UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Naiara Sachetti

Curriculum Learning aplicado a um fluxo de otimização lógica de circuitos digitais baseado em Programação Genética Cartesiana

Florianópolis
2024

Naiara Sachetti

Curriculum Learning aplicado a um fluxo de otimização lógica de circuitos digitais baseado em Programação Genética Cartesiana

Relatório da disciplina Trabalho de Conclusão de Curso II de Ciências da Computação da Universidade Federal de Santa Catarina.
Orientador: Prof. Jônata Tyska Carvalho, Dr.
Coorientadora: Profa. Cristina Meinhardt, Dra.
Coorientador: Augusto André Souza Berndt, Dr.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Sachetti, Naiara
Curriculum Learning aplicado a um fluxo de otimização lógica de circuitos digitais baseado em Programação Genética Cartesiana / Naiara Sachetti ; orientador, Jônata Tyska Carvalho, coorientadora, Cristina Meinhardt, coorientador, Augusto André Souza Berndt, 2024.
49 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2024.

Inclui referências.

1. Ciências da Computação. 2. Aprendizado de máquina. 3. Programação Genética Cartesiana. 4. Electronic Design Automation. 5. Curriculum Learning. I. Carvalho, Jônata Tyska. II. Meinhardt, Cristina. III. Berndt, Augusto André Souza IV. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. V. Título.

Naiara Sachetti

***Curriculum Learning* aplicado a um fluxo de otimização lógica de circuitos digitais baseado em Programação Genética Cartesiana**

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharela em Ciências da Computação e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação do Departamento de Informática e Estatística (INE), Centro Tecnológico (CTC) da Universidade Federal de Santa Catarina (UFSC).

Florianópolis, 18 de dezembro de 2024.

Profa. Lúcia Helena Martins Pacheco, Dra.
Coordenadora do Curso

Banca Examinadora:

Prof. Jônata Tyska Carvalho, Dr.
Orientador
Universidade Federal de Santa Catarina

Profa. Cristina Meinhardt, Dra.
Coorientadora
Universidade Federal de Santa Catarina

Profa. Jerusa Marchi, Dra.
Universidade Federal de Santa Catarina

Prof. Elder Rizzon Santos, Dr.
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Dedico este trabalho à minha mãe, Nair, e meu pai, Pedro (*in memoriam*), que me ensinaram não só a dar os primeiros passos, mas o valor da educação e resiliência. Juntamente a eles, também agradeço aos meus irmãos, Andressa e Anderson, com quem divido grande parte das lembranças felizes da minha infância e vida adulta. O apoio e incentivo de cada um de vocês foi fundamental para que eu pudesse chegar até aqui.

Aos professores, Dr. Jônata Tyska Carvalho e Dra. Cristina Meinhardt e ao coorientador Dr. Augusto Berndt pela orientação neste trabalho e pela paciência em ensinar. Muitas vezes vocês acreditaram mais em mim que eu mesma, e isso foi essencial. Também agradeço aos colegas que passaram pelo grupo de pesquisa ao longo do desenvolvimento deste trabalho, Bryan e João, pela troca de ideias e colaboração em artigos e apresentações.

Por fim, aos meus amigos, que tornaram a vida mais leve e feliz durante este período, e aos colegas com quem tive a oportunidade de desenvolver trabalhos, trocar ideias e aprender ao longo da graduação.

RESUMO

A Programação Genética Cartesiana (CGP, do inglês *Cartesian Genetic Programming*) é uma técnica de computação evolutiva para a geração automática de programas inspirada na Programação Genética e na qual os indivíduos são representados como grafos dirigidos e acíclicos, com nodos computacionais dispostos em uma grade bidimensional. Entre os diversos contextos em que esta técnica já foi utilizada está a otimização lógica de circuitos. Esta aplicação, proposta no contexto de *Logic Learning*, pode ser adaptada para a Síntese Lógica Aproximada (ALS, do inglês *Approximate Logic Synthesis*). A ALS tem como principal objetivo a sintetização de circuitos aproximados, os quais produzem algumas saídas imprecisas ou incorretas em troca de ganhos em outras métricas de desempenho. Estes são geralmente explorados na busca de eficiência energética no projeto de circuitos para aplicações tolerantes a erro, relevantes para o cenário atual de tecnologia. Exemplos de aplicações tolerantes a erro são aquelas que envolvem redes neurais, processamento de sinais, visão computacional e processamento de linguagem natural, que estão tipicamente ligadas ao processamento de um alto volume de informações, levando também a um aumento a passos largos do consumo de energia de sistemas computacionais. Apesar de uma versão inicial de um fluxo de otimização lógica baseado em CGP para síntese de circuitos aproximados ter se mostrado efetivo para boa parte dos *benchmarks* em que foi avaliado, ainda restam casos em que a acurácia dos circuitos finais sintetizados não é satisfatória, motivando a aplicação de novos mecanismos potencialmente capazes de melhorar sua evolucionabilidade. É neste contexto que o presente trabalho se insere, propondo uma estratégia baseada em Curriculum Learning para a escolha de linhas da tabela verdade utilizadas para avaliação dos circuitos durante a fase de busca do fluxo, objetivando assim a melhoria da acurácia dos circuitos gerados por tal fluxo. Os resultados obtidos indicam que uma estratégia de escolha de linhas para compor os *mini-batches*, seja ela qual for, tem o poder de influenciar na capacidade de generalização dos circuitos gerados pelo fluxo e a configuração de linhas aprendidas tanto direta, como indiretamente. Em última instância, tais comportamentos indicam que uma estratégia de escolha de linhas afeta tanto a acurácia final dos circuitos gerados, como a velocidade com que a síntese é feita, mostrando que a técnica é pertinente para o contexto em que é proposta. Por outro lado, também observou-se que a abordagem idealizada neste trabalho apresenta uma contribuição limitada para o aumento da acurácia dos circuitos gerados, sendo que para a maioria dos *benchmarks* analisados não foram observadas melhorias estatisticamente significantes e foi alcançada uma média de 1,71% de acréscimo para os casos significantes. Ainda assim, identificou-se *benchmarks* e uma classes de problemas que se beneficiaram da abordagem de forma destacada, indicando que, mesmo não havendo garantia de ganho em acurácia, esta é uma alternativa cujo uso pode ser considerado e testado.

Palavras-chave: Aprendizado de máquina. Programação Genética Cartesiana. *Electronic Design Automation. Curriculum Learning*

ABSTRACT

Cartesian Genetic Programming (CGP) is an evolutionary computing technique to the automatic program generation inspired by Genetic Programming, and in which the individuals are represented as directed acyclic graphs, with computational nodes disposed in a bidimensional grid. Between the different contexts in which this technique has already been used is the logic optimization of circuits. This application, proposed in the context of Logic Learning, can be adapted to the Approximate Logic Synthesis, whose main goal is the synthesis of approximate circuits, that produce incorrect or imprecise outputs in exchange for better values for other performance metrics. They are usually explored in the search for energy efficiency in the circuit design of error-tolerant applications, relevant to the current technology scenario. Examples of error-tolerant applications are those involving neural networks, signal processing, computer vision and natural language processing, typically linked with the processing of a high volume if information, leading to a rapidly increasing in energy consumption of computer systems. Although an initial version of a CGP-based logic optimization flow has proven to be effective for the most of the benchmarks in which it was evaluated, there still remain cases in which the accuracy of the final synthesized circuits is not satisfactory, motivating the application of new mechanisms potentially capable if improving its evolvability. This is the context in which this work is inserted, proposing a strategy based on Curriculum Learning for the choosing of truth table lines used in the evaluation of circuits during the search phase of the flow, aiming the improvement of the accuracy of the circuits generated by the flow. The results obtained indicate that a strategy for choosing rows to compose mini-batches, whatever it is, has the power of influence the generalization capability of circuits generated by the flow and the configuration of lines directly and indirectly learned. Ultimately, this behavior indicate that a strategy for choosing rows affects both the final accuracy of the circuits generated and the speed with which the synthesis is carried out, making it evident that the technique is relevant to the context in which is proposed. On the other hand, it was observed that the proposed approach presents a limited contribution to increasing the accuracy of the generated circuits, since for most of the *benchmarks* analyzed no statiscally significant improvements were observed and an average increase of 1.71% was achieved for the significant ones. Nevertheless, benchmarks and problem classes that benefited from the approach were identified, indicating that, even without guaranteeing gains in accuracy, this is an alternative whose use can be considered and tested.

Keywords: Machine Learning. Cartesian Genetic Programming. Electronic Design Automation. Curriculum Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIG	Grafo de ANDs e Inversores (do inglês, *AND-Inverter Graph*)
ALS	Síntese Lógica Aproximada (do inglês, *Approximate Logic Synthesis*)
CGP	Programação Genética Cartesiana (do inglês, *Cartesian Genetic Programming*)
CL	*Curriculum Learning*
EDA	*Electronic Design Automation*
IWLS 2020	*2020 International Workshop on Logic and Synthesis*
LDH	Linguagem de Descrição de Hardware
LUT	Look-Up Table
NAS	Neural Architecture Search
PG	Programação Genética
RTL	Nível de Transferência de Registro (do inglês, *Register Transfer Level*)

SUMÁRIO

1 INTRODUÇÃO

A Programação Genética trata-se de uma técnica de Computação Evolucionária para a geração automática de programas. Sua história remonta pelo menos do ano de 1958, no qual Richard Friedberg descreveu um algoritmo que avaliava a qualidade de um programa sucessivas vezes, fazendo alterações aleatórias ao mesmo entre uma avaliação e outra (FRIEDBERG, 1958 apud MILLER, 2011). Contudo, foi somente mais tarde que o tema se popularizou, após a publicação do livro de John R. Koza sobre o assunto (KOZA, 1992; MILLER, 2011).

Esta técnica é a base da Programação Genética Cartesiana, descrita pela primeira vez em Miller, Thomson e Fogarty (1997), no contexto do desenvolvimento de um método para a evolução de circuitos digitais, e nomeada como tal em Miller (1999). Nesta, os indivíduos são representados como grafos dirigidos e acíclicos, de modo que seus nodos intermediários são representados na forma de uma grade bidimensional com uma quantidade fixa de linhas e colunas (MILLER, 2011).

A adoção de Programação Genética Cartesiana foi proposta como solução para o problema abordado na competição do *2020 International Workshop on Logic and Synthesis* (IWLS 2020), que consiste na utilização de estratégias de aprendizado de máquina para a síntese lógica de circuitos no contexto de *Logic Learning* (RAI *et al.*, 2021).

Em *Logic Learning*, o desafio é a sintetização de circuitos que generalizem bem o comportamento de uma função lógica, partindo apenas de uma especificação parcial da mesma. Esta descrição é dada por um conjunto de combinações de entradas e saídas conhecidas de uma tabela verdade, denominado *care-set* (RAI *et al.*, 2021; BERNDT; FOGAÇA; MEINHARDT, 2022).

É importante salientar que, neste caso, na porção da tabela verdade desconhecida não se assume que as saídas sejam *don't cares*, ou seja, que se possa atribuir qualquer valor a elas como geralmente é feito na otimização lógica tradicional, mas, neste caso, as saídas não especificadas caracterizam um conjunto de valores desconhecidos, chamados na literatura de *don't knows* (COSTAMAGNA; DE MICHELI, 2023). Algoritmos de síntese lógica tradicionais podem assumir o conjunto de saídas desconhecidas como *don't cares*, ao passo que soluções de Logic Learning tentarão inferir os valores prováveis para elas (RAI *et al.*, 2021).

A solução em questão baseada em Programação Genética Cartesiana proposta para a competição do IWLS 2020 se trata da descrita em Berndt, Abreu *et al.* (2022), no qual foi desenvolvido um fluxo de otimização lógica que utiliza a técnica. Neste fluxo, uma população inicial de circuitos, representados por Grafo de ANDs e Inversores (do inglês, *AND-Inverter Graph*) (AIG), é dada como entrada para uma fase de otimização, dividida em duas outras etapas. Na primeira, prioriza-se a seleção de circuitos com maior acurácia e, na seguinte, com menor tamanho, buscando assim uma otimização destas duas

características na síntese de circuitos integrados combinacionais.

Apesar de esta solução ter sido concebida como voltada a *Logic Learning*, existe a possibilidade de adaptá-la a um contexto semelhante, à Síntese Lógica Aproximada de circuitos. Neste tipo de síntese, se parte do pressuposto que a natureza da aplicação pretendida é tolerante a erros, ou seja, que ela permite que o circuito erre uma certa quantidade de saídas. Aproveitando-se disso, na Síntese Lógica Aproximada troca-se a corretude completa dos circuitos por ganhos em outras métricas, como uma menor área, um menor atraso ou uma maior eficiência energética, por exemplo (XU; MYTKOWICZ; KIM, 2016).

Exemplos de aplicações tolerantes a erros são aquelas que envolvem redes neurais, processamento de sinais, visão computacional e processamento de linguagem natural, elementos de grande relevância na atualidade (BARUA; MONDAL, 2019) que estão tipicamente ligados à necessidade de processamento de um alto volume de informações. Essa demanda crescente de processamento vêm ocasionando nos últimos anos um aumento a passos largos do consumo de energia de sistemas computacionais, fazendo com que as soluções de Síntese Lógica Aproximada ganhem cada vez mais espaço no campo da pesquisa (XU; MYTKOWICZ; KIM, 2016).

Apesar de o fluxo de otimização lógica baseado em Programação Genética Cartesiana ter se mostrado efetivo para boa parte dos *benchmarks* analisados, ainda puderam ser observados casos em que a acurácia dos circuitos gerados não foi satisfatória (BERNDT; ABREU *et al.*, 2022). Uma estratégia adotada na tentativa de melhorar o aprendizado para estas funções foi a adoção de treinamento explorando a estratégia de utilização de *mini-batches* (BERNDT; ABREU *et al.*, 2022). Cada um dos *benchmarks* fornecidos pela competição do IWLS 2020, consiste em três tabelas verdade, disponibilizadas por meio de arquivos no formato PLA. A tabela de treinamento, utilizada na determinação da acurácia dos circuitos durante a fase de otimização, pode ser utilizada de forma completa ou parcial neste processo. No contexto em questão, a estratégia de *mini-batches* consiste no segundo caso.

Os *mini-batches*, subconjuntos da tabela verdade, tipicamente possuem tamanho fixo, de modo que, de tempos em tempos, têm as suas linhas completamente trocadas por uma escolha aleatória na tabela verdade original. De forma alternativa, os autores de Lima *et al.* (2023) aplicaram uma abordagem de incremento de *mini-batch*, na qual se aumenta periodicamente o seu tamanho apenas adicionando novas linhas, nunca substituindo as que já o compõem. Esta abordagem foi efetiva principalmente para a redução do tempo de execução do fluxo, mas também teve impacto positivo na acurácia dos circuitos gerados para os *benchmarks* do IWLS 2020.

Ainda assim, existem funções cuja acurácia não teve uma melhoria considerável mesmo quando considerada a utilização das técnicas supracitadas. Portanto, o fluxo em questão poderia se beneficiar da aplicação de novos mecanismos potencialmente capazes de

melhorar sua evolucionabilidade, ou seja, a sua capacidade de continuar achando soluções melhores. Nesse sentido, uma direção possível de pesquisa é tentar procurar por técnicas capazes de melhorar a acurácia de soluções baseadas em aprendizado de máquina, e investigar como elas podem ser exploradas no fluxo para melhorar a acurácia ou reduzir o tempo necessário para aprender uma função. Seguindo esta direção, este trabalho investiga a técnica de *Curriculum Learning* (MILANO; PAGLIUCA; NOLFI, 2019) aplicada ao fluxo de síntese lógica baseado em Programação Genética Cartesiana.

Tal técnica já foi amplamente utilizada em outros contextos na literatura (BENGIO *et al.*, 2009; MATIISEN *et al.*, 2020; GRAVES *et al.*, 2017; MILANO; NOLFI, 2021) e espera-se que ela seja capaz de melhorar a evolucionabilidade do fluxo de síntese lógica baseado em Programação Genética Cartesiana, ou seja, a sua capacidade de continuar buscando soluções cada vez melhores.

1.1 OBJETIVOS

O presente trabalho tem como objetivo principal a avaliação do impacto da utilização de *Curriculum Learning* em um fluxo de otimização lógica baseado em Programação Genética Cartesiana.

Além disso, o desenvolvimento deste trabalho se desdobra nos seguintes objetivos específicos:

- Propor e implementar abordagem baseada em *Curriculum Learning* para a seleção de exemplos de treinamento de um fluxo de otimização lógica fundamentado em Programação Genética Cartesiana.
- Avaliar a abordagem proposta via comparação da acurácia dos circuitos finais gerados com as versões do fluxo com e sem a estratégia de *Curriculum Learning* idealizada.

1.2 ORGANIZAÇÃO DO TEXTO

O texto a seguir é iniciado com a exposição da fundamentação teórica para este trabalho, no Capítulo 2, passando pelos temas de Computação Evolucionária, Programação Genética, Programação Genética Cartesiana, projeto de circuitos integrados auxiliado por software, Síntese Lógica Aproximada, *Logic Learning* e *Curriculum Learning*. Já o Capítulo 3 traz de forma detalhada os trabalhos com os quais este tem uma relação direta. Em seguida, no Capítulo 4, apresenta-se a estratégia de Curriculum Learning idealizada. Por último, no Capítulo 5, são apresentados resultados da avaliação preliminar da aplicação de *Curriculum Learning* a um fluxo de otimização lógica baseado em Programação Genética Cartesiana e da avaliação da abordagem proposta neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções a seguir, conceitos importantes para o entendimento completo do trabalho desenvolvido são apresentados. Na Seção 2.1 são apresentados os conceitos de Computação Evolucionária e Programação Genética, criando base para o entendimento da Programação Genética Cartesiana, descrita na Seção 2.2. Então, na Seção 2.3 é apresentado um fluxo clássico de projeto de circuitos integrados auxiliado por software. Em seguida, dois conceitos próximos, Sintese Lógica Aproximada e Logic Learning, são elucidados nas seções 2.4 e 2.5, respectivamente. Finalmente, este capítulo se encerra com a Seção 2.6, que apresentação do conceito de *Curriculum Learning*.

2.1 PROGRAMAÇÃO GENÉTICA

A Programação Genética (PG) é um ramo da Computação Evolucionária, que abriga uma série de algoritmos de busca inspirados na teoria evolucionista da seleção natural cunhada por Darwin (1859), o qual tem como característica particular e fundamental o tipo dos objetos otimizados: soluções algorítmicas para problemas específicos (MILLER, 2011). Ela foi idealizada há pelo menos cinco décadas, quando Richard Friedberg descreveu um algoritmo que avaliava a qualidade de um programa sucessivas vezes, fazendo alterações aleatórias ao mesmo entre uma avaliação e outra (FRIEDBERG, 1958 apud MILLER, 2011).

Como é possível observar através do Algoritmo 1, um algoritmo genérico de Computação Evolucionária se inicia com a criação de um conjunto inicial de soluções candidatas. Tal conjunto é chamado de população, enquanto cada componente seu é denominado indivíduo. Em seguida, as soluções são avaliadas quanto à sua qualidade, resumida em um valor denominado *fitness*. Então, com base nesta métrica, indivíduos são selecionados para participar na criação das soluções candidatas da próxima população, sendo, portanto, os pais desta nova população. Os indivíduos do novo conjunto de soluções ainda podem sofrer mutação, passando por mudanças pontuais e aleatórias na sua descrição. Este processo repete-se por um número fixo de iterações g, chamadas gerações, ou até que algum indivíduo atinja um valor de *fitness* aceitável.

A forma com que o(s) pai(s) da próxima geração são escolhidos e que os indivíduos de uma nova população são gerados depende da tática escolhida para tal, sendo este um campo ativo de pesquisa em Computação Evolucionária. A variedade de possibilidades pode ser percebida através do trabalho de Hansen, Arnold e Auger (2015), no qual uma série de estratégias evolutivas são detalhadas.

Durante a história da PG, seu uso foi sugerido não somente para a geração automática de código (KOZA, 1992), mas também redes lógicas, redes neurais, máquinas de estados finitas (POLI, 1997) e circuitos (LOUIS, 1993) (KOZA *et al.*, 2005), por exemplo. Ainda, é importante notar que variadas abordagens foram desenvolvidas para a esta

Algoritmo 1: Algoritmo genérico de Computação Evolucionária. Adaptado de (MILLER, 2011).

Entrada: g
1 Gera a população inicial
2 $i \leftarrow 0$
3 **repita**
4 Calcula o valor de *fitness* para cada indivíduo da população
5 Seleciona um número de pais de acordo com a qualidade das soluções
6 Cria soluções filhas a partir dos pais
7 Muta pais e filhos (alguns ou todos)
8 Forma uma nova população a partir de pais e filhos
9 $i \leftarrow i + 1$
10 **até** $i = g$ ***OU*** *fitness é aceitável*;

técnica de Computação Evolucionária, destacando-se as baseadas em código de máquina (conhecida como Programação Genética Linear), pilha, gramáticas, e grafos (MILLER, 2011). Entre as últimas está o Programação Genética Cartesiana (do inglês, *Cartesian Genetic Programming*) (CGP).

2.2 PROGRAMAÇÃO GENÉTICA CARTESIANA

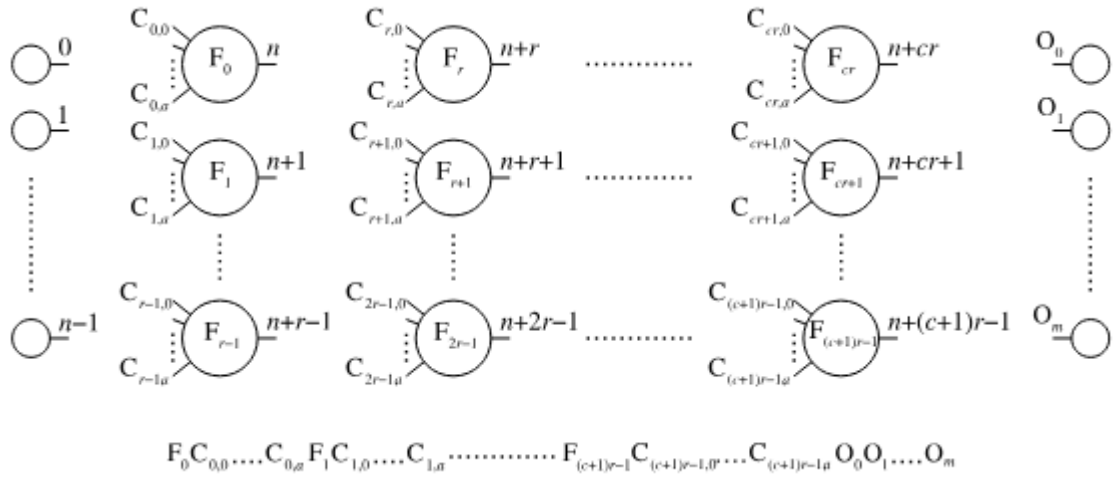
A técnica de CGP foi descrita pela primeira vez em Miller, Thomson e Fogarty (1997), no contexto do desenvolvimento de um método para a evolução de circuitos digitais, e nomeada de tal forma em Miller (1999).

O formato básico dos indivíduos, que inspira o nome da técnica, pode ser vislumbrado na Figura 1. Ele consiste num grafo acíclico de nodos, composto por uma coluna de n nodos de entrada, seguida por uma grade bidimensional de nodos computacionais com r linhas e c colunas, que lembra o formato de um plano cartesiano. Por último, como pode ser observado na extremidade direita da ilustração, ainda há uma coluna de m nodos de saída (MILLER, 2011). Cada um dos nodos de entrada é identificado de maneira única por um inteiro no intervalo $[0, n-1]$, enquanto os nodos intermediários são enumerados sequencialmente, coluna por coluna, com números no intervalo $[n, n + (c+1)(r-1)]$, conforme ilustrado na Figura 1 (MILLER, 2011).

O formato fixo dos indivíduos faz com que a técnica não sofra com *bloat*, conferindo uma vantagem importante ao CGP. O fenômeno em questão é comum em estratégias de PG e trata-se do crescimento do tamanho das soluções candidatas sem nenhum aumento em *fitness*. Quando este evento ocorre, tipicamente, programas com sub expressões ineficientes ou redundantes são gerados, ocasionando um aumento no tempo necessário para processá-los e, eventualmente, excedendo a capacidade de memória do computador utilizado para gerá-los (MILLER, 2011).

Ainda na mesma Figura 1, logo abaixo da representação geral em grafo das soluções, é possível observar o formato da *string* de números inteiros na qual estas são codificadas. Este é o genótipo de um indivíduo, enquanto seu fenótipo é o programa resultante da decodificação do genótipo (MILLER, 2011). Na *string* em questão estão descritos todos os nodos do grafo, sendo que a especificação de um nodo se inicia com um gene de função, número que identifica a função representada por ele, seguido dos genes de conexão, índices dos nodos cujas saídas são utilizadas como entrada para o nodo sendo especificado, e terminando com m índices de nodos de onde as saídas são coletadas. É importante notar que um nodo pode ter como entradas apenas saídas de nodos de colunas à sua esquerda (MILLER, 2011).

Figura 1 – Estrutura básica de uma solução em CGP



Fonte: Miller (2011).

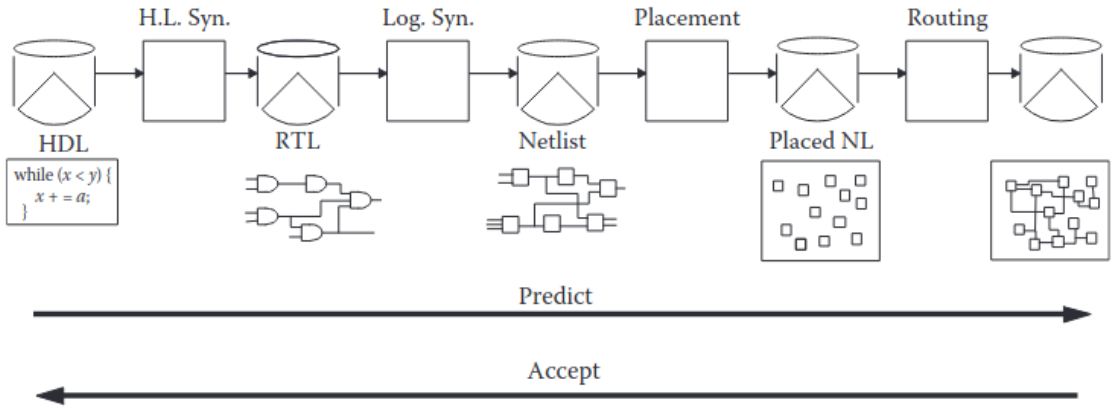
2.3 PROJETO DE CIRCUITOS INTEGRADOS AUXILIADO POR SOFTWARE

Projetos de circuitos integrados recentes tipicamente contém bilhões de portas lógicas, cada uma formada por múltiplos transistores, montadas em placas com diversas camadas de fios conectores (LAVAGNO *et al.*, 2016). Devido a esta complexidade, não é viável que tais projetos sejam feitos sem softwares que ofereçam suporte ao processo, chamados de ferramentas de EDA.

De fato, a necessidade de tais ferramentas foi identificada logo a partir da criação dos circuitos integrados. A sua história se inicia na década de 1960 (KAHNG *et al.*, 2011), praticamente junto à história destes, idealizados em 1952 por Geoffrey W. A. Dummer e construídos com sucesso e de forma independente em 1959, por Robert Noyce, e em 1960, por Jack Kilby, que dividem o reconhecimento de inventores do circuito integrado (WAZLAWICK, 2016).

Soluções de EDA focam, mais especificamente, em recursos para a modelagem colaborativa de circuitos e para a automatização de partes do processo de projeto de circuitos e da verificação da corretude dos mesmos, fazendo isso por meio de diversas etapas de síntese, verificação e teste (WANG; CHANG; CHENG, 2009). Estas podem ser distribuídas em um fluxo, que, segundo Lavagno *et al.* (2016), se modificou drasticamente ao longo das eras da EDA: era da invenção, implementação e da integração. A Figura 2 apresenta um fluxo sequencial de projeto de circuitos integrados auxiliado por ferramentas de EDA tipicamente implementado na era da implementação.

Figura 2 – Principais passos implementados por ferramentas de EDA.



Fonte: Lavagno *et al.* (2016).

Através dele é possível observar os principais passos implementados até os dias atuais. Tudo se inicia com uma descrição em alto nível do circuito, por meio de uma Linguagem de Descrição de Hardware (LDH). Esta passa por uma fase de síntese lógica, na qual é traduzida para uma descrição Nível de Transferência de Registro (do inglês, *Register Transfer Level*) (RTL), e então é traduzida para uma *netlist*, que consiste na coleção de todos os pinos e os componentes (transistores, resistores e capacitores, por exemplo) que eles conectam no design do circuito inteiro ou em uma subseção do mesmo (KAHNG *et al.*, 2011). Ainda nesta fase são feitas otimizações independentes de tecnologia, seguido do mapeamento dos componentes da *netlist* para elementos de uma tenologia específica e, finalmente, otimizações dependentes de tecnologia (LAVAGNO *et al.*, 2016).

Esta *netlist* é dada então como entrada para síntese física, a próxima grande fase do fluxo. Nesta, todos os componentes da *netlist* são instanciados nas posições exatas que ocuparão na área do chip (fase de posicionamento, ou *placement*) e as interconexões metálicas entre cada um deles são criadas na fase de roteamento (*routing*) (LAVAGNO *et al.*, 2016).

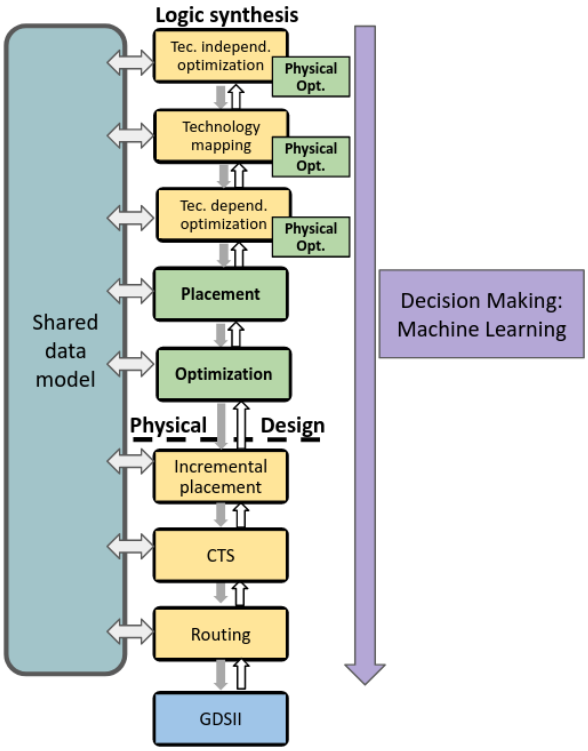
Ao final da execução destas fases, a descrição detalhada do circuito ainda é verificada e, caso esteja correta e respeite as restrições de projeto, aceita. O design final, com todos os

detalhes necessários para que circuito integrado seja fabricável, é tipicamente representado no formato *GDSII Stream* (LAVAGNO *et al.*, 2016) (KAHNG *et al.*, 2011).

Nos últimos 30 anos, tal fluxo sofreu alterações significativas. Inicialmente era possível a utilização do fluxo sequencial, como na Figura 2, entretanto, conforme os designs crescem em número de transistores e o tamanho destes componentes é reduzido, fica mais difícil de se achar funções de custo corretas para a previsão de métricas importantes para o projeto, como *delay* e área (LAVAGNO *et al.*, 2016). Cada vez mais os resultados de fases avançadas do fluxo da Figura 2 têm efeito em otimizações que podem ser feitas nas primeiras etapas da mesma. Por estes motivos e para que seja possível que o design final obedeça a restrições de tempo do projeto, as fases apresentadas estão cada vez mais integradas (LAVAGNO *et al.*, 2016).

A Figura 3 mostra um exemplo de fluxo de síntese atual, que contempla esse fenômeno. Através dela é possível notar também a existência de um modelo de dados compartilhado entre as fases e a introdução de abordagens de aprendizado de máquina para o apoio à tomada de decisão, auxiliando inclusive na previsão das métricas anteriormente mencionadas (BERNDT; FOGAÇA; MEINHARDT, 2022).

Figura 3 – Exemplo de fluxo de síntese atual, destacando a integração de ferramentas baseadas em aprendizado de máquina.



Fonte: Berndt, Fogaça e Meinhardt (2022).

Desta forma, um dos principais focos de fluxos modernos de EDA é evitar o reprocessamento de suas fases por conta de falhas na fase de verificação ocasionadas tipicamente por erros de predição (BERNDT; FOGAÇA; MEINHARDT, 2022). Além disso, outras

aplicações de aprendizado de máquina em EDA incluem o seu uso para a melhoria da exploração do espaço de design e para guiar a execução de tarefas realizadas durante a síntese de circuitos, como otimizações (ZHU *et al.*, 2020 apud BERNDT; FOGAÇA; MEINHARDT, 2022) (BERNDT; FOGAÇA; MEINHARDT, 2022).

2.4 SÍNTESE LÓGICA APROXIMADA

A área de Computação Aproximada abrange abordagens, métodos e algoritmos que geram resultados que podem conter "erros aceitáveis"(XU; MYTKOWICZ; KIM, 2016). Dada a existência de inúmeras aplicações resistentes a erros, tipicamente classificadas como tarefas de reconhecimento, mineração e síntese, o nascimento de tal paradigma de computação trouxe a possibilidade da troca da corretude absoluta em benefício da eficiência energética das soluções elaboradas (XU; MYTKOWICZ; KIM, 2016). Tal abordagem já foi explorada anteriormente em diferentes camadas de computação, desde o nível de software até aplicações em projeto de circuitos (XU; MYTKOWICZ; KIM, 2016), objeto de interesse deste trabalho.

Atualmente, desempenho, potência e área são as métricas de maior preocupação nos projetos de circuitos integrados (LAVAGNO *et al.*, 2016). Via de regra, para aumentar o desempenho dos circuitos é necessário haver um aumento no seu consumo de energia ou no seu tamanho, fazendo com que boa parte dos esforços de design sejam focados em atingir um bom equilíbrio entre estes três parâmetros (SCARABOTTOLO *et al.*, 2020). Neste contexto, a Computação Aproximada adiciona a dimensão da corretude das saídas ao espaço de design, permitindo assim que se obtenha ganhos em desempenho, área e potência em troca de perdas nessa nova métrica (SCARABOTTOLO *et al.*, 2020).

Uma possível abordagem para a redução do consumo de energia é o *overscaling*, que se trata da redução da tensão de alimentação sem reduzir a frequência de operação do circuito, o que ocasiona erros de *timing*, que, por sua vez, impactam as saídas dadas por tais componentes. Entretanto, os ganhos em eficiência energética neste modo de operação são pequenos, o que impulsiona o maior desenvolvimento da aproximação funcional de circuitos, que consiste numa mudança da função original do circuito em busca de menor consumo de energia e de área do *chip*, por exemplo (XU; MYTKOWICZ; KIM, 2016).

Uma forma de fazer isso é através da Síntese Lógica Aproximada (do inglês, *Approximate Logic Synthesis*) (ALS), que se traduz na síntese de circuitos que produzem saídas incorretas em troca de ganhos em outras métricas importantes, como área, delay e potência (XU; MYTKOWICZ; KIM, 2016) (SCARABOTTOLO *et al.*, 2020). Tais métodos têm como entrada a funcionalidade exata do circuito e um limite de aproximabilidade aceito (SCARABOTTOLO *et al.*, 2020).

O trabalho de Scarabottolo *et al.* (2020) identificou três estratégias de destaque em ALS: transformação de *netlist*, reescrita booleana, e síntese de alto nível aproximada. Na primeira delas, são feitas pequenas alterações a uma *netlist* existente, como a remoção

de alguns nós e a substituição de alguns fios por outros, reduzindo a área e o consumo de energia do circuito. Na segunda, o ponto de partida é a tabela verdade que descreve o circuito, sendo feitas alterações aos bits de saída para um subconjunto das entradas nela especificadas. Já na última, o alvo das alterações é a descrição em LDH do circuito.

2.5 LOGIC LEARNING

Problemas de *Logic Learning* envolvem a descoberta do comportamento geral de um circuito a partir de uma descrição incompleta do mesmo. Dada a natureza dos problemas, os circuitos dados como solução são aproximados (BERNDT; FOGAÇA; MEINHARDT, 2022).

Neste nicho, a descrição da parte do comportamento conhecido do circuito é dada por um conjunto combinações de entradas e saídas de uma tabela verdade, denominado *care-set*. A diferença de *Logic Learning* para o problema da otimização de funções lógicas especificadas de forma incompleta reside no fato de que as combinações de entradas e saídas desconhecidas, portanto fora do *care-set*, são tratadas como um conjunto de *don't knows*, em vez de *don't cares*. Neste último caso se pode assumir qualquer valor para as saídas desconhecidas, o que habilita que algoritmos de síntese lógica tradicionais efetuem uma série de otimizações com base nisso, porém na primeira situação isso não é possível, uma vez que existe um valor correto para as saídas do conjunto, apesar de elas serem desconhecidas. Dessa forma, algoritmos de *Logic Learning* devem tentar inferir o valor provável para os *don't knows* (BERNDT; FOGAÇA; MEINHARDT, 2022; COSTAMAGNA; DE MICHELI, 2023; RAI *et al.*, 2021).

Por conta da capacidade de generalização de técnicas de aprendizado de máquina, é frequente que elas sejam utilizadas para a busca de tais soluções (BERNDT; FOGAÇA; MEINHARDT, 2022). Incentivar a exploração da conexão entre estes dois campos foi justamente o objetivo da competição do IWLS 2020, de modo que a tarefa dos times participantes consistiu na geração de circuitos para 100 funções lógicas, dadas as suas tabelas verdade parcialmente especificadas como ponto de partida (RAI *et al.*, 2021).

Na literatura existem registros da utilização de diferentes técnicas de aprendizado de máquina em *Logic Learning*, como Árvores de Decisão (TENACE; CALIMERA, 2018; ABREU *et al.*, 2021; ZENG; DAVOODI; TOPALOGLU, 2021), Florestas Aleatórias (ZENG; DAVOODI; TOPALOGLU, 2021), redes de Look-Up Table (LUT) (CHATTERJEE, 2018; MIYASAKA *et al.*, 2021), Redes Neurais (MIYASAKA *et al.*, 2021) e CGP (BERNDT; ABREU *et al.*, 2021, 2022). Este trabalho foca na busca da melhoria desta última abordagem.

2.6 CURRICULUM LEARNING

De acordo com Bengio *et al.* (2009), que primeiro definiram o termo, um *curriculum* pode ser visto como uma sequência de critérios de treinamento, com cada critério associado a um conjunto diferente de pesos nos exemplos de treinamento. Nesse sistema, conforme se troca o critério ao qual um modelo é exposto, a quantidade e diversidade dos exemplos deve aumentar gradualmente, até que finalmente se possa utilizar o conjunto de treinamento completo. De acordo com tal definição, *Curriculum Leaning* é então uma técnica de manipulação do processo de treinamento de um modelo de aprendizado de máquina com base neste *curriculum*.

Entretanto, mais recentemente, Wang, Chen e Zhu (2022) analisaram que, conforme o campo se desenvolveu após a publicação original sobre o tema, este *curriculum* passou a ser idealizado de formas diversas e que não mais poderiam ser abarcadas por tal definição, o que os motivou a tecer uma acepção mais generalizada sobre o tema. Assim:

> ... um *curriculum* é uma sequência de critérios de treinamento sobre T passos. Cada critério Q_t inclui o *design* para todos os elementos de um treinamento de um modelo de aprendizado de máquina, por exemplo: dados/tarefas, capacidade do modelo, objetivo de aprendizado, etc. *Curriculum Learning* é a estratégia que treina um modelo com tal *curriculum*. [1] (WANG; CHEN; ZHU, 2022)

Esta definição é capaz de abarcar estratégias como a de Milano e Nolfi (2021), que inspira diretamente a abordagem adotada neste trabalho. Nela, o conceito é transportado para o treinamento evolucionário de agentes corporificados e estudado sobre dois problemas populares da área, conhecidos como *long double-pole balancing problem* e *bipedal walker hardcore problem*. A abordagem em questão utiliza como base a estratégia evolucionária de Salimans *et al.* (2017) e troca a seleção aleatória das condições ambientais iniciais sob as quais os agentes serão avaliados por uma baseada em CL.

Na mesma direção dos resultados observados por Milano e Nolfi (2021), Wang, Chen e Zhu (2022) notam que em diversos outros problemas onde CL foi aplicado pôde-se observar ganhos em desempenho nas tarefas para as quais os modelos foram treinados, além de aceleração do processo de treinamento. Com um escopo de aplicação vasto, a técnica vem sendo aplicada em tarefas de aprendizado supervisionado e aprendizado por reforço, além de aplicações como *graph learning* e Neural Architecture Search (NAS) (WANG; CHEN; ZHU, 2022). Tais fatos motivam a investigação da aplicação de CL ao contexto em que este trabalho está focado.

[1] No original: "... a curriculum is a sequence of training criteria over T training steps. Each criterion Qt includes the design for all the elements in training a machine learning model, e.g., data/tasks, model capacity, learning objective, etc. Curriculum learning is the strategy that trains a model with such a curriculum."

3 TRABALHOS CORRELATOS

Esta seção apresenta os trabalhos diretamente relacionados com o trabalho descrito neste relatório. A Seção 3.1 traz em detalhes o fluxo de otimização lógica baseado em CGP utilizado como fundamento para este trabalho. Já a Seção 3.2 traz uma abordagem alternativa para a melhoria da acurácia dos circuitos gerados e, especialmente, do tempo de execução do fluxo em questão. A Seção 3.3 expõe em detalhes a solução de CL utilizada como base para a elaboração da abordagem proposta neste trabalho. Finalmente, a Seção 3.4 traz em resumo as contribuições de cada uma das produções supracitadas, fazendo uma comparação com as contribuições deste trabalho.

3.1 FLUXO DE OTIMIZAÇÃO LÓGICA BASEADO EM CGP

No trabalho desenvolvido em (BERNDT; ABREU *et al.*, 2022), é apresentado um fluxo de otimização lógica baseado em CGP, sobre o qual o presente trabalho está fundamentado. Originalmente, a abordagem foi desenvolvida para atacar problemas de *Logic Learning*, mas é possível adaptá-la para a sintetização de circuitos aproximados por meio da utilização completa ou parcial da tabela verdade importantes para a descrição do comportamento esperado para o circuito, considerando que neste contexto tais linhas são conhecidas.

No fluxo de Berndt, Abreu *et al.* (2022), os nodos computacionais em questão representam portas lógicas. O que, na prática, permite que se utilize AIGs para a representação dos circuitos, uma estrutura de dados amplamente utilizada para otimizações independentes de tecnologia (RIENER *et al.*, 2019).

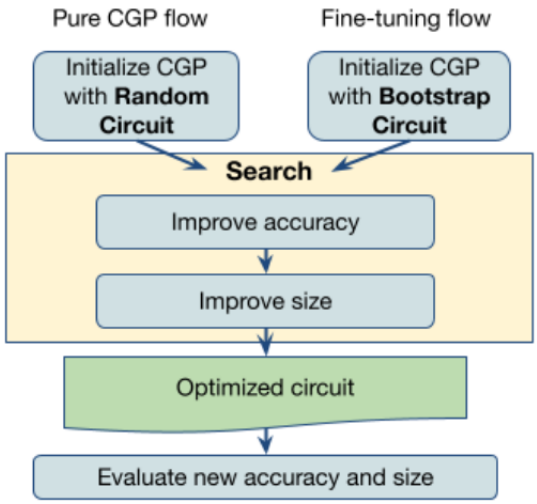
A Figura 4 mostra o funcionamento do fluxo citado. No modo puro de execução (*Pure CGP flow*, na figura), uma população destes circuitos é gerada de forma aleatória. Esta então é dada como entrada para uma fase de busca. Nesta, os circuitos são avaliados de acordo com uma métrica desejada, de forma que o indivíduo que tem maior *fitness*, ou seja, que tem o melhor valor para tal métrica, é então considerado o pai da próxima população de circuitos, formada por este pai e por circuitos gerados mediante mutações dele. A fase de busca é dividida em duas etapas. Na primeira, busca-se por circuitos com maior acurácia, avaliada com base em um conjunto de linhas da tabela verdade (o conjunto de treinamento do fluxo). Na segunda, prioriza-se a seleção de circuitos com menor tamanho. Assim, no fluxo exposto busca-se por circuitos que unam estas duas características (BERNDT; ABREU *et al.*, 2022).

Por fim, o circuito final sintetizado na etapa de busca é avaliado com relação a um conjunto de linhas da tabela verdade diferente do utilizado durante a fase anterior (BERNDT; ABREU *et al.*, 2022).

Há ainda a alternativa de utilizar o modo de *fine-tuning* do fluxo (*Fine-tuning* flow, na figura), o qual utiliza como ponto de partida uma população inicial gerada com base

em um circuito no formato AIG previamente sintetizado, podendo assim melhorar ainda mais circuitos anteriormente otimizados com diferentes técnicas (BERNDT; ABREU *et al.*, 2022).

Figura 4 – Fluxo de otimização lógica baseado em CGP desenvolvido por Berndt, Abreu *et al.* (2022).



Fonte: (BERNDT; ABREU *et al.*, 2022).

É importante notar ainda que o fluxo em questão possui pontos passíveis de melhoria. O principal deles é a ampla quantidade tempo necessária para rodar a fase de busca do algoritmo, devido ao tempo levado para a convergência e, acima de tudo, para a avaliação dos circuitos. Outro ponto de destaque é acurácia. Apesar de ter gerado circuitos com alta acurácia para boa parte dos *benchmarks* do IWLS 2020, restam casos em que as soluções geradas não foram satisfatórias neste quesito (BERNDT; ABREU *et al.*, 2022).

3.2 CGP COM TAMANHO ADAPTATIVO DE *BATCH*

Nesta estratégia, descrita em Lima *et al.* (2023), o *mini-batch* utilizado para a avaliação das soluções candidatas é incrementado com mais linhas da tabela verdade escolhidas de forma aleatória sempre que, após um certo período, não é mais observada uma melhoria significativa na acurácia dos circuitos gerados. Assim, o algoritmo em questão melhora a estimação do valor de *fitness* dos indivíduos ao longo do tempo.

Esta técnica teve impacto positivo no fluxo de otimização de Berndt, Abreu *et al.* (2022), sendo observados aumentos na acurácia dos circuitos de até 8,19% e uma diminuições na quantidade de avaliações de circuitos requeridas no processo de até 84,56%, afetando diretamente o tempo de execução do algoritmo. Além disso, para todos os circuitos gerados através do fluxo com a estratégia, não foram observadas reduções significativas em sua acurácia, quando comparadas com as soluções geradas utilizando a versão original do fluxo, que não utiliza *mini-batches*.

3.3 CL AUTOMATIZADO PARA AGENTES CORPORIFICADOS

A estratégia de Milano e Nolfi (2021), que inspira a abordagem proposta neste trabalho, foca no treinamento evolucionário de agentes corporificados. No contexto do trabalho em questão, tais agentes são robôs virtuais que, conforme as condições do ambiente em que se encontram, realizam ações em busca de algum objetivo específico.

Em sua abordagem, que utiliza como base a estratégia evolucionária de Salimans *et al.* (2017), os autores trocam a seleção aleatória das condições ambientais iniciais sob as quais os agentes serão avaliados por uma baseada em CL.

Para tal, as condições ambientais iniciais são escolhidas de forma aleatória no primeiro 1/10 do processo evolucionário. Após isso, é selecionada uma condição de cada um dos η subconjuntos definidos da seguinte forma: o desempenho obtido nas 5 últimas avaliações que uma condição ambiental foi utilizada é normalizado e cada condição é destinada a um conjunto diferente, de acordo com tal valor normalizado. O conjunto ao qual elas devem ser destinadas é definido através η intervalos de desempenho, calculados com base em uma função de dificuldade. Caso um subconjunto estiver vazio, as condições ambientais são escolhidas de um dos subconjuntos vizinhos não vazios mais próximos (MILANO; NOLFI, 2021).

Os dados analisados indicam que a abordagem propiciou ganhos em desempenho nas tarefas para as quais os modelos foram treinados, além de aceleração do processo de treinamento (MILANO; NOLFI, 2021).

3.4 COMPARAÇÃO COM TRABALHOS CORRELATOS

A Tabela 1 mostra uma comparação entre as contribuições dos trabalhos correlatos com as deste trabalho, em termos de aplicação em síntese lógica, utilização de CL e *mini-batches* e disponibilidade do código desenvolvido.

Tabela 1 – Comparação da abordagem proposta com as de trabalhos correlatos.

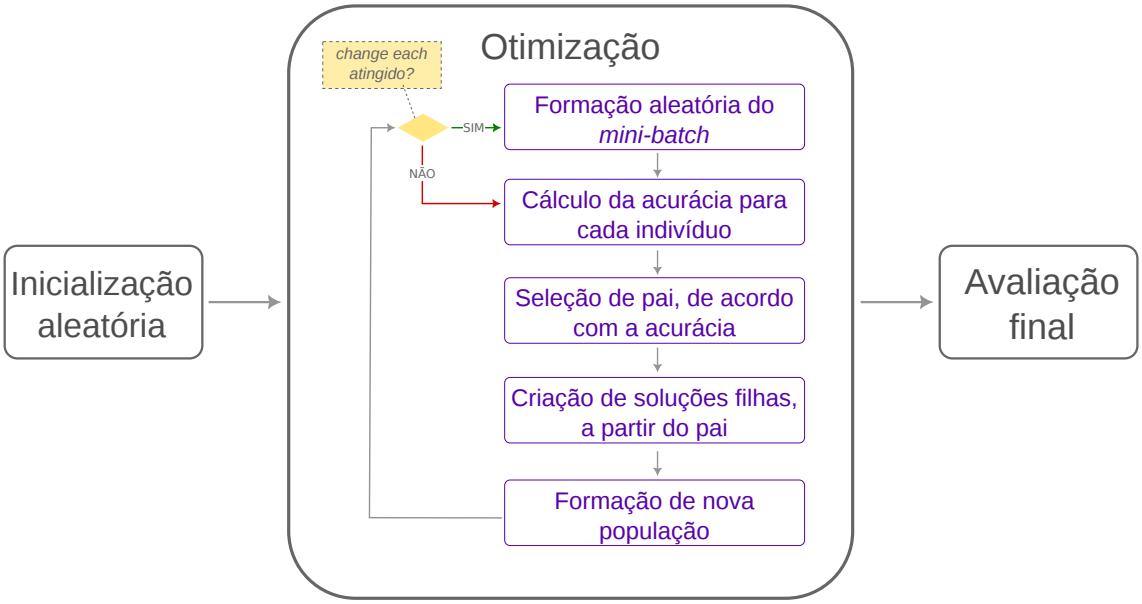
Abordagem	Aplicação em síntese lógica	Utiliza CL	Utiliza *mini-batches*	Código disponível
Berndt, Abreu *et al.* (2022)	X			X
Lima *et al.* (2023)	X		X	X
Milano e Nolfi (2021)		X		
Este trabalho	X	X	X	X

Fonte: Elaboração própria.

4 METODOLOGIA

Para o desenvolvimento deste trabalho, utilizou-se como base uma parte do fluxo de otimização lógica descrito em (BERNDT; ABREU *et al.*, 2022), implementado na linguagem de programação C++. A implementação da abordagem proposta, descrita nesta seção, foi realizada sobre o mesmo código, portanto na mesma linguagem. A Figura 5 mostra o funcionamento da versão do fluxo sem CL, utilizada como base de comparação na análise do efeito de CL sobre a acurácia dos circuitos gerados.

Figura 5 – Fluxo de otimização lógica baseado em CGP original.



Fonte: Elaboração própria.

Primeiramente, uma população de circuitos é gerada, na fase de "Inicialização aleatória", ou seja, é criado um conjunto aleatório de AIGs. Em seguida, tal população é dada como entrada à etapa de otimização, a qual consiste em um *loop* executado por um número fixo de iterações (também chamadas de gerações).

Em uma geração, a acurácia de cada circuito da população é avaliada com relação a linhas de uma tabela verdade, o que significa que entradas desta tabela são apresentadas aos circuitos, de forma que a saídas dadas por eles são comparadas com as saídas esperadas para tais entradas, conforme definido na tabela verdade. Quanto maior o número de saídas corretas, maior a acurácia do circuito. Na versão do fluxo apresentada na Figura 5, a acurácia dos circuitos é avaliada com relação a um *mini-batch*. Dessa forma, a acurácia dos circuitos ao longo da otimização é sempre estimada, uma vez que não são utilizadas todas as linhas da tabela verdade disponível para esta avaliação.

Cabe destacar que a definição de *mini-batch* para o contexto deste trabalho assemelha-se àquela utilizada no contexto de outros algoritmos empregados em aprendizado de máquina, como o Gradiente Descendente, por exemplo (GÉRON, 2019), apesar

de a técnica de otimização explorada ser diferente. Neste trabalho, o termo *mini-batch* designa estritamente um subconjunto de linhas da tabela verdade utilizada para a avaliação da acurácia dos circuitos durante a fase de otimização do fluxo.

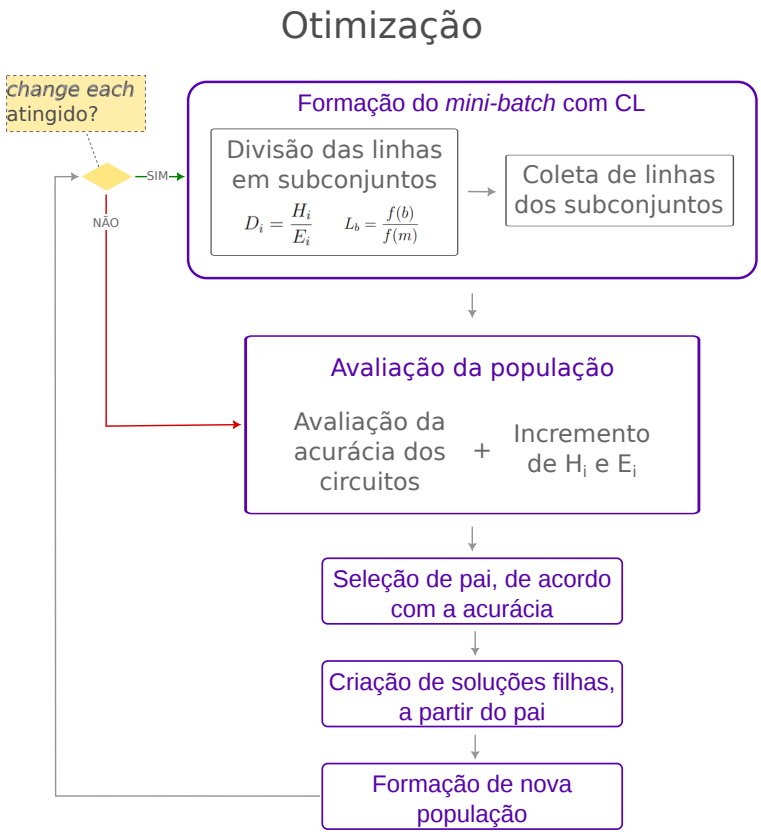
Esporadicamente, ocorre a formação de novos *mini-batches*, através da escolha aleatória de linhas do conjunto de treinamento. A frequência de execução desta operação é controlada através do hiperparâmetro *change each*.

Com base no valor de acurácia medido, o indivíduo com melhor desempenho sobre as linhas do *mini-batch* é escolhido como o pai da próxima população, a ser avaliada na iteração seguinte do *loop*. Tal população é formada por este pai e circuitos gerados através de mutações do mesmo.

Finalmente, é realizada uma avaliação final do circuito gerado pelo fluxo, utilizando uma tabela verdade diferente daquela usada durante a etapa de otimização. A otimização do tamanho dos circuitos gerados, capaz de ser realizada no fluxo original, não é utilizada.

A abordagem proposta neste trabalho consiste na substituição da seleção aleatória de linhas da tabela verdade para a composição dos *mini-batches* por uma baseada em CL. Inspirando-se no trabalho desenvolvido em (MILANO; NOLFI, 2021), descrito na Seção 2.6, as linhas são selecionadas utilizando um valor de dificuldade associado a cada linha da tabela verdade e uma função de dificuldade, conforme sumarizado na Figura 6.

Figura 6 – Fase de otimização do fluxo de otimização lógica baseado em CGP com CL.



Fonte: Elaboração própria.

A Equação 1 define a dificuldade D_i de uma linha i, considerando H_i o número de vezes que a saída esperada para uma linha i foi a mesma saída dada por um circuito em avaliação, e E_i como o número total de vezes que uma linha i participou de avaliações de circuitos. Assim, linhas com valores de D próximos de 1 são consideradas fáceis, enquanto aquelas com D próximo de 0 são as mais difíceis.

É importante notar que E_i não é o número de vezes que uma linha simplesmente foi selecionada para fazer parte de um *mini-batch*, uma vez que um mesmo subconjunto pode ser utilizado por mais de uma geração e, em uma mesma geração, é utilizado para avaliar mais de um circuito.

$$D_i = \frac{H_i}{E_i} \tag{1}$$

É necessário destacar também que as quantidades de acertos e de avaliações são acumulados ao longo de todas as gerações da evolução. Outra abordagem possível é a consideração apenas de tais valores obtidos a partir da avaliação da última população de circuitos gerada antes da troca de *mini-batch*. Optou-se pela primeira abordagem porque ela permite a captura do caráter temporal da dificuldade de uma linha, evitando situações em que linhas acertadas com frequência por uma população específica sejam consideradas fáceis, mesmo com um histórico de dificuldade para outras populações que demostra o contrário.

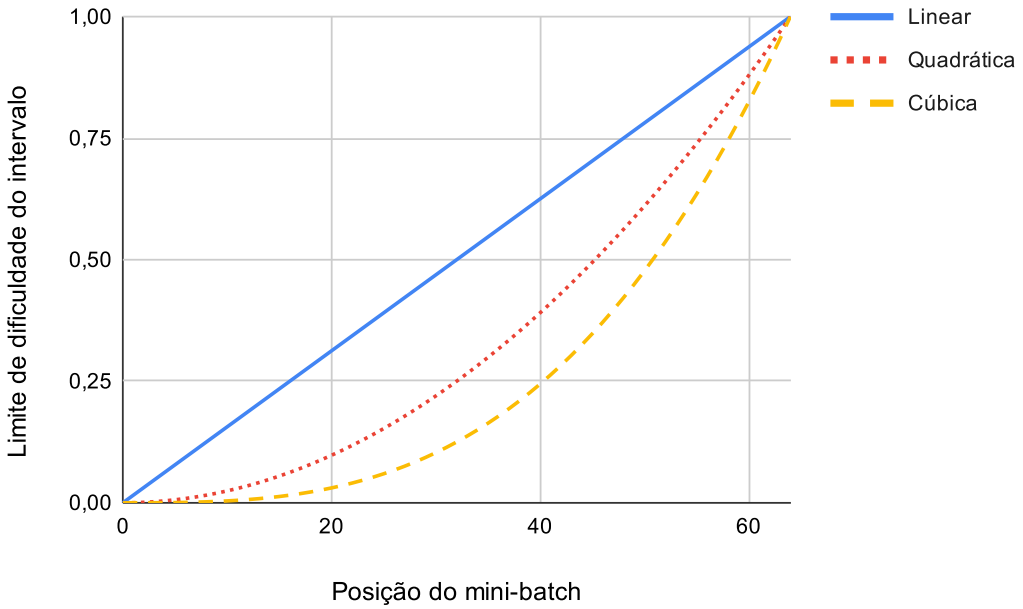
A função de dificuldade, por sua vez, define o intervalo de dificuldade que uma linha da tabela verdade deve possuir para ser posicionada em determinada posição de um *mini-batch*. O limite superior de dificuldade para cada índice do *mini-batch* é definido pela Equação 2, com b sendo índice da posição, m o valor de *batch size* e f a função de dificuldade. Considerando tal equação, o valor de dificuldade que uma linha da tabela verdade deve idealmente possuir para ser selecionada para uma posição i do *mini-batch* está entre $[0, L_1)$ para o primeiro índice, $[L_b - 1, L_b)$ para $1 < b < m$, e $[L_{m-1}, L_m]$ para o último índice.

$$L_b = \frac{f(b)}{f(m)} \tag{2}$$

A Figura 7 apresenta os valores de L para um *mini-batch* com 64 posições e as funções de dificuldade linear (em azul sólido), quadrática (em vermelho pontilhado) e cúbica (em amarelo tracejado). A partir dela é possível observar que a função linear guia a formação de *mini-batches* com as dificuldades das linhas que o compõem distribuídas de forma uniforme entre suas posições, enquanto as funções quadrática e cúbica favorecem a constituição de *mini-batches* difíceis.

Com estes elementos, têm-se um sistema no qual a dificuldade para cada linha é atualizada a cada geração, através do incremento de H_i e E_i conforme o que é observado em tempo de execução, e, a cada formação de *mini-batch*, as linhas que possuem o valor de

Figura 7 – Limites de dificuldade para as funções de dificuldade linear (em azul sólido), quadrática (em vermelho pontilhado) e cúbica (em amarelo tracejado).



Fonte: Elaboração própria.

dificuldade esperado para cada posição do arranjo (isto é, que se encaixam nos intervalos definidos pela Equação 2) são divididas em diferentes subconjuntos e então selecionadas para compor tal *mini-batch*. O exemplo a seguir ilustra o funcionamento da formação de um *mini-batch* utilizando a abordagem idealizada neste trabalho.

Considerando um *mini-batch* de quatro posições e uma função de dificuldade quadrática, têm-se os limites superiores de dificuldade para cada um dos índices do *mini-batch*, calculados com base na Equação 2, apresentados na Tabela 2.

Tabela 2 – Limites de dificuldade para um *mini-batch* de quatro posições, com uma função de dificuldade quadrática.

Índice do *mini-batch* (k)	L_k
4	1
3	0,5625
2	0,25
1	0,0625

Fonte: Elaboração própria.

Com base nestes valores e nos valores de dificuldade para cada uma das linhas de uma tabela verdade, calculados conforme a Equação 1, pode-se dividir estas linhas em diferentes intervalos de dificuldade. A Tabela 3 traz um exemplo de tabela verdade, com suas entradas e saídas, além de seus respectivos valores de dificuldade e o índice do *mini-batch* em que poderiam ser inseridas (ou seja, o intervalo de dificuldade em que poderiam ser posicionadas). Assim, como $D_2 = 0,01$, e, portanto, $0 \leq D_2 < 0,0625$, a

linha 2 é classificada no intervalo 1, por exemplo.

Tabela 3 – Exemplo de tabela verdade a ser utilizada na fase de otimização do fluxo com CL.

Índice (i)	Entrada	Saída esperada	D_i	Intervalo de dificuldade
1	0010000111101111	0	0,875	4
2	1001101001100010	0	0,01	1
3	0010101010011110	1	0,049	1
4	1101011010100111	1	0,9	4
5	0000101000111110	1	0,968	4
6	0100011110011010	1	0,620	4
7	1000000010010010	0	0,016	1

Fonte: Elaboração própria.

Em posse destas informações, pode-se então realizar a seleção de linhas que irão compor o *mini-batch*. Para o primeiro índice do subconjunto, é selecionada, de forma aleatória, uma linha cujo intervalo de dificuldade seja 1, como a linha 7. Já para a segunda posição, seleciona-se aleatoriamente uma linha do intervalo não vazio mais próximo (1, neste caso), uma vez que não existe nenhuma linha com intervalo igual a 2. Para as próximas posições, segue-se as mesmas regras, sendo selecionadas duas linhas do intervalo 4. Seguindo estes critérios, um possível *mini-batch* formado para a situação deste exemplo é ilustrado na Tabela 4.

Tabela 4 – Exemplo *mini-batch* de quatro posições formado com CL, utilizando uma função de dificuldade quadrática.

Índice (i)	Entrada	Saída esperada	D_i	Intervalo de dificuldade
1	0010000111101111	0	0,875	4
2	1001101001100010	0	0,01	1
4	1101011010100111	1	0,9	4
7	1000000010010010	0	0,016	1

Fonte: Elaboração própria.

É importante ressaltar ainda que podem ocorrer situações em que existem dois intervalos não vazios igualmente próximos para se coletar uma linha. Caso isso aconteça, é escolhida uma linha do intervalo com linhas mais difíceis, isto é, com valores de D menores.

Com a abordagem descrita neste capítulo, espera-se que a seleção de linhas difíceis aprimore a cobertura da tabela verdade, garantindo que os circuitos tenham acesso a linhas que ainda precisam ser aprendidas, enquanto a seleção de linhas fáceis previna o esquecimento daquelas que já foram aprendidas, melhorando assim a capacidade de generalização do fluxo de otimização lógica em questão.

5 RESULTADOS E DISCUSSÃO

Nesta seção é apresentada a implementação da abordagem proposta, bem como os protocolos experimentais elaborados para este trabalho e os resultados obtidos a partir de sua execução. Primeiramente, na Seção 5.1 são descritas as principais alterações feitas sobre o código do fluxo baseado em CGP original, com o intuito de guiar o leitor em um primeiro contato com a implementação realizada neste trabalho. Na Seção 5.2 é apresentada a fase de avaliação preliminar da aplicação de CL ao fluxo original. Esta foi realizada com o intuito de verificação da corretude do código desenvolvido e de fundamentação para a realização dos experimentos descritos e analisados na Seção 5.3, que versa sobre a avaliação da abordagem proposta.

5.1 IMPLEMENTAÇÃO

Em Sachetti e Berndt (2024) são apresentadas em detalhe as alterações feitas sobre o código original ao longo do desenvolvimento do trabalho. A imagem Docker utilizada para rodar os experimentos analisados neste documento pode ser acessada em Sachetti (2024).

As alterações centrais realizadas concentram-se nos arquivos *binaryPLA.cpp*, *mainPLA.cpp*, *AigPopulation.cpp* e *AIGraph.cpp*. Em *binaryPLA.cpp*, destaca-se a criação da função para a formação de *mini-batches* com CL, *setCLBatch*, substituta de *setRandomBatch*, e a implementação de *setSequentialBatch*, utilizada para a realização da avaliação inicial, tratada na Seção 5.2.

mainPLA.cpp contém o fluxo principal de otimização lógica. Alterou-se este arquivo e *AigPopulation.cpp*, onde ocorre a geração de soluções filhas (na função *generateOffspring*), para que a formação de *mini-batches* sequenciais e com CL fosse realizada quando necessário, de acordo com fluxo alterado descrito no Capítulo 4.

Ainda, foram realizadas alterações em *binaryPLA.cpp* e *AIGraph.cpp* para permitir a contagem de E_i e H_i para cada linha i da tabela verdade utilizada para a avaliação dos circuitos, conforme elucidado no capítulo anterior. Demais alterações referem-se a adaptações necessárias para a coleta de dados durante a sintetização dos circuitos.

5.2 AVALIAÇÃO INICIAL DE CL APLICADO AO FLUXO BASEADO EM CGP

Para a realização dos experimentos iniciais, foram selecionados os *benchmarks* com entradas pequenas (com no máximo 20 bits de entrada) do conjunto proposto para a competição do IWLS 2020, listados na Tabela 5. Tais *benchmarks* são formados por três tabelas verdade com um único bit de saída, dadas como conjuntos de treino, validação e teste. Assim, as linhas que compõem o *care-set*, fornecido à fase de otimização, são as dos conjuntos de treinamento, que possuem 6400 linhas.

O critério de escolha foi adotado visando maior rapidez na execução dos experimentos, visto que estes foram utilizados somente para se ter uma visão geral sobre o possível impacto da adoção de CL no contexto em questão, demandando menos recursos computacionais e tempo nas avaliações iniciais.

Tabela 5 – *Benchmarks* utilizados.

Funções	Tipo lógico	# Entradas		Domínio lógico
		Min	Max	
20,21	Multiplicadores	16	16	Aritmético
30	Comparadores	20	20	Aritmético
40,41,43	Raiz quadrada	10	18	Aritmético
50	Design PicoJava	19	19	Aleatório
65,69,73,74	*Benchmarks* MCNC	16	19	Aleatório
75-79	Funções simétricas	16	16	Aleatório

Fonte: Adaptado de Berndt, Abreu *et al.* (2022).

Executou-se uma sequência de evoluções para cada um dos *benchmarks* selecionados, cada qual com uma composição de *mini-batch* diferente, mas a partir da mesma população inicial de circuitos. Os hiperparâmetros utilizados para tal tarefa podem ser vislumbrados na Tabela 6, de modo que população indica a quantidade de circuitos gerados aleatoriamente e utilizados durante a evolução, profundidade lógica define o número máximo de nodos que podem ser encadeados nos circuitos, enquanto o resto dos hiperparâmetros indica a quantidade de gerações utilizada na seleção de indivíduos, o tamanho dos mini-batches e a frequência com que estes devem ser substituídos.

É importante pontuar ainda que, nestes experimentos iniciais, a forma de escolha das linhas para cada um dos subconjuntos ainda não segue a abordagem descrita na Seção 4, mas sim foi feita de forma sequencial, ou seja, para o primeiro *mini-batch* utilizaram-se as primeiras 64 linhas do conjunto de treinamento, para o segundo, as 64 seguintes, e assim por diante.

Tabela 6 – Hiperparâmetros utilizados.

Hiperparâmetro	Valor
População	5
Profundidade lógica	2000
Gerações	10000
Batch size	64
Change each	10000

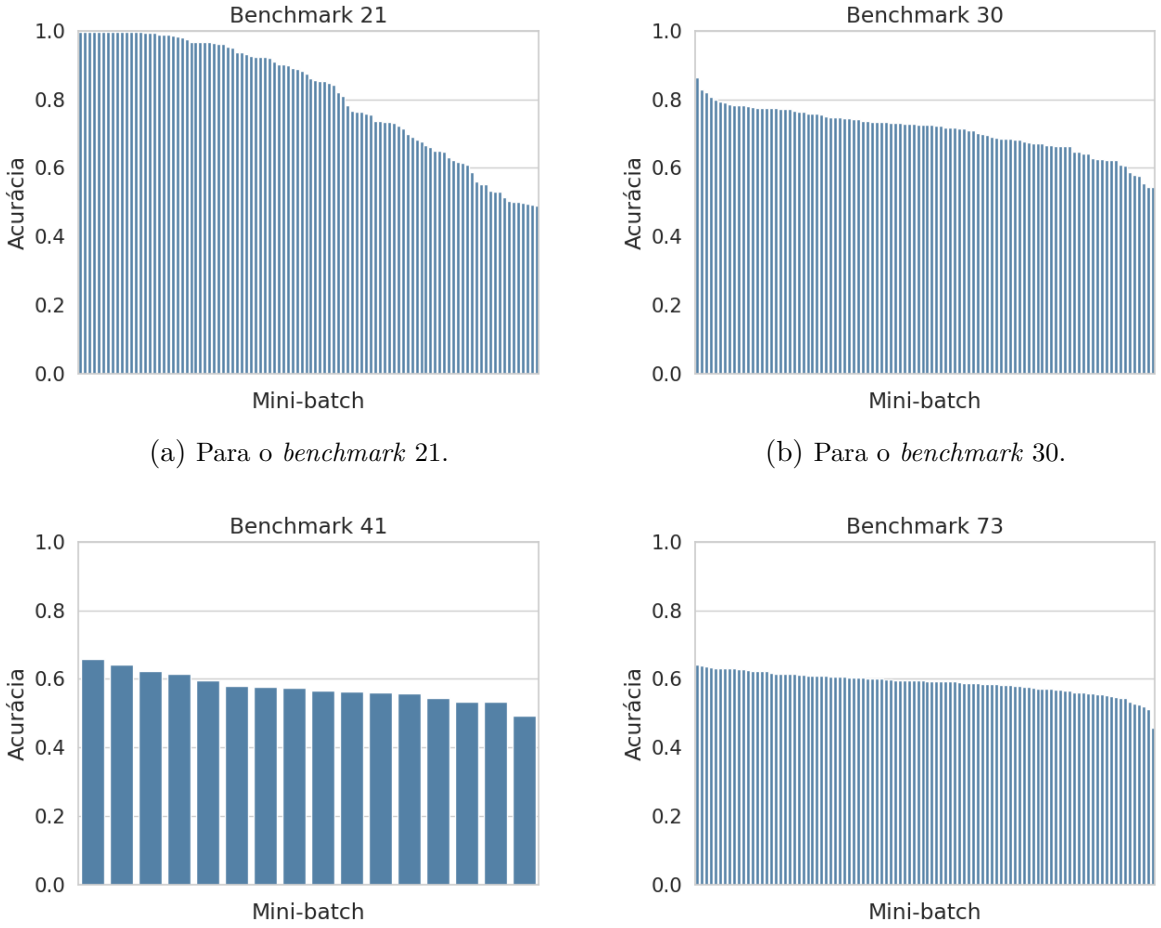
Fonte: Elaboração própria.

Para auxiliar na discussão da avaliação preliminar da adoção de CL no fluxo de síntese baseado em CGP, apresentam-se os gráficos das Figuras 8 e 9. Esta avaliação concentra-se na análise dos *benchmarks* 21, 30, 41 e 73. Para o desenvolvimento da Figura 8, mediu-se a acurácia dos circuitos finais gerados pelo fluxo utilizando as linhas do conjunto

de treinamento não pertencentes ao *mini-batch* usado durante a evolução, totalizando 100 valores diferentes para as figuras 8a, 8b e 8d e 16 para 8c, que se refere ao *benchmark* 41, o único do conjunto da competição do IWLS220 com uma tabela verdade completa, além de menor, com 1024 linhas. Originalmente, as linhas de tal tabela foram replicadas para a formação de três tabelas com 6400 linhas cada, visando a padronização dos *benchmarks*.

Tais valores, dispostos em ordem decrescente nos gráficos de barras apresentados, mostram que a escolha das linhas para os *mini-batches* influencia a capacidade de generalização do circuito para além das linhas utilizadas durante a evolução.

Figura 8 – Acurácia de circuitos finais gerados por evoluções com diferentes *mini-batches*, medida com o complemento de tal subconjunto dentro do conjunto de treinamento.



(a) Para o *benchmark* 21.

(b) Para o *benchmark* 30.

(c) Para o *benchmark* 41.

(d) Para o *benchmark* 73.

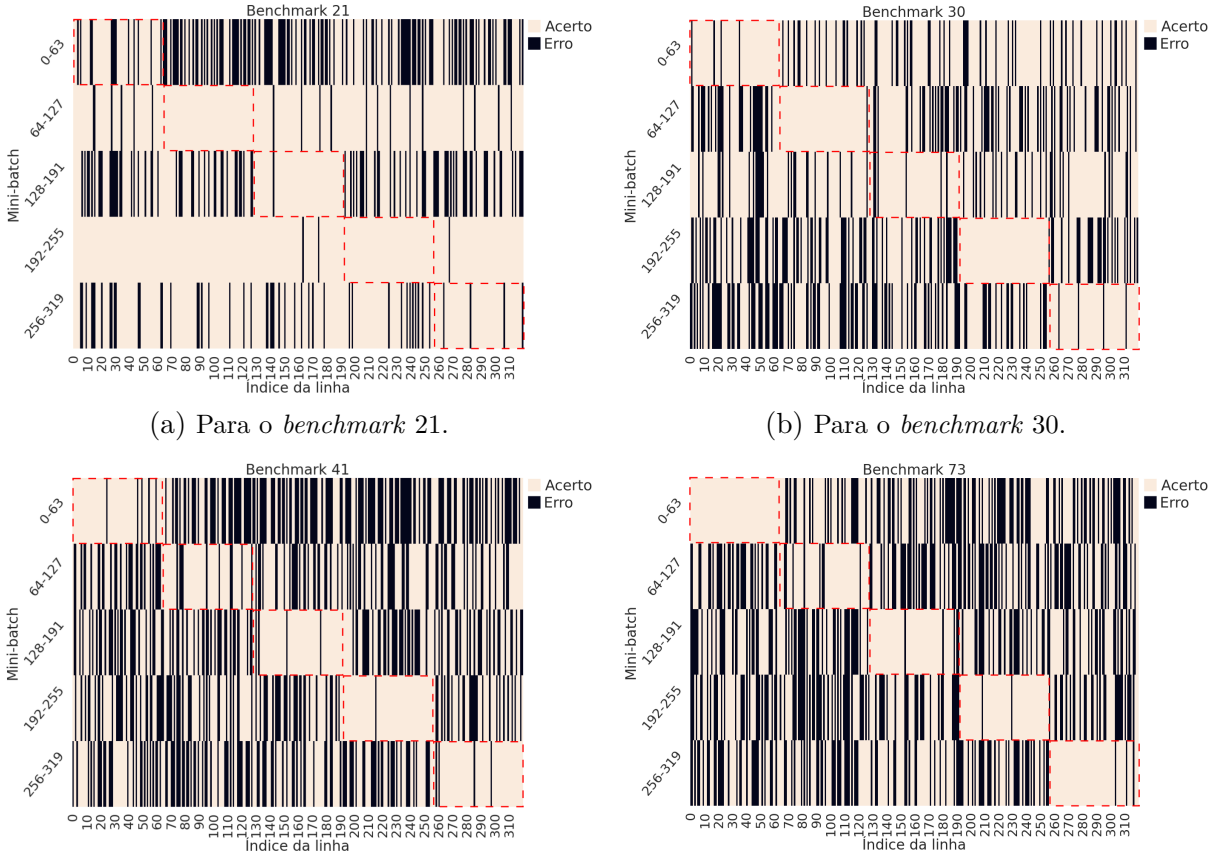
Fonte: Elaboração própria.

Já a Figura 9 mostra a configuração de acertos e erros dos circuitos finais gerados nas cinco primeiras evoluções para as 320 primeiras linhas da tabela verdade utilizada como conjunto de treinamento. A partir dela, é possível observar uma quantidade de acertos mais acentuada entre as linhas que compuseram os *mini-batches*, na diagonal principal da figura, além de configurações de acertos e erros distintas para as linhas não pertencentes aos *mini-batches*. Isso deixa visualmente claro o impacto que as linhas

escolhidas para o subconjunto têm tanto nas linhas aprendidas de forma intencional, como naquelas assimiladas indiretamente.

Em suma, tais comportamentos indicam que a utilização de uma estratégia de escolha de linhas para estes pequenos conjuntos tem o poder de afetar tanto a acurácia final dos circuitos sintetizados através do fluxo, como também a velocidade com que a síntese é feita, uma vez que este impacto na acurácia, se positivo, permite que menos linhas sejam avaliadas durante a fase de otimização.

Figura 9 – Configurações de acertos (em bege) e erros (em preto) de circuitos finais gerados por evoluções com diferentes *mini-batches*. As linhas que compõem cada um dos *mini-batches* estão destacadas por retângulos vermelhos tracejados.



(a) Para o *benchmark* 21.

(b) Para o *benchmark* 30.

(c) Para o *benchmark* 41.

(d) Para o *benchmark* 73.

Fonte: Elaboração própria.

Considerando isso, a abordagem proposta foi avaliada de forma mais robusta com relação à sua acurácia, utilizando-se todos os *benchmarks* da competição do IWLS 2020, conforme estabelecido na seção seguinte.

5.3 AVALIAÇÃO AMPLIADA DA ABORDAGEM PROPOSTA

Seguindo a avaliação inicial, esta segunda parte do trabalho apresenta uma avaliação ampliada da abordagem proposta. Para esta análise da abordagem descrita no

Capítulo 4, foram executados experimentos utilizando todos os *benchmarks* propostos para a competição do IWLS 2020. Ao todo são 100 *benchmarks*, descritos em detalhes na Tabela 7. Estes estão divididos nos domínios lógicos aritmético, aleatório e de aprendizado de máquina, são de diferentes tipos lógicos e possuem uma quantidade de entradas variada.

Tabela 7 – *Benchmarks* utilizados.

Funções	Tipo lógico	# Entradas		Domínio lógico
		Min	Max	
00-09	Somadores	32	512	Aritmético
10-19	Divisores, restos	32	512	Aritmético
20-29	Multiplicadores	16	256	Aritmético
30-39	Comparadores	20	200	Aritmético
40-49	Raiz quadrada	10	256	Aritmético
50-59	Design PicoJava	19	394	Aleatório
60-74	*Benchmarks* MCNC	16	52	Aleatório
75-79	Funções simétricas	16	16	Aleatório
80-89	MNIST	196	196	Aprendizado de máquina
90-99	CIFAR-10	768	768	Aprendizado de máquina

Fonte: Berndt, Abreu *et al.* (2022).

Executou-se uma série de evoluções para cada um dos *benchmarks* com as funções de dificuldade linear, quadrática, cúbica e raiz quadrada, além de evoluções sem CL, para permitir a comparação do desempenho entre estas abordagens. Os hiperparâmetros utilizados podem ser observados na Tabela 8. Conforme é possível observar nela, o penúltimo parâmetro foi otimizado para cada *benchmark* entre os valores indicados na tabela, levando em conta o objetivo de maximizar a acurácia dos circuitos finais gerados.

Tabela 8 – Hiperparâmetros utilizados para os experimentos com todos os *benchmarks*.

Hiperparâmetro	Valor
População	5
Profundidade lógica	2000
Gerações	50000
Batch size	256 - 2048
Change each	2500

Fonte: Elaboração própria.

Ainda, a fim de minimizar a influência da estocasticidade nos resultados obtidos, para cada um dos benchmarks foram rodadas vinte evoluções com sementes diferentes para a geração de números aleatórios durante o processo.

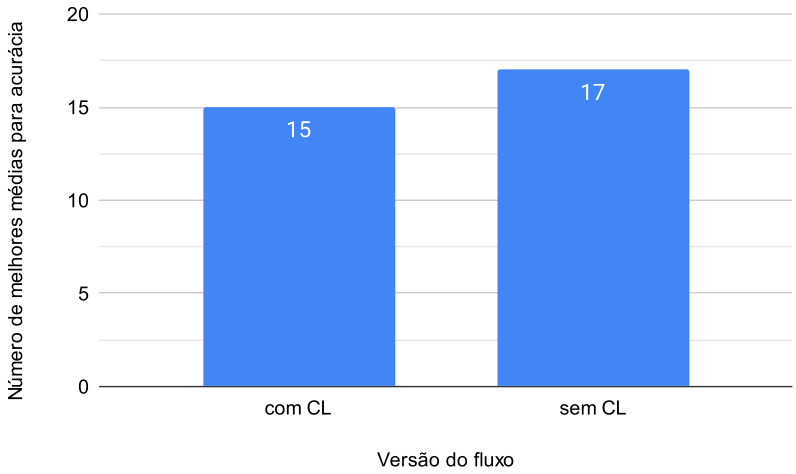
Para a análise dos resultados a seguir, foram considerados apenas *benchmarks* para os quais se obteve, através de pelo menos uma função de dificuldade ou da versão do fluxo sem CL, médias maiores ou iguais a 60% para a acurácia dos circuitos gerados, restando assim 81 dos 100 *benchmarks* originais. Tal decisão foi tomada visando evitar

que os *benchmarks* em que nenhuma versão do CGP produz soluções melhores que o nível casual (*chance level*) atrapalhem as análises dos demais casos.

Ainda, os testes de significância estatística foram realizados com o teste *Wilcoxon Signed-Rank*, designado para amostras dependentes e não-paramétricas, utilizando uma taxa de 95% de confiança (REY; NEUHÄUSER, 2011). Este teste foi escolhido pelo fato de os mesmos *benchmarks* terem sido utilizados para a avaliação da acurácia das versões do fluxo com e sem CL, configurando os conjuntos de dados como dependentes, e pelo fato de grande parte das distribuições dos dados analisados não serem normais, conforme o teste de Shapiro-Wilk, com 95% de confiança. O *p-valor* para cada uma das distribuições em questão pode ser observado no Anexo A.

A versão do fluxo com CL obteve médias de acurácia superiores às atingidas sem CL para 50 *benchmarks*, enquanto o contrário pôde ser observado para 31 dos exemplares. Entretanto, conforme exposto na Figura 10, considerando as diferenças entre as médias de acurácia para as abordagens com e sem CL, tomando o maior valor obtido com CL entre as diferentes funções de dificuldade, observou-se que, dos 50 *benchmarks* destacados anteriormente, apenas 15 apresentaram diferenças estatisticamente significantes ($p\text{-valor}<,05$ para o teste *Wilcoxon Signed-Rank*). Para os casos em que as médias sem CL foram superiores, houveram 17 casos com diferenças estatisticamente significantes. Assim, para a maioria dos *benchmarks* analisados não foram observadas diferenças significantes em acurácia.

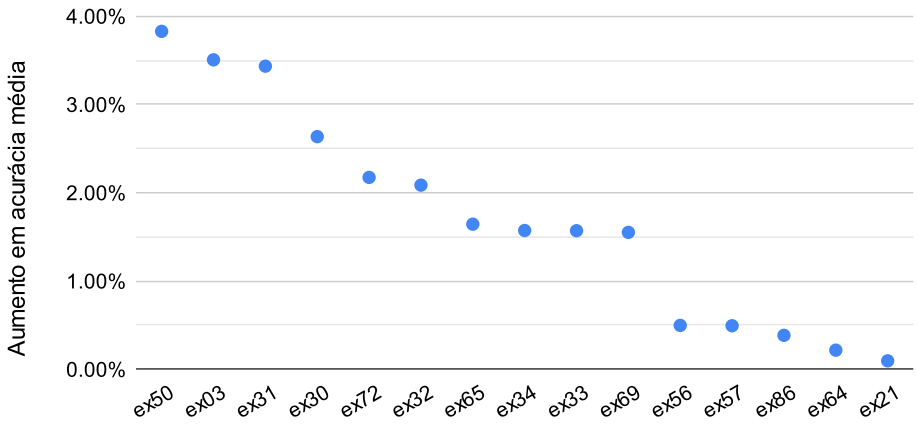
Figura 10 – Comparação do número de vezes que o fluxo sem CL e com CL apresentou melhor desempenho, conforme a média da acurácia.



Fonte: Elaboração própria.

Através da Figura 11 é possível observar as diferenças entre as acurácias médias obtidas com e sem CL para os casos em que CL foi a abordagem mais vantajosa. A média de aumento foi de 1,71%. O gráfico em questão também permite verificar que os *benchmarks* 50, 3 e 31 apresentaram os aumentos mais expressivos, de 3,83%, 3,51% e 3,43%, respectivamente.

Figura 11 – Diferença entre a melhor mediana para a acurácias obtidas com CL e melhor mediana obtida sem CL.



Fonte: Elaboração própria.

As maiores médias, a função de dificuldade utilizada para obtê-las, os aumentos na acurácia média atingidos com CL e o *p-valor* para cada um dos *benchmarks* em questão podem ser observados na Tabela 9. Através dela pode-se destacar a classe de problemas que compreende os *benchmarks* 30 a 39, que possui cinco representantes entre estes *benchmarks*, o maior número entre todas as classes. Isso enseja uma investigação mais detalhada dos efeitos de CL sobre este conjunto em trabalhos futuros.

Pode-se destacar ainda a função de dificuldade linear, que propiciou doze das quinze melhorias destacadas na Tabela 9, e fato de que apenas *benchmarks* com acurácias originalmente altas terem se beneficiado da abordagem proposta.

Tabela 9 – Detalhamento das médias para a acurácia obtidas com CL e do aumento representado por elas com relação às médias obtidas sem CL, para os *benchmarks* com aumentos estatisticamente significantes.

Benchmark	Maior média	Função de dificuldade	Melhora	*p-valor*
50	98,08%	Linear	3,83%	0,0003
3	94,60%	Linear	3,51%	0,0004
31	96,15%	Linear	3,43%	0,0005
30	97,22%	Linear	2,64%	0,0037
72	96,56%	Quadrática	2,17%	0,0107
32	94,44%	Linear	2,09%	0,0215
65	99,66%	Linear	1,64%	0,0008
34	93,04%	Linear	1,57%	0,0484
33	93,95%	Linear	1,57%	0,0400
69	99,68%	Linear	1,55%	0,0216
56	96,15%	Linear	0,50%	0,0006
57	98,49%	Cúbica	0,49%	0,0062
86	94,86%	Linear	0,39%	0,0362
64	99,81%	Raiz quadrada	0,22%	0,0218
21	99,75%	Linear	0,10%	0,0213

Fonte: Elaboração própria.

Adicionalmente, a Tabela 10 expõe o restante dos *benchmarks* que apresentaram um aumento em sua média com a utilização de CL, a função de dificuldade com a qual ela foi obtida, o aumento observado com relação à média atingida sem CL e o *p-valor* para a comparação entre os valores de acurácia obtidos com e sem CL. De acordo com tais valores, pode-se destacar os *benchmarks* 62, 81, 63, 73 e 88, cujos *p-valores* são próximos ao definido para a rejeição da hipótese nula do teste de significância estatística.

Tabela 10 – Detalhamento das médias para a acurácia obtidas com CL e do aumento representado por elas com relação às médias obtidas sem CL, para o restante dos *benchmarks*.

Benchmark	Maior média	Função de dificuldade	Melhora	*p-valor*
62	92,49%	Quadrática	1,29%	0,0583
81	86,85%	Linear	0,75%	0,0583
63	96,90%	Cúbica	1,21%	0,0637
73	84,44%	Linear	5,01%	0,0637
88	92,32%	Linear	0,88%	0,0759
01	96,39%	Linear	2,63%	0,0826
71	99,01%	Quadrática	1,44%	0,0936
10	80,68%	Linear	1,27%	0,0973
39	89,24%	Linear	1,35%	0,1327
41	92,86%	Cúbica	2,82%	0,1327
05	92,10%	Linear	1,29%	0,1650
82	91,93%	Linear	0,75%	0,1650
85	93,28%	Linear	0,61%	0,1650
55	98,11%	Linear	1,80%	0,1769
59	86,59%	Linear	0,50%	0,1893
89	93,11%	Linear	0,54%	0,2162
35	92,63%	Linear	1,54%	0,2305
00	89,39%	Linear	1,84%	0,2611
36	91,91%	Linear	0,71%	0,2611
58	93,54%	Cúbica	0,47%	0,2772
54	92,96%	Linear	0,28%	0,3488
70	98,78%	Cúbica	0,67%	0,3547
61	98,99%	Quadrática	1,57%	0,3793
37	91,65%	Linear	0,75%	0,4304
80	82,17%	Cúbica	0,16%	0,4524
60	90,06%	Raiz quadrada	0,79%	0,4980
29	91,16%	Cúbica	3,73%	0,5540
75	82,21%	Linear	0,13%	0,6215
68	97,26%	Linear	0,10%	0,6477
16	84,02%	Linear	0,26%	0,7841
84	91,97%	Linear	0,09%	0,8408
83	95,13%	Raiz quadrada	0,24%	0,9563
25	99,87%	Linear	0,05%	0,9592
87	93,43%	Raiz quadrada	0,02%	0,9854
23	99,997%	Cúbica	0,00001%	1,0000

Fonte: Elaboração própria.

Dessa forma, de modo geral, a abordagem de CL proposta neste trabalho apresenta uma contribuição limitada para o aumento da acurácia dos circuitos gerados com o fluxo baseado em CGP. Ainda assim, a análise dos resultados mostrou que houve *benchmarks* que se beneficiaram da abordagem e uma classe de funções que apresentou aumentos em acurácia estatisticamente significativos para metade de seus *benchmarks*. Isso indica que, apesar de o ganho em acurácia não ser garantido, esta é uma alternativa cujo uso pode ser considerado e testado, especialmente para funções com características em comum com a classe de funções lógicas identificada neste trabalho.

6 CONCLUSÕES

Este trabalho apresentou uma abordagem de *Curriculum Learning* aplicada a um fluxo de otimização lógica baseado em Programação Genética Cartesiana, bem como resultados de experimentos realizados, visando verificar a eficácia da solução para o aprimoramento da acurácia dos circuitos gerados.

Os dados dos experimentos iniciais analisados mostram que *mini-batches* diferentes têm a capacidade de fomentar a sintetização de circuitos com acurácias bastante díspares entre si, quando avaliados sob as linhas não pertencentes aos subconjuntos utilizados para treinamento. Também se observou que *mini-batches* diferentes levam à sintetização de circuitos que erram e acertam linhas distintas da tabela verdade.

Tais resultados indicam que a utilização de uma estratégia de escolha de linhas para os *mini-batches* tem o poder de afetar a acurácia dos circuitos sintetizados e, além disso, a velocidade com que ocorre a síntese, caso se considere que um impacto positivo na acurácia permite que menos linhas sejam avaliadas durante a fase de busca. Assim, pode-se concluir que a técnica é pertinente para o contexto proposto.

Por outro lado, dados de experimentos realizados para a avaliação da solução desenvolvida neste trabalho em si indicam que ela apresenta uma contribuição limitada para a melhora da acurácia dos circuitos gerados, sendo que para a maioria dos *benchmarks* analisados não foram observadas diferenças em acurácia estatisticamente significantes. Entre os casos significantes, houve uma média de 1,71% de acréscimo.

Ainda assim, através dos mesmos experimentos também foi possível identificar *benchmarks* e uma classe de problemas que se beneficiou da abordagem de forma destacada, indicando que, mesmo não havendo garantia de que ao aplicar a estratégia proposta sejam observados ganhos em acurácia, esta é uma alternativa cujo uso pode ser considerado e testado, especialmente para funções com características comuns às classes de funções destacadas na análise.

O partir do que foi observado, uma direção de pesquisa possível é a elaboração de novas abordagens de CL que foquem na construção de *mini-batches* com dificuldades bem distribuídas, uma vez que esta foi a função de dificuldade para a qual se observou o maior número de casos de melhora em acurácia.

Outra possibilidade é a mudança do foco de investigação, da solução para o problema, buscando entender quais as características-chave dos *benchmarks* do IWLS 2020 que fazem com que eles sejam difíceis de serem aprendidos nas condições testadas neste trabalho, fundamentando a elaboração de estratégias mais eficazes para a melhoria da acurácia dos circuitos gerados, sejam elas baseadas em CL ou não.

REFERÊNCIAS

ABREU, Brunno A. de *et al.* Fast Logic Optimization Using Decision Trees. *In:* 2021 IEEE International Symposium on Circuits and Systems (ISCAS). [*S.l.*: *s.n.*], 2021. P. 1–5. DOI: 10.1109/ISCAS51556.2021.9401664.

BARUA, Hrishav Bakul; MONDAL, Kartick Chandra. Approximate Computing: A Survey of Recent Trends—Bringing Greenness to Computing and Communication. **Journal of The Institution of Engineers (India): Series B**, v. 100, n. 3, p. 619–626, 2019. DOI: https://doi.org/10.1007/s40031-019-00418-8.

BENGIO, Yoshua *et al.* Curriculum Learning. *In:* PROCEEDINGS of the 26th Annual International Conference on Machine Learning. Montreal, Quebec, Canada: Association for Computing Machinery, 2009. (ICML '09), p. 41–48. DOI: 10.1145/1553374.1553380. Disponível em: https://doi.org/10.1145/1553374.1553380.

BERNDT, Augusto; ABREU, Brunno A. de *et al.* A CGP-based Logic Flow: Optimizing Accuracy and Size. **Journal of Integrated Circuits and Systems (JICS)**, 2022. DOI: 10.29292/jics.v17i1.546.

_____. Accuracy and Size Trade-off of a Cartesian Genetic Programming Flow for Logic Optimization. *In:* PROCEEDINGS of the 34th Symposium on Integrated Circuits and Systems Design. Brazil: [*s.n.*], 2021. (SBCCI '21). DOI: 10.1109/SBCCI53441.2021.9529968.

BERNDT, Augusto André Souza; FOGAÇA, Mateus; MEINHARDT, Cristina. A review of machine learning in logic synthesis. **Journal of Integrated Circuits and Systems**, v. 17, n. 3, p. 1–12, 2022.

CHATTERJEE, Satrajit. Learning and Memorization. *In:*_____. **Proceedings of the 35th International Conference on Machine Learning**. [*S.l.*]: PMLR, out. 2018. (Proceedings of Machine Learning Research), p. 755–763. Disponível em: https://proceedings.mlr.press/v80/chatterjee18a.html.

COSTAMAGNA, Andrea; DE MICHELI, Giovanni. Accuracy recovery: A decomposition procedure for the synthesis of partially-specified Boolean functions. **Integration**, v. 89, p. 248–260, 2023. ISSN 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2022.12.008. Disponível em: https://www.sciencedirect.com/science/article/pii/S0167926022001791.

DARWIN, Charles. On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. **London: Murray**, 1859.

FRIEDBERG, R. M. A Learning Machine: Part I. **IBM Journal of Research and Development**, v. 2, n. 1, p. 2–13, 1958. DOI: 10.1147/rd.21.0002.

GÉRON, Aurélien. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow**. 2. ed. [*S.l.*]: O'Reilly Media, Inc., 2019. ISBN 9781492032649.

GRAVES, Alex *et al.* Automated Curriculum Learning for Neural Networks. *In:* PROCEEDINGS of the 34th International Conference on Machine Learning - Volume 70. Sydney, NSW, Australia: JMLR.org, 2017. (ICML'17), p. 1311–1320.

HANSEN, Nikolaus; ARNOLD, Dirk V; AUGER, Anne. Evolution strategies. *In:* SPRINGER handbook of computational intelligence. [*S.l.*]: Springer, 2015. P. 871–898.

KAHNG, Andrew B *et al.* **VLSI physical design: from graph partitioning to timing closure**. [*S.l.*]: Springer Science & Business Media, 2011.

KOZA, John R. **Genetic programming: on the programming of computers by means of natural selection**. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262111705.

KOZA, John R. *et al.* **Genetic Programming IV: Routine Human-Competitive Machine Intelligence**. [*S.l.*]: Springer New York, 2005.

LAVAGNO, Luciano *et al.* **Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology**. 2. ed. [*S.l.*]: CRC Press, 2016. ISBN 9781315215112. DOI: https://doi.org/10.1201/9781315215112.

LIMA, Bryan Martins *et al.* Adaptive Batch Size CGP: Improving Accuracy and Runtime for CGP Logic Optimization Flow. *In:* GENETIC Programming. [*S.l.*]: Springer Nature Switzerland, 2023. P. 149–164. DOI: https://doi.org/10.1007/978-3-031-29573-7_10.

LOUIS, Sushil John. **Genetic algorithms as a computational tool for design**. 1993. Tese (Doutorado) – Department of Computer Science, Indiana University - EUA.

MATIISEN, Tambet *et al.* Teacher–Student Curriculum Learning. **IEEE Transactions on Neural Networks and Learning Systems**, v. 31, n. 9, p. 3732–3740, 2020. DOI: 10.1109/TNNLS.2019.2934906.

MILANO, Nicola; NOLFI, Stefano. Automated curriculum learning for embodied agents a neuroevolutionary approach. **Scientific Reports**, v. 11, n. 8985, p. 1–14, abr. 2021. Disponível em: https://doi.org/10.1038/s41598-021-88464-5.

MILANO, Nicola; PAGLIUCA, Paolo; NOLFI, Stefano. Robustness, evolvability and phenotypic complexity: insights from evolving digital circuits. **Evolutionary Intelligence**, Springer, v. 12, n. 1, p. 83–95, 2019.

MILLER, Julian F; THOMSON, Peter; FOGARTY, Terence. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. **Genetic**

algorithms and evolution strategies in engineering and computer science, Wiley Chechester, UK, p. 105–131, 1997.

MILLER, Julian F. An Empirical Study of the Efficiency of Learning Boolean Functions Using a Cartesian Genetic Programming Approach. *In:* PROCEEDINGS of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2. Orlando, Florida: Morgan Kaufmann Publishers Inc., 1999. (GECCO'99), p. 1135–1142.

_____. Cartesian Genetic Programming. *In:* CARTESIAN Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. P. 17–34. ISBN 978-3-642-17310-3. DOI: 10.1007/978-3-642-17310-3_2. Disponível em: https://doi.org/10.1007/978-3-642-17310-3%5C_2.

MIYASAKA, Yukio *et al.* Logic Synthesis for Generalization and Learning Addition. *In:* 2021 Design, Automation Test in Europe Conference Exhibition (DATE). [*S.l.*: *s.n.*], 2021. P. 1032–1037. DOI: 10.23919/DATE51398.2021.9474169. Disponível em: https://doi.org/10.23919/DATE51398.2021.9474169.

POLI, Riccardo. Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming. *In:* INTERNATIONAL Conference on Genetic Algorithms. [*S.l.*: *s.n.*], 1997.

RAI, Shubham *et al.* Logic Synthesis Meets Machine Learning: Trading Exactness for Generalization. *In:* IEEE. 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). [*S.l.*: *s.n.*], 2021.

REY, Denise; NEUHÄUSER, Markus. Wilcoxon-Signed-Rank Test. *In:* **International Encyclopedia of Statistical Science**. Edição: Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. P. 1658–1659. ISBN 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_616. Disponível em: https://doi.org/10.1007/978-3-642-04898-2_616.

RIENER, Heinz *et al.* On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis. *In:* 2019 Design, Automation Test in Europe Conference Exhibition (DATE). [*S.l.*: *s.n.*], 2019. P. 1649–1654. DOI: 10.23919/DATE.2019.8715185.

SACHETTI, Naiara. **Código fonte deste projeto**. [*S.l.*: *s.n.*], 2024. Disponível em: https://codigos.ufsc.br/naiara.sachetti/cgpv3-image/-/tree/07446a31da4da577ff5866c4a280320d7e8d56dc.

SACHETTI, Naiara; BERNDT, Augusto. **Código fonte deste projeto**. [*S.l.*: *s.n.*], 2024. Disponível em: https://gitlab.com/gudeh/cgpv3/-/tree/728ad1ba31d1b993ff0eaf222affc7fca08f63e8.

SALIMANS, Tim *et al.* **Evolution Strategies as a Scalable Alternative to Reinforcement Learning**. [*S.l.*: *s.n.*], 2017. arXiv: 1703.03864.

SCARABOTTOLO, Ilaria *et al.* Approximate Logic Synthesis: A Survey. **Proceedings of the IEEE**, v. 108, n. 12, p. 2195–2213, 2020. DOI: 10.1109/JPROC.2020.3014430.

TENACE, Valerio; CALIMERA, Andrea. Inferential Logic: a Machine Learning Inspired Paradigm for Combinational Circuits. *In:* 2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC). [*S.l.*: *s.n.*], 2018. P. 149–154. DOI: 10.1109/VLSI-SoC.2018.8644808. Disponível em: https://doi.org/10.1109/VLSI-SoC.2018.8644808.

WANG, Laung-Terng; CHANG, Yao-Wen; CHENG, Kwang-Ting Tim. **Electronic design automation: synthesis, verification, and test**. [*S.l.*]: Morgan Kaufmann, 2009.

WANG, Xin; CHEN, Yudong; ZHU, Wenwu. A Survey on Curriculum Learning. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 44, n. 9, p. 4555–4576, 2022. DOI: 10.1109/TPAMI.2021.3069908.

WAZLAWICK, R.S. **História da Computação**. [*S.l.*]: Elsevier Brasil, 2016. ISBN 9788535285468.

XU, Qiang; MYTKOWICZ, Todd; KIM, Nam Sung. Approximate Computing: A Survey. **IEEE Design Test**, v. 33, n. 1, p. 8–22, 2016. DOI: 10.1109/MDAT.2015.2505723.

ZENG, Wei; DAVOODI, Azadeh; TOPALOGLU, Rasit Onur. Lorax: Machine Learning-Based Oracle Reconstruction With Minimal I/O Patterns. *In:* 2021 IEEE COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI (ISVLSI). [*S.l.*: *s.n.*], 2021. P. 126–131. DOI: 10.1109/ISVLSI51109.2021.00033. Disponível em: https://doi.org/10.1109/ISVLSI51109.2021.00033.

ZHU, Keren *et al.* Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network. *In:* PROCEEDINGS of the 2020 ACM/IEEE Workshop on Machine Learning for CAD. [*S.l.*: *s.n.*], 2020. P. 145–150.

Anexos

ANEXO A – RESULTADOS DO TESTE DE NORMALIDADE DE SHAPIRO-WILK

Resultados do teste de Shapiro-Wilk para os *benchmarks* com médias de acurácia acima de 60% (*p-valor>,05* indica que a distribuição é normal).

Benchmark	Melhor função (CL)	*p-valor* com CL	*p-valor* sem CL
ex00	Linear	0,3674	0,2004
ex01	Linear	0,0144	$1 * 10^{-5}$
ex02	Linear	0,02049	0,00125
ex03	Linear	$9 * 10^{-6}$	0,0209
ex05	Linear	0,013	0,0053
ex07	Linear	0,0012	0,0686
ex09	Linear	0,3728	0,2327
ex10	Linear	0,033	0,6279
ex11	Quadrática	0,3098	0,9088
ex12	Linear	0,0102	0,2733
ex13	Raiz quadrada	0,0005	0,5851
ex14	Quadrática	0,1523	0,3809
ex15	Linear	0,0108	0,6395
ex16	Linear	0,2269	0,7554
ex17	Raiz quadrada	0,81778	0,9136
ex18	Raiz quadrada	$6 * 10^{-7}$	0,007
ex19	Raiz quadrada	$2 * 10^{6}$	0,3777
ex21	Linear	0,0004	$1 * 10^{-8}$
ex23	Cúbica	$1 * 10^{-7}$	$2 * 10^{-8}$
ex25	Linear	$1 * 10^{-8}$	$3 * 10^{-8}$
ex27	Quadrática	$3 * 10^{-8}$	$4 * 10^{-6}$
ex29	Cúbica	$4 * 10^{-7}$	$4 * 10^{-6}$
ex30	Linear	0,0077	0,0005
ex31	Linear	0,7759	0,0008
ex32	Linear	0,0001	0,0008
ex33	Linear	0,4883	0,0005
ex34	Linear	0,006	0,0275
ex35	Linear	0,0031	0,0007
ex36	Linear	0,0004	0,0424
ex37	Linear	0,0278	0,0004
ex38	Linear	0,0011	0,0031
ex39	Linear	0,0023	0,0644
ex40	Quadrática	0,6606	0,8562
ex41	Cúbica	0,1312	0,0011
ex43	Linear	0,5245	0,1288
ex50	Linear	0,0011	0,0027
ex52	Linear	0,3919	0,2661
ex53	Cúbica	0,1556	0,273
ex54	Cúbica	$2 * 10^{-7}$	$8 * 10^{-7}$
ex55	Linear	0,6511	0,0003
ex56	Linear	0,5263	0,0001

ex57	Cúbica	0,7333	0,0036
ex58	Cúbica	0,0001	0,0043
ex59	Linear	$3 * 10^{-5}$	0,1256
ex60	Raiz quadrada	0,2283	0,0057
ex61	Quadrática	0,0001	$8 * 10^{-7}$
ex62	Quadrática	0,0135	0,4788
ex63	Cúbica	0,0858	0,0004
ex64	Raiz quadrada	0,0003	$2 * 10^{-5}$
ex65	Linear	0,0014	0,0091
ex66	Raiz quadrada	0,0001	0,0789
ex67	Cúbica	0,1637	0,0931
ex68	Linear	0,1092	0,1857
ex69	Linear	$2 * 10^{-6}$	0,0002
ex70	Cúbica	0,0005	$7 * 10^{-6}$
ex71	Quadrática	$2 * 10^{-6}$	0,0006
ex72	Quadrática	0,0001	0,0019
ex73	Linear	0,059	0,3682
ex75	Linear	0,6089	0,2522
ex76	Raiz quadrada	$5 * 10^{-6}$	0,0012
ex77	Raiz quadrada	$5 * 10^{-6}$	0,007
ex78	Linear	0,0004	0,1144
ex79	Raiz quadrada	$5 * 10^{-6}$	0,007
ex80	Cúbica	0,0001	0,1828
ex81	Linear	0,3799	0,296
ex82	Linear	0,5296	0,0001
ex83	Raiz quadrada	0,0165	0,0007
ex84	Linear	0,0686	0,4768
ex85	Linear	0,2172	0,0863
ex86	Linear	$1 * 10^{-6}$	0,0034
ex87	Raiz quadrada	0,5117	0,0628
ex88	Linear	0,8569	0,0013
ex89	Linear	0,4565	0,002
ex91	Linear	0,4078	0,002
ex92	Linear	0,9974	0,0034
ex93	Linear	0,0853	0,1284
ex95	Raiz quadrada	0,0087	0,5262
ex96	Linear	0,0006	0,1457
ex97	Linear	0,0943	0,1777
ex98	Linear	0,4135	0,508
ex99	Linear	0,0009	0,2334

Fonte: Elaboração própria.

XIV Computer on the Beach
30 de Março a 01 de Abril de 2023, Florianópolis, SC, Brasil

Aprendizagem Curricular Aplicada à Programação Genética Cartesiana para Otimização Lógica de Circuitos Digitais

Naiara Sachetti
Universidade Federal de Santa Catarina
Florianopolis, Santa Catarina, Brasil
naiara.sachetti@grad.ufsc.br

Bryan Martins Lima
Universidade Federal de Santa Catarina
Florianopolis, Santa Catarina, Brasil
bryan.l@grad.ufsc.br

Augusto Berndt
Universidade Federal de Santa Catarina
Florianopolis, Santa Catarina, Brasil
augusto.berndt@posgrad.ufsc.br

Cristina Meinhardt
Universidade Federal de Santa Catarina
Florianopolis, Santa Catarina, Brasil
cristina.meinhardt@ufsc.br

Jonata Tyska Carvalho
Universidade Federal de Santa Catarina
Florianopolis, Santa Catarina, Brasil
jonata.tyska@ufsc.br

RESUMO

Atualmente, o número elevado de entradas em circuitos digitais tem se tornado um problema cada vez mais comum, demandando novas soluções para a otimização lógica dos mesmos. Uma técnica que vem sendo utilizada nos últimos anos é a de *Logic Learning*, que tem como base o uso de técnicas de *Machine Learning* (ML) para a geração de circuitos aproximados a partir de descrições parciais das funções lógicas. Uma das técnicas de ML já utilizada em *Logic Learning* é a Programação Genética Cartesiana (CGP). Apesar de fluxos de síntese lógica baseados em CGP já terem se mostrado efetivos, podem ter dificuldades em atingir uma evolucionabilidade satisfatória dentro de uma restrição de tempo de execução e para certas funções. Neste contexto, o presente trabalho busca investigar a aplicação de uma técnica denominada aprendizagem curricular (*Curriculum Learning*) para melhorar a evolucionalidade, convergindo para uma melhor acurácia. Para avaliar de forma preliminar a solução proposta foi utilizada uma porção dos *benchmarks* da competição de síntese lógica da conferência IWLS de 2020, através dos quais observou-se que, quando são priorizados exemplos de treinamento com dificuldade igualitariamente distribuída ou exemplos mais difíceis, a técnica de aprendizado curricular pode trazer benefícios para o processo evolutivo. Entre os resultados que colaboram para esta hipótese estão ganhos de até 20% na acurácia dos circuitos gerados (se considerada uma diferença absoluta) e *benchmarks* que só apresentaram ganhos em acurácia. Ainda assim, demais resultados evidenciam a importância de uma avaliação mais detalhada sobre a abordagem.

PALAVRAS-CHAVES

Programação Genética Cartesiana, *Logic Learning*, *Machine Learning*, *Curriculum Learning*, otimização lógica

1 INTRODUÇÃO

Com o passar dos anos, a implementação de funções lógicas complexas em circuitos digitais tem se tornado cada vez mais comum. Tal cenário demanda novas soluções para ferramentas de otimização lógica, já que problemas que exigem tais funções, como redes neurais, normalmente possuem um número elevado de entradas. Neste contexto, uma técnica que vem sendo utilizada recentemente é a de *Logic Learning*, que se baseia na utilização de algoritmos de *Machine Learning* (ML) para a geração de circuitos aproximados, ou seja, circuitos capazes de generalizar o comportamento de funções descritas por tabelas-verdade especificadas meramente de forma parcial.

Uma das técnicas de ML já utilizadas em *Logic Learning* é a Programação Genética Cartesiana (*Cartesian Genetic Programming* - CGP), uma técnica de computação evolutiva que utiliza grafos dirigidos acíclicos representados como uma grade de duas dimensões de nodos computacionais para representar programas [6]. Tipicamente, a cada geração, uma população de programas é avaliada para que se estabeleça a acurácia de cada um de seus indivíduos e, em seguida, uma nova população é formada pelo programa com maior acurácia (ou seja, que produz o maior número de saídas esperadas) e novos indivíduos, chamados de filhos, gerados a partir de mutações de tal programa [6]. Em *Logic Learning*, tais nodos representam portas lógicas, de forma que os circuitos podem ser representados por AIGs (do inglês, AND-Inverter Graphs, Grafos de ANDs e Inversores).

Um fluxo de otimização lógica baseado em CGP é apresentado em [2] e [3], com resultados promissores para boa parte das funções lógicas consideradas, quando levados em conta simultaneamente acurácia e tamanho dos circuitos gerados. Entretanto, restam situações em que a solução não produz circuitos com uma acurácia satisfatória. Sendo assim, fluxos de otimização lógica baseados em CGP poderiam se beneficiar da aplicação de novos mecanismos potencialmente capazes de melhorar sua evolucionabilidade, ou seja, a sua capacidade de continuar encontrando soluções melhores. Seguindo esta direção, decidiu-se investigar o uso da técnica de *Curriculum Learning* (CL) [5].

Como formalizado em [1], um *curriculum* pode ser visto, de forma abstrata, como uma sequência de critérios de treinamento, com cada critério sendo associado com um conjunto diferente de pesos nos exemplos de treinamento. Os autores de [4] trazem este conceito para o contexto de treinamento evolucionário de agentes corporificados (isto é, que possuem um corpo e estão situados em um ambiente), usando CL para manipular as condições ambientais que desafiam as fraquezas dos agentes em evolução, mantendo um nível apropriado de dificuldade.

XIV Computer on the Beach
30 de Março a 01 de Abril de 2023, Florianópolis, SC, Brasil

Neste trabalho, aplicou-se uma estratégia de *Curriculum Learning* semelhante à utilizada em [4] a um fluxo de otimização lógica baseado em CGP inspirado em [2] e [3]. No presente resumo são apresentados resultados de experimentos preliminares que visam entender se tal abordagem pode ser efetiva para melhorar a acurácia dos circuitos em evolução.

2 SOLUÇÃO PROPOSTA

A Figura 1 mostra o fluxo de otimização lógica baseado em CGP utilizado como base para os experimentos realizados. Tal fluxo parte de populações iniciais aleatórias, seleciona os circuitos com maior acurácia e avalia a acurácia final do circuito gerado. Durante a etapa de seleção, a avaliação dos circuitos é feita a partir de *mini-batches*, compostos por uma parcela aleatoriamente selecionada das linhas da tabela-verdade utilizada como conjunto de treinamento. A quantidade de linhas que compõem cada *mini-batch* é dada pelo hiperparâmetro *batch size*, enquanto a quantidade de gerações que um mini-batch deve permanecer inalterado é definida pelo hiperparâmetro *change each*.

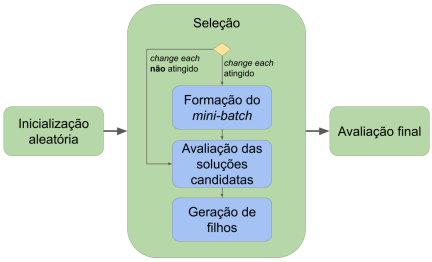


Figura 1: Fluxo de otimização baseado em CGP utilizado como base.

A estratégia de *Curriculum Learning* (CL) proposta neste trabalho substitui a seleção aleatória de linhas para os *mini-batches*. De modo semelhante à abordagem utilizada em [4], tal seleção é feita com base em uma *função de dificuldade* e em um valor de dificuldade atribuído para cada uma das linhas da tabela-verdade. As alterações no fluxo necessárias para o suporte de CL foram feitas diretamente no código original, na linguagem C++.

Na aplicação de *Curriculum Learning*, o valor de dificuldade é definido por meio da Equação 1, de modo que H_i é o número de vezes que uma linha foi corretamente solucionada por circuitos em avaliação nas gerações anteriores, enquanto E_i é o número total de vezes que uma linha participou de avaliações de circuitos.

$$D_i = \frac{H_i}{E_i} \tag{1}$$

Já a função de dificuldade define os intervalos de dificuldade nos quais as linhas que compõem os *mini-batches* devem se encaixar. O limite de dificuldade para cada posição do *mini-batch* é dado pela Equação 2, sendo b o índice da posição do *mini-batch* e m o valor de *batch size*.

O valor inicial de dificuldade para cada uma das linhas é definido através de uma avaliação das mesmas antes da primeira geração de seleção de circuitos. Esta avaliação é feita com a população inicial de circuitos gerada aleatoriamente e que deverá, logo em seguida, passar pela seleção.

$$L_b = \frac{f(b)}{f(m)} \tag{2}$$

3 AVALIAÇÃO E RESULTADOS

Para avaliar preliminarmente a abordagem idealizada utilizou-se dezesseis *benchmarks* da competição de síntese lógica da conferência IWLS de 2020. O critério de escolha foi o tamanho das entradas, considerando os recursos computacionais disponíveis no momento. Assim, considerou-se apenas os *benchmarks* com no máximo 20 entradas.

As funções de dificuldade consideradas foram linear e quadrática. No primeiro caso, são gerados mini-batches com uma dificuldade igualitariamente distribuída, evitando que somente linhas fáceis formem o conjunto. No segundo caso, prioriza-se a seleção de linhas difíceis. Em ambos os casos, garante-se que os circuitos, durante o processo de seleção, sejam expostos a linhas não triviais e possam se adaptar às mesmas. Assim, tal técnica evita que os indivíduos sejam mal avaliados e haja *overfitting*.

Já os hiperparâmetros considerados são listados na Tabela 1, onde a *População* indica a quantidade de circuitos gerados aleatoriamente antes da seleção e utilizados durante a mesma, a *Profundidade lógica* define o número máximo de nodos que podem ser encadeados nos circuitos gerados, enquanto os demais hiperparâmetros indicam a quantidade de gerações utilizada na seleção de indivíduos, o tamanho dos *mini-batches* e a frequência com que estes devem ser substituídos.

Ainda, a fim de minimizar a influência da estocasticidade nos resultados obtidos, para cada um dos *benchmarks* foram rodadas cinco evoluções com sementes diferentes para a geração de números aleatórios durante o processo.

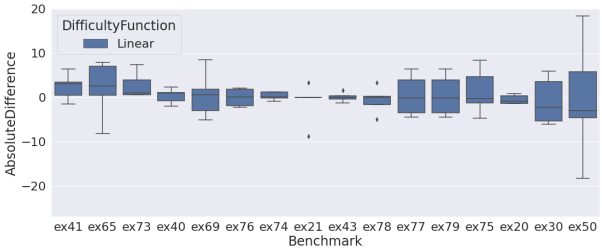
Tabela 1: Hiperparâmetros utilizados.

Hiperparâmetro	Valor
População	5
Profundidade lógica	250
Gerações	50.000
Batch size	64
Change each	250

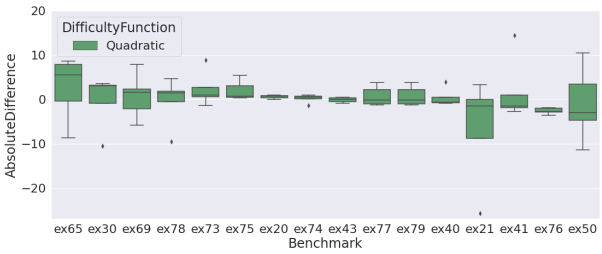
A Figura 2 apresenta as diferenças absolutas entre acurácias obtidas a partir das versões do fluxo com e sem CL, separadas por *benchmark*. Na Figura 2a, que considera uma função de dificuldade linear, o ex73 apresentou somente ganhos em acurácia e, como é possível observar através do ex50, ganhos de até aproximadamente 20% foram observados. Ainda assim, também foram observadas perdas de aproximadamente 20%.

Já a Figura 2b apresenta as diferenças absolutas tendo em conta uma função de dificuldade quadrática. Aqui, pode-se destacar o ex75, que somente teve ganhos, além dos limites de ganho e perda, de aproximadamente 15% e 25%, respectivamente.

XIV Computer on the Beach
30 de Março a 01 de Abril de 2023, Florianópolis, SC, Brasil



(a) Para uma função de dificuldade linear.



(b) Para uma função de dificuldade quadrática.

Figura 2: Diferenças absolutas com relação às acurácias em CGP-noCL.

4 CONSIDERAÇÕES FINAIS

Neste trabalho, apresentou-se resultados preliminares da aplicação de *Curriculum Learning* a um fluxo de otimização lógica baseado em CGP. A partir dos dados observados, pode-se concluir que a utilização de *Curriculum Learning* em conjunto com CGP tem a possibilidade de ser uma estratégia eficaz de aumento da acurácia dos circuitos gerados. Sendo assim, uma avaliação mais detalhada sobre a abordagem apresentada é não somente necessária, como também valorosa.

Para tanto pretende-se otimizar os hiperparâmetros utilizados, verificando quais valores produzem os melhores resultados, bem como avaliar quais características dos próprios *benchmarks* podem definir se a técnica proposta trará ou não maior efetividade. Pretende-se ainda explorar novas funções de dificuldade como a função raiz, por exemplo.

REFERÊNCIAS

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada) *(ICML '09)*. Association for Computing Machinery, New York, NY, USA, 41–48. https://doi.org/10.1145/1553374.1553380

[2] Augusto Berndt, Brunno A. de Abreu, Isac S. Campos, Bryan Lima, Mateus Grellert, Jonata T. Carvalho, and Cristina Meinhardt. 2021. Accuracy and Size Trade-off of a Cartesian Genetic Programming Flow for Logic Optimization. In *Proceedings of the 34th Symposium on Integrated Circuits and Systems Design* (Brazil) *(SBCCI '21)*. https://doi.org/10.1109/SBCCI53441.2021.9529968

[3] Augusto Berndt, Brunno A. de Abreu, Isac S. Campos, Bryan Lima, Mateus Grellert, Jonata T. Carvalho, and Cristina Meinhardt. 2022. A CGP-based Logic Flow: Optimizing Accuracy and Size. *Journal of Integrated Circuits and Systems (JICS)* (2022). https://doi.org/10.29292/jics.v17i1.546

[4] Nicola Milano and Stefano Nolfi. 2021. Automated curriculum learning for embodied agents a neuroevolutionary approach. *Scientific Reports* 11, 8985 (April 2021), 1–14. https://doi.org/10.1038/s41598-021-88464-5

[5] Nicola Milano, Paolo Pagliuca, and Stefano Nolfi. 2019. Robustness, evolvability and phenotypic complexity: insights from evolving digital circuits. *Evolutionary Intelligence* 12, 1 (2019), 83–95.

[6] Julian F. Miller. 2011. *Cartesian Genetic Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, 17–34. https://doi.org/10.1007/978-3-642-17310-3_2