também indicam a existência de possíveis padrões nas escolhas do tipo de troca de porta lógica, ou elemento de nó a sofrer mutação. Essa característica, se devidamente estudada, pode levar a métodos mais elaborados e eficientes de mutação na CGP.

A *Node* CGP-RL e CGP-RL atuam em momentos diferentes da mutação de um nó e em tese podem atuar em conjunto. Porém o desempenho da CGP-RL e da *Node* CGP-RL em conjunto não foi investigado.

Em trabalhos futuros, pretende-se investigar o uso conjugado dos mesmos. Outra possibilidade de investigação é o uso de estratégias mais complexas de aprendizado por reforço nos algoritmos propostos.

REFERÊNCIAS

1 Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*, volume 2. Springer, 1 edition, 1987.

2 Xinye Cai, Stephen L. Smith, and Andy M. Tyrrell. Positional independence and recombination in cartesian genetic programming. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Genetic Programming*, pages 351–360, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

3 Janet Clegg, James Alfred Walker, and Julian Frances Miller. A new crossover technique for cartesian genetic programming. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, page 1580–1587, New York, NY, USA, 2007. Association for Computing Machinery.

4 Carlos A. Coello Coello, Arturo. Aguirre, and Bill. Buckles. Evolutionary multiobjective design of combinational logic circuits. In *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*, pages 161–170, 2000.

5 Carlos A. Coello Coello, Alan D. Christiansen, and Arturo Hernández Aguirre. Towards automated evolutionary design of combinational circuits. *Computers & Electrical Engineering*, 27(1):1–28, 2000.

6 José Eduardo da Silva and Helder Soares Bernardino. Cartesian genetic programming with crossover for designing combinational logic circuits. *Brazilian Conf. on Intelligent Systems (BRACIS)*, pages 145—150, 2018.

7 José Eduardo da Silva, Francisco Manfrini, Heder S. Bernardino, and Helio Barbosa. Biased mutation and tournament selection approaches for designing combinational logic circuits via cartesian genetic programming. In *Anais do Encontro Nacional de Inteligência Artificial e Computacional*, pages 835–846. SBC, 2018.

8 Lucas Augusto Müller de Souza, José Eduardo Henriques da Silva, Luciano Jerez Chaves, and Heder Soares Bernardino. A benchmark suite for designing combinational logic circuits via metaheuristics. *Applied Soft Computing*, 91:106246, 2020.

9 Agoston E. Eiben and James E. Smith. *Introduction to evolutionary computing.* Natural Computing Series. Springer, Berlin, Germany, 2 edition, July 2015.

10 Halit Eren. *Electronic portable instruments : design and applications.* CRC Press, Boca Raton, FL, 2003.

11 George Jerry Friedman. *Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy.* University of California, Los Angeles–Engineering, 1956.

12 Brian W. Goldman and William F. Punch. Reducing wasted evaluations in cartesian genetic programming. In Krzysztof Krawiec, Alberto Moraglio, Ting Hu, A. Şima Etaner-Uyar, and Bin Hu, editors, *Genetic Programming*, pages 61–72, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

13 C. Guiducci, Alexandre Schmid, Frank Gurkaynak, and Yusuf Leblebici. Novel front-end circuit architectures for integrated bio-electronic interfaces. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pages 1328–1333, 03 2008.

14 José Eduardo Henriques da Silva and Heder Soares Bernardino. A 3-step cartesian genetic programming for designing combinational logic circuits with multiplexers. In Paulo Moura Oliveira, Paulo Novais, and Luís Paulo Reis, editors, *Progress in Artificial Intelligence*, pages 762–774, Cham, 2019. Springer International Publishing.

15 David Hodan, Vojtech Mrazek, and Zdenek Vasicek. Semantically-oriented mutation operator in cartesian genetic programming for evolutionary circuit design. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 940–948, New York, NY, USA, 2020. Association for Computing Machinery.

16 Radek Hrbacek and Lukas Sekanina. Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, page 1015–1022, New York, NY, USA, 2014. Association for Computing Machinery.

17 Christopher Jankee, Sébastien Verel, Bilel Derbel, and Cyril Fonlupt. On the design of a master-worker adaptive algorithm selection framework. In Evelyne Lutton, Pierrick Legrand, Pierre Parrend, Nicolas Monmarché, and Marc Schoenauer, editors, *Artificial Evolution*, pages 1–15, Cham, 2018. Springer International Publishing.

18 Roman Kalkreuth, Günter Rudolph, and Andre Droschinsky. A new subgraph crossover for cartesian genetic programming. In *Genetic Programming*, 04 2017.

19 Saso. Karakatic, Vili. Podgorelec, and Marjan. Hericko. Optimization of combinational logic circuits with genetic programming. *Elektronika ir Elektrotechnika*, 19(7):86–89, Sep. 2013.

20 Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Trans. Amer. Inst. Electr. Eng. Part I: Commun. Electron*, 72(5):593–599, 1953.

21 Michael Katehakis and Veinott Jr. The multi- armed bandit problem: Decomposition and computation. *Math. Oper. Res.*, 12:262–268, 05 1987.

22 Richard Langlois. Computers and semiconductors. *Technological Innovation and Economic Performance*, 01 2002.

23 Sushil John Louis. *Genetic Algorithms as a Computational Tool for Design*. PhD thesis, Indiana University, USA, 1993. UMI Order No. GAX94-04378.

24 Jorge Maturana, Alvaro Fialho, Frederic Saubion, Marc Schoenauer, and Michele Sebag. Extreme compass and dynamic multi-armed bandits for adaptive operator selection. In *Congress on Evolutionary Computation (CEC)*, pages 365 – 372, 06 2009.

25 Jorge Maturana and Frédéric Saubion. A compass to guide genetic algorithms. In Günter Rudolph, Thomas Jansen, Nicola Beume, Simon Lucas, and Carlo Poloni, editors, *Parallel Problem Solving from Nature – PPSN X*, pages 256–265, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

26 Edward Joseph McCluskey. Minimization of boolean functions. *Bell Labs Tech. J.*, 35(6):1417–1444, 1956.

27 Julian Miller, P. Thomson, T. Fogarty, and I Ntroduction. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, 10 1999.

28 Julian F. Miller. *Cartesian Genetic Programming*. Natural Computing Series. Springer, Berlin, Heidelberg, 2011.

29 Julian F. Miller, Dominic Job, and Vesselin K. Vassilev. Principles in the evolutionary design of digital circuits—part i. *Genetic Programming and Evolvable Machines*, 1(1/2):7–35, 2000.

30 Julian.F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

31 Tom Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

32 Frederico José Dias Möller, Heder Soares Bernardino, Stênio Sã Rosário Furtado, and Lucas Augusto Müller de Souza. An adaptive mutation for cartesian genetic programming using an ϵ-greedy strategy (em revisão). In *Applied Inteligence*, pages –. Springer International Publishing, 2022.

33 Frederico José Dias Möller, Heder Soares Bernardino, Luciana Brugiolo Gonçalves, and Stênio Sã Rosário Furtado Soares. A reinforcement learning based adaptive mutation for cartesian genetic programming applied to the design of combinational logic circuits. In Ricardo Cerri and Ronaldo C. Prati, editors, *Intelligent Systems*, pages 18–32, Cham, 2020. Springer International Publishing.

34 S. Salivahanan. *Digital Circuits and Design*. Oxford University Press, 5 edition, 2018.

35 Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. A Bradford Book, 2 edition, 2018.

36 Moss Gregory Tocci Ronald, Widmer Neal. *Sistemas Digitais: Princípios e Aplicações*. Pearson, 12 edition, 2019.

37 Andrew Turner and Julian Miller. Neutral genetic drift: an investigation using cartesian genetic programming. *Genetic Programming and Evolvable Machines*, 16, 05 2015.

38 Chris Umans, Tiziano Villa, and Alberto Sangiovanni-Vincentelli. Complexity of two-level logic minimization. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*, 25(7):1230—-1246, 2006.

39 Frank Vahid. *Digital Design*. John Wiley & Sons, Nashville, TN, December 2004.

40 James Walker, Julian Miller, and Rachel Cavill. A multi-chromosome approach to standard and embedded cartesian genetic programming. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 903–910, 01 2006.

41 S. Yang. Logic synthesis and optimization benchmarks user guide: Version 3.0. Technical report, MCNC Technical Report, January 1991.