Leonardo Muttoni

Desenvolvimento e Avaliação de um Framework Modular para Síntese Automática de Circuitos Analógicos: Aplicação do Algoritmo Recozimento Simulado com Evolução Geométrica de Circuitos

Uberlândia-MG

2024

Leonardo Muttoni

Desenvolvimento e Avaliação de um Framework Modular para Síntese Automática de Circuitos Analógicos: Aplicação do Algoritmo Recozimento Simulado com Evolução Geométrica de Circuitos

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciências.

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Elétrica – FEELT

Programa de Pós-Graduação em Engenharia Elétrica - PPGEELT

Orientador: Prof. Dr. Antônio Cláudio Paschoarelli Veiga

Uberlândia-MG

2024

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

M993 Muttoni, Leonardo, 1984-
2024 Desenvolvimento e Avaliação de um Framework Modular para Síntese Automática de Circuitos Analógicos: Aplicação do Algoritmo Recozimento Simulado com Evolução Geométrica de Circuitos [recurso eletrônico] / Leonardo Muttoni. - 2024.

Orientador: Antônio Cláudio Paschoarelli Veiga.
Tese (Doutorado) - Universidade Federal de Uberlândia, Pós-graduação em Engenharia Elétrica.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.te.2024.74
Inclui bibliografia.
Inclui ilustrações.

1. Engenharia elétrica. I. Veiga, Antônio Cláudio Paschoarelli,1963-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

Leonardo Muttoni

Desenvolvimento e Avaliação de um Framework Modular para Síntese Automática de Circuitos Analógicos: Aplicação do Algoritmo Recozimento Simulado com Evolução Geométrica de Circuitos

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciências.

Uberlândia-MG, 29 de janeiro de 2024

Banca examinadora:

Prof. Dr. Antônio Cláudio Paschoarelli Veiga – UFU (Orientador)

Prof. Dr. Gilberto Arantes Carrijo – UFU

Prof. Dr. Carlos Paula Lemos – IFTM

Prof. Dr. Daniel Moraes Santos - UFVJM

Prof. Dr. Eduardo Silva Vasconcelos- IF Goiano

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Engenharia Elétrica
Av. João Naves de Ávila, 2121, Bloco 3N - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4707 - www.posgrad.feelt.ufu.br - copel@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Engenharia Elétrica				
Defesa de:	Tese de Doutorado, 331, PPGEELT				
Data:	Vinte e nove de janeiro de dois mil e vinte e quatro	Hora de início:	09:30	Hora de encerramento:	12:30
Matrícula do Discente:	11813EEL005				
Nome do Discente:	Leonardo Muttoni				
Título do Trabalho:	Desenvolvimento e Avaliação de um Framework Modular para Síntese Automática de Circuitos Analógicos: Aplicação do Algoritmo Recozimento Simulado com Evolução Geométrica de Circuitos				
Área de concentração:	Processamento da Informação				
Linha de pesquisa:	Processamento Digital de Sinais				
Projeto de Pesquisa de vinculação:	Coordenador do projeto: Antonio C. P. Veiga. Título do projeto: Aplicações de machine learning para predição de sinais, estimação de parâmetros e reconhecimentos de padrões. Agência financiadora: não se aplica. Número do processo na agência financiadora: não se aplica. Vigência do projeto: 26/02/2018 – atual.				

Reuniu-se por meio de videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Engenharia Elétrica, assim composta:

Professores Doutores: Gilberto Arantes Carrijo (UFU), Carlos Paula Lemos (IFTM), Eduardo Silva Vasconcelos (IFG), Daniel Moraes Santos (UFVJM) e Antonio Cláudio Paschoarelli Veiga, orientador do discente.

Iniciando os trabalhos o presidente da mesa, Dr. Antonio Cláudio Paschoarelli Veiga, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao discente a palavra para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

APROVADO.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Doutor. O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme, foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Eduardo Silva Vasconcelos**, **Usuário Externo**, em 30/01/2024, às 12:07, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Gilberto Arantes Carrijo**, **Usuário Externo**, em 30/01/2024, às 15:01, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Carlos Paula Lemos**, **Usuário Externo**, em 30/01/2024, às 15:06, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Daniel Moraes Santos**, **Usuário Externo**, em 08/02/2024, às 11:15, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Antonio Claudio Paschoarelli Veiga**, **Professor(a) do Magistério Superior**, em 08/02/2024, às 11:44, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5143934** e o código CRC **6475FFE2**.

Referência: Processo nº 23117.001556/2024-97 SEI nº 5143934

Dedico este trabalho ao meu filho, Eduardo.

Agradecimentos

A Deus, pela minha vida, saúde e por me guiar onde quer que eu esteja.

Aos meus pais por sempre me apoiarem na busca de novos horizontes.

À minha esposa, Marília, por acreditar que esta árdua caminhada teria um fim.

Gostaria de expressar meus agradecimentos ao meu orientador, Prof. Paschoarelli, pelo apoio no desenvolvimento deste trabalho e pelas palavras motivadoras que me permitiram atravessar os momentos mais difíceis deste trabalho.

Dedico também agradecimentos à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica pela confiança no desenvolvimento deste trabalho.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro concedido.

Também agradeço ao programa *AWS Cloud Credit for Research* da *Amazon Web Services* que disponibilizou créditos para utilização de seus recursos de computação em nuvem, indispensáveis para a obtenção célere de muitos dados utilizados neste trabalho.

"Quando tudo parecer estar indo contra você, lembre-se que o avião decola contra o vento, e não a favor dele"

Henry Ford

Resumo

Este trabalho apresenta o desenvolvimento e a avaliação de um framework modular para síntese automática de circuitos eletrônicos analógicos intitulado `circ_autoproj`. O componente principal do framework é o SANN-GCE, uma meta-heurística orientada por simulações SPICE que gera automaticamente a topologia e o dimensionamento dos componentes de um circuito eletrônico a partir de um comportamento desejado definido pelo usuário. O SANN-GCE é composto pelo algoritmo de busca Recozimento Simulado (*Simulated Annealing - SANN*) e pela representação da solução denominada de Evolução Geométrica de Circuitos (*Geometric Circuit Evolution - GCE*). O GCE é um novo esquema de codificação que utiliza graus de liberdade categorizados que permitem que características distintas de um circuito sofram mutação, durante a evolução da solução, com diferentes probabilidades de acordo com sua categoria. Neste trabalho também são apresentados o simulador de circuitos Ngspice, utilizado pelo SANN-GCE para avaliar as soluções candidatas, e os conceitos da computação em nuvem e seu uso como ferramenta para acelerar a execução do `circ_autoproj`. O SANN-GCE foi testado em sete casos de uso: sensor de temperatura, função gaussiana, referência de tensão, função quadrática, raiz quadrada, função cúbica e raiz cúbica. Cada configuração foi executada 50 vezes e o desempenho nestes circuitos foi avaliado através de 12 métricas, comparadas com o algoritmo de referência ACID-MGE. Também foi feito um estudo do efeito dos ajustes nos parâmetros do algoritmo. Nele, 7 parâmetros selecionados foram alterados em 41 configurações. A significância estatística dos resultados destes ajustes foi avaliada por meio de um Teste de Permutação. Este estudo revelou que, dentre as 41 configurações avaliadas, 19 apresentaram resultados estatisticamente significantes abrangendo todos os casos de uso, com valor p médio de 0,00993, cerca de cinco vezes inferior ao nível de significância fixado habitualmente em 0,05. Os dados obtidos indicaram que o SANN-GCE apresentou um desempenho melhor e com menos variabilidade entre as execuções quando comparado ao ACID-MGE, pois foram obtidos resultados melhores em 11 das 12 métricas. Foi obtida uma mediana 1,52× melhor para a taxa de sucesso, 13,94× melhor para a aptidão média, 64,75× melhor para o desvio padrão da aptidão e 7,83× melhor para o desvio padrão da porcentagem de êxito. Além disso, a mediana do tempo de execução do SANN-GCE foi 15,9× menor quando comparado com o tempo de execução normalizado do ACID-MGE.

Palavras-chave: Síntese de circuitos analógicos, Projeto automático, Algoritmo evolutivo, Meta-heurística, Recozimento Simulado.

Abstract

This work presents the development and evaluation of a modular framework for automatic synthesis of analog electronic circuits entitled `circ_autoproj`. The main component of the framework is SANN-GCE, a metaheuristic driven by SPICE simulations that automatically generates the topology and dimensioning of the components of an electronic circuit based on a user-defined desired behavior. SANN-GCE is composed of the Simulated Annealing search algorithm (SANN) and the solution representation called Geometric Circuit Evolution (GCE). GCE is a new coding scheme that uses categorized degrees of freedom that allow distinct characteristics of a circuit to mutate, during the evolution of the solution, with different probabilities according to their category. This work also presents the Ngspice circuit simulator, used by SANN-GCE to evaluate candidate solutions, and the concepts of cloud computing and its use as a tool to accelerate the execution of `circ_autoproj`. SANN-GCE was tested in seven use cases: temperature sensor, Gaussian function, voltage reference, quadratic function, square root, cube function and cube root. Each configuration was run 50 times and performance on these circuits was evaluated using 12 metrics, compared to the ACID-MGE reference algorithm. A study was also carried out on the effect of adjustments to the algorithm parameters. In it, 7 selected parameters were changed in 41 distinct configurations. The statistical significance of the results of these adjustments was evaluated using a Permutation Test. This study revealed that, among the 41 configurations evaluated, 19 presented statistically significant results covering all use cases, with an average p value of 0.00993, approximately five times lower than the significance level usually set at 0.05. The data obtained indicated that SANN-GCE performed better and with less variability between runs when compared to ACID-MGE, as better results were obtained in 11 of the 12 metrics. The following best medians obtained were highlighted: 1.52× for success rate, 13.94× for average fitness, 64.75× for standard deviation of fitness and 7.83× for the standard deviation of the Hits percentage. Also, the median execution time of SANN-GCE was 15.9× shorter when compared to the normalized execution time of ACID-MGE.

Keywords: Analog circuit synthesis, Automatic design, Evolutionary algorithm, Metaheuristic, Simulated Annealing.

Lista de ilustrações

Lista de tabelas

Sumário

Capítulo 1

Introdução

1.1 Considerações Iniciais

Neste capítulo serão abordados e contextualizados os elementos fundamentais do tema proposto. Além disso, será realizada uma revisão bibliográfica sobre o assunto em questão. Serão destacados o problema de pesquisa, os objetivos, as contribuições e a motivação que embasou a escolha desse tema pelo autor. Por fim, será apresentada a estrutura deste trabalho, explicitando o encadeamento dos capítulos.

1.2 Contextualização do Tema

O uso de ferramentas de automação de projetos eletrônicos (em inglês *Electronic Design Automation - EDA*) se tornou indispensável na criação de sistemas eletrônicos modernos, pois automatizam algumas tarefas que são complexas e demoradas, reduzindo o tempo e custo de desenvolvimento. Seu uso tem contribuído para o avanço da indústria eletrônica, principalmente no desenvolvimento de dispositivos eletrônicos mais complexos, conferindo um maior desempenho e confiabilidade nos seus produtos. De maneira geral, em projetos de circuitos analógicos não integrados, as ferramentas EDA compreendem um conjunto de softwares que permitem, de forma integrada, a captura de esquemas, a simulação de circuitos e o desenho da placa de circuito impresso.

Nesse contexto, o fluxo de projeto se inicia com o software para a captura de esquemáticos, onde o projetista inclui os componentes e desenha as suas interligações, gerando o *esquema* (também chamado de *esquemático* ou *diagrama*) do circuito. Depois são feitas simulações de seu funcionamento para verificar se o desempenho do circuito atingiu os requisitos de projeto. Caso o resultado não seja satisfatório, o esquemático é alterado e novas simulações são executadas. Com o esquema pronto, há a etapa de layout, onde a posição física de cada componente e suas interligações são desenhadas em uma Placa de Circuito Impresso (PCI), obedecendo o esquemático. Neste ponto a etapa de

projeto se encerra, e a etapa de fabricação da PCI e posterior montagem dos componentes na mesma pode ser iniciada.

Projetos de circuitos digitais se beneficiam de métodos e ferramentas amplamente utilizadas que possibilitam a síntese automática do esquemático do circuito a partir de uma especificação prévia do comportamento desejado do mesmo. Porém, para circuitos analógicos, o problema da síntese automática é mais complexo, não havendo ferramentas amplamente empregadas neste fim (SORKHABI; ZHANG, 2017). Por isso, os diagramas de circuitos analógicos são normalmente desenvolvidos manualmente com base no conhecimento especializado. Neste caso, as ferramentas EDA são usadas para o desenho e a validação do diagrama, e normalmente são necessárias diversas iterações de simulações e alterações no esquemático até que um circuito com as características desejadas seja obtido.

Diante deste contexto, é de grande valia a pesquisa e desenvolvimento de algoritmos e ferramentas computacionais que possam auxiliar o projetista na criação de diagramas de circuitos analógicos a partir de uma prescrição de suas características pretendidas. Por isso, este tema é objeto de diversas pesquisas e desenvolvimentos experimentais de algoritmos, muitos deles baseados em meta-heurísticas.

O problema da síntese automática de circuitos eletrônicos analógicos pode ser dividido em duas partes:

1. Síntese de topologia: determina a quantidade e os tipos de componentes eletrônicos no circuito, bem como suas interconexões. É um problema combinatório muito difícil, pois o espaço de busca é enorme e uma simples mudança de uma única variável como a conexão de um terminal de componente pode alterar completamente a resposta do circuito.

2. Dimensionamento: envolve a determinação dos parâmetros numéricos (ou valores) de todos os componentes do circuito, por exemplo, o valor da resistência nos resistores. Geralmente são parâmetros contínuos dos componentes previamente dispostos em um circuito. Algumas ferramentas comerciais de EDA têm a capacidade de otimizar automaticamente o dimensionamento com base na meta de desempenho desejada especificada pelo usuário.

1.3 Revisão Bibliográfica

Esta seção visa examinar o panorama atual da síntese automática de circuitos eletrônicos, um problema intricado e diverso que tem despertado o interesse de pesquisadores globalmente. Partindo dos pioneiros que utilizavam abordagens fundamentadas no conhecimento humano até as técnicas contemporâneas de otimização por meio de

meta-heurísticas, esta análise visa traçar o desenvolvimento das metodologias e enfatizar os progressos notáveis, bem como os obstáculos que ainda persistem.

A geração automática de circuitos eletrônicos analógicos é foco de várias pesquisas, especialmente o problema de síntese topológica. De acordo com a extensa pesquisa feita por Sorkhabi e Zhang (2017), os primeiros trabalhos sobre este assunto usaram principalmente *seleção humana de topologia e baseada em conhecimento* (HARJANI; RUTENBAR; CARLEY, 1989; KOH; SEQUIN; GRAY, 1990). Trabalhos mais recentes foram categorizados como sendo do tipo *refinamento de topologia* ou *geração de topologia*. No primeiro, o método modifica um dado circuito inicial para melhorar seu desempenho (JIAO; DOBOLI, 2016). Já no segundo, uma topologia é gerada sem utilizar um circuito inicial (MEISSNER; HEDRICH, 2015).

As contribuições de Koza et al. (1999), Koza et al. (2000) são consideradas um marco por apresentarem uma técnica geral para abordar o problema de síntese de circuitos eletrônicos analógicos. Nestes trabalhos foi utilizada a Programação Genética (ou *Genetic Programming - GP*) para evoluir tanto a topologia quanto os valores dos componentes, criando um circuito completo a partir de uma declaração de alto nível do comportamento desejado do circuito.

Nos anos seguintes, várias técnicas surgiram neste assunto, dentre as quais citamos o trabalho de Mattiussi e Floreano (2007), que mostrou uma codificação inspirada em *redes regulatórias genéticas* biológicas que podem ser usadas para evoluir circuitos analógicos. Foi testado em três problemas de circuitos analógicos, usando um algoritmo genético como método de busca.

McConaghy et al. (2011) apresentaram um sistema que usa GP para criar circuitos analógicos usando um grande banco de dados de *blocos de construção confiáveis* preparado previamente por um especialista humano para resolver um problema específico.

Castejón e Carmona (2018) utilizaram Evolução Gramatical (*Grammatical Evolution - GE*) para expressar uma solução candidata por meio de uma definição gramatical utilizando o Formalismo de Backus-Naur (em inglês *Backus-Naur Form - BNF*) (O'NEILL; RYAN, 2001). As soluções são então evoluídas usando um algoritmo genético. Este trabalho se destaca por apresentar de forma detalhada e reproduzível a solução proposta, e por testá-la em sete problemas clássicos (casos de uso) com 50 rodadas de execução cada, permitindo assim uma comparação estatística mais fidedigna com outras técnicas. Este algoritmo foi posteriormente referido pelos autores como ACID-GE (do inglês *analog circuit design based on grammatical evolution*).

Rojec, Burmen e Fajfar (2019) propuseram uma técnica que utiliza uma matriz de conexão para representar a topologia e otimizá-la juntamente com os valores dos componentes por meio de um algoritmo evolutivo, com operações de cruzamento e mutação.

A solução proposta foi demonstrada utilizando três circuitos de teste.

Castejón e Carmona (2020) apresentaram a Evolução Multigramatical (*Multi Grammatical Evolution - MGE*), conseguindo melhores resultados em comparação com o trabalho publicado em 2018 pelos mesmos autores. Neste trabalho, os autores dividiram o problema original em dois subproblemas distintos: dimensionamento de componentes e topologia do circuito. O algoritmo foi testado novamente nos sete casos de uso utilizados em seu artigo anterior, cada um com 50 rodadas. Este método foi chamado pelos referidos autores de ACID-MGE (do inglês *analog circuit design based on multi-grammatical evolution*).

Através desta revisão bibliográfica, é possível observar que o campo de síntese automática de circuitos eletrônicos analógicos passou por um notável desenvolvimento ao longo dos anos. Desde os métodos que utilizavam blocos de construção confiáveis até as estratégias contemporâneas que fazem uso de algoritmos avançados otimização e geração de topologia, os avanços são notáveis. A Programação Genética, a Evolução Gramatical e a Evolução Multigramatical são apenas algumas das abordagens inovadoras que demonstraram potencial para transformar a forma como os circuitos analógicos são projetados.

1.4 Objetivos e Contribuições

O presente trabalho tem como objetivo geral desenvolver e avaliar um framework modular utilizando o algoritmo de Recozimento Simulado em conjunto com a Evolução Geométrica de Circuitos para a síntese automática de circuitos eletrônicos analógicos.

A Evolução Geométrica de Circuitos (chamada a partir daqui de GCE, sigla em inglês[1] para *Geometric Circuit Evolution*) é um esquema de codificação que representa soluções candidatas para o problema de síntese de circuitos analógicos. O GCE introduz o conceito de *Grau de Liberdade* (ou *Degrees of Freedom - DOF*) categorizados que permitem que diferentes características de um circuito sofram mutações com diferentes probabilidades ao longo da construção da solução, conforme a categoria designada para o DOF.

O GCE foi implementado em conjunto com o algoritmo clássico de busca *Recozimento Simulado*, que conduz o processo de otimização através da mutação dos DOFs do GCE. A composição do GCE com o recozimento simulado é referida neste trabalho a partir daqui como *SANN-GCE*[2].

O SANN-GCE foi construído sobre um *framework* modular chamado de

[1] Optou-se pelo uso de alguns termos em inglês neste trabalho para uniformizar com outro trabalho publicado pelo autor nesta mesma língua (MUTTONI; VEIGA, 2023)

[2] SANN é o acrônimo escolhido pelo autor para o termo em inglês *Simulated Annealing* (recozimento simulado). O acrônimo SA também poderia ser utilizado para tal, mas o algoritmo de busca (*Search Algorithm*) já havia sido batizado com esta sigla

`circ_autoproj`[3], que carrega o algoritmo e suas partes em tempo de execução através de *plugins*. Isto permite uma experimentação de novos algoritmos de maneira mais rápida e fácil durante o desenvolvimento.

A única entrada do SANN-GCE é uma formulação que especifica o comportamento desejado do circuito, e a saída é o melhor circuito encontrado, projetado no nível de seus componentes discretos, representado por meio de uma lista de componentes com seus atributos e conexões, chamada de *netlist*. Durante a execução do algoritmo, a aptidão das soluções candidatas (circuitos eletrônicos) são obtidas por meio de simulações SPICE.

Assim como outros algoritmos que não limitam o espaço de busca, os circuitos projetados pelo SANN-GCE não possuem restrições. Isto possibilita a criação de projetos incomuns que podem, por acaso, ser inéditos. Isso contrasta com outras técnicas que usam informações de especialistas, como a *seleção humana de topologia e baseada em conhecimento* (HARJANI; RUTENBAR; CARLEY, 1989; KOH; SEQUIN; GRAY, 1990), o *refinamento de topologia* (JIAO; DOBOLI, 2016) e o uso de *banco de dados de blocos de construção confiáveis* (MCCONAGHY et al., 2011). Estas técnicas naturalmente incorrem em restrições nas possibilidades de circuitos que podem ser gerados.

Para se atingir o objetivo geral deste trabalho, são elencados os seguintes objetivos específicos:

- Desenvolver e implementar o framework `circ_autoproj`, que suporte a integração do algoritmo Recozimento Simulado e a representação GCE, permitindo a fácil adaptação e expansão para diferentes tipos de problemas de síntese de circuitos;
- Desenvolver a representação de solução GCE e aplicá-la em conjunto com o algoritmo de busca Recozimento Simulado, obtendo assim o SANN-GCE, uma metodologia robusta para a evolução e otimização de circuitos eletrônicos analógicos;
- Integrar o simulador de circuitos Ngspice no `circ_autoproj` para a avaliar a aptidão das soluções candidatas geradas pelo SANN-GCE;
- Executar o `circ_autoproj` na *nuvem* para aproveitar os recursos de escalabilidade e desempenho computacional, acelerando a síntese de circuitos eletrônicos;
- Validar o framework e o algoritmo SANN-GCE por meio de casos de uso representativos, aplicando diversas métricas de desempenho e comparando os resultados com os algoritmos de referência ACID-GE e ACID-MGE para demonstrar a eficácia do método proposto.

O framework e seus algoritmos desenvolvidos devem atender aos seguintes requisitos:

[3] Origem do nome do *framework*: do inglês, de forma não rigorosa, *circuit automatic project*

- Projetar automaticamente circuitos eletrônicos analógicos a partir de uma descrição de alto-nível do desempenho esperado do circuito, com desempenho competitivo ao *estado da arte*;
- O framework deve ser modular, com possibilidade de se alterar o algoritmo, configurações do algoritmo ou o problema sem precisar ajustar o código-fonte e recompilar;
- Utilizar recursos de programação que confiram alto desempenho e fácil manutenção do software, por exemplo, processamento paralelo, linguagem compilada e orientada a objetos.

1.5 Justificativa

O estudo da síntese automática de circuitos eletrônicos analógicos se justifica por diversas razões científicas e tecnológicas. Em primeiro lugar, a automação do projeto de circuitos enfrenta desafios únicos, como a necessidade de integrar de forma eficiente a topologia e o dimensionamento dos componentes, o que requer abordagens inovadoras. Além disso, o desenvolvimento de métodos automáticos abre caminho para a exploração de soluções que podem potencializar a inovação e a descoberta de novos circuitos.

O framework `circ_autoproj` que será descrito neste trabalho apresenta uma abordagem modular e, em conjunto com o algoritmo SANN-GCE, possibilita sínteses mais rápidas e precisas de circuitos eletrônicos analógicos. É esperado que este trabalho possa contribuir para o avanço do conhecimento no campo da Engenharia Elétrica, fornecendo inspiração para futuras pesquisas e aplicações práticas nesta área que apresenta crescente demanda por eficiência e inovação.

1.6 Motivação

A motivação para o desenvolvimento deste trabalho ocorreu em razão dos seguintes fatores:

Desafio : O projeto completo de um circuito eletrônico analógico (incluindo a sua topologia e o dimensionamento de seus componentes) a partir da descrição de seu desempenho desejado, é um problema de elevada complexidade. A construção de um software (e seus algoritmos associados) que execute esta tarefa com desempenho competitivo aos algoritmos considerados *estados da arte*, foi assimilada pelo autor como um desafio motivador.

Aplicabilidade em outros problemas de engenharia : O software desenvolvido neste trabalho pode ser futuramente adaptado para outras tarefas como, por exemplo, o projeto

de antenas, layout de circuitos integrados e projetos de códigos corretores de erros. Isto é possível devido à sua capacidade de otimização por meio de meta-heurísticas e sua modularidade.

Confluência de áreas de interesse : O projeto de circuitos eletrônicos analógicos, suas simulações através do SPICE e o desenvolvimento de software para engenharia são alguns temas de interesse do autor.

1.7 Estrutura da Tese

Os capítulos seguintes deste trabalho estão organizados da seguinte forma: Primeiro, o capítulo 2 apresenta uma visão geral da estrutura do framework desenvolvido, chamado de `circ_autoproj`. Em seguida, o capítulo 3 introduz o simulador de circuitos Ngspice, sua aplicação no `circ_autoproj`, os principais tipos de simulações, a sintaxe da netlist, os modelos de componentes e os seus principais comandos. Depois, o capítulo 4 apresenta o algoritmo SANN-GCE, considerado o núcleo deste trabalho, e suas duas partes: o algoritmo de busca Recozimento Simulado e a representação de solução GCE. Na sequência, o capítulo 5 expõe uma breve introdução dos conceitos da computação em nuvem e seu uso como ferramenta para acelerar a execução do `circ_autoproj`. Depois, o capítulo 6 apresenta os sete casos de uso selecionados para testar o algoritmo SANN-GCE, as métricas utilizadas para mensurar o desempenho dos circuitos e a metodologia utilizada para a seleção dos parâmetros do algoritmo. Em seguida, o capítulo 7 exibe e discute os resultados obtidos pelo SANN-GCE utilizando parâmetros iniciais e ajustados e os compara com o algoritmo ACID-MGE. Por fim, o capítulo 8 relata as considerações finais deste trabalho.

Há também cinco apêndices, que contêm informações complementares referenciadas em alguns dos capítulos deste trabalho. O apêndice A apresenta um exemplo de um arquivo de configuração do `circ_autoproj`. Já o apêndice B exibe um exemplo de um arquivo de resultado do `circ_autoproj`. O apêndice C mostra os modelos SPICE dos componentes eletrônicos utilizados nas simulações. O apêndice D apresenta os melhores circuitos encontrados em cada caso de uso, na forma de netlists. O apêndice E, por sua vez, contém tutoriais para a criação de máquinas virtuais na nuvem. Este apêndice é um complemento do capítulo 5.

Capítulo 2

Framework gerador de circuitos eletrônicos analógicos

2.1 Introdução

Este capítulo é dedicado a uma exploração detalhada do framework gerador de circuitos eletrônicos analógicos, conhecido como `circ_autoproj`. Será apresentada uma visão abrangente da arquitetura e das características de cada componente desse software, destacando como ele, juntamente com seus plugins, realiza a síntese de circuitos eletrônicos.

No início do capítulo, são apresentadas as justificativas para o desenvolvimento deste framework e seus principais fundamentos. Depois é apresentada uma introdução ao *plugin SR*, que codifica as soluções candidatas. Em seguida, é abordado o *problema SPICE*, um componente que contém as especificações do circuito a ser projetado automaticamente, incluindo as diretivas para a execução das simulações SPICE e as fórmulas para o cálculo da função objetivo. Na sequência, é apresentada a integração do simulador Ngspice com o framework, necessário no contexto do problema SPICE. Depois é apresentado o *plugin SA*, responsável pela condução da otimização.

Ainda neste capítulo, a *execução* do software é apresentada, explicando o seu funcionamento e os processos internos do `circ_autoproj`. Depois, são apresentados o formato e a estrutura dos arquivos de entrada e de saída do framework. Por fim, é apresentado o conceito de *rodadas de execução* e como ele é utilizado para combinar diversas execuções em um único arquivo de saída.

2.2 Justificativa e Fundamentos

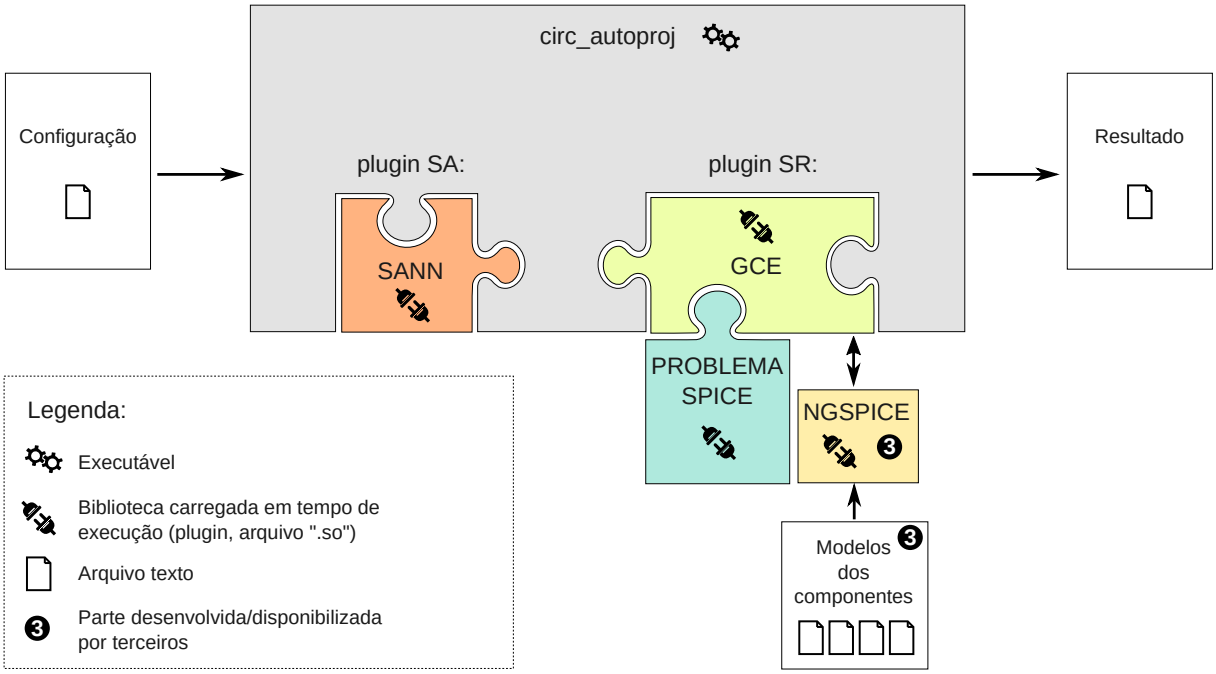
Ao longo do desenvolvimento deste trabalho – mais especificamente durante a implementação computacional de diversas meta-heurísticas que culminaram no desenvolvimento do algoritmo SANN-GCE – foi observada uma redundância nos códigos-fonte dos diferentes

algoritmos desenvolvidos. Por exemplo, todos continham chamadas de função para: 1. criar uma solução candidata aleatória; 2. obter a sua aptidão; 3. induzir uma mutação aleatória. Esta repetição de código-fonte causa problemas de manutenção, tornando *bugs* mais difíceis de se detectar e corrigir, além de tornar a implementação de melhorias no programa menos ágeis.

Estes problemas de redundância de código-fonte indicaram a necessidade de uma reorganização do software desenvolvido, no sentido de se dissociar a parte comum em uma estrutura de suporte genérica para os algoritmos meta-heurísticos. Esta estrutura (ou *framework*) age como um esqueleto, impondo uma interface padronizada para os algoritmos, e permite que eles sejam carregados em tempo de execução como *plugins*. Isto provoca uma separação clara entre os papéis de cada componente, permitindo a troca, experimentação e comparação de diferentes algoritmos de maneira rápida e fácil, além de tornar a manutenção do software menos dispendiosa.

O framework desenvolvido foi chamado de `circ_autoproj` e foi escrito em C++ utilizando o paradigma de orientação a objetos. Nele, o algoritmo meta-heurístico a ser executado é dividido em duas partes principais (plugins): 1. Algoritmo de Busca (ou *Search Algorithm - SA*); 2. Representação da Solução (ou *Solution Representation - SR*). Esta divisão permitiu, durante o desenvolvimento deste trabalho, uma experimentação ágil de diferentes combinações de algoritmos e representações. Tanto o SA quanto o SR são carregados em tempo de execução (não é necessário recompilar todo o software a cada alteração de plugin), de acordo com a configuração fornecida pelo usuário. A Figura 1 apresenta a arquitetura do `circ_autoproj`, indicando seus principais componentes e suas relações, que são detalhadas nas seções a seguir.

Figura 1 – Framework desenvolvido e seus componentes



Fonte: o autor (2023)

2.3 Plugin SR

A atribuição principal de um plugin SR é a de construir objetos que representam *soluções candidatas*. Isto é feito utilizando o padrão de projeto *Fábrica* (VALENTE, 2020, cap. 6): o `circ_autoproj` chama uma função do SR que retorna um *objeto fábrica*, o qual possui um método que, quando invocado, cria e retorna objetos *soluções candidatas*.

Cada *objeto solução candidata* possui métodos para modificar a solução, como os operadores `cruza()` e `muta()`. Há também o método `aptidão()`, utilizado para obter a aptidão da solução, mas este é delegado para o *problema SPICE* (ou simplesmente *problema*): um plugin subordinado ao SR que também é carregado em tempo de execução. Esta separação adicional das funcionalidades do SR em outro plugin proporciona facilidades na troca do problema, pois não há a necessidade de se recompilar o programa quando se deseja testar o mesmo algoritmo com outro problema.

2.4 Problema SPICE

O problema contém todas as especificações pretendidas para o circuito eletrônico a ser projetado automaticamente. Estas características desejadas são codificadas em uma função `P()`, cuja entrada é o resultado de uma simulação SPICE (que contém tensões e correntes do circuito) e a saída é a aptidão calculada em função de quão próximas as correntes e/ou tensões de determinados nós do circuito estão de uma curva ideal.

O problema também fornece ao SR os comandos SPICE (que determinam o tipo da análise a ser executada, por exemplo), a netlist do dispositivo de teste e as coordenadas da curva de resposta ideal do circuito. A seção 3.8 apresenta em detalhes a utilização do *problema SPICE* no cálculo da aptidão de uma solução candidata.

2.5 Ngspice e modelos dos componentes

Para calcular a aptidão de uma solução candidata é necessário primeiro simular o circuito por ela representada. Para este fim, o plugin SR se utiliza dos serviços da biblioteca do Ngspice (normalmente disponibilizada através do arquivo `libngspice.so` em sistemas Linux). O caminho para esta biblioteca é especificado no campo `path_libngspice` do arquivo de configuração do `circ_autoproj`. Ela é carregada em tempo de execução e os ponteiros de algumas de suas funções (VOGT et al., 2022, cap. 19) são armazenados para uso posterior pelo SR. São elas:

- `ngSpice_Init()`: Função para Inicializar o simulador SPICE;
- `ngSpice_Command()`: Executa um comando;
- `ngSpice_Circ()`: Envia linhas de um circuito, em forma de netlist, para o simulador;
- `ngSpice_running()`: Verifica se a simulação está sendo executada.

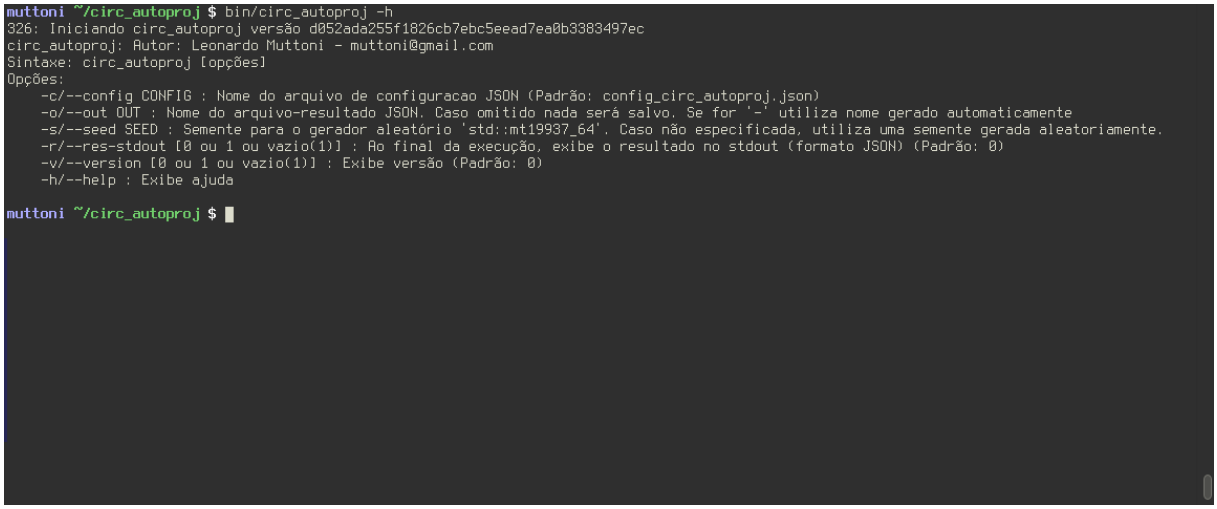
Os comandos SPICE definidos no *problema SPICE* normalmente solicitam o carregamento, via diretiva `.include`, de arquivos de modelos de componentes eletrônicos para serem utilizados nas simulações. Estes modelos especificam, para transistores bipolares, por exemplo, parâmetros como o ganho de corrente direto e a resistência de emissor (VOGT et al., 2022, seção 8.2.1). O apêndice C apresenta a listagem de todos os modelos SPICE dos componentes eletrônicos utilizados neste trabalho.

2.6 Plugin SA

O plugin SA é responsável por conduzir a otimização do problema. É normalmente um algoritmo meta-heurístico que faz uso do *objeto fábrica* retornado pelo SR para criar soluções candidatas. O SA pode armazenar os *objetos soluções* em um vetor e invocar seus métodos `cruza()`, `muta()`, `clona()` e `aptidão()`, quando aplicável. Este último método é delegado pelo SR para o plugin *problema*, conforme detalhado na seção 2.3.

A SA disponibiliza para o `circ_autoproj` alguns métodos. Os principais são: `avanca_geracao()` e `obtem_melhor_solucao()`. O primeiro executa um passo do algoritmo SA, manipulando os *objetos soluções* em busca de uma melhor solução. O segundo retorna o melhor *objetos solução* encontrado até o momento.

Figura 2 – Opções de linha de comando do `circ_autoproj`



Fonte: o autor (2023)

2.7 Execução

O framework `circ_autoproj` é o único arquivo executável de todo o conjunto de software aqui descrito. É um programa que possui interface com o usuário do tipo *linha de comando*, sem interface gráfica, portanto a interação com o usuário se dá através do seu arquivo de entrada (arquivo de *configuração*), do arquivo de saída (*arquivo resultado*) e da saída padrão do console, onde são exibidas diversas informações sobre o progresso da otimização. A Figura 2 apresenta uma captura de tela do programa sendo executado em um terminal, exibindo suas opções de linha de comando após ser invocado com o argumento `-h` (*help*).

Na fase de inicialização, o `circ_autoproj` processa os argumentos passados via linha de comando, depois lê o arquivo de configuração, gera uma semente aleatória, carrega o plugin SR (que por sua vez carrega o plugin *problema*), carrega o plugin SA, passando a referência do SR para ele e, enfim, cria o arquivo de resultados.

Depois da inicialização, ocorre a fase de execução, onde há um laço que, a cada iteração, invoca o método `avanca_geracao()` do plugin SA e o framework registra os resultados no arquivo de saída. Quando o critério de parada é alcançado, o laço se encerra e o framework entra na fase de encerramento, onde a melhor solução encontrada é salva no arquivo de saída em conjunto com diversas outras informações, com o número de rodadas, data e hora e o tempo gasto.

A Figura 3 apresenta a captura de tela do início da execução do `circ_autoproj`. Nela é possível observar as mensagens da saída padrão do console informando o carregamento dos plugins (SA, SR e problema) e da `libngspice.so` pelo objeto `SOLoader`. Já a Figura 4 apresenta a saída de console durante a fase de execução. Ali são exibidas

Figura 3 – Tela do `circ_autoproj`: início da execução

```
muttoni ~/circ_autoproj $ bin/circ_autoproj -c config-SANN-GCE-Temperatura-MIN-03.json -o out.json
27453: Iniciando circ_autoproj versão d052ada255f1826cb7ebc5eead7ea0b3383497ec
27453: Engine aleatório: std::mt19937_64
27453: Utilizando semente aleatória gerada: 385680130
27453: Abrindo arquivo de configuração 'config-SANN-GCE-Temperatura-MIN-03.json'
SOLoader: Carregando 'plugins/sr/libplugin_sr-spice-gce.so'
SOLoader.get_symbol(): Obtendo simbolo 'plugin_sr_info'
SOLoader: Carregando 'plugins/problemas-spice/libplugin_problema-spice-temperatura.so'
SOLoader.get_symbol(): Obtendo simbolo 'problema_spice'
SOLoader: Carregando 'libngspice.so'
SOLoader.get_symbol(): Obtendo simbolo 'ngSpice_Init'
SOLoader.get_symbol(): Obtendo simbolo 'ngSpice_Command'
SOLoader.get_symbol(): Obtendo simbolo 'ngSpice_Circ'
SOLoader.get_symbol(): Obtendo simbolo 'ngSpice_running'
Versão do ngspice:
stdout ******
stdout ** ngspice-40 : Circuit level simulation program
stdout ** The U. C. Berkeley CAD Group
stdout ** Copyright 1985-1994, Regents of the University of California.
stdout ** Copyright 2001-2023, The ngspice team.
stdout ** Please get your ngspice manual from https://ngspice.sourceforge.io/docs.html
stdout ** Please file your bug-reports at http://ngspice.sourceforge.net/bugrep.html
stdout ** Creation Date: Fri Apr 14 19:10:16 UTC 2023
stdout **
stdout ** CIDER 1.b1 (CODECS simulator) included
stdout ** XSPICE extensions included
stdout ** Relevant compilation options (refer to user's manual):
stdout ** OpenMP multithreading for BSIM3, BSIM4 enabled
stdout ** X11 interface not compiled into ngspice
stdout **
stdout ** ngspice shared library.
stdout ******

SOLoader: Carregando 'plugins/sa/libplugin_sa-sann.so'
SOLoader.get_symbol(): Obtendo simbolo 'plugin_sa_info'
SAnn.SAnn(): Instanciando 2 individuos 'GCE-Temperatura'
SAnn.SAnn(): objetivo=MIN
Algoritmo de busca (SA): SAnn
Representação da solução (SR): GCE-Temperatura
27453: Abrindo arquivo 'out.json' para salvar os resultados
```

Fonte: o autor (2023)

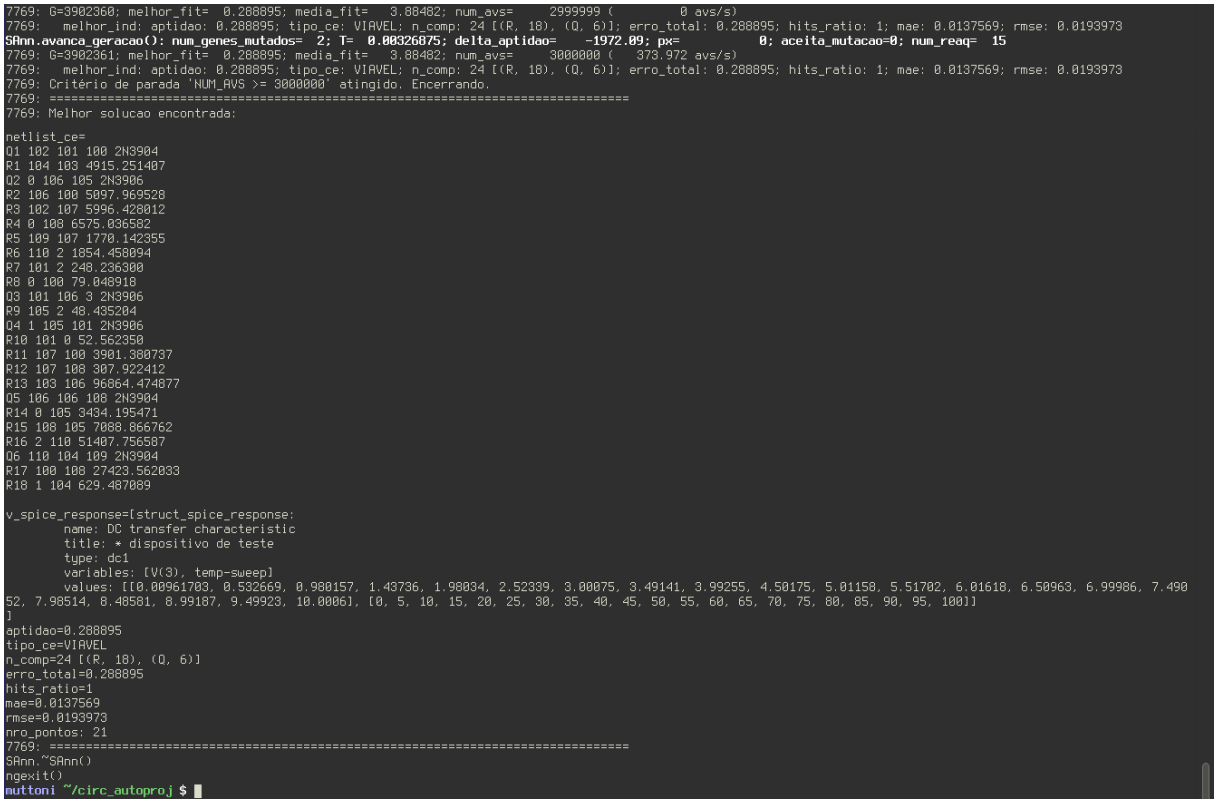
Figura 4 – Tela do `circ_autoproj`: meio da execução

```
SAnn.avanca_geracao(): num_genes_mutados=  2; T=     84.3152; delta_aptidao=    -260.493; px=   0.0455239; aceita_mutacao=0; num_reaq=   3
7769: G=761523; melhor_fit=  0.555571; media_fit=   294.048; num_avs=      622264 (   373.274 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  3; T=     84.3067; delta_aptidao=    -272.693; px=   0.0393783; aceita_mutacao=0; num_reaq=   3
7769: G=761524; melhor_fit=  0.555571; media_fit=   294.048; num_avs=      622265 (   309.023 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  1; T=     84.2983; delta_aptidao=    -4.35822; px=    0.949614; aceita_mutacao=1; num_reaq=   3
7769: G=761525; melhor_fit=  0.555571; media_fit=   296.227; num_avs=      622266 (   315.557 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  1; T=     84.2899; delta_aptidao=    -8.54779; px=    0.903563; aceita_mutacao=1; num_reaq=   3
7769: G=761526; melhor_fit=  0.555571; media_fit=   300.501; num_avs=      622267 (   283.206 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  3; T=     84.2814; delta_aptidao=    -20.8007; px=    0.781296; aceita_mutacao=1; num_reaq=   3
7769: G=761527; melhor_fit=  0.555571; media_fit=   310.902; num_avs=      622268 (   282.725 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  2; T=      84.273; delta_aptidao=  -0.0933228; px=    0.998893; aceita_mutacao=1; num_reaq=   3
7769: G=761528; melhor_fit=  0.555571; media_fit=   310.948; num_avs=      622269 (   239.006 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  1; T=     84.2646; delta_aptidao=     1.00745; px=         nan; aceita_mutacao=1; num_reaq=   3
7769: G=761529; melhor_fit=  0.555571; media_fit=   310.445; num_avs=      622270 (   345.185 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  5; T=     84.2562; delta_aptidao=      6.6734; px=         nan; aceita_mutacao=1; num_reaq=   3
7769: G=761530; melhor_fit=  0.555571; media_fit=   307.108; num_avs=      622271 (   256.213 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  2; T=     84.2477; delta_aptidao=     30.2994; px=         nan; aceita_mutacao=1; num_reaq=   3
7769: G=761531; melhor_fit=  0.555571; media_fit=   291.958; num_avs=      622272 (   235.294 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
SAnn.avanca_geracao(): num_genes_mutados=  2; T=     84.2393; delta_aptidao=    0.0297241; px=         nan; aceita_mutacao=1; num_reaq=   3
7769: G=761532; melhor_fit=  0.555571; media_fit=   291.943; num_avs=      622273 (   279.018 avs/s)
7769:   melhor_ind: aptidao: 0.555571; tipo_ce: VIAVEL; n_comp: 33 [(R, 24), (Q, 10)]; erro_total: 0.555571; hits_ratio: 1; mae: 0.0264558; rmse: 0.0322284
```

Fonte: o autor (2023)

mensagens sobre o andamento da otimização, por exemplo, a melhor aptidão encontrada até o momento (variável `melhor_fit`), o número de gerações já passadas e a quantidade de avaliações já executadas (`G` e `num_avs`, respectivamente). Há também informações sobre o número de componentes do melhor circuito gerado (`n_comp`), desmembrado por tipo de componentes (Resistor (R) e Transistor Bipolar (Q), no caso específico do problema da figura) e as métricas secundárias, como `hits_ratio`, `mae` e `rmse`. O cálculo destas métricas é apresentado em detalhes no capítulo 6.

Figura 5 – Tela do `circ_autoproj`: final da execução



Fonte: o autor (2023)

A Figura 5 mostra o encerramento da execução do `circ_autoproj`. Nela é possível observar a mensagem de que o critério de parada foi atingido (neste caso o número de avaliações da função de aptidão se tornou maior ou igual a 3.000.000), e também a listagem da netlist da melhor solução encontrada (`netlist_ce`). Também são exibidas a aptidão, as métricas secundárias e a curva de resposta SPICE (`v_spice_response`).

2.8 Arquivo de entrada

O arquivo de entrada (configuração) possui o formato *JavaScript Object Notation - JSON*. Este formato, conforme descrito na RFC 8259, é:

> (...) um formato leve de intercâmbio de dados, baseado em texto e independente de linguagem. É derivado do Padrão da Linguagem de Programação ECMAScript. JSON define um pequeno conjunto de regras de formatação para a representação portátil de dados estruturados. (BRAY, 2017, tradução nossa)

Este formato foi escolhido em razão da sua sintaxe simples, rica e compacta, sendo adequado tanto para edição manual quanto automatizada. A natureza baseada em texto do JSON torna-o acessível em qualquer editor de textos e sua estrutura flexível permite a representação clara e organizada de dados complexos.

São utilizados os seguintes campos[1] para o arquivo de entrada:

- `registrar_props_melhor_ind`: Vetor contendo os nomes das propriedades (métricas) das melhores soluções encontradas que devem ser registradas no arquivo de saída periodicamente;
- `sr`: Caminho para o plugin SR (arquivo de extensão .so);
- `params_sr`: Dicionario de parâmetros a ser repassado para o plugin SR na sua inicialização;
- `sa`: Caminho para o plugin SA (arquivo de extensão .so);
- `params_sa`: Dicionario de parâmetros a ser repassado para o plugin SA na sua inicialização;
- `criterio_parada`: Critério de parada da otimização;
- `criterio_registro`: Critério de registro no arquivo de saída. Normalmente indica a periodicidade de registro dos resultados.

O apêndice A desse trabalho (pág. 116) apresenta a listagem completa de um arquivo de configuração do `circ_autoproj`, a título de exemplo. Nele é possível observar como estes campos são utilizados. Os parâmetros `params_sr` e `params_sa` são detalhados nas seções 4.1 e 4.2.

2.9 Arquivo de saída

O arquivo de saída (resultado), assim com o arquivo de entrada, possui o formato JSON. Isso permite que o mesmo seja facilmente aberto e analisado por outras ferramentas de software ou manualmente, se necessário. O apêndice B apresenta um exemplo de um arquivo de resultado do `circ_autoproj`. Os campos do arquivo de saída são:

1. `appconfig`: Configuração utilizada. É uma cópia do arquivo de configuração fornecido;
2. `file_format`: Formato do arquivo, normalmente circ_autoproj_cpp;
3. `file_version`: Versão do arquivo;
4. `host_info`: Informações da máquina que executou o `circ_autoproj`. Contém os seguintes campos:

[1] No formato JSON, cada campo é um par nome/valor, normalmente associado a estrutura de dados *dicionário*

   a) `argv`: Argumentos do programa;
   b) `cpu_info`: Informações da CPU;
   c) `hostname`: Nome do host;
   d) `path_exe`: Caminho do executável utilizado;
   e) `username`: Nome de usuário;
   f) `version`: Versão do circ_autoproj;

5. `rnd`: Informações sobre a semente aleatória utilizada;
6. `run_name`: Nome da configuração atual, depende dos plugins carregados (SA, SR e problema);
7. `sa`: Informações do plugin SA ao final da execução, como seu nome e variáveis de status;
8. `sr`: Informações do plugin SR ao final da execução. Contém os seguintes campos:
   a) `netlist_comandos`: Parte da netlist que contém os comandos de simulação;
   b) `netlist_dt`: Parte da netlist que especifica o Dispositivo de Teste (DT);
   c) `ngspice_version_banner`: Mensagens de inicialização do Ngspice, contendo sua versão;
   d) `nome`: Nome do plugin SR carregado;
   e) `num_avs`: Quantidade total de avaliações executadas (chamadas da função aptidão);
   f) `referencia`: Curva de Referência do problema que foi carregado pelo SR;
9. `rodada`: Número de rodadas executadas;
10. `vs_rodadas`: Vetor contendo o resultado das rodadas. Cada rodada contém os seguintes campos:
    a) `datahora_final_rodada`: Timestamp do final da execução;
    b) `tempo_gasto_segundos`: Tempo total de execução, em segundos;
    c) `eval`: Quantidade total de avaliações executadas (chamadas da função aptidão);
    d) `geracao`: Número de iterações executadas do SA;
    e) `melhor_ind`: Informações sobre a melhor solução encontrada. Contém os seguintes campos:
       i. `aptidao`: Aptidão;

ii. `cromossomo`: Representação completa da solução em formato de texto, incluindo o conteúdo de todas as estruturas de dados utilizadas para codificar a solução;

iii. `n_comp`: Número de componentes;

iv. `netlist_ce`: Netlist do Circuito Evolutivo (CE);

v. `spice_response`: Resposta SPICE;

vi. `tempo_simulacao_spice_us`: Tempo gasto pela simulação no Ngspice, em microssegundos;

vii. `tipo_ce`: Classificação do circuito evolutivo, normalmente VIÁVEL;

viii. `to_string_resumo`: Texto-resumo da melhor solução encontrada, e suas propriedades;

f) `v_evals`: Vetor contendo o número total de avaliações executadas em cada instante;

g) `v_geracoes`: Vetor contendo o número total de iterações executadas do SA em cada instante;

h) `v_media_fit`: Vetor contendo a aptidão média das soluções encontradas em cada instante;

i) `v_melhor_fit`: Vetor contendo a aptidão da melhor solução encontrada em cada instante;

j) `v_melhor_props`: Objeto contendo vetores, cada um armazena as propriedades (métricas) solicitadas no parâmetro `registrar_props_melhor_ind` do arquivo de configuração.

2.10 Rodadas de execução

Cada execução concluída com sucesso do `circ_autoproj` dá origem a uma única *rodada* de resultados. É possível combinar vários arquivos de saída (de diferentes rodadas, executadas de forma independente) em um único arquivo de saída de mesmo formato. Isto facilita a análise posterior dos resultados, permitindo, por exemplo, um processamento mais fácil para cálculos estatísticos.

Para mesclar os resultados, foi desenvolvida uma ferramenta auxiliar chamada de `merge_circ_autoproj_cpp_out` que concatena os campos `vs_rodadas` de cada arquivo resultado em diferentes posições de um vetor único, gerando um arquivo resultado unificado. Isto só é permitido se todos os campos (com exceção do `vs_rodadas`, `host_info`, `rnd.seed` e `sr.num_avs`) dos arquivos-resultado a serem agrupados forem idênticos. Isto é feito para se evitar equívocos como o de se agrupar resultados de problemas SPICE diferentes, ou mesmo configurações diferentes para problemas SPICE iguais.

2.11 Conclusão

Neste capítulo foram apresentadas a arquitetura do framework `circ_autoproj`, a motivação para seu desenvolvimento, o detalhamento dos seus componentes, as suas entradas e saídas e a sua execução. Neste trabalho, o referido framework foi configurado para usar o algoritmo de Recozimento Simulado como SA (ver seção 4.1) e o GCE como SR (descrito na seção 4.2). Os problemas utilizados foram: 1. sensor de temperatura; 2. função Gaussiana; 3. referência de tensão; 4. função quadrática; 5. raiz quadrada; 6. função cúbica; 7. raiz cúbica. Eles são descritos com detalhes no capítulo 6.

Capítulo 3

Simulação de circuitos eletrônicos

Neste capítulo será introduzido o simulador de circuitos Ngspice, precedido de uma apresentação de alguns aspectos históricos de seu antecessor SPICE. Serão apresentados os principais tipos de análises executadas pelo Ngspice, a sintaxe da netlist para a especificação de elementos, o uso de modelos e seus principais comandos. Depois, a aplicação das simulações SPICE no `circ_autoproj` é abordada.

3.1 Introdução

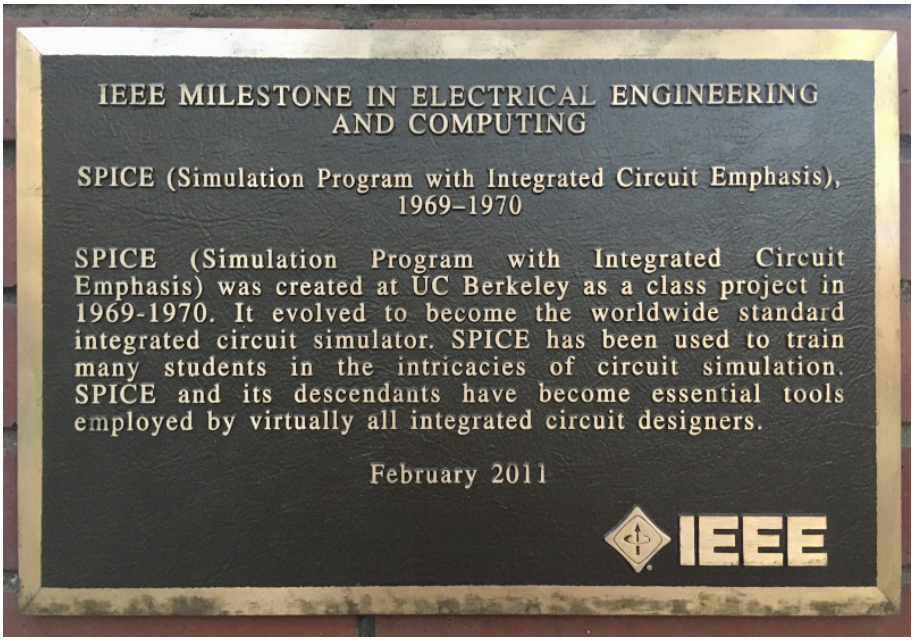
O projeto típico de um circuito eletrônico envolve normalmente diversas simulações. Nesta etapa, o diagrama do circuito é inserido em um software simulador e cada componente é configurado com parâmetros semelhantes ao do componente real. No simulador, as possíveis entradas do circuito são conectadas a fontes de sinais e o comportamento do circuito é analisado, por exemplo observando-se a sua resposta em frequência.

A simulação economiza tempo de projeto, pois permite que diversas alterações sejam feitas e os resultados analisados de forma muito mais rápida do que seria possível com protótipos em bancada. Esta agilidade permite com que o projeto possa ser mais otimizado, sendo possível obter, por exemplo, um circuito com menor número de componentes, menor consumo de energia, menor distorção ou ruído.

3.2 SPICE

O SPICE (sigla em inglês para *Simulation Program with Integrated Circuit Emphasis*: Programa de simulação com ênfase em circuitos integrados) é um simulador de circuitos criado em 1969 na Universidade da Califórnia em Berkeley. Ele foi desenvolvido inicialmente como um projeto de classe e lançado sob domínio público, se tornando uma semente para uma classe de inúmeros simuladores derivados, tanto comerciais como gratuitos. Hoje esta classe de simuladores é considerada um padrão de fato na indústria, sendo introduzida aos

Figura 6 – Fotografia da placa em homenagem ao marco da criação do SPICE



Fonte: o autor (2023)

alunos de virtualmente todos os cursos de graduação em eletrônica no mundo (NAGEL, 2013).

Em fevereiro de 2011 o SPICE foi reconhecido pelo IEEE[1] com um *marco na engenharia elétrica e da computação*[2] (Engineering and Technology History Wiki, 2011; SPANOS, 2010). A Figura 6 mostra a fotografia da placa criada em homenagem ao marco. A placa foi afixada em um espaço da Universidade da Califórnia em Berkeley e permanece exposta para visitação pública.

3.3 Ngspice

O simulador de circuitos eletrônicos utilizado neste trabalho foi o Ngspice (NGSPICE, 2021). Ele foi desenvolvido a partir do *Spice 3f5*, o último da família *Spice 3* lançado em 1993 pela Universidade da Califórnia em Berkeley. O Ngspice ao longo de seu desenvolvimento integrou o *Xspice* e o *CIDER* ao seu código. O primeiro provê um simulador baseado em eventos, útil para simulações digitais e mistas, e o último permite simulações físicas de dispositivos dos circuitos integrados.

O Ngspice é um software de código aberto que pode ser compilado e executado em diversas plataformas, como BSD, Linux, MAC e Windows. É um simulador de propósito

[1] *Institute of Electrical and Electronics Engineers* ou Instituto de Engenheiros Eletricistas e Eletrônicos

[2] O IEEE possui um programa de marcos que registra grandes feitos em áreas afins do instituto. Alguns outros marcos estabelecidos pelo programa foram: a ARPANET (rede precursora da internet), o CD player e a WaveLAN (precursor do WiFi)

geral para análises lineares e não lineares, compatível com PSPICE e LTSPICE. Dentre os dispositivos mais comuns suportados, os circuitos podem conter resistores, capacitores, indutores, fontes de tensão ou corrente, diodos, transistores bipolares e MOSFETs. O Ngspice contém internamente diversos modelos para os semicondutores.

As simulações analógicas são feitas resolvendo-se um conjunto de equações simultâneas, as leis de Kirchhoff, postas sobre a forma de matriz. Os seguintes tipos de simulações (ou *tipos de análises*) são suportados (VOGT et al., 2022):

1. Análise C.C.
2. Análise C.A.
3. Análise transitória

As seções seguintes descrevem estes três tipos de análises.

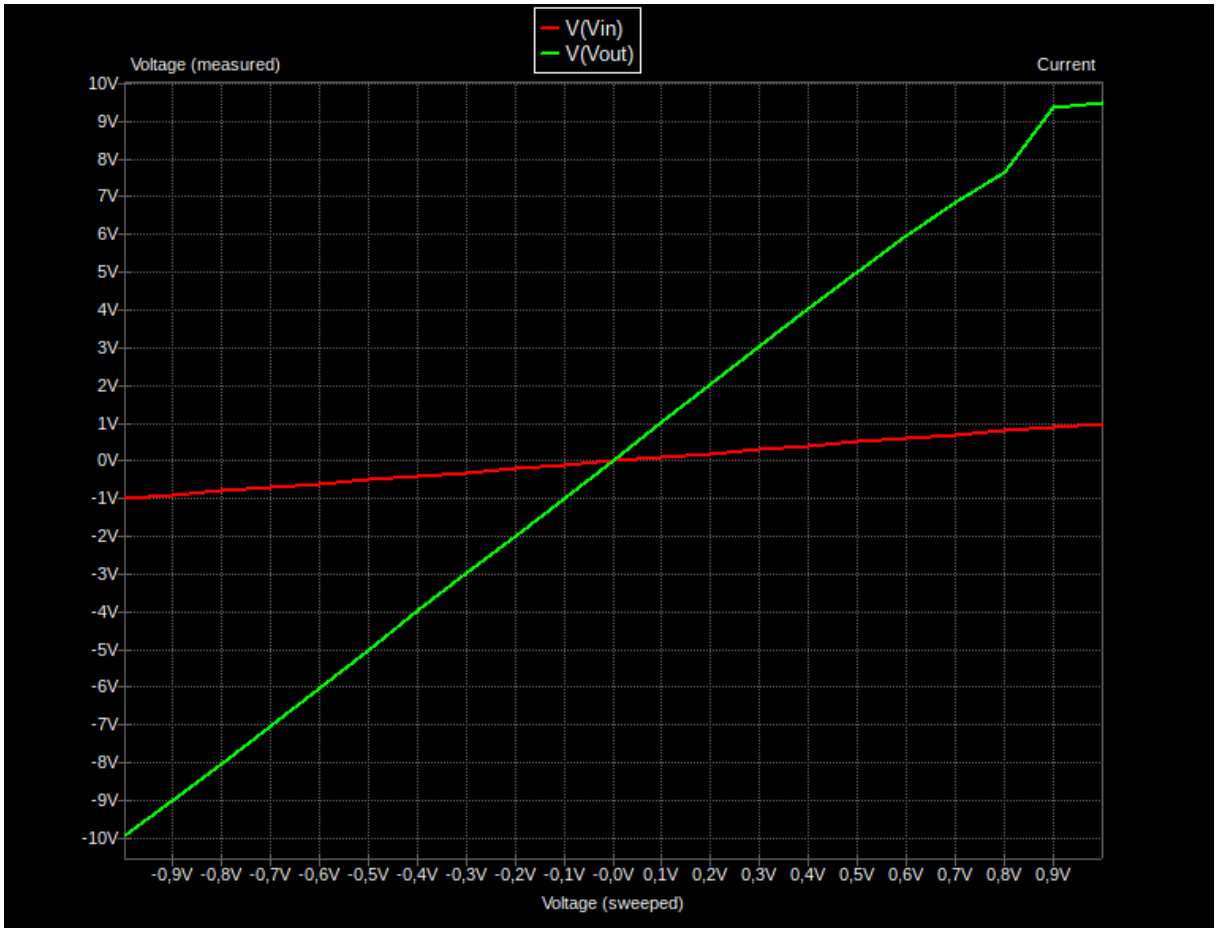
3.4 Análise C.C.

A análise de corrente contínua determina o ponto de operação C.C. em regime permanente de um circuito. Ele ignora indutores e capacitores e também quaisquer dependências da variável tempo das fontes. Este tipo de análise é executada automaticamente no início de uma análise C.A. ou transitória, pois o seu resultado é utilizado como ponto de operação ou condição inicial da simulação subsequente.

É possível solicitar ao simulador uma análise de varredura C.C, que consiste em executar diversas análises C.C. independentes com a alteração de apenas alguns parâmetros de entrada.

A Figura 7 apresenta o exemplo do resultado de uma análise C.C. para um circuito amplificador que possui acoplamento C.C. O eixo das abcissas denota a tensão aplicada na entrada, em forma de varredura. A curva mostrada em vermelho (*Vin*) repete a entrada (por isso é uma rampa linear, podendo ser omitida). A tensão de saída do circuito em função da entrada é mostrada na curva verde (Vout).

Figura 7 – Exemplo do resultado de uma análise C.C.



Fonte: o autor (2023)

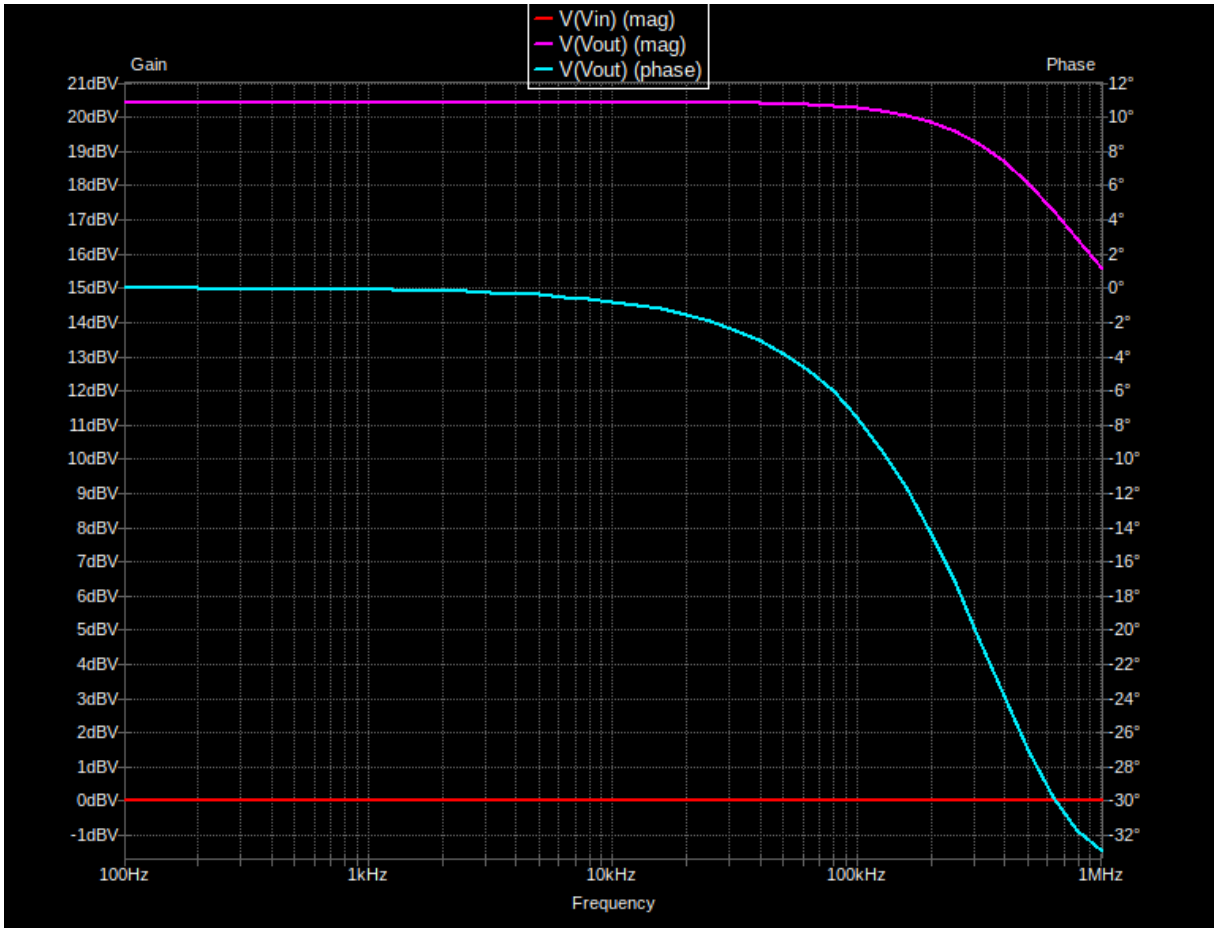
3.5 Análise C.A.

A análise de corrente alternada é um tipo de simulação que considera a solução senoidal de *pequenos sinais*, em uma ou várias frequências solicitadas. O ponto de operação é obtido através de uma análise C.C. prévia, onde são obtidos modelos lineares dos dispositivos.

A saída deste tipo de simulação é uma função de transferência, com o ganho e fase calculados em função da frequência de entrada.

A Figura 8 apresenta o exemplo do resultado de uma análise C.A. de um circuito amplificador. A curva exibida em vermelho (*Vin*) é a tensão de entrada usada como referência (em razão disso possui ganho constante de 0 dB, portanto poderia ser omitida). As curvas mostradas em violeta (Vout mag) e ciano (Vout phase) são, respectivamente, o ganho e fase da saída em função da entrada.

Figura 8 – Exemplo do resultado de uma análise C.A.



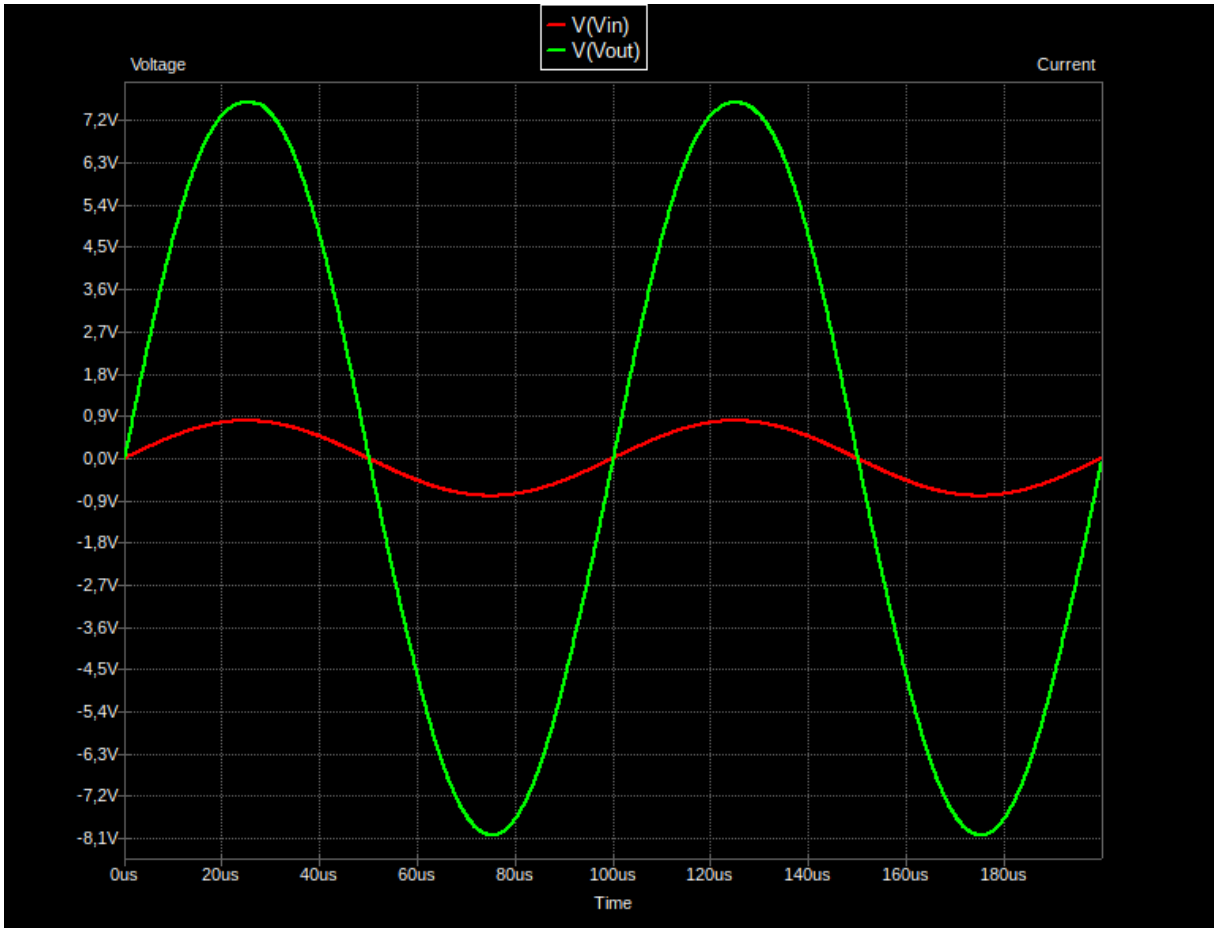
Fonte: o autor (2023)

3.6 Análise transitória

A análise transitória começa com uma análise de corrente contínua que determina as condições iniciais do circuito. A simulação então passa a considerar todos os dispositivos com aspectos que dependem do tempo, sendo executada repetidamente com um determinado passo de tempo, obtendo-se a evolução das tensões e correntes dos nós do circuito ao longo do tempo.

A Figura 9 apresenta o exemplo do resultado de uma análise transitória de um amplificador de tensão. Neste gráfico a curva em vermelho (*Vin*) é a entrada do circuito e a em verde (*Vout*) é a sua saída.

Figura 9 – Exemplo do resultado de uma análise transitória

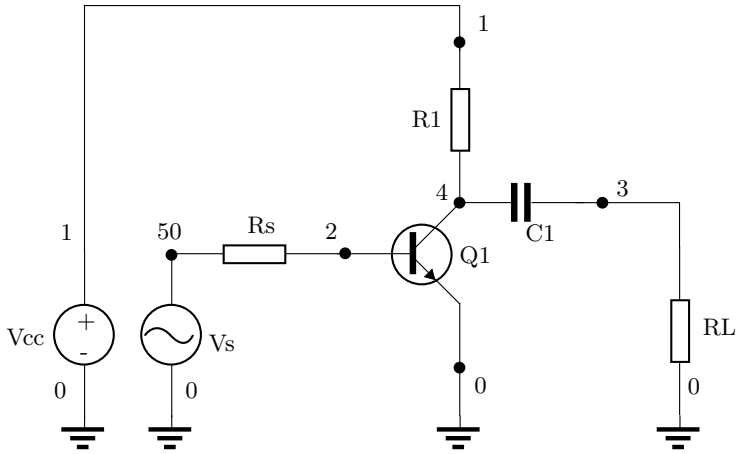


Fonte: o autor (2023)

3.7 Netlist

No SPICE, os circuitos eletrônicos são descritos utilizando uma *netlist*, um arquivo texto onde cada linha define um componente ou invoca um comando. Por exemplo, o circuito da Figura 10 pode ser representado com a seguinte netlist:

Figura 10 – Circuito de exemplo para a netlist



Fonte: o autor (2023), baseada em Castejón e Carmona (2018)

```
CIRCUITO DE TESTE
Vcc 1 0 dc 5
Vs 50 0 0.0 ac 1 sin(0 2 10k)
Rs 50 2 100
RL 3 0 2k
Q1 4 2 0 2N3904
R1 1 4 2.0e3
C1 4 3 2.0e-9
.include 2N3904.sp3
.tran 100ns 200us
.save v(3)
```

A primeira linha da netlist é o título do circuito, utilizado apenas para identificação nas saídas das simulações. Alternativamente o título pode ser especificado através do comando `.title <titulo>`. Linhas de comentários podem ser escritas começando com um asterisco. No final da linha, estes podem ser inseridos utilizando os caracteres `;` ou `$` para iniciar o comentário.

O comando `.include <arquivo>` é utilizado para copiar todo o conteúdo de um segundo arquivo para o primeiro. Normalmente é empregado para carregar modelos de componentes ou mesmo partes de um circuito. Já o comando `.save <v1> <v2> ... <vn>` marca vetor(es) para ser(em) gravado(s) no arquivo de saída. Por exemplo, o comando `.save v(3)` instrui o Ngspice a salvar a tensão do nó 3 do circuito após a simulação.

Os componentes eletrônicos do circuito (chamados de *elementos* no Ngspice) são especificados cada um em uma linha na netlist. Cada linha contém:

Tabela 1 – Sintaxe dos elementos SPICE em uma netlist

Elemento	Sintaxe
Fonte de tensão	`VXXXXXXX N+ N- «DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE»>`
Transistor BJT	`QXXXXXXX nc nb ne mname`
Resistor	`RXXXXXXX n+ n- value`
Capacitor	`CXXXXXXX n+ n- value`
Indutor	`LXXXXXX n+ n- value`

Fonte: o autor (2023), com informações obtidas de Vogt et al. (2022)

1. O nome da instância do elemento;
2. Os nós do circuito que o elemento é conectado;
3. Os valores dos parâmetros que determinam as características elétricas do elemento.

A Tabela 1 apresenta a sintaxe de alguns tipos de elementos SPICE. O primeiro caractere da linha é uma letra que determina o tipo do elemento, seguido por outros caracteres alfanuméricos que o identificam unicamente no circuito. O identificador `N+`, `N-`, `nc`, `nb` e `ne` na sintaxe indica o nó ao qual o terminal do elemento está conectado. Pode ser qualquer número inteiro não negativo, sendo reservado o zero para o terra (*ground*). Já o item `value` especifica o valor de um parâmetro do componente: para um resistor isto seria a sua resistência elétrica em ohms. O item `mname` especifica o nome de modelo (ou *model name*) do transistor, cujos parâmetros são normalmente definidos em um arquivo externo à netlist, carregado através da diretiva `.include <arquivo>`.

Os componentes mais complexos, especialmente os semicondutores, normalmente possuem muitos parâmetros. Por exemplo, um transistor bipolar pode possuir até 100 parâmetros[3]. Frequentemente, vários elementos em uma netlist utilizam muitos parâmetros idênticos. Com isso, seu uso direto em uma netlist demandaria muitas repetições ao longo das instâncias de elementos iguais ou similares. O uso de modelos permite eliminar esta redundância, simplificando a netlist. Um exemplo de modelo para um transistor bipolar é dado a seguir:

```
.model BC548 NPN BF=200 IS=1e-10 VBF=50
```

O comando para introduzir o modelo é o `.model`, seguido pelo nome do modelo (que deve ser único em todo o circuito a ser simulado) e pelo tipo do modelo. Em seguida são listados os parâmetros através de pares *parâmetro=valor*, separados por espaços em branco. Este exemplo foi propositalmente simplificado para apresentar o assunto de forma

[3] Modelo Gummel-Poon, de acordo com Vogt et al. (2022, seção 8.2.1)

Tabela 2 – Comandos de análise SPICE em uma netlist

Análise	Sintaxe	Exemplo
Análise transitória	`.tran tstep tstop`	`.tran 100ns 200us`
Análise C.A.	`.ac dec nd fstart fstop`	`.ac dec 20 20 200k`
Análise C.C.	`.dc srcnam vstart vstop vincr`	`.dc vs 0 1.5 0.01`

Fonte: o autor (2023), com informações obtidas de Vogt et al. (2022)

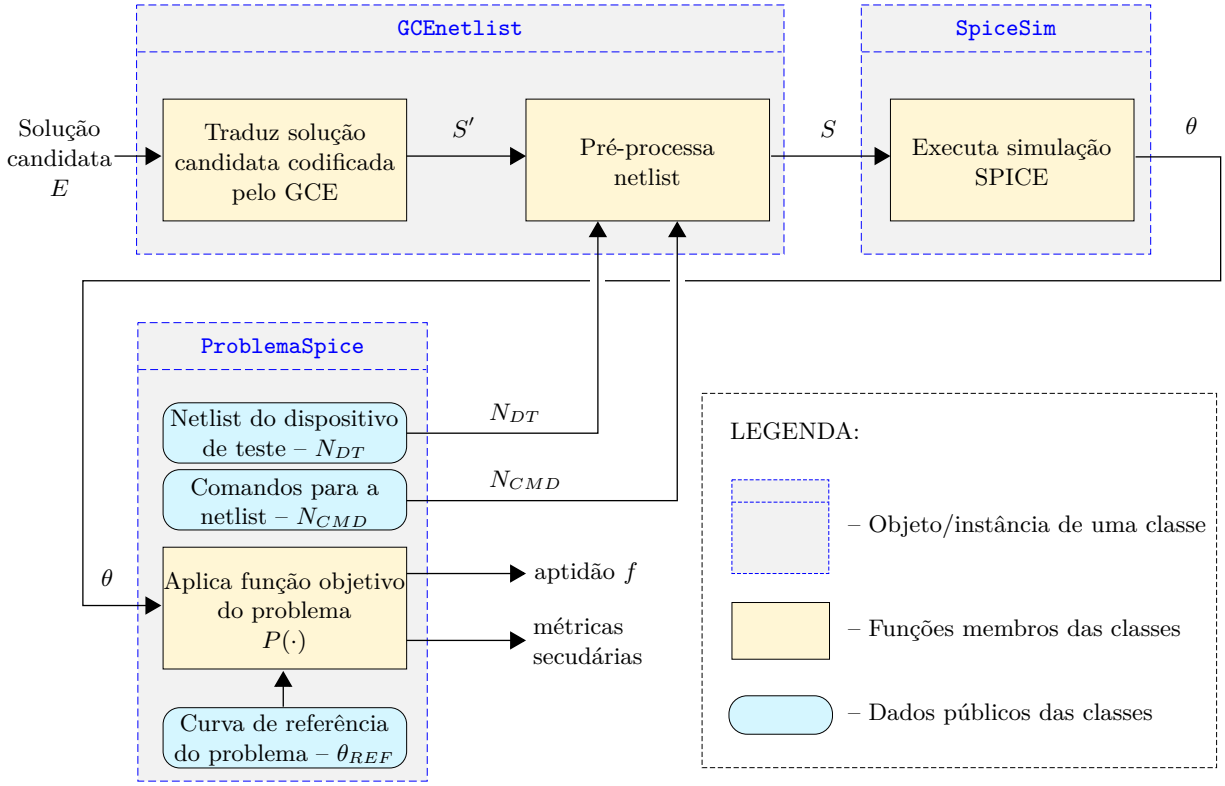
resumida. Modelos reais, normalmente disponibilizado por fabricantes de componentes, possuem muitos parâmetros, sendo muito mais longos. Por esse motivo, cada especificação de um modelo é normalmente contida em um arquivo próprio, e o seu uso em uma netlist é feito através do comando `.include <arquivo do modelo>`. O apêndice C apresenta a listagem de todos os modelos SPICE dos componentes eletrônicos utilizados neste trabalho.

O tipo de análise a ser executada é especificada através de um *comando* SPICE. A Tabela 2 apresenta a sintaxe simplificada dos comandos de três tipos mais comuns de análise, incluindo um exemplo em cada caso. O identificador `tstep` e `tstop` determinam o tamanho do passo e o tempo de parada da simulação transitória, respectivamente. Na análise C.A., o identificador `nd` especifica o número de pontos por década, `fstart` e `fstop` determinam a frequência inicial e final da simulação. Na análise C.C., `srcnam` determina o nome de uma fonte independente de tensão ou corrente, um resistor, ou a temperatura do circuito a ser variada. `vstart`, `vstop` e `vincr` especificam o valor inicial, final e de incremento do parâmetro de `srcnam`.

3.8 Aplicação das simulações SPICE no circ_autoproj

O framework `circ_autoproj` utiliza o Ngspice em uma parte do cálculo da aptidão de uma solução candidata: os circuitos sintetizados pelo algoritmo de busca meta-heurístico são enviados para o simulador Ngspice, e os resultados da simulação são utilizados para calcular a aptidão (métrica principal) e algumas outras métricas secundárias opcionais.

Figura 11 – Diagrama do cálculo da aptidão de uma solução candidata



Fonte: o autor (2023)

A aptidão f é um número real não negativo que indica o *custo*, *adequação* ou o *erro* da solução em relação a uma referência desejada. Quanto menor o valor da aptidão, melhor é a solução. Assim, o objetivo da meta-heurística é otimizar o circuito, minimizando f. A Figura 11 mostra o processo de cálculo da aptidão a partir de uma solução candidata. Nela são apresentadas as classes de objetos do `circ_autoproj` responsáveis por cada função. O cálculo consiste em:

1. Obter a netlist SPICE S' da solução candidata E através da tradução do circuito representado/codificado pelo GCE, usando o Algoritmo 3;

2. Pré-processar S', resultando em S. Isto consiste em enumerar os componentes, remover aqueles que possuem todos os terminais em curto-circuito, adicionar resistores aos nós pendentes e concatenar seu resultado com a netlist do dispositivo de teste N_{DT} e os comandos da netlist N_{CMD} especificados pelo objeto `ProblemaSpice`. A enumeração consiste em identificar os elementos da netlist através da atribuição de numerais consecutivos para um mesmo tipo de componente, por exemplo: à primeira ocorrência do elemento resistor (R) é atribuído o numeral 1, à segunda, 2, assim por diante. Isto é necessário, pois cada componente precisa estar identificado unicamente em uma netlist. Durante a síntese da netlist pelo algoritmo meta-heurístico, pode haver a ocorrência de componentes com todos os seus terminais curto-circuitados (com

o mesmo número de nó). Estes componentes em uma simulação não são funcionais, portanto são removidos para o simples saneamento da netlist. Além disso, podem haver alguns terminais de componentes desconectados. Estes são chamados de *nós pendentes* e são conectados ao terra através de resistores de alto valor (1 GΩ) para tornar a simulação SPICE possível;

3. Executar o simulador SPICE utilizando a netlist S como entrada, obtendo θ. Este resultado da simulação inclui tensões e/ou correntes solicitadas para alguns ou todos os nós do circuito ao longo do tempo, frequência ou temperatura, dependendo do tipo de análise solicitada;

4. Comparar θ com a referência de desempenho desejada do circuito θ_{REF} através da função objetivo do problema $P(\cdot)$. Seu resultado é a aptidão f e, opcionalmente, métricas secundárias.

3.9 Conclusão

Este capítulo apresentou um breve histórico do SPICE, um simulador que deu origem a uma classe de programas, se tornando um padrão na indústria. Dentre eles, foi abordado o Ngspice, o simulador utilizado neste trabalho. Foram apresentados os tipos de análises, a netlist, o uso de modelos e alguns comandos, com o fechamento na aplicação do referido simulador no framework `circ_autoproj`.

Capítulo 4

Algoritmo SANN-GCE

Este capítulo apresenta o algoritmo[1] para síntese de circuitos eletrônicos analógicos SANN-GCE. Este algoritmo é dividido em duas partes, apresentadas nas seções subsequentes: 1. Algoritmo de busca, implementado através do Recozimento Simulado (*Simulated Annealing - SANN*); 2. Representação da solução, chamada de Evolução Geométrica de Circuitos (*Geometric Circuit Evolution - GCE*). Esta divisão em duas partes é aderente ao esquema de *plugins* SA/SR do framework `circ_autoproj`, conforme detalhado no capítulo 2.

4.1 Algoritmo de busca: Recozimento simulado

O recozimento simulado (ou *simulated annealing*) (KIRKPATRICK; GELATT; VECCHI, 1983) é uma meta-heurística clássica para otimização global que opera em uma solução candidata por vez (em contraste com algoritmos baseados em população, como os Algoritmos Genéticos). É adequado para aplicações onde a avaliação da função objetivo é complexa e não é explicitamente conhecida (como nos *problemas caixa-preta*). Suas principais vantagens são a simplicidade e o baixo uso de memória, pois opera em uma solução candidata por vez (DELAHAYE; CHAIMATANAN; MONGEAU, 2019).

Seu funcionamento baseia-se em uma analogia com o processo de recozimento em sólidos, que consiste em aquecer o material e depois resfriá-lo lentamente de forma controlada, com o objetivo de aliviar suas tensões internas. Este processo permite que os domínios do material sejam reorganizados em um estado de baixa energia. A *mecânica estatística* versa sobre a teoria deste processo físico, e afirma, em linhas gerais, que a probabilidade do sólido estar em um estado de energia $E_s = E$ é dada pela Equação (4.1), onde $Z(T)$ é um fator de normalização, T é a temperatura e k_B é a constante de Boltzmann (LAARHOVEN; AARTS, 1987).

[1] Rigorosamente, o SANN-GCE é composto de diversos algoritmos, mas arbitrou-se neste trabalho como sendo adequado se referir ao conjunto simplesmente como *algoritmo SANN-GCE*

$$P(E_s = E) = \frac{1}{Z(T)} e^{-\frac{E}{k_B T}} \tag{4.1}$$

O recozimento simulado utiliza o *critério de Metropolis* (METROPOLIS et al., 1953) durante a evolução da solução. Ele prescreve que se uma *solução nova* possuir aptidão melhor que a *solução atual*, ela será sempre aceita, se tornado a solução atual. No sentido contrário (quando a *solução nova* possuir aptidão pior que a *solução atual*), ela ainda poderá ser aceita, mas agora a aceitação é condicionada a uma probabilidade que é função da diferença entre as aptidões ($\Delta F = F_{new} - F_{curr}$). Mais especificamente, a aceitação se dará com probabilidade $e^{-\Delta F/T}$. Caso não haja aceitação da solução nova, a solução atual é mantida. Esta probabilidade de aceitação é baseada na distribuição de Boltzmann, relacionada ao processo de recozimento em sólidos (PHAM; KARABOGA, 2000).

O algoritmo do recozimento simulado usado neste trabalho é apresentado formalmente através do Algoritmo 1 e na forma de fluxograma, de maneira simplificada, na Figura 12. O algoritmo inicia definindo a temperatura inicial $T = T_0$ e criando uma solução candidata aleatória S_0. A cada iteração, uma solução vizinha S_1 é criada a partir de uma *mutação* de S_0. Se S_1 for melhor que S_0, ela é atribuída como a solução atual. Se for pior, o critério Metropolis é utilizado para decidir sua aceitação. No final do laço de iteração, a temperatura T é reduzida por um fator α e o laço recomeça na próxima iteração.

Observe que quanto maior a temperatura (presente no início da execução), mais provavelmente uma solução pior será aceita. À medida que as iterações avançam, essa probabilidade diminui seguindo a redução da temperatura. Quando T se aproxima de zero, o recozimento simulado torna-se semelhante ao algoritmo de Monte Carlo (DELAHAYE; CHAIMATANAN; MONGEAU, 2019).

O esquema de resfriamento usado no Algoritmo 1 foi o *resfriamento geométrico*, dado por $T_{new} = \alpha T_{old}$, onde $0 < \alpha < 1$ é uma constante chamada de *fator de resfriamento*, geralmente muito próxima de 1. Além disso, quando a temperatura cai abaixo do *limite de reaquecimento* T_r, a temperatura é ajustada de volta à temperatura inicial T_0.

O Algoritmo 1 utilizou o critério do *elitismo* para manter a melhor solução candidata (S_B) encontrada ao longo do processo de evolução. Ele funciona comparando, a cada iteração, as aptidões das soluções antes e depois da mutação (S_0 e S_1). A melhor delas é copiada para S_B e retornada no final da execução do algoritmo.

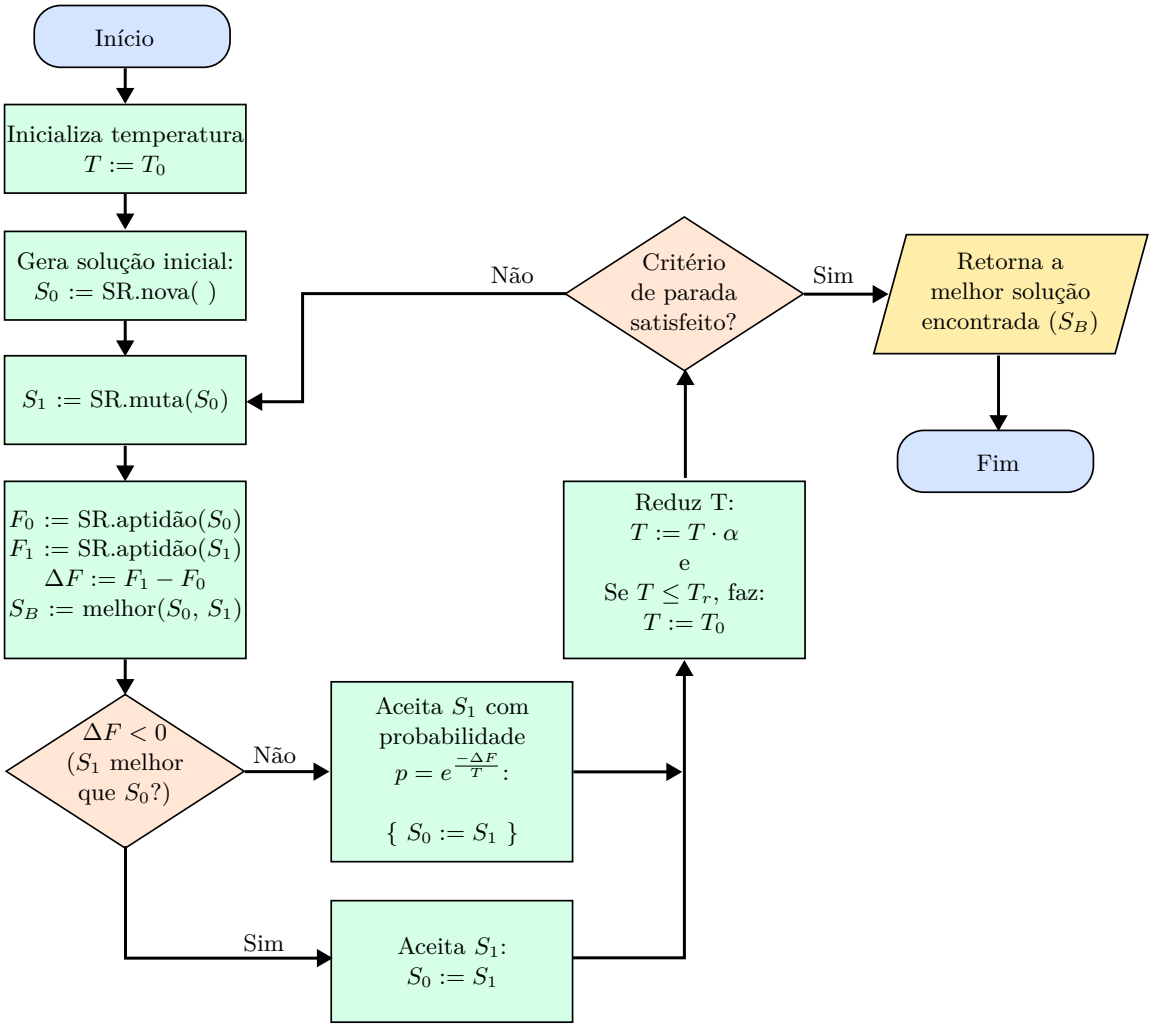
O laço principal é executado até que uma *condição de parada* seja satisfeita. Esta é configurável, podendo ser limitada a um determinado número de iterações ou a uma quantidade limite de chamadas da função interna que calcula a aptidão. Convém salientar que não há necessariamente uma relação direta desta última com a primeira, pois há um mecanismo de *cache* implementado no *objeto solução* que evita que a *função interna de aptidão* seja acionada quando a solução não tiver sido modificada desde o último cálculo

Algoritmo 1 Algoritmo do recozimento simulado

Requisito: SR: classe que representa uma solução candidata (Veja a seção 4.2). Esta classe precisa implementar os métodos *nova()*, *muta()* e *aptidão()*.
Requisito: R(): gera um número aleatório a partir de uma distribuição uniforme $0 \leq x \leq 1$
Entrada: T_0: Temperatura inicial
Entrada: T_r: Limiar de reaquecimento
Entrada: α: Fator de resfriamento. Precisa ser um número positivo menor que 1 (tipicamente muito próximo de 1).

1: **function** sann(T_0, T_r, α)
2: $T = T_0$
3: $S_0 \leftarrow$ SR.nova() ▷ Cria uma instância de uma nova solução candidata
4: $S_B \leftarrow S_0$ ▷ Armazena a melhor solução (elitismo)
5: **while** (condição de parada não atingida) **do**
6: $S_1 \leftarrow S_0$ ▷ Cria uma cópia de trabalho de S_0
7: S_1.muta( )
8: $F_0 \leftarrow S_0$.aptidão( )
9: $F_1 \leftarrow S_1$.aptidão( )
10: **if** $F_0 < F_1$ **then** ▷ Compara a aptidão: menor é melhor
11: $S_B \leftarrow S_0$
12: **else**
13: $S_B \leftarrow S_1$
14: **end if**
15: $\Delta F \leftarrow F_1 - F_0$ ▷ $\Delta F < 0$ significa que S_1 é melhor que S_0
16: $A \leftarrow$ false ▷ A: aceitação da mutação
17: **if** $\Delta F = 0$ **then**
18: $A \leftarrow$ false
19: **else if** $\Delta F < 0$ **then**
20: $A \leftarrow$ true
21: **else**
22: $p = e^{-\Delta F/T}$
23: **if** R( ) $< p$ **then**
24: $A \leftarrow$ true
25: **end if**
26: **end if**
27: **if** A **then**
28: $S_0 \leftarrow S_1$
29: **end if**
30: $T \leftarrow T \cdot \alpha$
31: **if** $T \leq T_r$ **then** ▷ se abaixo de um certo limiar...
32: $T = T_0$ ▷ ...aplica o reaquecimento
33: **end if**
34: **end while**
35: **return** S_B ▷ retorna a melhor solução encontrada
36: **end function**

Figura 12 – Fluxograma simplificado do algoritmo do recozimento simulado



Fonte: o autor (2024)

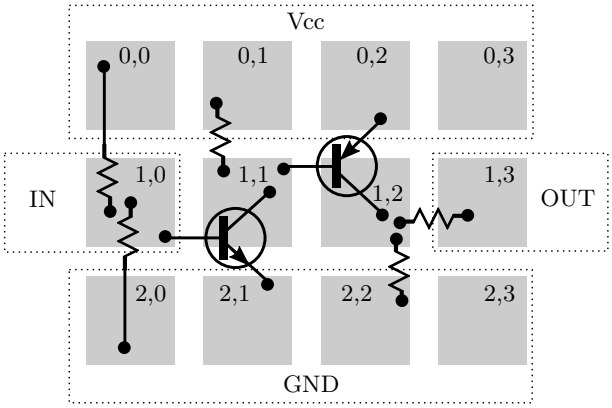
real da aptidão (por exemplo, quando uma mutação não ocorre devido a sua probabilidade associada), sendo retornado o valor armazenado em *cache* neste caso.

Neste trabalho, a condição de parada do algoritmo foi definida como um número máximo de avaliações da função de aptidão, conforme detalhamento fornecido no capítulo 6.

4.2 Representação da solução: Evolução Geométrica de Circuitos

A representação da solução (ou *Solution Representation - SR*) codifica uma solução candidata em um formato conveniente para que o algoritmo de busca funcione com eficiência. Neste trabalho, foi desenvolvido uma SR denominada Evolução Geométrica de Circuitos (ou *Geometric Circuit Evolution - GCE*). O GCE armazena uma solução por meio da composição hierárquica de objetos, cada um contendo informações associadas

Figura 13 – Exemplo de placa GCE



Fonte: o autor (2023)

a um elemento do circuito. Os principais objetos de um circuito eletrônico (ou solução candidata) E no GCE são: B – a *placa* e C – um array de *componentes*.

A *placa* representa uma matriz de dimensões fixas $M \times N$. Cada posição da matriz é equivalente a um contato elétrico – ou *pad* – lembrando uma placa de circuitos perfurada. Uma posição na placa é representada por um par ordenado (x, y) que indica sua localização. Cada *pad* pode ter zero ou mais terminais de componentes associados a ele. A Figura 13 mostra um exemplo de uma placa GCE preenchida com alguns componentes. Nessa figura, os quadrados cinzas são as ilhas da placa e as áreas pontilhadas indicam os nós externos.

A *placa* armazena as seguintes informações:

- número de componentes do circuito: n;
- um array contendo n objetos *componentes*: $C = \{c_0, c_1, \ldots, c_n\}$;
- um conjunto para mapeamento dos nós externos: $\Omega = \{\omega_a, \omega_b, \ldots\}$, onde ω é uma lista de *pads* e o seu subscrito é um numeral que indica o nó externo atribuído à referida lista. Por exemplo, $\omega_0 = \{(2,0), (2,1), (2,2), (2,3)\}$ indica que esses quatro *pads* estão associados ao nó 0 (na simulação SPICE, o nó zero é o *terra* (GND)). Esses nós externos são representados como áreas pontilhadas na Figura 13.

Cada objeto componente c no GCE armazena:

- Seu tipo: c_t (por exemplo: resistor, transistor, etc.);
- A posição de cada um de seus terminais na *placa*;
- Seus parâmetros (por exemplo: resistência elétrica, modelo de um transistor BJT, largura de canal em um transistor FET), armazenados como um array de tipo de dados variante (inteiro, float ou string dependendo do tipo de parâmetro).

Os objetos do GCE possuem algumas variáveis designadas que podem ser *perturbadas* externamente. Essas variáveis são chamadas neste trabalho de *graus de liberdade* (ou sua sigla em inglês, DOF - *Degress Of Freedom*). Quando um DOF é perturbado, ele invoca sua própria *rotina geradora* para, possivelmente, gerar um novo valor. Após isso, se configurada, é invocada a sua *rotina ouvinte* para agir sobre o valor recém-alterado.

Por exemplo, o DOF n do objeto *placa* (que representa o número de componentes do circuito) quando perturbado, invoca sua *rotina geradora* que produz um número aleatório a partir de uma distribuição uniforme dentro do intervalo $[n_{min}, n_{max}]$ e atribui este valor a n. Depois disso, sua rotina ouvinte é invocada e cria ou destrói componentes para manter exatamente n componentes no circuito.

No GCE, todos os DOFs possuem um tipo associado T, que pode ser um dos seguintes:

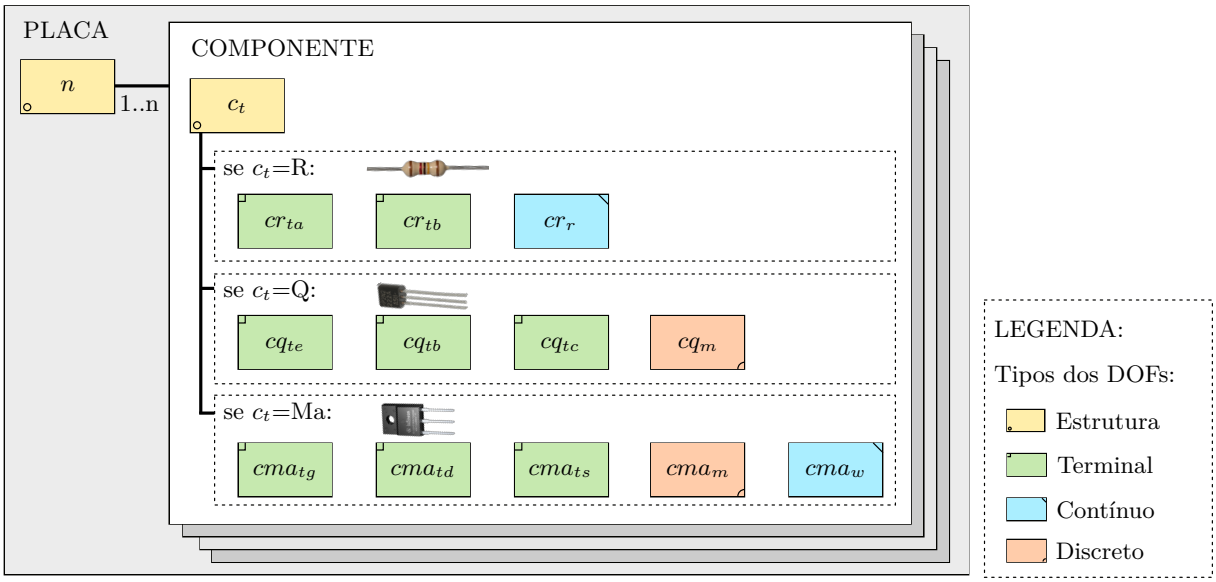
- *Terminal*: representa a posição (x, y) de ligação de um terminal de componente na placa. É esperado que a perturbação deste tipo de DOF ocasione uma grande variação (ou descontinuidade) na resposta do circuito, pois a alteração na ligação dos terminais de um componente no circuito pode alterar completamente o seu funcionamento;
- *Contínuo*: define uma variável contínua (por exemplo, resistência de um resistor). Sua mudança pode ter um impacto suave na resposta do circuito;
- *Discreto*: define uma variável discreta (por exemplo, o modelo SPICE de um transistor, escolhido de uma lista de opções). Sua alteração pode implicar em variação moderada na resposta do circuito;
- *Estrutura*: controla a estrutura do circuito (por exemplo, o número de componentes). Sua alteração pode causar grande variação na resposta do circuito e, inclusive, alterar o número de DOFs.

A Tabela 3 enumera todos os DOFs usados no GCE e apresenta, para cada um, o objeto que o contém, o seu tipo, a sua descrição, os possíveis valores que ele pode assumir e uma explicação sobre o funcionamento da sua função geradora e ouvinte (se configurada).

A Figura 14 apresenta, de forma resumida, a relação hierárquica dos objetos de uma solução candidata no GCE e seus DOFs, incluindo seus tipos. A existência de alguns DOFs dependem do tipo de componente c_t. Quando o tipo do componente é alterado, os DOFs subordinados ao tipo antigo são destruídos, e outros específicos do novo tipo de componente são criados.

O algoritmo de busca mantém em memória soluções candidatas (codificadas pela SR, o GCE) e as otimiza (evolui) ao longo das suas iterações. Para isso, o SA se utiliza

Figura 14 – Hierarquia dos objetos de uma solução candidata do GCE e seus graus de liberdade



Fonte: o autor (2023)

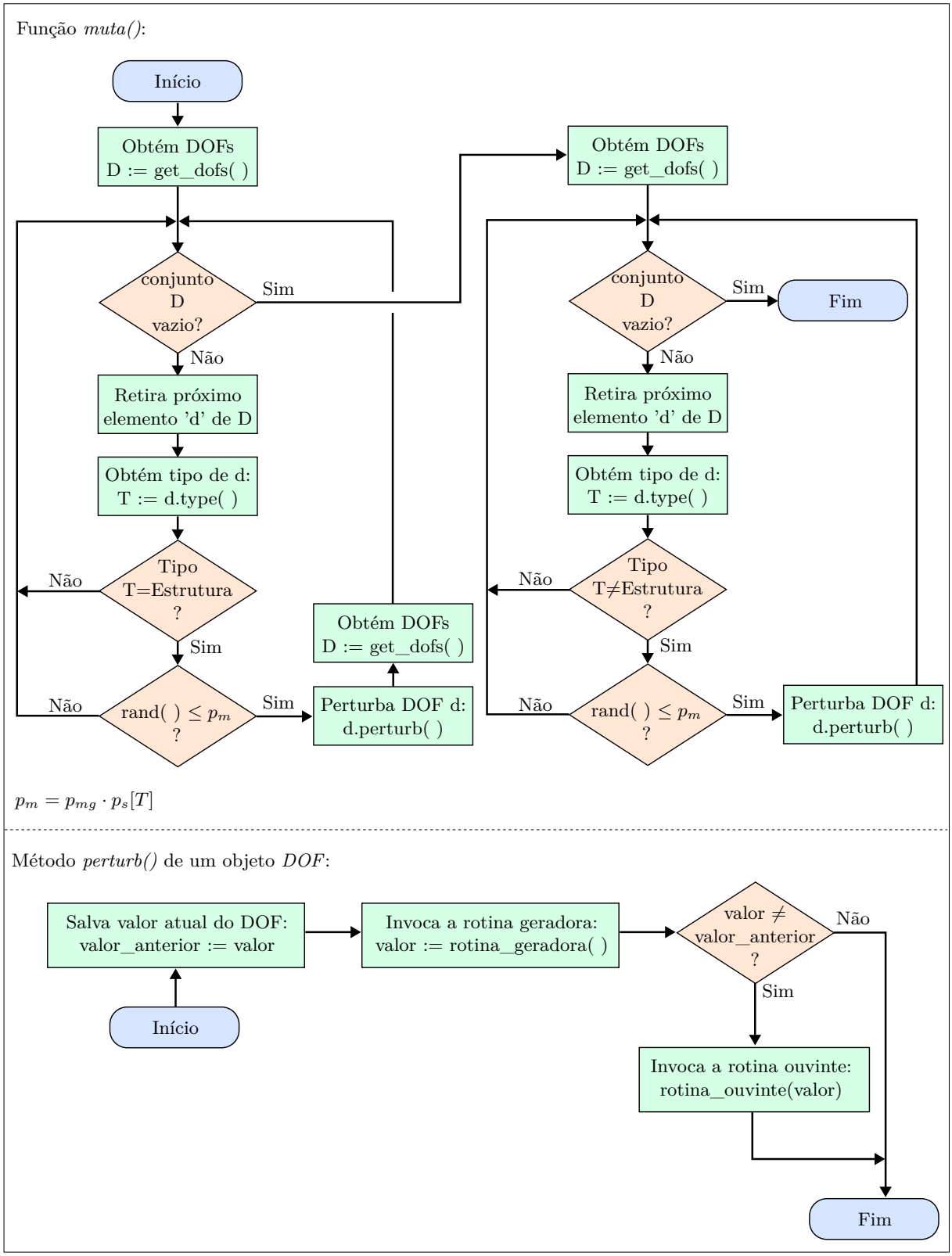
principalmente da função de *mutação* da solução candidata. O objetivo desta função é alterar a solução para um *estado vizinho* da solução anterior, permitindo que o SA explore o espaço de busca de maneira eficiente. O Algoritmo 2 apresenta formalmente o funcionamento desta função, enquanto a Figura 15 ilustra de maneira simplificada a sua execução.

O SANN-GCE não utiliza operador de *recombinação*, pois o recozimento simulado não o requer. Isso traz o benefício de permitir que a implementação da SR seja mais simples, pois a operação de recombinação não é trivial para estruturas de dados complexas como as do GCE.

O procedimento de mutação se inicia com a chamada de função `get_dofs()`, que obtém todos os DOFs de todos os objetos de uma solução candidata. Depois, os DOFs são visitados um a um. Em uma visita, o DOF pode ou não ser perturbado (ou seja, sofrer mutação). Os DOFs do tipo estrutura são processados primeiro de uma maneira especial, pois, quando perturbados, podem criar ou destruir outros DOFs. A mutação ocorre com uma probabilidade $p_m = p_{mg} \cdot p_s(T)$, onde p_{mg} é uma probabilidade de mutação geral e $p_s(T)$ é uma probabilidade de mutação específica para o tipo DOF T. Ambos p_{mg} e $p_s(T)$ são configuráveis.

A discriminação de diferentes $p_s(T)$ para T distintos permite que a mutação seja menos frequente para os tipos de DOFs que, quando perturbados, possam causar variações disruptivas na resposta do circuito. Desta forma, o algoritmo de busca pode explorar o espaço do problema de forma mais eficiente.

Figura 15 – Fluxograma simplificado do algoritmo de mutação GCE



Fonte: o autor (2024)

Algoritmo 2 Algoritmo de mutação GCE

Requisito: get_dofs(): obtém uma lista atualizada de todos os DOFs
Requisito: R(): gera um número aleatório a partir de uma distribuição uniforme $0 \leq x \leq 1$
Entrada: p_{mg}: probabilidade de mutação geral, $0 \leq p_{mg} \leq 1$
Entrada: $p_s[T]$: probabilidade de mutação específica para o DOF tipo T, $0 \leq p_s[T] \leq 1, \forall\ T$

```
1:  procedure muta(p_mg, p_s)
2:      D ← get_dofs()
3:      for all d ∈ D do                          ▷ 1ª iteração: somente DOFs do tipo estrutura
4:          T ← d.type( )                                          ▷ obtém o tipo do DOF d
5:          if T = estrutura then
6:              if R( ) ≤ p_mg · p_s[T] then
7:                  d.perturb( )
8:                  D ← get_dofs()    ▷ obtém todos os DOFs novamente, pois eles podem ter
                                          sido alterados pelo comando anterior
9:              end if
10:         end if
11:     end for
12:     D ← get_dofs()
13:     for all d ∈ D do                          ▷ 2ª iteração: outros tipos de DOFs
14:         T ← d.type( )
15:         if T ≠ estrutura then
16:             if R( ) ≤ p_mg · p_s[T] then
17:                 d.perturb( )
18:             end if
19:         end if
20:     end for
21: end procedure
22: procedure perturb( )                              ▷ Método de um objeto DOF
23:     valor_anterior ← valor                  ▷ armazena o valor atual do DOF
24:     valor ← rotina_geradora( )                    ▷ invoca a rotina geradora
25:     if valor ≠ valor_anterior then          ▷ se o valor do DOF foi alterado
26:         rotina_ouvinte(valor)                       ▷ invoca a rotina ouvinte
27:     end if
28: end procedure
```

A criação de uma solução candidata começa com um objeto *placa* contendo seu único DOF n (que representa a quantidade de componentes no circuito, tipo *estrutura*) definido com o valor zero. Na sequência, este DOF é perturbado, o que faz com que n assuma um valor aleatório positivo $n_{min} \leq n \leq n_{max}$. Sua rotina ouvinte atua sobre o valor alterado com o propósito de criar n objetos filhos, do tipo *componente*.

Cada objeto componente possui seus próprios DOFs. Estes também são perturbados na inicialização, começando pelo c_t (DOF tipo *estrutura*) que definirá o tipo do componente eletrônico e criará seus DOFs específicos. Em seguida, todos os DOFs restantes dos componentes são perturbados, resultando em um circuito eletrônico aleatório.

Todas as informações necessárias para expressar uma solução candidata em uma netlist SPICE estão contidas nos DOFs dos objetos. Esta operação é chamada de *tradução*

e está apresentada no Algoritmo 3 através da função `translation()`.

Algoritmo 3 Tradução do GCE para uma netlist SPICE

Requisito: Cada tipo de componente eletrônico precisa implementar o método get_netlist().
Requisito: START_NODE: inteiro que indica o primeiro número do nó a ser utilizado nos nós internos
Requisito: get_external_node(P): função que retorna o número do nó externo (≥ 0) para a posição da placa P. Retorna -1 se P não possuir um número de nó associado
Entrada: C: vetor de objetos componentes

1: **function** translation(C)
2: $S \leftarrow$ ‘’ ▷ String vazia
3: **for all** c ∈ C **do**
4: $s \leftarrow$ c.get_netlist( )
5: $S \leftarrow S +\ s +$ ‘\n’ ▷ Concatena e adiciona fim de linha
6: **end for**
7: **return** S ▷ retorna a netlist SPICE do circuito
8: **end function**
9: **function** Q.get_netlist( ) ▷ Exemplo para um transistor bipolar
10: **return** ‘Q ’ + pos2node(cq_{tc}) + ‘ ’ + pos2node(cq_{tb}) + ‘ ’ + pos2node(cq_{te}) + ‘ ’ + cq_m
11: **end function**
12: **function** pos2node(P) ▷ Obtém o número do nó a partir da posição na placa $P = (x,y)$
13: static $M \leftarrow$ map[] ▷ Cria um mapa estático (mantém seu conteúdo entre as chamadas desta função)
14: static $c \leftarrow$ START_NODE ▷ contador estático
15: $e \leftarrow$ get_external_node(P)
16: **if** e ≥ 0 **then**
17: $n \leftarrow e$
18: **else**
19: **if** P ∈ M **then**
20: $n \leftarrow M[P]$
21: **else**
22: $n \leftarrow c$
23: M.add[P]
24: $M[P] \leftarrow n$
25: $c \leftarrow c + 1$
26: **end if**
27: **end if**
28: **return** n ▷ retorna o número do nó
29: **end function**

A saída da etapa de tradução é a netlist da solução candidata que é então enviado para o simulador SPICE para obter seu desempenho e calcular a sua aptidão. A seção 3.8 apresenta o funcionamento desta etapa em detalhes.

Tabela 3 – Lista dos graus de liberdade do GCE

Variável	Objeto	Tipo[1]	Descrição	Valores possíveis	Gerador/Ouvinte
n	Placa	E	Número de componentes no circuito	$n \in \mathbb{N}^+, n_{min} \leq n \leq n_{max}$. Configurado em cada problema.	Número aleatório uniformemente distribuído dentro da faixa prescrita. Rotina ouvinte: Cria ou destrói uma certa quantidade de componentes para, no final, se ter exatamente n componentes
c_t	Componente	E	Tipo do componente	$c_t \in \{R, Q, Ma\}$, onde R é um resistor, Q é um transistor bipolar (BJT) e Ma é um MOSFET com comprimento do canal fixo (10 μm)	Escolhe aleatoriamente um dos valores discretos prescritos, considerando os pesos especificados de cada item. Rotina ouvinte: Regenera todos os DOFs específicos do componente
cr_{ta}, cr_{tb}	Componente c_t=R	T	A posição de cada um dos dois terminais do resistor	$(x, y); 0 \leq x \leq M-1; 0 \leq y \leq N-1$	Na primeira geração, retorna um inteiro aleatório dentro dos limites permitidos. Gerações subsequentes somam cada eixo de forma independente com o valor -1, 0 ou 1 (escolha aleatória)
cr_r	Componente c_t=R	C	Resistência elétrica do resistor, em Ω	$1 \leq cr_r \leq 60 \times 10^6$	$cr_r = a \times 10^b$, onde a é um número aleatório uniforme no intervalo $[1, 10]$ e b é um inteiro aleatório no intervalo [0,6] escolhido considerando os pesos [5/135, 20/135, 30/135, 40/135, 25/135, 10/135, 5/135]
cq_{te}, cq_{tb}, cq_{tc}	Componente c_t=Q	T	Posição dos terminais emissor, base e coletor do transistor	$(x, y) : 0 \leq x \leq M-1; 0 \leq y \leq N-1$	Na primeira geração, retorna um inteiro aleatório dentro dos limites permitidos. Gerações subsequentes somam cada eixo de forma independente com o valor -1, 0 ou 1 (escolha aleatória)
cq_m	Componente c_t=Q	D	O modelo SPICE do transistor	$cq_m \in \{M_0, M_1, \ldots, M_{n-1}\}$, onde M é o nome do modelo SPICE (string). Configurado em cada problema. Exemplo: $cq_m \in \{2N3904, 2N3906\}$	Escolha aleatória de um item do conjunto de strings, com pesos iguais
cma_{tg}, cma_{td}, cma_{ts}	Componente c_t=Ma	T	Posição dos terminais porta, dreno e fonte do MOSFET	$(x, y); 0 \leq x \leq M-1; 0 \leq y \leq N-1$	Na primeira geração, retorna um inteiro aleatório dentro dos limites permitidos. Gerações subsequentes somam cada eixo de forma independente com o valor -1, 0 ou 1 (escolha aleatória)
cma_m	Componente c_t=Ma	D	O modelo SPICE do MOSFET	$cma_m \in \{M_0, M_1, \ldots, M_{n-1}\}$, onde M é o nome do modelo SPICE (string). Configurado em cada problema. Exemplo: $cma_m \in \{NMOS1, PMOS1\}$	Escolha aleatória de um item do conjunto de strings, com pesos iguais
cma_w	Componente c_t=Ma	C	Largura do canal do MOSFET (μm)	$cma_w \in \mathbb{N}^+, w_{min} \leq n \leq w_{max}$. Configurado em cada problema.	Número aleatório uniformemente distribuído dentro da faixa prescrita

[1] : Legenda: (E) Estrutura; (T) Terminal; (C) Contínuo; (D) Discreto.

Fonte: o autor (2023)

4.3 Conclusão

Este capítulo apresentou o SANN-GCE, um algoritmo para a síntese automática de circuitos analógicos, composto pelo algoritmo Recozimento Simulado e pela representação de solução Evolução Geométrica de Circuitos. Foram apresentadas a divisão entre o algoritmo de busca e a representação da solução, que se justifica pela modularidade imposta pelo framework em que o SANN-GCE está baseado.

A representação de solução GCE foi apresentada, e seus principais conceitos, como a sua hierarquia de objetos, incluindo a placa, componentes e os DOFs, que possibilitam que características distintas de um circuito sejam alteradas com probabilidades diferentes durante o processo de construção da solução, foram detalhados.

Também foram evidenciados o funcionamento do algoritmo de busca recozimento simulado, que conduz o processo de otimização, e sua interface com o GCE. Foi apresentado o algoritmo de mutação, que modifica a solução candidata para um estado vizinho, possibilitando que o algoritmo de busca explore o espaço de busca de forma eficaz. Também foi apresentado o algoritmo de tradução de uma solução candidata para uma netlist SPICE, que permite que a solução candidata seja enviada para um simulador de circuitos e tenha as suas métricas de desempenho, como a aptidão, calculadas.

O SANN-GCE, em conjunto com o framework modular `circ_autoproj` e o simulador de circuitos Ngspice, apresentados nos capítulos 2 e 3, respectivamente, compõe o desenvolvimento central deste trabalho: um software que sintetiza projetos de circuitos analógicos automaticamente, a partir de uma descrição de alto-nível do desempenho esperado do circuito.

Os capítulos seguintes a este irão aplicar os conceitos apresentados até aqui, começando pelo capítulo 5, que introduz o uso de recursos da computação em nuvem, imprescindível para acelerar a execução do SANN-GCE. Posteriormente, os capítulos 6 e 7 irão abordar a aplicação desta solução em diversos casos de uso e discutirão seus resultados.

Capítulo 5

Execução na Nuvem

Devido à grande demanda computacional, as simulações descritas neste trabalho foram, na maioria, executadas utilizando recursos de *computação em nuvem*. Neste capítulo, serão introduzidos os principais conceitos deste tema, seguido por um breve histórico e características dos dois provedores utilizados. Depois, será explicado como se deu a aplicação dos recursos da nuvem neste trabalho, e por fim será apresentada uma descrição detalhada das configurações de hardware e software empregadas.

O apêndice E é um complemento importante deste capítulo, pois apresenta dois tutoriais que explicam o passo-a-passo para a criação e acesso de máquinas virtuais nos provedores Google Cloud e Amazon Web Services. Estes podem ser úteis ao leitor em caso de necessidade do uso de recursos semelhantes em outros trabalhos.

5.1 Introdução

A definição do NIST (*National Institute of Standards and Technology*) para computação em nuvem (do inglês *Cloud computing*) é:

> Computação em nuvem é um modelo para permitir acesso de rede onipresente, conveniente e sob demanda a um conjunto compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e liberados com esforço mínimo de gerenciamento ou interação do provedor de serviços. Este modelo de nuvem é composto por cinco características essenciais, três modelos de serviço e quatro modelos de implantação. (MELL; GRANCE, 2011, p. 2, tradução nossa)

As cinco características essenciais, segundo o autor supracitado, são:

1. Autosserviço sob demanda: o consumidor pode, unilateralmente, provisionar recursos de computação;

2. Acesso amplo à rede: recursos estão disponíveis pela rede (internet) e podem ser acessados por mecanismos padrões (como por uma página web ou via *ssh*);

3. Agrupamento de recursos: os recursos de computação do provedor são agrupados para atender vários consumidores usando um modelo compartilhado, com diferentes recursos físicos e virtuais atribuídos dinamicamente conforme a demanda dos consumidores;

4. Elasticidade rápida: as capacidades podem ser provisionadas e liberadas de forma elástica, em alguns casos automaticamente, para escalar rapidamente proporcionalmente à demanda. Para o consumidor, as capacidades disponíveis para provisionamento muitas vezes parecem ilimitadas e podem ser apropriadas em qualquer quantidade e a qualquer momento;

5. Serviço medido: o uso de recursos (por exemplo, armazenamento, processamento, largura de banda) é monitorado, controlado e relatado pelo provedor, proporcionando transparência para o consumidor.

Ainda de acordo com o referido autor, os modelos de serviço são:

1. Software como serviço (do inglês *Software as a Service – SaaS*): o provedor fornece aplicativos executados em sua infraestrutura. Estes são acessíveis a partir de diversos dispositivos clientes por meio de interfaces como um navegador da Web (por exemplo um *webmail*) ou um programa nativo;

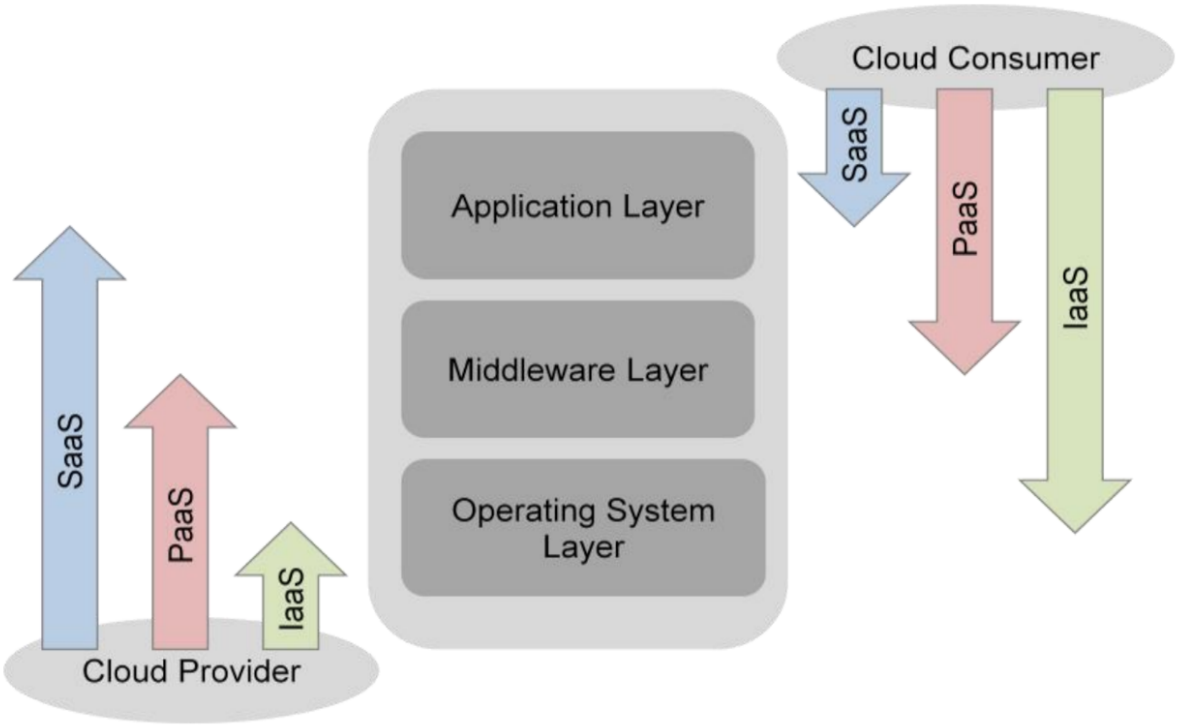
2. Plataforma como serviço (*Platform as a Service – PaaS*): o provedor fornece acesso para que o cliente possa implantar na nuvem aplicativos criados ou adquiridos pelo consumidor, utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor;

3. Infraestrutura como serviço (*Infrastructure as a Service – IaaS*): o provedor de nuvem fornece acesso direto aos recursos computacionais fundamentais (como processamento, armazenamento e redes) de forma que o cliente seja capaz de implantar e executar software arbitrário (exemplos: sistemas operacionais e aplicativos).

A Figura 16 ilustra o grau de controle entre o provedor e o consumidor para os modelos de serviço SaaS, PaaS e IaaS. É notável que o modelo IaaS provê maior controle para o consumidor da nuvem, pois este permite que o cliente acesse diretamente os recursos computacionais fundamentais, ao nível do sistema operacional.

Mell e Grance (2011) citam quatro modelos de implantação. São eles:

Figura 16 – Graus de controle entre o provedor e o consumidor para os diferentes modelos de serviço



Fonte: Liu et al. (2011)

1. Nuvem privada: a infraestrutura em nuvem é fornecida para uso exclusivo por uma única organização composta por vários consumidores (por exemplo, unidades de negócios). Pode pertencer, ser gerenciado e operado pela organização, por terceiros ou por alguma combinação deles, e pode existir dentro ou fora das instalações;

2. Nuvem comunitária: a infraestrutura em nuvem é fornecida para uso exclusivo por uma comunidade específica de consumidores de organizações que compartilham de requisitos semelhantes (por exemplo: requisitos de segurança ou políticas de conformidade). Pode pertencer, ser gerido e operado por uma ou mais organizações da comunidade, por um terceiro, ou alguma combinação deles, e pode existir dentro ou fora das instalações;

3. Nuvem pública: A infraestrutura em nuvem é fornecida para uso aberto pelo público geral. Pode pertencer, ser gerenciado e operado por uma organização empresarial, acadêmica ou governamental, ou alguma combinação delas. Existe nas instalações do provedor de nuvem;

4. Nuvem híbrida: composição de duas ou mais infraestruturas de nuvem distintas.

De acordo com Erl e Monroy (2023), as principais motivações para o uso da computação em nuvem em vez do emprego de uma solução tradicional no local são a

busca pela redução de custos e a crescente necessidade de agilidade nos negócios para enfrentar novas demandas computacionais. De forma complementar, Ruparelia (2016) aponta como vantagens a não necessidade de investimento inicial em infraestrutura e a alta disponibilidade dos serviços providos pela nuvem.

A redução de custos ocorre pela maior eficiência no uso da infraestrutura de um provedor quando comparado com uma estrutura equivalente montada no cliente, pois a infraestrutura (servidores e equipamentos de rede, por exemplo) de um provedor é compartilhada por muitos clientes. Além disso, os serviços providos pela nuvem são pagos pelo uso, portanto não há necessidade de mobilização de capital.

A maior agilidade é alcançada através da facilidade e rapidez em que é possível aumentar a escala de uma solução na nuvem – por exemplo, se o desempenho de uma aplicação estiver abaixo do aceitável em razão de alguma limitação do hardware, na nuvem é possível rapidamente trocar o tipo de máquina virtual (com mais memória e processadores, por exemplo), ou mesmo distribuir a carga entre várias máquinas. Isto fora da nuvem, em uma solução tradicional dentro das instalações do cliente, demandaria a aquisição, instalação e comissionamento de novos equipamentos, o que certamente levaria muito mais tempo.

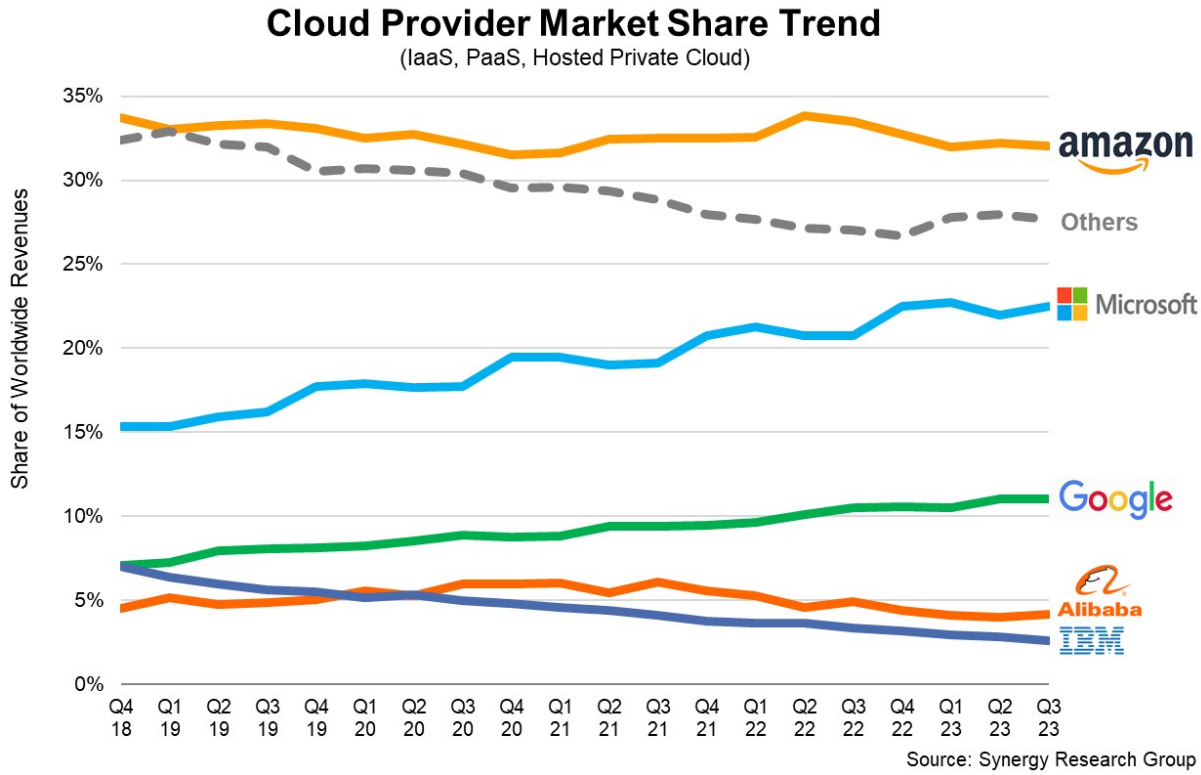
Como os provedores de nuvem normalmente são grandes empresas de tecnologia que dispõe de mão de obra qualificada e grande capacidade de investimento em infraestrutura, é de se esperar que seus serviços possuam alto grau de disponibilidade. De fato, é comum para os maiores provedores de nuvem garantirem em seus Acordos de Nível de Serviço taxas de disponibilidade entre 99,5% e 99,99% (GOOGLE, 2023a; AMAZON, 2022).

5.2 Principais provedores

Os recursos de computação em nuvem utilizados neste trabalho empregaram o modelo de serviço *infraestrutura como serviço*, pois a instalação e execução do framework `circ_autoproj` requer acesso a nível de sistema operacional (especificamente o Linux). O modelo de implantação empregado foi o da *nuvem pública*, onde diversos provedores de nuvem competem fornecendo serviços equivalentes para o público geral. Isto nos permite escolher o fornecedor de menor custo.

O mercado de provedores públicos de serviços em nuvem é composto por diversas empresas nacionais e estrangeiras. Segundo o Synergy Research Group (2023), os cinco provedores com maior participação de mercado mundial no terceiro trimestre de 2023 são, em ordem decrescente de receita: Amazon, Microsoft, Google, Alibaba e IBM. O provedor de nuvem do Alibaba está sediado na China e os demais estão nos Estados Unidos da América. A Figura 17 apresenta a evolução da fatia de mercado destes provedores em cinco anos, até o terceiro trimestre de 2023.

Figura 17 – Evolução da participação de mercado dos provedores de serviços em nuvem



Fonte: Synergy Research Group (2023)

Neste trabalho foram utilizados serviços de infraestruturas em nuvem (IaaS) do Google e da Amazon. Portanto, o restante deste capítulo será dedicado aos principais recursos de computação em nuvem fornecidos por estas duas empresas.

A nuvem da Amazon se chama *Amazon Web Services* – AWS e seu serviço de computação em nuvem (IaaS) se chama *Elastic Compute Cloud* – EC2, lançado em agosto de 2006 como um serviço em fase *beta* para um público restrito. Em outubro de 2007, ainda em fase beta, o EC2 foi aberto para o público geral. Um ano depois, a fase beta se encerrou, iniciando-se a oferta do serviço com um Acordo de Nível de Serviço (BARR, 2006; AMAZON, 2007; AMAZON, 2008).

Atualmente a AWS possui *data centers* em 32 regiões geográficas distribuídas pelo mundo. O EC2 permite a criação de máquinas virtuais (conhecido também como *instâncias*) a partir da escolha de um sistema operacional pré-configurado (chamado de *Amazon Machine Images* – AMIs). Há diversas opções de AMIs, como o Microsoft Windows além de diferentes distribuições Linux, como o Amazon Linux, Ubuntu, Red Hat Enterprise Linux, CentOS, SUSE e o Debian.

O EC2 possui diversos tipos de instâncias: 1. propósito geral; 2. otimizado para computação; 3. otimizado para memória; 4. computação acelerada; 5. otimizado para armazenamento; 6. otimizado para HPC (sigla do inglês *High Performance Computing*). Cada

tipo de instância possui diferentes tamanhos (ou configurações) de máquinas disponíveis: desde 1 vCPU[1] e 0,5 GB de memória RAM até máquinas com 192 vCPUs e 1.536 GB de memória RAM. Os processadores das instâncias podem ser AMD, Intel ou Graviton[2].

O Google, por sua vez, fornece serviços em nuvem através da marca *Google Cloud*. O primeiro serviço oferecido foi o *App Engine*, no modelo PaaS, lançado em abril de 2008 como uma versão prévia. Em novembro de 2011 o serviço deixa de ser considerado uma versão prévia passando a ser um *produto totalmente suportado* (MCDONALD, 2008; GOOGLE, 2011).

O *Compute Engine* é serviço de computação em nuvem da Google Cloud que provê acesso ao nível do sistema operacional (IaaS). Foi lançado em junho de 2012 como versão prévia, sendo disponibilizado para o público geral em dezembro de 2013 (MAGNUSSON, 2012; BALOGH, 2013). Assim como no seu concorrente Amazon EC2, há uma grande variedade de sistemas operacionais Windows e Linux pré-configurados disponíveis para a criação de instâncias.

Atualmente o Compute Engine possui data centers em 39 regiões geográficas do mundo. Suas instâncias são agrupadas em 4 famílias: 1. propósito geral; 2. otimizado para computação; 3. otimizado para memória; 4. otimizado para aceleração. São ofertadas máquinas virtuais desde 0,25 vCPUs[3] e 0,5 GB de memória RAM até 416 vCPUs e 11.776 GB de memória RAM. O processador pode ser AMD, Intel ou Ampere Altra[4].

5.3 Aplicação neste trabalho

No início, os testes do framework `circ_autoproj` e seus algoritmos foram feitos em um computador local que possuía uma configuração até então considerada razoável[5]. No entanto, ao longo do tempo, foi observado que as simulações computacionais levavam muito tempo para terminar, o que atrasava a obtenção dos resultados para os casos de uso selecionados, prejudicando os ciclos de desenvolvimento e testes do referido software.

Em vista disso, buscou-se utilizar serviços de nuvem para acelerar as simulações computacionais. O primeiro uso foi do Google Cloud, por meio do seu programa de avaliação gratuita que concede US$ 300 para uso em alguns serviços, no período de até 90 dias (GOOGLE, 2023b). Uma restrição importante deste programa gratuito é que a máquina virtual é limitada a 8 vCPUs – a mesma quantidade de *threads* de CPU do hardware disponível localmente. Apesar desta restrição gerar um aparente empate, o uso

[1] Uma vCPU é tipicamente uma thread de um núcleo de CPU
[2] Graviton é uma CPU de arquitetura ARM desenvolvida pela própria Amazon
[3] A fração de uma vCPU é possível no modelo shared-core, onde uma vCPU é compartilhada com outras instâncias
[4] Processador com arquitetura ARM do fabricante Ampere Computing
[5] Processador Intel Core i7 2,4 GHz e 8 GB de memória RAM

da nuvem se mostrou muito benéfico, pois os processadores da máquina virtual eram mais rápidos, da classe de servidor (Intel Xeon ou AMD EPYC), e o uso de uma máquina (virtual) dedicada para as simulações deixava o computador local livre para outras tarefas de desenvolvimento.

A sua concorrente AWS também possui um programa de testes gratuito com duração de 12 meses chamado de *AWS Free Tier*, onde são oferecidas 750 horas/mês para instâncias tipo t2.micro ou t3.micro (AMAZON, 2023b), que possuem apenas 1 ou 2 vCPUs e 1 GB de memória RAM. Evidentemente, esta oferta gratuita da Amazon não era interessante neste trabalho, pois seu desempenho seria muito inferior ao da máquina local.

No entanto, a AWS possui um programa de créditos para pesquisas denominado *AWS Public Sector Cloud Credit for Research* (AMAZON, 2023a), no qual o autor deste trabalho se candidatou e, após 20 meses de espera, foi contemplado com US$ 9.856,00 para uso em 12 meses, sem restrições importantes[6]. Sendo assim, após a avaliação gratuita do Google Cloud ter se encerrado, decidiu-se migrar para uma instância na AWS com grande poder computacional, custeada por meio do referido crédito de pesquisa obtido.

5.3.1 Queue Manager

Para otimizar o uso do recurso computacional fornecido pela nuvem, foi desenvolvido um programa chamado *Queue Manager*. Ele é instalado na máquina virtual (servidor) criada na nuvem, e tem como função gerenciar uma fila de execução de tarefas (*jobs*).

O *Queue Manager* agenda a execução das tarefas de modo que haja uma ocupação ideal dos processadores disponíveis no servidor[7]. Além disso, se configurado, quando todas as tarefas agendadas se encerram ele desliga a máquina virtual, evitando deixá-la ligada e ociosa, reduzindo o custo.

As tarefas são arquivos-pacotes padronizados (`.tar.gz`) contendo um arquivo de configuração da tarefa (`job_config.json`) e um script *bash* (`run`) que pode invocar qualquer outro programa. No caso deste trabalho, o script executa o framework `circ_autoproj`, com as devidas configurações para sintetizar um circuito eletrônico analógico específico, e ao final grava o resultado da execução em um arquivo para posterior análise.

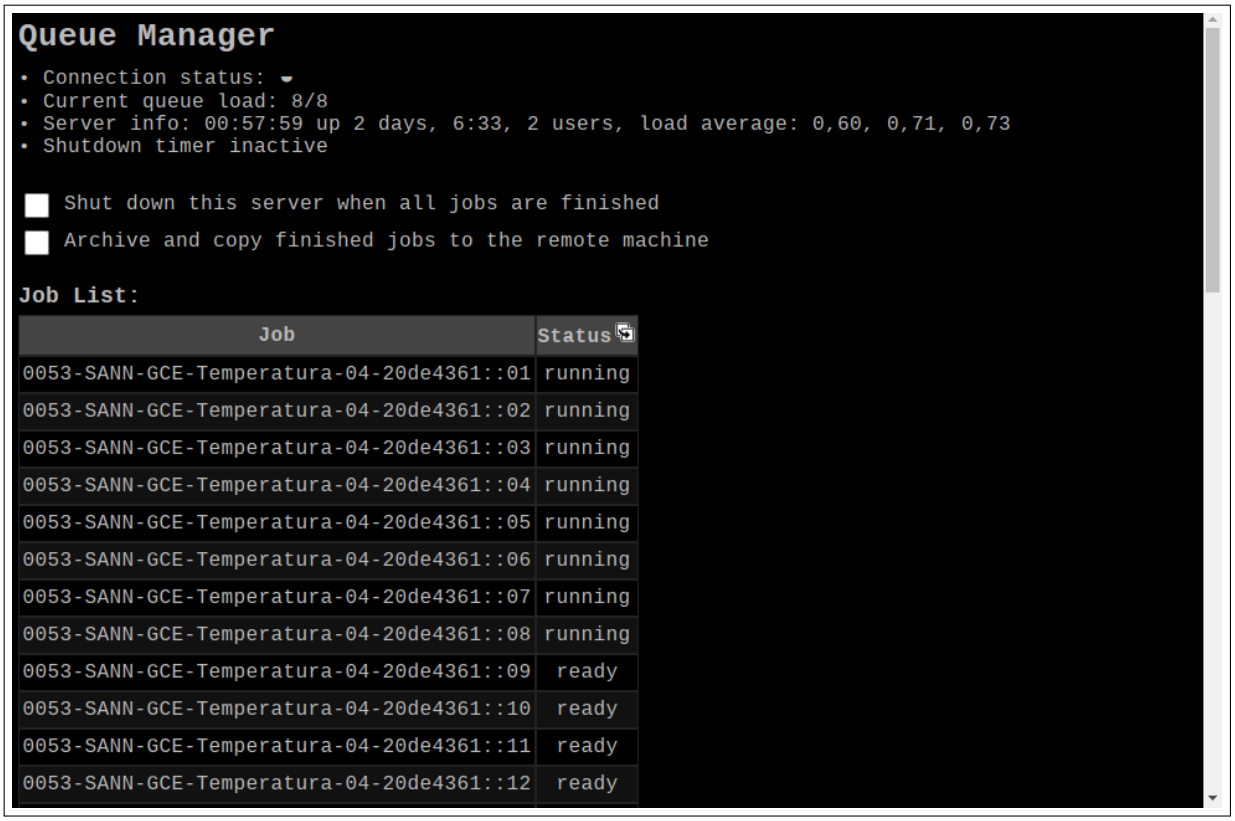
A Figura 18 apresenta a tela do *Queue Manager*. Este software foi desenvolvido na linguagem *Javascript* e é executado no ambiente *Node.js*. Sua interface com o usuário é uma página web, que pode ser acessada por um navegador comum.

---

[6] Considera-se uma restrição importante, por exemplo, a eventual limitação do número de processadores. Isto não ocorre no programa de créditos para pesquisas da AWS.

[7] Ocupação ideal dos processadores foi definida como sendo a que ocorre quando há exatamente um processo de alta demanda (`circ_autoproj`) em execução para cada núcleo de processador. Assim, consegue-se uma ocupação de 100% de cada núcleo, evitando a sobrecarga da troca de contexto que ocorreria caso houvesse mais de um processo de alta demanda para cada núcleo

Figura 18 – Tela do *Queue Manager*



Fonte: o autor (2023)

Para agendar a execução de uma tarefa (ou conjunto de tarefas) no servidor, basta abrir a página do *Queue Manager* no navegador web e arrastar-e-soltar o(s) arquivo(s) pacote(s) de tarefa(s) previamente preparado(s) na página do programa. Assim que isso ocorre, o *Queue Manager* analisa o arquivo de configuração da tarefa `job_config.json` de cada pacote e a coloca no final da fila de execução.

O arquivo de configuração da tarefa é escrito no formato JSON (BRAY, 2017), e utiliza os seguintes campos:

```
{
        "desc": string,
        "load_per_run": number,
        "runs": number
}
```

O campo `desc` é o nome da tarefa. Já o `load_per_run` é a carga de processador declarada de uma execução desta tarefa, ou simplesmente *carga da tarefa*. O `runs` é o número de vezes que esta tarefa deve ser executada. Este último é um parâmetro muito útil, pois neste trabalho cada simulação é sempre repetida 50 vezes.

O *Queue Manager* possui diversos parâmetros configuráveis, dentre eles o `max_load`. Este parâmetro é a carga de processador máxima admitida, ou simplesmente *carga máxima da fila*. Normalmente este valor é ajustado para a quantidade de núcleos de CPU da máquina.

O algoritmo de agendamento começa executando as tarefas por ordem de criação, somando os valores das cargas das tarefas até que seja atingido a carga máxima da fila. Quando isto ocorre, as tarefas seguintes ficam aguardando a sua vez para um agendamento posterior. Quando uma tarefa se encerra, a carga da fila cai abaixo da carga máxima da fila e a tarefa seguinte é agendada caso a sua carga somada à carga atual da fila não ultrapasse a sua carga máxima.

Outra funcionalidade importante deste programa é o *Archive and copy finished jobs to the remote machine* (vide Figura 18). Se ativada, quando uma tarefa termina, o seu diretório de trabalho (que normalmente inclui o resultado da sua execução) é empacotado e copiado para outra máquina especificada na configuração. Isto é particularmente útil quando combinado com a opção *Shut down this server when all jobs are finished*, pois reduz custos em função do desligamento automático do servidor e, ao mesmo tempo, deixa os resultados disponíveis em uma segunda máquina (normalmente a máquina local) para posterior análise.

O uso do *Queue Manager* agilizou as execuções das simulações computacionais deste trabalho, pois possibilitou a execução de diversas tarefas (simulações) em paralelo e de forma organizada, sem deixar o processador ocioso, sempre desligado a máquina virtual após a execução de um lote de tarefas programadas. Isto reduziu o tempo de execução do conjunto de simulações desejado e minimizou o número de horas em que a máquina virtual esteve ligada, reduzindo os custos e consequentemente prolongando o uso dos créditos obtidos (considerando a AWS).

5.4 Google Cloud

Esta seção descreve as configurações da plataforma Google Cloud utilizadas neste trabalho.

Foi configurada uma instância com o maior número de CPUs possível dentro do programa de avaliação gratuita do Google. Esta instância é denominada `e2-highcpu-8`, pertence à família *propósito geral* e possui as seguintes características:

- CPU: Intel(R) Xeon(R) CPU @ 2.30GHz ou AMD EPYC 7B12;
- vCPUs: 8 (4 núcleos, 2 *threads* por núcleo);
- Memória RAM: 8 GB;

- Sistema Operacional: Debian GNU/Linux 10 (buster).

As CPUs de fabricação AMD e Intel descritas acima possuem desempenho semelhante. A escolha de uma ou outra não é feita pelo cliente, mas sim automaticamente pelo provedor, conforme a disponibilidade de hardware no momento da partida da instância.

A instância foi configurada na região `us-east1`, localizada no estado da Carolina do Sul, nos Estados Unidos da América. A escolha por esta região se deu exclusivamente pelo menor custo. Cada hora de utilização desta máquina virtual custa US$ 0,197872 (GOOGLE, 2023c), portanto os US$ 300 disponíveis no programa de avaliação gratuita foram suficientes para o uso equivalente de 63 dias ininterruptos.

Dentre diversas opções de distribuições Linux (como CentOS, Debian, Fedora, Red Hat, SUSE e Ubuntu) foi escolhido o sistema operacional Debian versão 10 (codinome buster) – a versão mais recente disponível na época de criação da máquina virtual. Esta escolha se deu pela familiaridade do autor, pela vasta quantidade de pacotes disponíveis e pela sua grande estabilidade.

5.4.1 Configuração do Sistema Operacional

A seção E.1 do apêndice E contém um tutorial detalhado para a criação de uma instância computacional na nuvem Google Cloud. Após ter acesso à máquina virtual criada, é necessário configurar o seu sistema operacional para executar o framework `circ_autoproj`. Considerando o sistema operacional Debian, isto pode ser feito através dos seguintes comandos:

```
# cópia dos modelos de componentes spice da máquina local para a máquina
↪ virtual:
scp -r ~/ngspice <usuario>@<end_ip>:.

# atualização do sistema operacional da máquina virtual:
sudo apt-get update
sudo apt-get upgrade
sudo reboot

# instalação de pacotes adicionais:
sudo apt-get install gcc g++ screen git cmake wget libboost-all-dev
↪ libboost-dev parallel

# download, compilação e instalação do ngspice (somente shared lib)
cd /tmp
```

```
mkdir ngspice
cd ngspice
wget https://downloads.sourceforge.net/project/ngspice/ng-spice-rework/
↪  41/ngspice-41.tar.gz
cd ngspice-41/
./configure  --prefix=/usr --libdir=/usr/lib --sysconfdir=/etc
↪  --with-ngshared  --disable-debug  --enable-xspice --enable-cider
↪  --enable-openmp
make -j8
sudo make install
```

5.5 Amazon Web Services

Nesta seção são apresentadas as configurações da plataforma AWS utilizadas neste trabalho.

Com a obtenção dos créditos de pesquisa da AWS, foi configurada uma instância do tipo `c6i.32xlarge` no *Amazon Elastic Compute Cloud* - EC2 no regime de cobrança *sob demanda*. Esta instância computacional foi criada na região denominada `us-east-1`, localizada fisicamente no estado da Virgínia nos Estados Unidos da América. Ela possui as seguintes características:

- CPU: Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz
- vCPUs: 128 (64 núcleos, 2 *threads* por núcleo);
- Memória RAM: 256 GB;
- Sistema Operacional: Amazon Linux 2.

Este tipo de instância pertence à classe de instâncias otimizadas para a computação, que tipicamente são cargas de trabalho em lote, como transcodificação de vídeo, modelagem científica, inferência no aprendizado de máquina, dentre outras aplicações computacionalmente intensivas (AMAZON, 2023c).

A tarifação do uso desta instância ocorre pelo tempo contado entre as suas partidas e paradas, ao custo de US$ 5,44/hora, registrado com a granularidade de 1 segundo, obedecendo uma cobrança mínima de 60 segundos por partida (AMAZON, 2023d). Considerando este custo horário para uso da instância, o valor de US$ 9.856,00 obtido como crédito de pesquisa da AWS pôde custear aproximadamente 75 dias de uso contínuo.

O Amazon Linux 2 é baseado na família de distribuições Red Hat (assim como o CentOS e o Fedora) portanto possui facilidades semelhantes como o gerenciamento de pacotes RPM através do comando `yum`.

5.5.1 Configuração do Sistema Operacional

A seção E.2 do apêndice E apresenta um passo-a-passo a criação de uma máquina virtual na nuvem AWS. Após ter criado a instância na AWS e configurado o seu acesso através do `ssh`, é necessário configurar o seu sistema operacional para executar o framework `circ_autoproj`. No Amazon Linux (derivado do Fedora Linux), isto pode ser feito executando os comandos abaixo:

```
# cópia dos modelos de componentes spice da máquina local para a máquina
↪ virtual:
scp -r ~/ngspice <usuario>@<end_ip>:.

# atualização do sistema operacional da máquina virtual:
sudo yum update
sudo reboot

# instalação de pacotes adicionais:
sudo yum install gcc10 gcc10-c++ libboost boost boost-devel cmake3 git
↪ jq
sudo ln -s /usr/bin/gcc10-gcc /usr/bin/gcc
sudo ln -s /usr/bin/gcc10-g++ /usr/bin/g++
sudo ln -s /usr/bin/cmake3 /usr/bin/cmake

# download, compilação e instalação do ngspice (somente shared lib)
cd /tmp
mkdir ngspice
cd ngspice
wget https://downloads.sourceforge.net/project/ngspice/ng-spice-rework/
↪ 41/ngspice-41.tar.gz
cd ngspice-41/
./configure  --prefix=/usr --libdir=/usr/lib --sysconfdir=/etc
↪ --with-ngshared  --disable-debug  --enable-xspice --enable-cider
↪ --enable-openmp
make -j8
sudo make install
```

Capítulo 6

Casos de uso

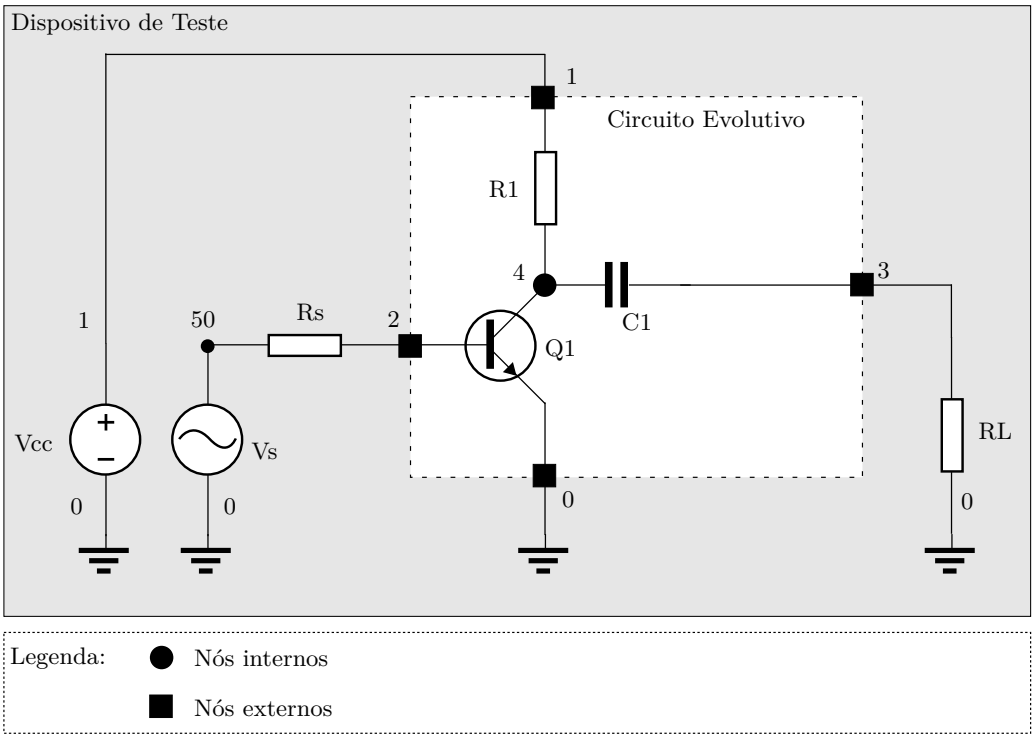
Este capítulo apresenta os casos de uso (ou *problemas SPICE*) selecionados para testar o algoritmo SANN-GCE e os parâmetros escolhidos para estes problemas. Antes, porém, serão introduzidos os principais conceitos para a especificação de um problema, seguido pela apresentação das fórmulas das métricas de desempenho dos circuitos calculadas pelo algoritmo.

6.1 Especificação de um problema SPICE

A especificação de um circuito para ser sintetizado no SANN-GCE (*problema SPICE*) compreende os seguintes passos:

- Definir um Dispositivo de Teste (DT), que é um circuito fixo que fornece energia e testa o circuito que será criado – chamado aqui *circuito evolutivo* (CE). A interface entre DT e CE são os *nós externos*. A Figura 19 apresenta um exemplo de um circuito eletrônico com a demarcação do DT, do CE e o apontamento dos nós externos e internos. Os nós internos são os nós do CE que não fazem interface com o DT;
- Especificar a(s) Variável(eis) Manipulada(s) (VM). São parâmetros do circuito DT alterados para testar o CE;
- Prescrever a(s) Variável(eis) de Resposta (VR). Geralmente é a tensão ou corrente no(s) nó(s) do DT que indica a resposta do CE ao estímulo dado pelas VMs;
- Escrever o script SPICE que conduz a simulação, baseado nos itens anteriores desta lista. O script deve conter o tipo de análise a ser executada e as variáveis a serem salvas para o cálculo da aptidão. Pode conter modelos de componentes e opções de simulação;
- Escrever a função para calcular a aptidão. Sua entrada é a saída da simulação SPICE, normalmente uma coleção de arrays que será comparada com uma saída de referência.

Figura 19 – Exemplo de circuito indicando a demarcação do Dispositivo de Teste, do Circuito Evolutivo e o apontamento dos nós internos e externos



Fonte: o autor (2023), baseada em Castejón e Carmona (2018)

O resultado desta comparação será a saída da função: um número real que representa a aptidão do CE. Quanto menor a aptidão, melhor o circuito. Esta é a *métrica primária* do problema;

- Escrever funções para o cálculo das métricas secundárias. Isso é opcional e seus resultados são utilizados apenas para fins de registro (diferentemente da métrica primária, estes não são usados para guiar o algoritmo de busca).

6.2 Métricas de desempenho

Durante o processo de otimização, cada solução candidata é avaliada para se obter algumas métricas de desempenho. Estas são divididas em duas categorias: primária e secundária. A métrica primária (ou principal) é unicamente a *aptidão* f da solução. Ela é calculada a cada iteração do algoritmo de busca e o orienta para que o valor numérico de f seja minimizado, o que consequentemente produz circuitos eletrônicos com uma resposta mais próxima da desejada. As métricas secundárias são opcionais e não são usadas para guiar o algoritmo de busca, servindo apenas como indicadores extras do desempenho da solução.

Esta seção apresenta as fórmulas de cálculo destas métricas e seus parâmetros, compartilhados por todos os circuitos de teste utilizados neste trabalho. Suas fórmulas e nomenclaturas seguiram os artigos de Castejón e Carmona (2018) e Castejón e Carmona (2020) (chamados a partir daqui de *artigos de referência*) com o objetivo de tornar possível a comparação de seus resultados.

A aptidão f é calculada através da seguinte equação:

$$f = \sum_{i=0}^{N_p-1} w_i e_i \tag{6.1}$$

Onde N_p é o número de pontos no resultado da simulação SPICE, e e_i é o erro absoluto entre a variável de resposta do circuito (saída) X_i e o valor desejado (referência) $\tilde{X}_i$, calculado da seguinte forma:

$$e_i = |X_i - \tilde{X}_i| \tag{6.2}$$

Na Equação (6.1), w_i é um *fator de ponderação* calculado conforme a Equação (6.3). Este fator aumenta a penalidade para a aptidão f quando há um erro absoluto acima de um limite predefinido X_{th}. Na mesma equação, w_b é a *ponderação básica* que pode ser definida de forma diferente para cada circuito de teste.

$$w_i = \begin{cases} w_b & e_i \leq X_{th} \\ 10 \cdot w_b & \text{caso contrário} \end{cases} \tag{6.3}$$

Em todos os casos de uso selecionados, são calculadas duas métricas secundárias. A primeira delas é o *Mean Absolute Error - MAE* (erro médio absoluto), obtida de acordo com a seguinte equação:

$$MAE = \frac{1}{N_p} \sum_{i=0}^{N_p-1} e_i \tag{6.4}$$

A segunda métrica secundária é a porcentagem de êxito *hits %*, que indica a proporção dos pontos do resultado SPICE que tiveram erro absoluto abaixo do limite X_{th}. É calculada pela equação:

$$hits_{\%} = \frac{1}{N_p} \sum_{i=0}^{N_p-1} [e_i \leq X_{th}] \tag{6.5}$$

6.3 Circuitos de teste

Para testar o SANN-GCE, foram selecionados sete problemas (circuitos de teste) como casos de uso para o algoritmo. Esses circuitos de teste foram os mesmos dos artigos de referência supracitado, pois os referidos trabalhos apresentaram 7 casos de uso e mostraram resultados detalhados. Além disso, todos os testes foram executados 50 vezes, o que permitiu fazer uma comparação mais robusta.

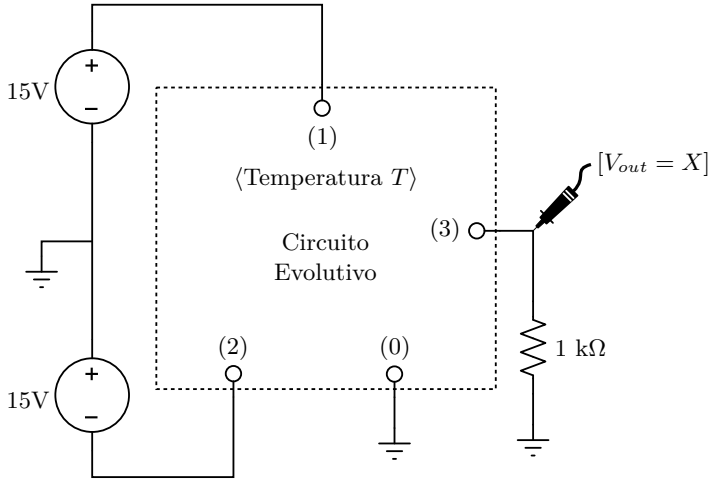
Os circuitos de teste foram divididos em dois conjuntos: 1. Circuitos não computacionais, compostos por um circuito sensor de temperatura, uma função Gaussiana e uma referência de tensão; 2. Circuitos computacionais: função quadrática, raiz quadrada, função cúbica e raiz cúbica.

A subseções seguintes descrevem cada um destes circuitos de teste em detalhes.

6.3.1 Sensor de temperatura

O problema do sensor de temperatura pretende projetar automaticamente um circuito eletrônico analógico que gere uma tensão de saída desejada $\tilde{X}$ (em volts) proporcional à temperatura do circuito T (em graus Celsius), seguindo a relação $\tilde{X} = T/10$. A Figura 20 apresenta o dispositivo de teste usado neste problema. Nessa figura, os números dos nós externos estão entre parênteses, as variáveis manipuladas estão entre parênteses angulares e as variáveis de resposta estão entre colchetes. O circuito é avaliado através da sua simulação SPICE para cada uma das temperaturas $T = [0, 5, 10, \ldots, 100]$. O resultado da simulação é um conjunto de $N_p = 21$ valores de tensão medidas no nó #3 (sua tensão de saída X).

Figura 20 – Dispositivo de teste para o circuito do sensor de temperatura



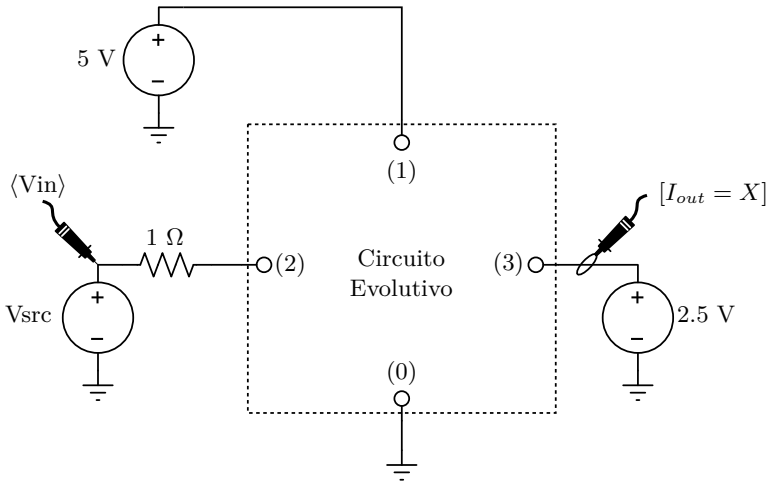
Fonte: o autor (2023)

6.3.2 Função Gaussiana

O circuito da função Gaussiana tenta aproximar a curva da função matemática de mesmo nome. A Figura 21 apresenta o dispositivo de teste usado neste circuito. Nessa figura, a representação dos nós externos, variáveis manipuladas e variáveis de resposta é a mesma utilizada na figura do dispositivo de teste apresentado na subseção anterior. A fonte de entrada muda sua tensão linearmente seguindo os valores de tensão $Vin = [2; 2{,}01; 2{,}02; \ldots; 3]$, resultando em $N_p = 101$ pontos de simulação. A saída X do circuito é a corrente que flui do nó #3. A saída desejada para a i-ésima entrada é dada pela Equação (6.6), onde $a = 80 \times 10^{-9}$ é o valor de pico da função Gaussiana, $b = 2.5$ é o valor médio da distribuição e $c = 0{,}1$ é o seu desvio padrão.

$$\tilde{X}_i = a \exp\left(- \frac{(Vin_i - b)^2}{2c^2} \right) \tag{6.6}$$

Figura 21 – Dispositivo de teste para o circuito da função gaussiana

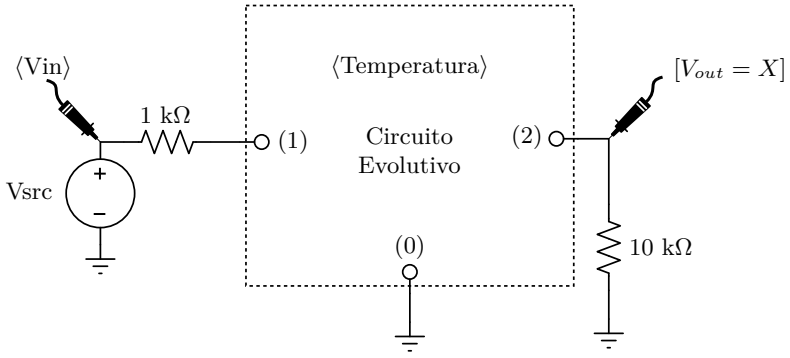


Fonte: o autor (2023)

6.3.3 Referência de tensão

O circuito de referência de tensão visa manter sua tensão de saída o mais constante possível quando sujeito às variações na tensão de alimentação e na temperatura do circuito. Seu dispositivo de teste é mostrado na Figura 22. Nessa figura, a representação dos nós externos, variáveis manipuladas e variáveis de resposta segue as demais figuras dos dispositivos de teste anteriores. A temperatura muda de acordo com os valores $T = [0, 25, 50, 75, 100]$, e para cada temperatura, a tensão de alimentação muda segundo a sequência $Vin = [4; 4{,}1; 4{,}2; \ldots; 6]$, totalizando assim $5 \times 21 = 105$ pontos de simulação. A saída do circuito é a tensão X, medida no nó #2. A saída desejada, $\tilde{X}$, é 2 volts para todos os 105 pontos de simulação, qualquer que sejam as entradas.

Figura 22 – Dispositivo de teste para o circuito de referência de tensão

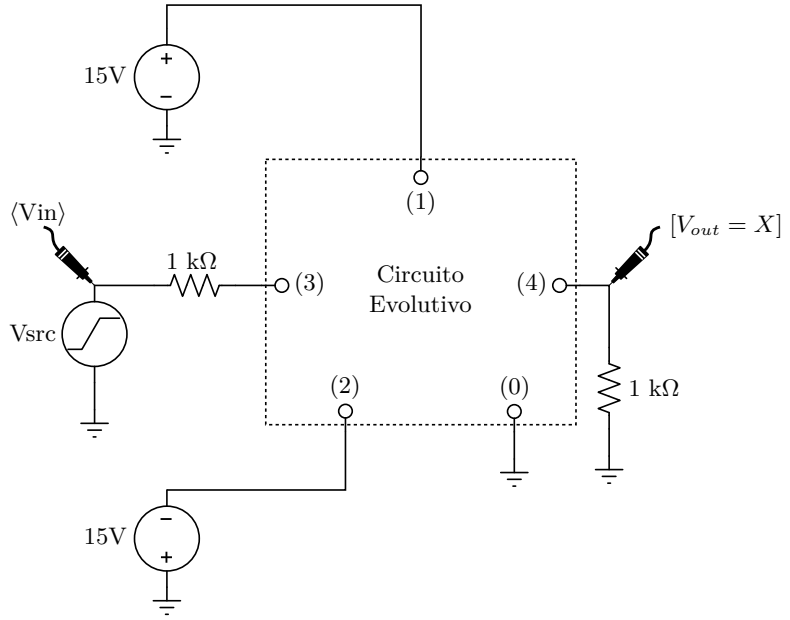


Fonte: o autor (2023)

6.3.4 Circuitos computacionais

Os quatro circuitos computacionais (função quadrática, raiz quadrada, função cúbica e raiz cúbica) tentam aproximar a função matemática do mesmo nome. A Figura 23 mostra o dispositivo de teste usado nesta classe de problema. A representação dos nós externos, variáveis manipuladas e de resposta dessa figura é a mesma das demais figuras dos dispositivos de teste já apresentados nesta seção. A fonte de entrada Vin muda sua tensão em uma rampa linear amostrada em $N_p = 21$ pontos de acordo com os limites dados na Tabela 4. A saída X_i é a tensão do nó #4. A tensão de saída desejada $\tilde{X}_i$ também é fornecida na tabela citada.

Figura 23 – Dispositivo de teste para os circuitos computacionais



Fonte: o autor (2023)

Tabela 4 – Faixa de tensão de entrada e a tensão de saída desejada para cada um dos circuitos computacionais

Circuito de teste	Faixa da tensão de entrada Vin	Tensão de saída desejada $\tilde{X}_i$
Função quadrática	$[-0{,}25; 0{,}25]$	Vin^2
Raiz quadrática	$[0; 0{,}5]$	$\sqrt{Vin}$
Função cúbica	$[-0{,}25; 0{,}25]$	Vin^3
Raiz cúbica	$[-0{,}25; 0{,}25]$	$\sqrt[3]{Vin}$

Fonte: o autor (2023), com dados obtidos de Castejón e Carmona (2018)

6.4 Parâmetros utilizados

Os parâmetros do algoritmo SANN-GCE e de seus casos de uso foram, em um primeiro momento, escolhidos após algumas poucas tentativas, utilizando, na maioria das vezes, parâmetros iguais em todos os casos de uso. Depois, em um segundo momento, os parâmetros foram ajustados por meio de diversas tentativas. As subseções seguintes descrevem os parâmetros utilizados nestes dois momentos.

6.4.1 Parâmetros iniciais

A Tabela 5 apresenta os valores de N_p, X_{th} e w_b, referenciados nas Equações (6.1) e (6.3) a (6.5), para cada circuito de teste. Estes são os mesmos utilizados pelos artigos de referência, novamente pelo motivo de permitir a comparação posterior dos resultados.

A Tabela 6 apresenta os parâmetros do SANN-GCE usados em cada circuito de teste. Estes parâmetros foram *tentativas iniciais*, escolhidos após poucos testes manuais.

O apêndice C lista os modelos SPICE dos transistores bipolares utilizados em alguns circuitos de teste onde este tipo de componente é aplicável.

Tabela 5 – Parâmetros utilizados no cálculo das métricas dos circuitos de teste

Circuito de teste	N_p	X_{th}	w_b
Sensor de temperatura	21	0,1	1
Função gaussiana	101	5×10^{-9}	10^6
Referência de tensão	105	0,02	1
Função quadrática	21	$0{,}05 \times 0{,}25^2$	1
Raiz quadrada	21	$0{,}05 \times \sqrt{0{,}5}$	1
Função cúbica	21	$0{,}05 \times 0{,}25^3$	1
Raiz cúbica	21	$0{,}05 \times \sqrt[3]{0{,}25}$	1

Fonte: o autor (2023), com dados obtidos de Castejón e Carmona (2018)

Tabela 6 – Parâmetros usados nos circuitos de teste

Parâmetro	Circuito de teste						
	Sensor de temperatura	Função gaussiana	Referência de tensão	Função quadrática	Raiz quadrada	Função cúbica	Raiz cúbica
Condição de parada	3×10^6 chamadas da função de aptidão (número de simulações SPICE)						
T_0	100						
T_r	1×10^{-9}						
α	0,9999						
p_{mg}	0,015						
$p_s[T]$	Estrutura $\mapsto$ 0,25; Outros $\mapsto$ 1,0						
$M \times N$	4×4					5×5	6×6
Dispositivo de teste	Veja Figuras 20 a 23						
Mapa dos nós externos	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(2,3) \mapsto 3$ $(3,1) \mapsto 2$ $(3,2) \mapsto 0$	$(0,1) \mapsto 1$ $(0,2) \mapsto 1$ $(1,0) \mapsto 2$ $(2,3) \mapsto 3$ $(3,1) \mapsto 0$ $(3,2) \mapsto 0$	$(1,0) \mapsto 1$ $(1,3) \mapsto 2$ $(3,1) \mapsto 0$ $(3,2) \mapsto 0$	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(1,0) \mapsto 3$ $(2,3) \mapsto 4$ $(3,1) \mapsto 2$ $(3,2) \mapsto 0$	$(0,1) \mapsto 0$ $(0,2) \mapsto 1$ $(2,0) \mapsto 3$ $(2,4) \mapsto 4$ $(4,2) \mapsto 2$ $(4,3) \mapsto 0$	$(0,2) \mapsto 0$ $(0,3) \mapsto 1$ $(2,0) \mapsto 3$ $(3,5) \mapsto 4$ $(5,2) \mapsto 2$ $(5,3) \mapsto 0$
$n_{min}; n_{max}$	$7; 42$						
Probabilidade dos tipos de componentes	R $\mapsto$ 70% Q $\mapsto$ 30%	R $\mapsto$ 50% Ma $\mapsto$ 50%	R $\mapsto$ 70% Q $\mapsto$ 30%				
cq_m	{2N3904, 2N3906}	N.A.	{2N3904, 2N3906}				
cma_m	N.A.	{NMOS, PMOS}	N.A.				
$w_{min}; w_{max}$	N.A.	10; 199	N.A.				

Fonte: o autor (2023)

6.4.2 Ajustes dos parâmetros

Esta subseção descreve o ajuste feito nos parâmetros do SANN-GCE visando obter melhores soluções. O ponto de partida deste ajuste são os parâmetros iniciais (ou *parâmetros de referência*) descritos na seção 6.4.1. A partir destes, os seguintes parâmetros foram alterados para cada um dos sete circuitos de teste: número de componentes $[n_{min}, n_{max}]$, fator de resfriamento (α), probabilidade dos tipos de componentes, probabilidade de mutação geral (p_{mg}), limiar de reaquecimento (T_r), temperatura inicial (T_0) e probabilidade de mutação específica ($p_s[T]$).

Os parâmetros foram alterados utilizando o método *Um fator de cada vez* (*One factor at a time - OFAT*). Nele, partindo dos parâmetros de referência, é escolhido um único parâmetro, seu valor é alterado, o algoritmo é executado 50 vezes e seus resultados são coletados. A Tabela 7 apresenta todos os valores testados para cada parâmetro. Nela,

a *probabilidade dos tipos de componentes* foi dividida em duas porque o circuito da função Gaussiana requer resistores (R) e MOSFETs (Ma), enquanto os demais circuitos de teste requerem resistores e transistores bipolares (Q). Neste parâmetro, a tupla representada como (R% ; Q%) ou (R% ; Ma%) indica a probabilidade de se gerar um tipo específico de componente. Ainda na tabela supracitada, o parâmetro de probabilidade de mutação específica $p_s[T]$ é mostrado como um vetor de comprimento 4, onde as posições *(T ; C ; D ; S)* são, respectivamente, as probabilidades relativas para os DOFs tipo Terminal, Contínuo, Discreto e Estrutura.

6.4.2.1 Teste estatístico

É pretendido saber se a alteração de um parâmetro realmente produziu um resultado melhor quando comparado ao parâmetro de referência ou se foi apenas um acaso. A ferramenta apropriada para responder a esta questão é um teste de hipótese estatística, que permite avaliar a significância das diferenças entre dois grupos distintos. Seu uso requer primeiro a definição de dois grupos:

- Grupo A: Este grupo contém os resultados de uma alteração de parâmetro específica no SANN-GCE. São coletadas amostras deste grupo executando o algoritmo 50 vezes. Isso resulta em um array com 50 elementos contendo a melhor aptidão de cada execução;
- Grupo B: Contém os resultados do SANN-GCE ao utilizar os parâmetros de referência. As amostras coletadas aqui também formam uma matriz contendo os 50 melhores valores de aptidão.

Depois, as hipóteses nula e alternativa são definidas:

- Hipótese Nula: A média do grupo A não é menor que a média do grupo B. Se for verdade, um acaso aleatório pode ser responsável pela aparente melhoria do resultado;
- Hipótese Alternativa: A média do grupo A é menor que a média do grupo B. Se for verdade, um acaso aleatório não é uma explicação razoável para a melhoria observada no resultado do grupo A, quando comparado ao grupo B. É o complemento da hipótese nula.

A referida média de cada grupo é a $\overline{BF}$, a média da métrica BF (*Best Fitness*). Esta métrica é a aptidão da melhor solução encontrada, calculada de acordo com a Equação (6.1).

Tabela 7 – Parâmetros e valores ajustados

Parâmetro	Valores testados
$[n_{min}, n_{max}]$	[7,14] [14,21] [21,28] [28,35] [35,42] [42,49]
α	0,99 0,999 0,99999 0,999999
Probabilidade dos tipos de componentes (para todos os casos de uso, exceto para a função Gaussiana) (R% ; Q%)	10 ; 90 30 ; 70 50 ; 50 90 ; 10
Probabilidade dos tipos de componentes (somente para a função Gaussiana) (R% ; Ma%)	10 ; 90 30 ; 70 70 ; 30 90 ; 10
p_{mg}	0,0015 0,003 0,0075 0,03 0,075 0,15
T_r	0 1×10^{-12} 1×10^{-11} 1×10^{-10} 1×10^{-8} 1×10^{-7} 1×10^{-6}
T_0	10 20 50 200 500 1000
$p_s[T]$ (T ; C ; D ; S)	0,25 ; 1,00 ; 0,50 ; 0,25 0,25 ; 1,00 ; 0,50 ; 0,50 0,25 ; 1,00 ; 0,75 ; 0,25 0,50 ; 1,00 ; 0,50 ; 0,25 0,50 ; 1,00 ; 0,50 ; 0,50 0,50 ; 1,00 ; 0,75 ; 0,25 1,00 ; 1,00 ; 1,00 ; 0,50 1,00 ; 1,00 ; 1,00 ; 1,00

Fonte: o autor (2023)

Com essas definições estabelecidas, foi aplicado o Teste de Permutação nos grupos A e B utilizando 50000 permutações. O Teste de Permutação é uma técnica estatística não paramétrica empregada para avaliar a significância estatística de um resultado observado por meio da realização de permutações aleatórias nos dados. Os dados são aleatoriamente permutados e a estatística de teste (como a diferença entre as médias, por exemplo) é calculada após cada permutação.

O Teste de Permutação é vantajoso porque não estabelece suposições sobre a distribuição subjacente dos dados e simplesmente demanda que as observações sejam intercambiáveis (GOOD, 2004). É particularmente útil quando os dados não seguem uma distribuição conhecida ou quando o tamanho da amostra é reduzida, circunstâncias em que as abordagens paramétricas habituais podem não ser apropriadas.

O resultado do teste é o valor p (ou *p-value*) que indica quantitativamente a significância estatística do resultado. Se p for inferior a um nível de significância α[1], rejeita-se a hipótese nula e afirma-se que o resultado tem significância estatística, ou seja, é assumido que o parâmetro alterado no SANN-GCE melhorou a qualidade da solução. Neste trabalho foi utilizado $\alpha = 0{,}05$.

6.5 Conclusão

Este capítulo introduziu os conceitos de um problema SPICE, apresentou as fórmulas das métricas de desempenho, detalhou os sete problemas selecionados como casos de uso neste trabalho e apresentou seus parâmetros. Na sequência, o capítulo 7 apresentará os resultados da execução do SANN-GCE utilizando os referidos circuitos de teste descritos neste capítulo.

[1] Não confundir com o fator de resfriamento α usado no recozimento simulado

Capítulo 7

Resultados e Discussões

Este capítulo apresenta os resultados obtidos do algoritmo SANN-GCE utilizando os 7 circuitos de teste descritos na seção 6.3. Os resultados são divididos em duas seções: a primeira utiliza os parâmetros iniciais apresentados na seção 6.4.1 e a segunda indica os resultados após o ajuste dos parâmetros, conforme os métodos descritos na seção 6.4.2.

O desempenho dos circuitos sintetizados pelo SANN-GCE, avaliados através das métricas discutidas na seção 6.2 e de outras métricas derivadas apresentadas a seguir, foi comparado com os algoritmos ACID-GE (CASTEJÓN; CARMONA, 2018) e ACID-MGE (CASTEJÓN; CARMONA, 2020). Na discussão que segue, foi dado enfoque ao ACID-MGE em detrimento do ACID-GE, pois o primeiro apresenta o melhor desempenho quando comparado ao segundo. Os dados dos resultados desses dois algoritmos foram obtidos das tabelas 4, 5 e 7 do artigo supracitado que apresenta o ACID-MGE.

Assim como no ACID-GE e ACID-MGE, o algoritmo SANN-GCE foi executado 50 vezes em cada configuração e em cada circuito de teste. Este elevado número de execuções permite obter resultados que representem mais fielmente o comportamento do algoritmo, facilitando as comparações de seus resultados.

O critério de parada utilizado pelos algoritmos ACID-GE e ACID-MGE foi a conclusão de 3000 gerações. Já no SANN-GCE, o critério de parada foi definido para a condição de se concluir 3×10^6 avaliações da função de aptidão. Isto é equivalente às 3000 gerações de uma população com 1000 soluções candidatas, parâmetros utilizados pelos algoritmos ACID-GE e ACID-MGE.

Esse critério de parada do SANN-GCE foi escolhido visando tornar a comparação entre os algoritmos mais justa, pois assim é contabilizada a quantidade de trabalho executada pela etapa de maior custo computacional (e que leva mais tempo) de um ciclo do algoritmo: a simulação SPICE. Em contraposição a isso, uma eventual escolha do critério de parada que estabelecesse um limite no número de iterações do algoritmo de busca (equivalente ao número de gerações em um algoritmo genético, por exemplo) possivelmente

penalizaria algoritmos que são mais eficientes, pois estes normalmente possuem uma razão menor entre o número de chamadas do simulador SPICE e o número de iterações.

A Tabela 8 apresenta a comparação dos resultados dos algoritmos SANN-GCE (utilizando os parâmetros iniciais e também os parâmetros ajustados), ACID-GE e ACID-MGE. Todos os algoritmos utilizaram os mesmos 7 casos de uso e foram executados 50 vezes cada. Na tabela, as linhas SANN-GCE' indicam o uso do algoritmo com os parâmetros iniciais, enquanto as linhas SANN-GCE sinalizam que foram utilizados os parâmetros ajustados. O significado das colunas dessa tabela são:

- SR%: *Success Rate* (taxa de sucesso): Percentual de execuções bem-sucedidas. Uma execução bem-sucedida é aquela que a métrica *Hits %* atinge o valor de 100 %;
- BF: *Best Fitness* (melhor aptidão), é a aptidão da melhor solução encontrada dentre todas as execuções, calculada de acordo com a Equação (6.1);
- MAE: *Mean Absolute Error* (erro médio absoluto), é uma métrica secundária calculada de acordo com a Equação (6.4);
- Hits %: Porcentagem das amostras do resultado SPICE que tiveram erro absoluto abaixo de um limiar predefinido, calculado de acordo com a Equação (6.5);
- #Ger. sucesso: Número de gerações necessárias para se obter o sucesso ($hits = 100\%$). Como o SANN-GCE não utiliza uma população de 1000 soluções candidatas como o ACID-GE e ACID-MGE, comparar o número de gerações destes algoritmos com o número de iterações do SANN-GCE não faz sentido. Para tornar a comparação entre estes algoritmos possível, o número de gerações no SANN-GCE para o cálculo desta métrica foi assumido como sendo a contagem de *avaliações da função de aptidão* dividida por 1000 (tamanho da população utilizada nos outros 2 algoritmos);
- NCBC: *Number of Components of the Best Circuit* (número de componentes do melhor circuito). É a quantidade de componentes utilizados na melhor solução encontrada. A contagem leva em consideração apenas o Circuito Evolutivo, desconsiderando os componentes fixos especificados pelo Dispositivo de Teste;
- DP: Indica o desvio padrão de uma métrica, calculado sobre as amostras obtidas em cada uma das 50 execuções do algoritmo.

Tabela 8 – Resultados do algoritmo SANN-GCE comparados com ACID-MGE e ACID-GE

Circuito de teste	Algoritmo	SR (%)*	MAE (mV)			BF			Hits (%)		#Ger. sucesso		NCBC
			Média	± DP	mín.	Média	± DP	mín.	Média*	± DP	Média	± DP	
Sensor de temperatura	ACID-MGE	70,0	ND	ND	ND	2,13	3,34	0,033	97,1	5,3	711,2	726,2	42
	ACID-GE	42,0	ND	ND	ND	5,29	6,53	0,169	93,0	7,9	1278,9	884,6	**28**
	SANN-GCE'	**100,0**	14,41	5,398	**0,5605**	0,3027	0,1145	**0,01177**	**100,0**	**0,0**	492,0	407,8	41
	SANN-GCE	**100,0**	**10,89**	**4,865**	2,799	**0,2286**	**0,1032**	0,05878	**100,0**	**0,0**	**274,0**	**191,6**	36
Função gaussiana	ACID-MGE	58,0	ND	ND	ND	1	1,75	0,028	94,1	9,8	1023,2	719,6	37
	ACID-GE	24,0	ND	ND	ND	6,7	6,78	0,04	77,3	17,6	1367,4	688,5	41
	SANN-GCE'	88,0	9,526e-07	5,912e-07	3e-07	0,1493	0,263	0,0303	99,3	2,6	961,4	832,1	**29**
	SANN-GCE	**100,0**	**5,841e-07**	**2,649e-07**	**2,215e-07**	**0,05899**	**0,02703**	**0,02237**	**100,0**	**0,0**	**628,0**	**560,4**	42
Referência de tensão	ACID-MGE	8,0	ND	ND	ND	19,35	14,04	0,053	64,2	23,3	**1205,0**	**863,3**	42
	ACID-GE	8,0	ND	ND	ND	26,57	16,23	0,112	53,8	25,0	1581,0	917,0	**32**
	SANN-GCE'	12,0	10,99	3,725	2,84	4,63	3,184	0,2982	90,4	7,0	1583,3	1018,9	35
	SANN-GCE	**30,0**	**7,574**	**3,462**	**0,4583**	**2,011**	**1,856**	**0,04812**	**96,3**	**4,5**	1206,7	894,0	41
Função quadrática	ACID-MGE	84,0	0,53	0,24	**0,08**	0,06	0,14	**0,002**	97,7	6,5	**671,5**	**574,9**	42
	ACID-GE	52,0	0,93	0,41	ND	0,73	1,17	0,009	81,0	28,5	1176,9	828,3	**28**
	SANN-GCE'	98,0	0,613	0,3069	0,1194	0,01488	0,01884	0,002507	99,8	1,3	706,1	634,5	35
	SANN-GCE	**100,0**	**0,5016**	**0,2175**	0,1447	**0,01053**	**0,004615**	0,003038	**100,0**	**0,0**	732,0	638,6	33
Raiz quadrada	ACID-MGE	66,0	4,01	2,47	**0,23**	2,71	5,78	**0,003**	89,0	24,2	1154,9	655,0	44
	ACID-GE	44,0	5,55	4,32	ND	6,25	7,61	0,028	74,0	32,3	1329,3	797,3	**35**
	SANN-GCE'	**100,0**	3,612	2,729	0,5273	0,07585	0,05789	0,01107	**100,0**	**0,0**	816,0	671,8	38
	SANN-GCE	**100,0**	**2,529**	**1,361**	0,3295	**0,05311**	**0,02888**	0,006919	**100,0**	**0,0**	**666,0**	**652,3**	**35**
Função cúbica	ACID-MGE	84,0	**0,1**	0,04	0,05	0,03	0,08	0,001	95,7	13,7	**539,8**	**361,7**	47
	ACID-GE	76,0	0,18	0,07	ND	0,05	0,12	0,002	92,2	19,3	1008,0	703,6	40
	SANN-GCE'	98,0	0,1871	0,1019	0,0523	0,004465	0,005039	0,001098	99,8	1,3	1200,0	787,7	37
	SANN-GCE	**100,0**	0,1024	**0,03712**	**0,03866**	**0,002151**	**0,0007874**	**0,0008119**	**100,0**	**0,0**	568,0	508,5	**33**
Raiz cúbica	ACID-MGE	22,0	7,13	3,7	2,04	13,87	25,02	0,036	70,2	29,4	1561,1	587,8	57
	ACID-GE	2,0	10,81	**0**	ND	76,95	23,74	0,227	11,5	20,2	1885,0	**0,0**	**41**
	SANN-GCE'	66,0	11,88	7,924	**0,956**	0,7892	1,175	**0,02008**	95,5	7,8	1297,0	918,3	**41**
	SANN-GCE	**90,0**	**6,454**	4,693	1,153	**0,2173**	**0,3594**	0,02422	**99,1**	**2,8**	**1093,3**	768,7	48

Fonte: o autor (2023), com dados de Castejón e Carmona (2018) e Castejón e Carmona (2020)

Nota: Métricas com valores menores indicam resultados melhores, exceto para aquelas marcadas com um asterisco (*) no cabeçalho. Os melhores resultados de uma métrica em cada circuito de teste estão destacados em negrito. As linhas SANN-GCE' indicam o uso do algoritmo com os parâmetros iniciais, enquanto as linhas SANN-GCE assinalam o uso dos parâmetros ajustados.

Os dados do MAE dos circuitos não computacionais não estão disponíveis na Tabela 8 para o ACID-GE e o ACID-MGE, pois o artigo fonte não informa esses valores. Pelo mesmo motivo, $\text{MAE}_{\text{mín}}$ não está disponível para ACID-GE em todos os circuitos de teste. Estas ausências estão marcadas na referida tabela com a sigla ND (*Não Disponível*).

7.1 Relação de desempenho

Em todas as métricas mostradas na Tabela 8, menores valores indicam resultados melhores, exceto naquelas marcadas com um asterisco no cabeçalho (SR% e $\text{Hits}_{\text{média}}$), onde essa lógica se inverte: valores maiores significam melhores resultados. Considerando estas diferentes lógicas, para facilitar a comparação de uma determinada métrica M entre o SANN-GCE e outro algoritmo α qualquer, definimos a *relação de desempenho do SANN-GCE* $R_{M,\alpha}$ da seguinte forma:

$$R_{M,\alpha} = \begin{cases} \dfrac{M_{\text{SANN-GCE}}}{M_\alpha} & \text{se } M \text{ maior indicar melhor resultado, ou} \\ \\ \dfrac{M_\alpha}{M_{\text{SANN-GCE}}} & \text{se } M \text{ menor indicar melhor resultado} \end{cases} \tag{7.1}$$

Assim, $R < 1$ sempre significa um resultado pior e $R > 1$ sempre indica um resultado melhor para SANN-GCE, independente da métrica seguir a lógica de *menor é melhor* ou *maior é melhor*.

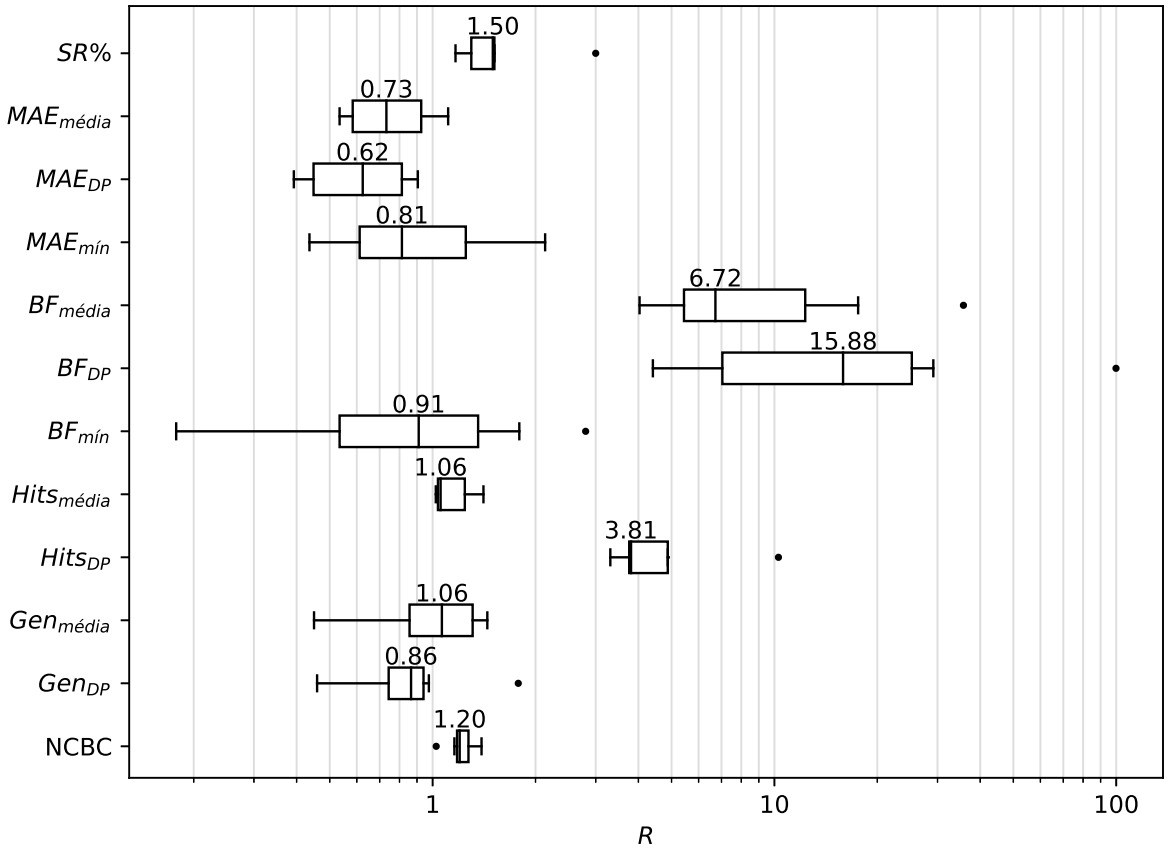
7.2 Resultados obtidos com os parâmetros iniciais

Esta seção discute os resultados obtidos pelo SANN-GCE e sua comparação com o ACID-MGE. Nesta seção, o SANN-GCE utilizou os parâmetros iniciais fornecidos na Tabela 6 e demais configurações apresentadas na seção 6.4.1.

A relação de desempenho R do SANN-GCE com o ACID-MGE foi calculada individualmente para cada uma das 12 métricas da Tabela 8 (colunas 3 a 14 da tabela) em todos os 7 circuitos de teste, utilizando a Equação (7.1). Em seguida, os 7 valores de R obtidos em uma determinada métrica foram utilizados para gerar uma dada linha (caixa) no diagrama de caixas (*boxplot*) mostrado na Figura 24. Nesta figura, as estatísticas para cada métrica foram calculadas sobre todos os circuitos de teste. Valores indicados sobre as caixas são as medianas de cada métrica. Valores $R > 1$ indicam melhores resultados para o SANN-GCE na comparação com o ACID-MGE.

O boxplot é uma representação visual da distribuição de uma variável numérica em várias métricas e permite visualizar de forma rápida e fácil a mediana, os quartis e os valores discrepantes de uma distribuição. O corpo da caixa (parte retangular) se estende

Figura 24 – Relação de desempenho R entre o SANN-GCE, utilizando os parâmetros iniciais, e o ACID-MGE em cada métrica



Fonte: o autor (2023)

do primeiro quartil (Q1) até o terceiro quartil (Q3) dos dados. A linha central vertical em seu corpo indica a mediana (Q2). As linhas horizontais que extrapolam o corpo indicam o limite teórico inferior (LTI) e o limite teórico superior (LTS) dos dados, calculados conforme as equações abaixo:

$$\text{LTI} = \text{Q1} - 1{,}5 \cdot \text{AIQ} \tag{7.2}$$

$$\text{LTS} = \text{Q3} + 1{,}5 \cdot \text{AIQ} \tag{7.3}$$

Onde AIQ é a *Amplitude Interquartil*, calculada da seguinte forma:

$$\text{AIQ} = \text{Q3} - \text{Q1} \tag{7.4}$$

Os pontos que extrapolam as linhas horizontais supracitadas, quando indicados, são os *valores discrepantes* (*outliers*) e indicam dados que estão fora do intervalo $[\text{LTI}, \text{LTS}]$.

É possível constatar nos dados mostrados pela Tabela 8 que os circuitos sintetizados pelo SANN-GCE, mesmo utilizando configurações iniciais, apresentaram uma vantagem de desempenho consistente nas métricas SR%, BF (média e desvio padrão) e Hits% (média e desvio padrão), pois nestas o SANN-GCE obteve números melhores em todos os circuitos de teste quando comparadas com o ACID-MGE e o ACID-GE.

As relações de desempenho R apresentados pela Figura 24 fornecem comparações de cada métrica agrupando os dados de todos os casos de uso. O tamanho das caixas e a presença de valores discrepantes indicam a variabilidade no desempenho relativo do SANN-GCE com diferentes circuitos de teste. Nesta figura é possível concluir que, quando comparado com o ACID-MGE, o SANN-GCE apresenta relações de desempenho que se destacam positivamente nas seguintes métricas:

- SR%: mediana de 1,5, com baixa dispersão entre o Q1 e Q3 e LTI acima de 1;
- $\text{BF}_{\text{média}}$: mediana foi 6,72, com o LTI acima de 5;
- BF_{DP}: a mediana foi 15,88, com o LTI acima de 7;
- $\text{Hits}_{\text{média}}$: mediana 1,06, com LTI acima de 1 e baixa dispersão entre o Q1 e Q3;
- Hits_{DP} mediana foi 3,81, com LTI acima de 3;
- $\text{Gen}_{\text{média}}$: mediana de 1,06, com LTI acima de 0,8;
- NCBC: mediana 1,2, com baixa dispersão entre o Q1 e Q3 e LTI acima de 1.

O SANN-GCE apresentou relações de desempenho piores nas métricas MAE (média, desvio padrão, e valor mínimo), $\text{BF}_{\text{mín}}$ e Gen_{DP}, onde foram obtidas as medianas 0,73, 0,62, 0,81, 0,91, 0,86, respectivamente. O motivo para tais resultados não foi evidenciado, mas é presumido que, no caso da métrica MAE, esta apresentou resultado ligeiramente inferior por ser uma métrica secundária, portanto não sendo utilizada para guiar o algoritmo de busca.

O conjunto de resultados apresentados acima evidenciam, de maneira geral, que o SANN-GCE, mesmo utilizando os parâmetros iniciais, possui um desempenho notável em todos os casos de uso selecionados, especialmente uma menor média e um menor desvio padrão da sua métrica primária (a aptidão), que refletiu em um valor de BF_{DP} 15,88× menor, e $\text{BF}_{\text{média}}$ 6,72× menor, quando observada a sua mediana calculada entre todos os problemas. Isto implica que, quando comparado com o ACID-MGE, é esperado que, em média, o SANN-GCE exija um número menor de execuções para se encontrar um circuito com uma aptidão menor.

7.3 Resultados obtidos com os parâmetros ajustados

Esta seção apresenta e discute os resultados obtidos utilizando a metodologia descrita na seção 6.4.2, onde diversos parâmetros utilizados em um primeiro momento são modificados e testados, visando encontrar a configuração que produza os melhores resultados em cada circuito de teste.

7.3.1 Comparação com o SANN-GCE usando os parâmetros iniciais

As Tabelas 9 a 15 apresentam comparações da média da melhor aptidão obtida no SANN-GCE utilizando parâmetros iniciais de referência e parâmetros ajustados. Nestas tabelas, p é o *p-value* do teste estatístico entre estas duas configurações, executado conforme a seção 6.4.2.1. Valores de p iguais ou inferiores ao nível de significância de 0,05 estão marcados em negrito. Em cada tabela, a primeira linha de resultados reproduz o parâmetro de referência para comodidade do leitor. Ainda nestas tabelas, $\overline{BF}$[1] indica a média da melhor aptidão obtida e o quociente $\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$ é a *relação de melhoria* entre a configuração inicial e a ajustada. Ela indica o quanto a aptidão média se modificou após a alteração de um parâmetro. A relação $\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}} < 1$ indica que o parâmetro alterado pode ter aperfeiçoado o resultado, pois valores menores de aptidão (BF) representam melhores soluções. Isso em conjunto com $p \leq 0{,}05$ indica um resultado melhor e estatisticamente significativo.

É possível observar neste conjunto de tabelas supracitadas que a mudança no número de componentes $[n_{min}, n_{max}]$ de [7,42] para [35,42] ou [42,49] melhorou o resultado em 5 dos 7 circuitos de teste. A relação de melhoria ficou entre 0,28 e 0,76. O circuito da função Gaussiana não se beneficia de nenhuma alteração neste parâmetro. O resultado do circuito raiz quadrada foi melhorado por um fator de 0,7 quando $[n_{min}, n_{max}]$ foi definido para [28,35].

Mudanças no fator de resfriamento α forneceram melhores resultados apenas para o circuito da função cúbica: quando $\alpha = 0{,}99999$, sua aptidão média melhorou por um fator de 0,58.

Ao investigar o resultado das mudanças na probabilidade dos tipos de componentes, ficou evidente que o circuito da função cúbica teve sua aptidão média reduzida pela metade quando a probabilidade de geração do componente foi definida como $[R\%; Q\%] = [10; 90]$ ou $[30; 70]$. Para o circuito de referência de tensão, foi observada uma melhoria de 0,68 quando o referido parâmetro foi definido como $[50; 50]$. O circuito da função Gaussiana, por sua vez, foi beneficiado com $[R\%; Ma\%] = [10; 90]$, onde sua aptidão média melhorou com uma proporção de 0,40. Os outros quatro circuitos de teste não apresentaram melhorias ao alterar este parâmetro.

[1] $\overline{BF}$ é o mesmo que $\mathrm{BF}_{\mathrm{média}}$, porém utiliza uma notação mais compacta, principalmente quando é necessário o uso do subscrito *ref*, como em $\overline{\mathrm{BF}}_{\mathrm{ref}}$

Tabela 9 – Teste de alteração do parâmetro número de componentes $[n_{min}, n_{max}]$

Parte 1 de 2:

Parâmetro alterado $[n_{min}, n_{max}]$	Raiz cúbica			Func. cúbica			Func. Gaussiana			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
[7,42] (referência)	0,7892	-	-	0,0045	-	-	0,1493	-	-	0,0759	-	-
[7,14]	60,9394	77,21	-	0,2107	47,18	-	2,9384	19,68	-	0,8239	10,86	-
[14,21]	33,6185	42,60	-	0,0454	10,16	-	0,2529	1,69	-	0,1402	1,85	-
[21,28]	5,7541	7,29	-	0,0100	2,23	-	0,2770	1,86	-	0,0732	0,97	0,4069
[28,35]	1,4970	1,90	-	0,0048	1,09	-	7,1853	48,12	-	0,0531	0,70	**0,0056**
[35,42]	0,3377	0,43	**0,0078**	0,0058	1,29	-	1,8706	12,53	-	0,0658	0,87	0,1715
[42,49]	0,2173	0,28	**0,0003**	0,0046	1,03	-	14,0450	94,05	-	0,0629	0,83	0,1151

Parte 2 de 2:

Parâmetro alterado $[n_{min}, n_{max}]$	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
[7,42] (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
[7,14]	0,2092	14,06	-	1,8147	6,00	-	15,6586	3,38	-
[14,21]	0,0277	1,86	-	0,5710	1,89	-	7,4353	1,61	-
[21,28]	0,0143	0,96	0,4858	0,3477	1,15	-	5,2440	1,13	-
[28,35]	0,0105	0,71	**0,0178**	0,2976	0,98	0,4115	3,5349	0,76	**0,0271**
[35,42]	0,0106	0,71	**0,0211**	0,2286	0,76	**0,0005**	2,2985	0,50	**0,0000**
[42,49]	0,0106	0,71	**0,0315**	0,2294	0,76	**0,0001**	2,0111	0,43	**0,0000**

Fonte: o autor (2023)

A mudança na probabilidade geral de mutação p_{mg} não mostrou melhora para nenhum dos circuitos de teste. A alteração do limiar de reaquecimento T_r de 1×10^{-9} para 1×10^{-7} beneficiou apenas o circuito do sensor de temperatura por um fator de 0,86. Já o ajuste do parâmetro de temperatura inicial T_0 de 100 para 10 no circuito de referência de tensão resultou em uma aptidão média de 0,72 × o valor de referência. Os outros seis circuitos não se beneficiaram das alterações deste parâmetro.

A probabilidade de mutação específica $p_s[T]$ quando alterada de $(T; C; D; S) = (1{,}00; 1{,}00; 1{,}00; 0{,}25)$ para $(0{,}25; 1{,}00; 0{,}75; 0{,}25)$ melhorou (reduziu) a aptidão do circuito sensor de temperatura pelo fator de 0,83.

A Tabela 16 resume os melhores resultados encontrados nas Tabelas 9 a 15 após o ajuste de parâmetros do SANN-GCE. Nessa tabela-resumo, em sua coluna central, é apresentada a melhor *relação de melhoria* encontrada em cada problema e a alteração específica dos parâmetros que conduz a isso. Pode-se observar que apenas dois parâmetros foram responsáveis por estas melhorias: 1. n_{min}, n_{max}; e 2. probabilidade de tipos de componentes. Ambos têm em comum a característica de, quando alterados, afetarem a *estrutura* do circuito eletrônico gerado: alterar o número de componentes em um circuito

Tabela 10 – Teste de alteração do parâmetro α

Parte 1 de 2:

Parâmetro alterado	Raiz cúbica			Func. cúbica			Func. Gaussiana			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
0,9999 (referência)	0,7892	-	-	0,0045	-	-	0,1493	-	-	0,0759	-	-
0,99	4,7119	5,97	-	0,0086	1,93	-	2,6480	17,73	-	1,1300	14,90	-
0,999	1,2352	1,57	-	0,0061	1,37	-	0,4965	3,32	-	0,2151	2,84	-
0,99999	0,6273	0,79	0,2723	0,0026	0,58	**0,0002**	0,1966	1,32	-	0,0926	1,22	-
0,999999	2,1042	2,67	-	0,3352	75,09	-	7,9066	52,95	-	1,1768	15,51	-

Parte 2 de 2:

Parâmetro alterado	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
0,9999 (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
0,99	0,3145	21,13	-	0,4754	1,57	-	19,8127	4,28	-
0,999	0,0275	1,85	-	0,3661	1,21	-	10,4522	2,26	-
0,99999	0,0231	1,55	-	0,4429	1,46	-	3,7459	0,81	0,2144
0,999999	2,1493	144,40	-	0,5811	1,92	-	4,6669	1,01	-

Fonte: o autor (2023)

envolve criar ou destruir componentes, enquanto a modificação do tipo de componente requer a destruição do componente antigo e a criação de um novo. Estes eventos de criação e destruição de componentes podem alterar muito a resposta do circuito, portanto o ajuste destes parâmetros é mais crítico.

Estes resultados evidenciam que a restrição do número de componentes de um circuito para uma faixa mais estreita facilita o trabalho do algoritmo em encontrar uma solução melhor em 5 dos 7 circuitos de teste. Para explicar isso, é levantada a seguinte hipótese: uma faixa mais estreita de número de componentes implica em uma menor probabilidade de alteração no número de componentes do circuito (que são eventos de criação ou destruição de componentes). Essa menor perturbação estrutural no circuito tornaria então a resposta do circuito mais estável ao longo da evolução da solução, e isso então permitiria que o algoritmo de busca encontrasse uma solução melhor de forma mais eficiente.

7.3.2 Comparação com o ACID-MGE

A Figura 25 apresenta a comparação do SANN-GCE, utilizando os parâmetros ajustados, com o ACID-MGE. Assim como na Figura 24, a comparação foi feita através da relação de desempenho R, dada pela Equação (7.1) e as estatísticas de cada métrica foram calculadas utilizando todos os 7 circuitos de teste. O desempenho relativo do algoritmo

Tabela 11 – Teste de alteração do parâmetro probabilidade do tipo de componente

Parte 1 de 3:

Parâmetro alterado (R% ; Q%)	Raiz cúbica			Func. cúbica			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
70 ; 30 (referência)	0,7892	-	-	0,0045	-	-	0,0759	-	-
10 ; 90	0,9604	1,22	-	0,0022	0,50	**0,0000**	0,1302	1,72	0,9979
30 ; 70	0,8950	1,13	-	0,0022	0,48	**0,0000**	0,0670	0,88	0,2104
50 ; 50	0,7131	0,90	0,3833	0,0041	0,93	0,4238	0,0666	0,88	0,2535
90 ; 10	9,4802	12,01	-	0,0339	7,60	-	0,4351	5,74	-

Parte 2 de 3:

Parâmetro alterado (R% ; Q%)	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
70 ; 30 (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
10 ; 90	0,0147	0,99	0,4915	0,4919	1,63	-	5,0951	1,10	-
30 ; 70	0,0114	0,77	0,1009	0,3332	1,10	-	3,6670	0,79	0,0702
50 ; 50	0,0123	0,83	0,2489	0,3280	1,08	-	3,1641	0,68	**0,0091**
90 ; 10	0,0255	1,71	-	0,3570	1,18	-	8,1459	1,76	-

Parte 3 de 3:

Parâmetro alterado (R% ; Ma%)	Func. Gaussiana		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
50 ; 50 (referência)	0,1493	-	-
10 ; 90	0,0590	0,40	**0,0000**
30 ; 70	0,1095	0,73	0,2368
70 ; 30	0,2898	1,94	0,9830
90 ; 10	1,6823	11,27	1,0000

Fonte: o autor (2023)

Tabela 12 – Teste de alteração do parâmetro p_{mg}

Parte 1 de 2:

Parâmetro alterado	Raiz cúbica			Func. cúbica			Func. Gaussiana			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
0,015 (referência)	0,7892	-	-	0,0045	-	-	0,1493	-	-	0,0759	-	-
0,0015	1,0978	1,39	-	0,0111	2,50	-	0,3922	2,63	-	0,3206	4,23	-
0,003	1,1410	1,45	-	0,0098	2,19	-	0,2705	1,81	-	0,2355	3,11	-
0,0075	0,7181	0,91	0,3752	0,0126	2,83	-	0,1318	0,88	0,3705	0,1083	1,43	-
0,03	0,9880	1,25	-	0,0083	1,85	-	0,2198	1,47	-	0,0747	0,98	0,4575
0,075	2,0059	2,54	-	0,0151	3,39	-	0,5046	3,38	-	0,2028	2,67	-
0,15	5,6079	7,11	-	0,0541	12,12	-	2,5104	16,81	-	0,9850	12,99	-

Parte 2 de 2:

Parâmetro alterado	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
0,015 (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
0,0015	0,0394	2,65	-	0,3091	1,02	-	10,8265	2,34	-
0,003	0,0214	1,44	-	0,2944	0,97	0,3579	6,5142	1,41	-
0,0075	0,0166	1,11	-	0,3064	1,01	-	5,4507	1,18	-
0,03	0,0126	0,85	0,3070	0,3947	1,30	-	5,1090	1,10	-
0,075	0,0249	1,67	-	0,9322	3,08	-	8,4020	1,81	-
0,15	0,1866	12,54	-	3,9433	13,03	-	26,0002	5,62	-

Fonte: o autor (2023)

SANN-GCE é medido pela razão das métricas obtidas nele dividido pelas métricas obtidas pelo ACID-MGE. Neste sentido, valores $R > 1$ evidenciam resultados favoráveis para o SANN-GCE. A mediana dessa razão para cada métrica é exibida sobre as respectivas caixas no gráfico.

Ainda na Figura 25, é possível observar que a mediana de R em 11 das 12 métricas foi superior a 1, o que sugere que o SANN-GCE tem um desempenho melhor do que o ACID-MGE nesses casos. As métricas $\mathrm{BF}_{\text{média}}$ e $\mathrm{BF}_{\mathrm{DP}}$ apresentaram medianas maiores que 10, indicando uma vantagem considerável do SANN-GCE sobre o ACID-MGE nessas áreas. Além disso, com exceção da NCBC, todas as métricas foram superiores às obtidas com as configurações iniciais, o que evidencia uma melhora significativa no desempenho do algoritmo depois do ajuste de parâmetros. Os dados da Tabela 8, em outra perspectiva, corroboram com estas observações.

7.3.3 Melhores circuitos encontrados

As Figuras 26 a 32 apresentam os gráficos contendo a resposta do melhor circuito eletrônico encontrado em cada um dos 7 casos de uso selecionados, utilizando o algoritmo

Tabela 13 – Teste de alteração do parâmetro T_r

Parte 1 de 2:

Parâmetro alterado	Raiz cúbica			Func. cúbica			Func. Gaussiana			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
1×10^{-9} (referência)	0,7892	-	-	0,0045	-	-	0,1493	-	-	0,0759	-	-
0	32,9435	41,74	-	0,1177	26,36	-	4,0106	26,86	-	1,8635	24,57	-
1×10^{-12}	1,2666	1,60	-	0,0127	2,85	-	0,2250	1,51	-	0,1058	1,39	-
1×10^{-11}	1,1924	1,51	-	0,0100	2,24	-	0,1744	1,17	-	0,1013	1,34	-
1×10^{-10}	0,8749	1,11	-	0,0063	1,41	-	0,1210	0,81	0,2881	0,0962	1,27	-
1×10^{-8}	1,2371	1,57	-	0,0072	1,60	-	0,1399	0,94	0,4289	0,0908	1,20	-
1×10^{-7}	0,6476	0,82	0,2635	0,0078	1,75	-	0,0922	0,62	0,0769	0,0733	0,97	0,4228
1×10^{-6}	0,7489	0,95	0,4384	0,0097	2,16	-	0,1140	0,76	0,2358	0,0645	0,85	0,1384

Parte 2 de 2:

Parâmetro alterado	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
1×10^{-9} (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
0	0,4242	28,50	-	2,1993	7,27	-	20,3028	4,38	-
1×10^{-12}	0,0139	0,94	0,4240	0,3124	1,03	-	4,5749	0,99	0,4608
1×10^{-11}	0,0136	0,91	0,3924	0,3219	1,06	-	4,2214	0,91	0,2698
1×10^{-10}	0,0125	0,84	0,2824	0,3775	1,25	-	4,9426	1,07	-
1×10^{-8}	0,0114	0,77	0,0862	0,2951	0,97	0,3612	3,6237	0,78	0,0551
1×10^{-7}	0,0134	0,90	0,4155	0,2601	0,86	**0,0340**	4,2697	0,92	0,2755
1×10^{-6}	0,0124	0,83	0,2630	0,2918	0,96	0,3122	3,9933	0,86	0,1634

Fonte: o autor (2023)

SANN-GCE após os ajustes de parâmetros. Em cada gráfico deste conjunto, há três curvas:

1. Referência: Estabelece a resposta desejada para o circuito, de acordo com as especificações de cada problema detalhadas na seção 6.3. É o alvo do algoritmo SANN-GCE, no sentido de que o algoritmo tem como objetivo gerar um circuito que produza em sua saída valores os mais próximos possíveis desta curva. É indicada pela linha azul pontilhada. Utiliza como escala o eixo das ordenadas da esquerda;

2. Resultado: Apresenta a resposta real do circuito sintetizado pelo algoritmo, medida após a sua simulação SPICE. É exibido através de uma linha azul contínua e também utiliza o eixo das ordenadas localizado à esquerda do gráfico;

3. Erro[2]: É calculado através da subtração da curva *resultado* com a curva *referência*, indicando o desvio do resultado real com o desejado. É apresentado com uma linha laranja contínua, com sua escala localizada do lado direito do gráfico.

[2] A curva do erro não é apresentada no gráfico do problema *referência de tensão* (Figura 28) pois haveriam 6 curvas de erro, e julgou-se que isso deixaria o gráfico muito poluído, dificultando a leitura.

Tabela 14 – Teste de alteração do parâmetro T_0

Parte 1 de 2:

Parâmetro alterado	Raiz cúbica			Func. cúbica			Func. Gaussiana			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
100 (referência)	0,7892	-	-	0,0045	-	-	0,1493	-	-	0,0759	-	-
10	0,9507	1,20	-	0,0064	1,44	-	0,0983	0,66	0,1127	0,0730	0,96	0,3946
20	0,5672	0,72	0,1401	0,0077	1,73	-	0,1192	0,80	0,2799	0,1085	1,43	-
50	1,3459	1,71	-	0,0139	3,11	-	0,1366	0,91	0,3884	0,0772	1,02	-
200	1,3344	1,69	-	0,0108	2,42	-	0,1656	1,11	-	0,0905	1,19	-
500	0,5135	0,65	0,0808	0,0067	1,50	-	0,1407	0,94	0,4339	0,1175	1,55	-
1000	1,1730	1,49	-	0,0096	2,15	-	0,1551	1,04	-	0,1023	1,35	-

Parte 2 de 2:

Parâmetro alterado	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\mathrm{BF}}_{\mathrm{ref}}}$	p
100 (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
10	0,0116	0,78	0,1154	0,4168	1,38	-	3,3442	0,72	**0,0281**
20	0,0184	1,24	-	0,3250	1,07	-	4,4959	0,97	0,4232
50	0,0114	0,77	0,0944	0,2837	0,94	0,2048	4,4336	0,96	0,3794
200	0,0152	1,02	-	0,2818	0,93	0,1807	4,5251	0,98	0,4406
500	0,0138	0,93	0,3526	0,3067	1,01	-	5,2738	1,14	-
1000	0,0138	0,93	0,3957	0,3222	1,06	-	3,8264	0,83	0,1168

Fonte: o autor (2023)

Neste conjunto de gráficos é possível observar que foi possível obter bons circuitos com o SANN-GCE em todos os casos de uso selecionados, pois as repostas de todos eles estão muito próximas de suas curvas de referência. Por exemplo, o melhor circuito para o problema *sensor de temperatura* apresentou um erro máximo, em módulo, da ordem de 7,5 milivolts, o que representa 0,075 % do fundo de escala da curva de referência. No problema *função Gaussiana*, este erro máximo relativo à escala máxima foi da ordem de 1,5 %. No problema *referência de tensão* o erro foi de 0,7 %. Já para os problemas *função quadrática*, *raiz quadrada*, *função cúbica* e *raiz cúbica*, os erros máximos foram de 1,3 %, 0,09 %, 1,0 % e 1,2 %, respectivamente.

O apêndice D apresenta a netlist completa destes circuitos, contendo o circuito evolutivo (CE), o dispositivo de teste (DT) e também os comandos que normalmente são utilizados para incluir modelos SPICE, invocar a simulação e salvar as tensões e/ou correntes pertinentes. Isto permite a reprodução e comprovação externa do desempenho do circuito sintetizado automaticamente pelo SANN-GCE.

Tabela 15 – Teste de alteração do parâmetro $p_s[T]$

Parte 1 de 2:

Parâmetro alterado (T ; C ; D ; S)	Raiz cúbica			Func. cúbica			Func. Gaussiana			Raiz quadrada		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	BF	$\frac{BF}{\text{BF}_{\text{ref}}}$	p	BF	$\frac{BF}{\text{BF}_{\text{ref}}}$	p	BF	$\frac{BF}{\text{BF}_{\text{ref}}}$	p
1,00 ; 1,00 ; 1,00 ; 0,25 (referência)	0,7892	-	-	0,0045	-	-	0,1493	-	-	0,0759	-	-
0,25 ; 1,00 ; 0,50 ; 0,25	1,2530	1,59	-	0,0151	3,39	-	0,0993	0,66	0,1109	0,1410	1,86	-
0,25 ; 1,00 ; 0,50 ; 0,50	0,9954	1,26	-	0,0079	1,76	-	0,1475	0,99	0,4855	0,1388	1,83	-
0,25 ; 1,00 ; 0,75 ; 0,25	1,6721	2,12	-	0,0079	1,77	-	0,1538	1,03	-	0,0783	1,03	-
0,50 ; 1,00 ; 0,50 ; 0,25	0,9973	1,26	-	0,0082	1,84	-	0,1339	0,90	0,3819	0,0783	1,03	-
0,50 ; 1,00 ; 0,50 ; 0,50	1,2313	1,56	-	0,0067	1,49	-	0,1076	0,72	0,1739	0,1450	1,91	-
0,50 ; 1,00 ; 0,75 ; 0,25	1,2193	1,54	-	0,0135	3,03	-	0,1279	0,86	0,3330	0,1070	1,41	-
1,00 ; 1,00 ; 1,00 ; 0,50	0,7890	1,00	-	0,0067	1,49	-	0,1121	0,75	0,2257	0,1437	1,89	-
1,00 ; 1,00 ; 1,00 ; 1,00	0,8537	1,08	-	0,0058	1,31	-	0,0984	0,66	0,0940	0,0919	1,21	-

Parte 2 de 2:

Parâmetro alterado (T ; C ; D ; S)	Func. quadrática			Sensor de temp.			Ref. de tensão		
	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p	$\overline{BF}$	$\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	p
1,00 ; 1,00 ; 1,00 ; 0,25 (referência)	0,0149	-	-	0,3027	-	-	4,6304	-	-
0,25 ; 1,00 ; 0,50 ; 0,25	0,0147	0,99	0,4854	0,2744	0,91	0,0944	5,3715	1,16	-
0,25 ; 1,00 ; 0,50 ; 0,50	0,0141	0,95	0,4309	0,2819	0,93	0,1759	5,8607	1,27	-
0,25 ; 1,00 ; 0,75 ; 0,25	0,0163	1,09	-	0,2514	0,83	**0,0055**	5,7802	1,25	-
0,50 ; 1,00 ; 0,50 ; 0,25	0,0130	0,88	0,3602	0,2886	0,95	0,2771	4,3377	0,94	0,3345
0,50 ; 1,00 ; 0,50 ; 0,50	0,0148	0,99	0,4908	0,2823	0,93	0,2232	4,8462	1,05	-
0,50 ; 1,00 ; 0,75 ; 0,25	0,0179	1,20	-	0,3027	1,00	-	4,5384	0,98	0,4439
1,00 ; 1,00 ; 1,00 ; 0,50	0,0206	1,39	-	0,3038	1,00	-	3,9660	0,86	0,1477
1,00 ; 1,00 ; 1,00 ; 1,00	0,0132	0,89	0,3351	0,2833	0,94	0,2047	4,3657	0,94	0,3385

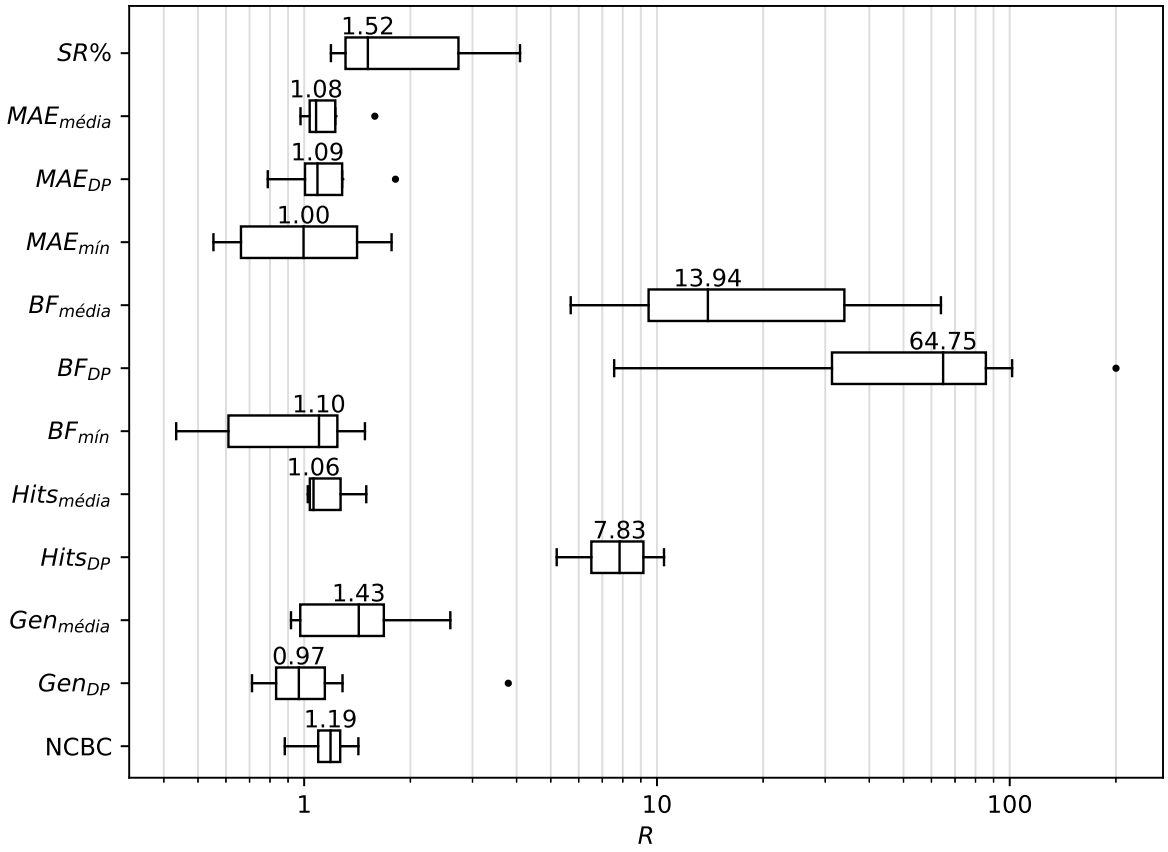
Fonte: o autor (2023)

Tabela 16 – Melhor *relação de melhoria* encontrada em cada problema e sua alteração de parâmetro relacionada

Problema	Melhor $\frac{\overline{BF}}{\overline{\text{BF}}_{\text{ref}}}$	Parâmetro alterado
Raiz cúbica	0,28	$[n_{min}, n_{max}] = [42{,}49]$
Função cúbica	0,48	$(R\%; Q\%) = 30; 70$
Função Gaussiana	0,40	$(R\%; Ma\%) = 10; 90$
Raiz quadrada	0,70	$[n_{min}, n_{max}] = [28{,}35]$
Função quadrática	0,71	$[n_{min}, n_{max}] = [28{,}35]$
Sensor de temperatura	0,76	$[n_{min}, n_{max}] = [35{,}42]$
Referência de tensão	0,43	$[n_{min}, n_{max}] = [42{,}49]$
Média	0,54	

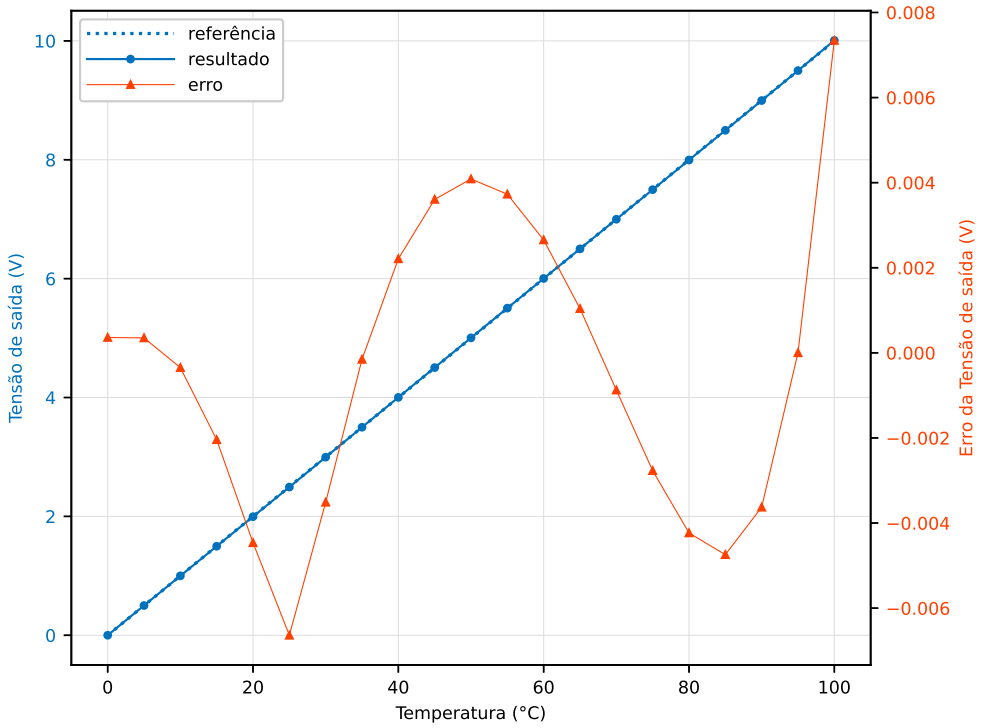
Fonte: o autor (2023)

Figura 25 – Relação de desempenho R entre o SANN-GCE, utilizando os parâmetros ajustados, e o ACID-MGE em cada métrica



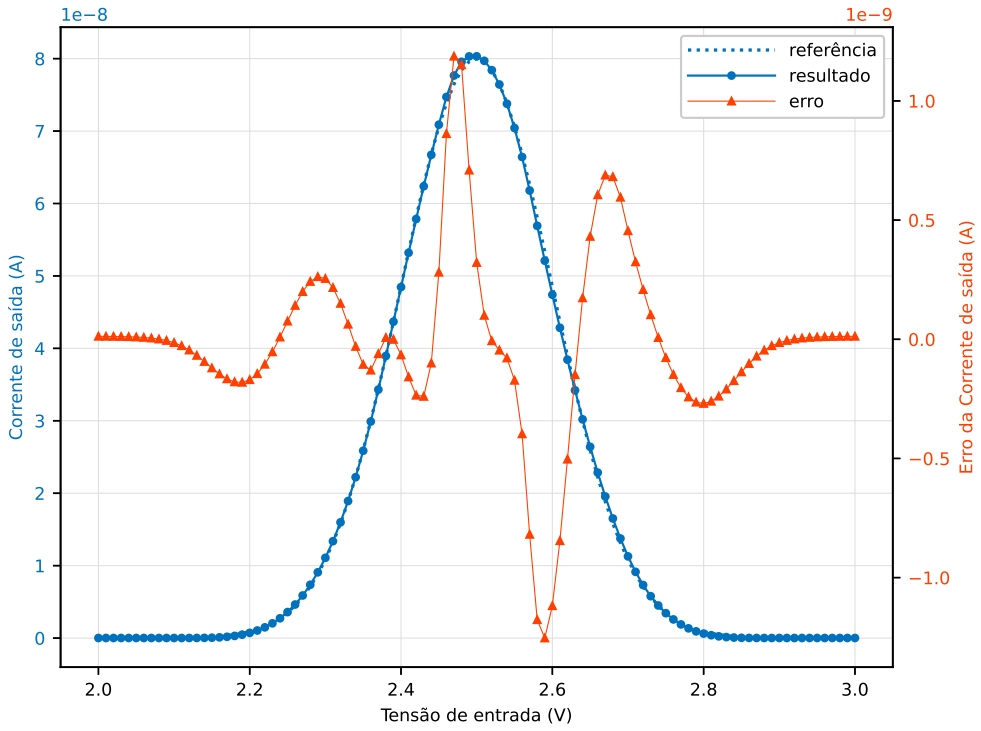
Fonte: o autor (2023)

Figura 26 – Resposta do melhor circuito para o problema *sensor de temperatura* obtido com o SANN-GCE



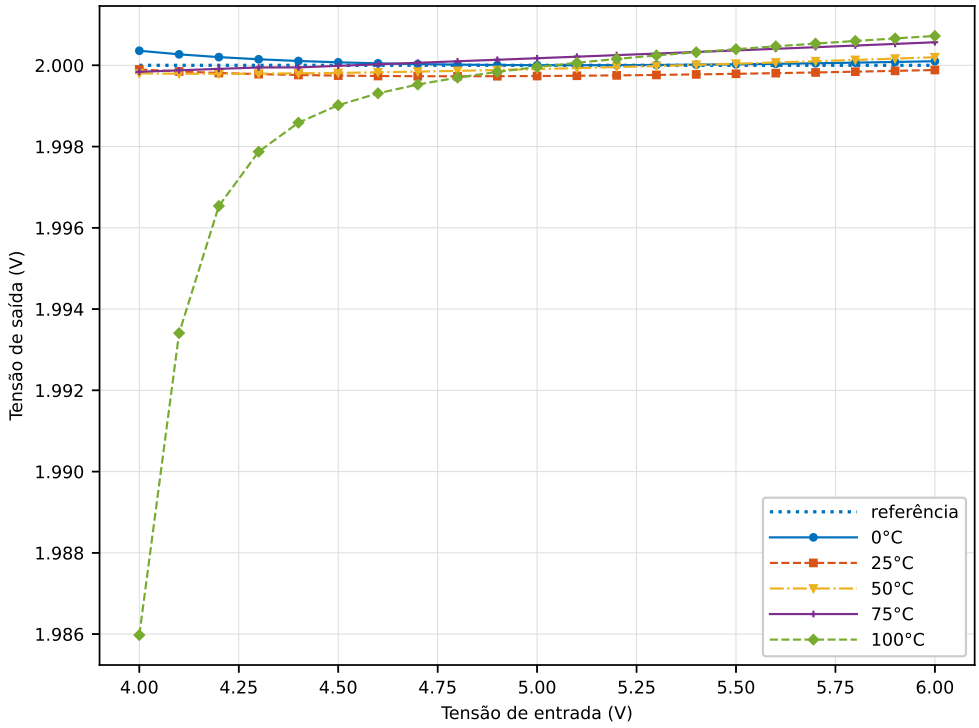
Fonte: o autor (2023)

Figura 27 – Resposta do melhor circuito para o problema *função Gaussiana* obtido com o SANN-GCE



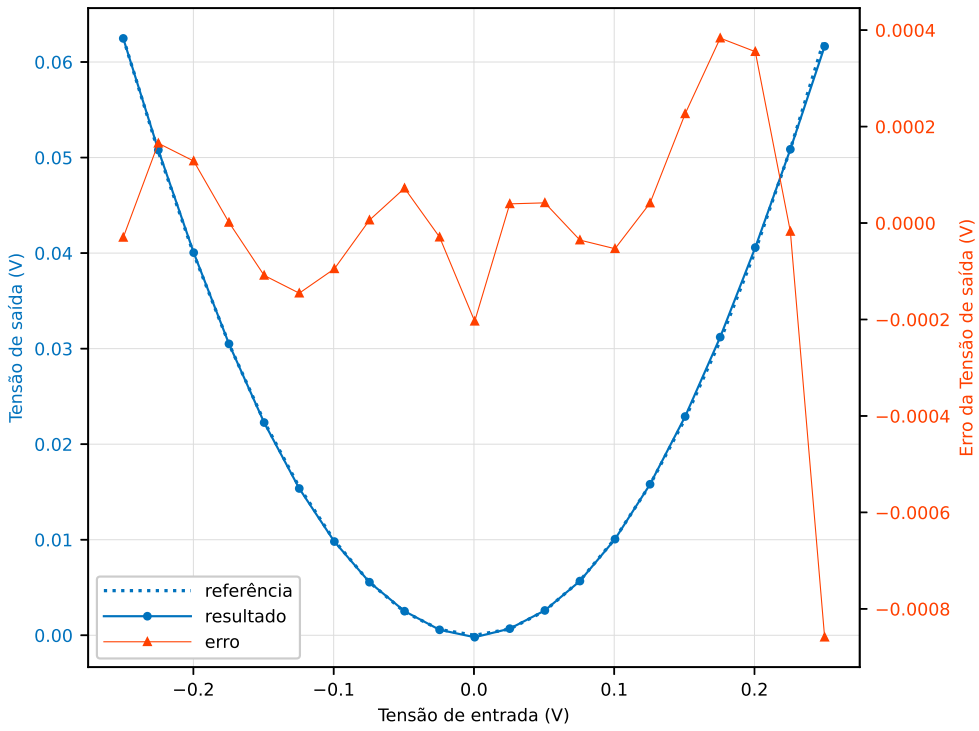
Fonte: o autor (2023)

Figura 28 – Resposta do melhor circuito para o problema *referência de tensão* obtido com o SANN-GCE



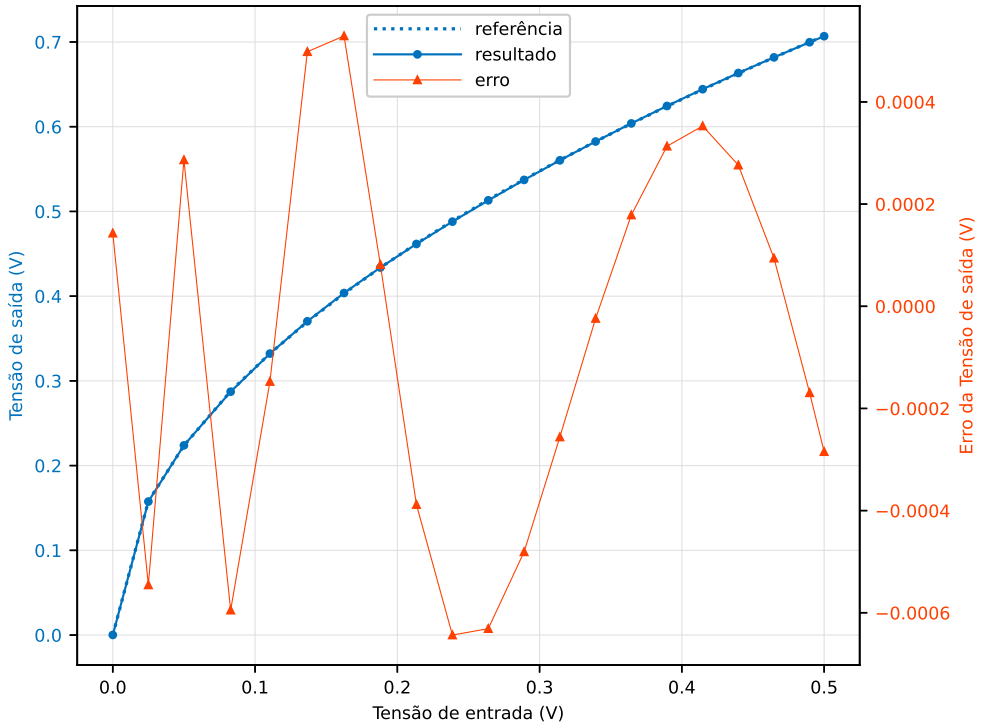
Fonte: o autor (2023)

Figura 29 – Resposta do melhor circuito para o problema *função quadrática* obtido com o SANN-GCE



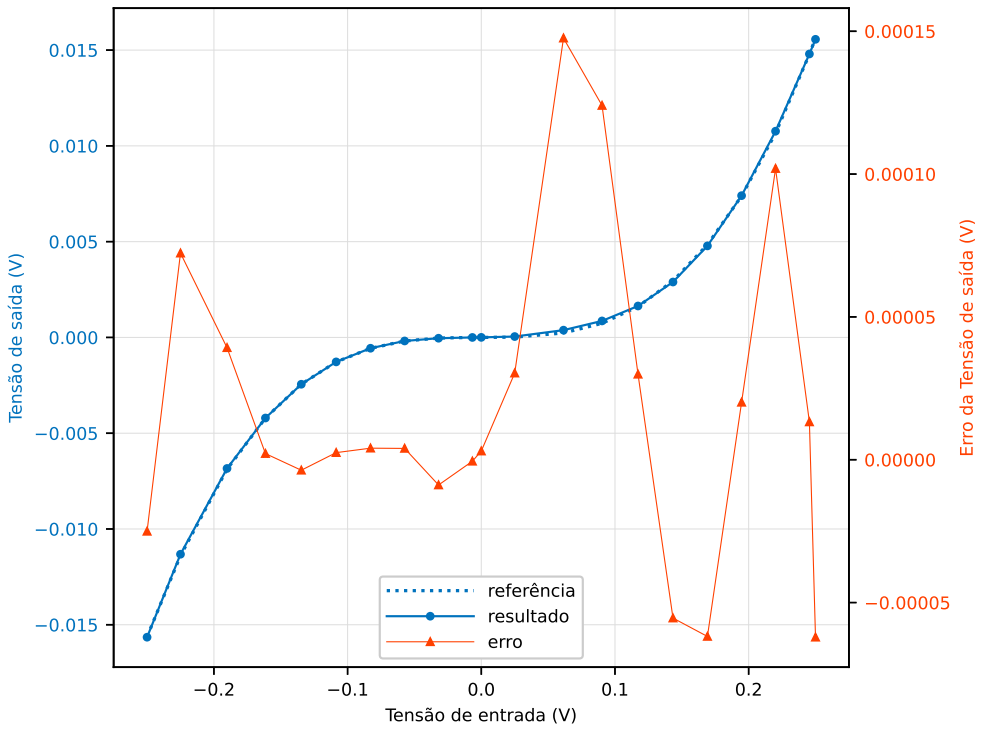
Fonte: o autor (2023)

Figura 30 – Resposta do melhor circuito para o problema *raiz quadrada* obtido com o SANN-GCE



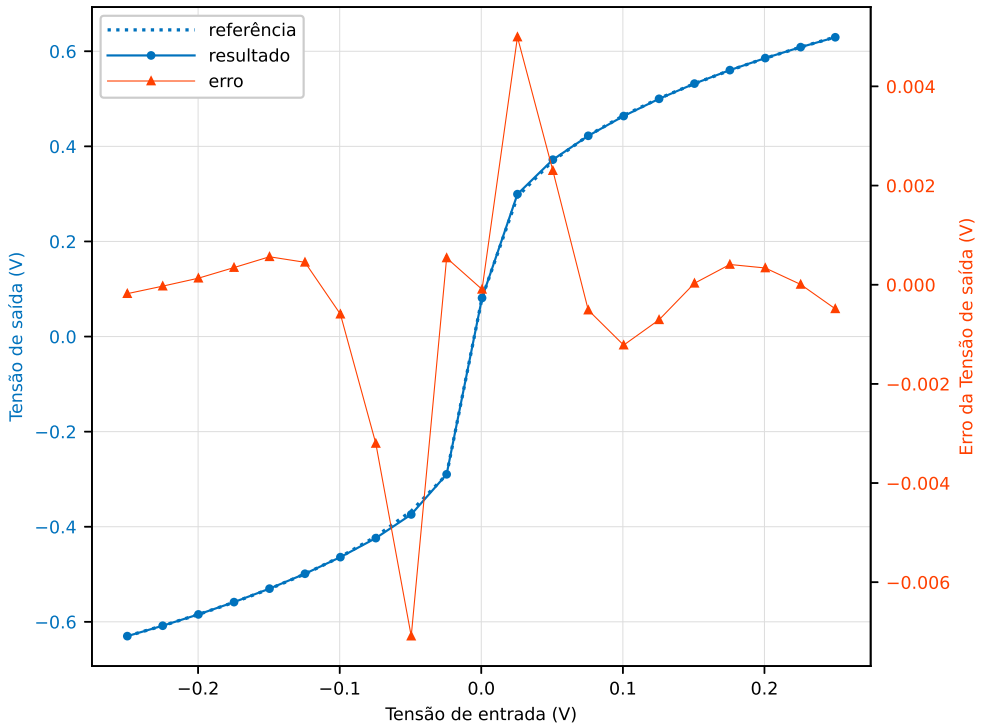
Fonte: o autor (2023)

Figura 31 – Resposta do melhor circuito para o problema *função cúbica* obtido com o SANN-GCE



Fonte: o autor (2023)

Figura 32 – Resposta do melhor circuito para o problema *raiz cúbica* obtido com o SANN-GCE



Fonte: o autor (2023)

7.4 Tempo de execução

Esta seção tem como objetivos a avaliação do tempo de execução do algoritmo SANN-GCE utilizando os parâmetros ajustados e a posterior comparação com o ACID-MGE. Considerando que há diferenças na forma como esses algoritmos utilizam a *computação paralela*, que há diferenças substanciais entre os computadores utilizados para executar o ACID-MGE e o SANN-GCE, e visando tentar obter uma comparação justa do tempo de execução entre estes dois algoritmos, neste trabalho foi utilizado como métrica para o tempo de execução dos algoritmos o *tempo de execução normalizado para executar uma rodada utilizando um núcleo do processador*. Isto difere da metodologia empregada pelo autor em (MUTTONI; VEIGA, 2023), onde foi utilizado o *tempo de execução normalizado para executar 50 rodadas em um computador*, que não considerou diferenças na quantidade de núcleos dos processadores dos computadores utilizados.

Inicialmente, será analisado o tempo de execução do ACID-MGE. Em seu artigo de apresentação, é afirmado que:

> O tempo para uma execução no ACID-MGE depende do circuito evolutivo, do número de gerações e do hardware utilizado. Por exemplo, para 3000 gerações, o tempo médio de execução é de 60-70 min, usando um *cluster* de oito computadores: dois deles com processador Core i5-7500@3,40 GHz, dois com processador Core i5-6500@3,2 GHz e quatro com processador Core 2 Quad@2,66 GHz. (CASTEJÓN; CARMONA, 2020, p. 11, tradução nossa)

Nesta descrição, é aparente que o ACID-MGE utiliza *computação paralela*, distribuindo a carga de *uma única execução* por todos os 8 computadores do *cluster*, cuja configuração foi sumarizada na Tabela 17. Nesta tabela, é indicada a quantidade de núcleos de cada computador, e é possível concluir que este *cluster* contém 32 núcleos de processamento. Como o tempo médio de *uma execução* (uma rodada) do ACID-MGE foi de 65 minutos[3] quando utilizados 32 núcleos, aplicando-se um modelo linear, é possível afirmar que, se o ACID-MGE fosse executado em apenas um núcleo, esse tempo de execução seria 32 vezes maior, ou seja, seria de 2080 minutos (34,67 horas ou 34h40m).

Já o SANN-GCE, após o ajuste de parâmetros, utilizou o hardware da *nuvem* AWS para ser executado, conforme detalhado na seção 5.5. Este ambiente de execução foi composto por uma instância (máquina virtual) do tipo `c6i.32xlarge` que utilizou 2 processadores *Intel Xeon Platinum 8375C @ 2,90 GHz*[4], perfazendo um total de 64 núcleos[5].

[3] Média do intervalo informado (60 a 70 minutos)

[4] <https://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%208375C.html>

[5] Os 64 núcleos dão origem a 128 *threads* (também chamadas de vCPUs no contexto das máquinas virtuais), pois a tecnologia *Simultaneous multithreading (SMT)* (chamada de Hyper-Threading pela Intel) estava habilitada por padrão naquela máquina virtual, e desse modo cada núcleo do processador expõe 2 núcleos lógicos para o sistema operacional, totalizando as 128 vCPUs

Tabela 17 – Detalhe do *cluster* de computadores utilizado pelo ACID-MGE

Quantidade de computadores (A)	Processador utilizado	Núcleos por computador (B)	Núcleos (subtotal) ($A \times B$)
2	Core i5-7500 @ 3,40 GHz	4[1]	8
2	Core i5-6500 @ 3,20 GHz	4[2]	8
4	Core 2 Quad @ 2,66 GHz	4[3]	16
		Total de núcleos:	32

[1] <https://www.intel.com.br/content/www/br/pt/products/sku/97123/intel-core-i57500-processor-6m-cache-up-to-3-80-ghz/specifications.html>
[2] <https://www.intel.com.br/content/www/br/pt/products/sku/88184/intel-core-i56500-processor-6m-cache-up-to-3-60-ghz/specifications.html>
[3] <https://ark.intel.com/content/www/br/pt/ark/products/38512/intel-core-2-quad-processor-q8400-4m-cache-2-66-ghz-1333-mhz-fsb.html>

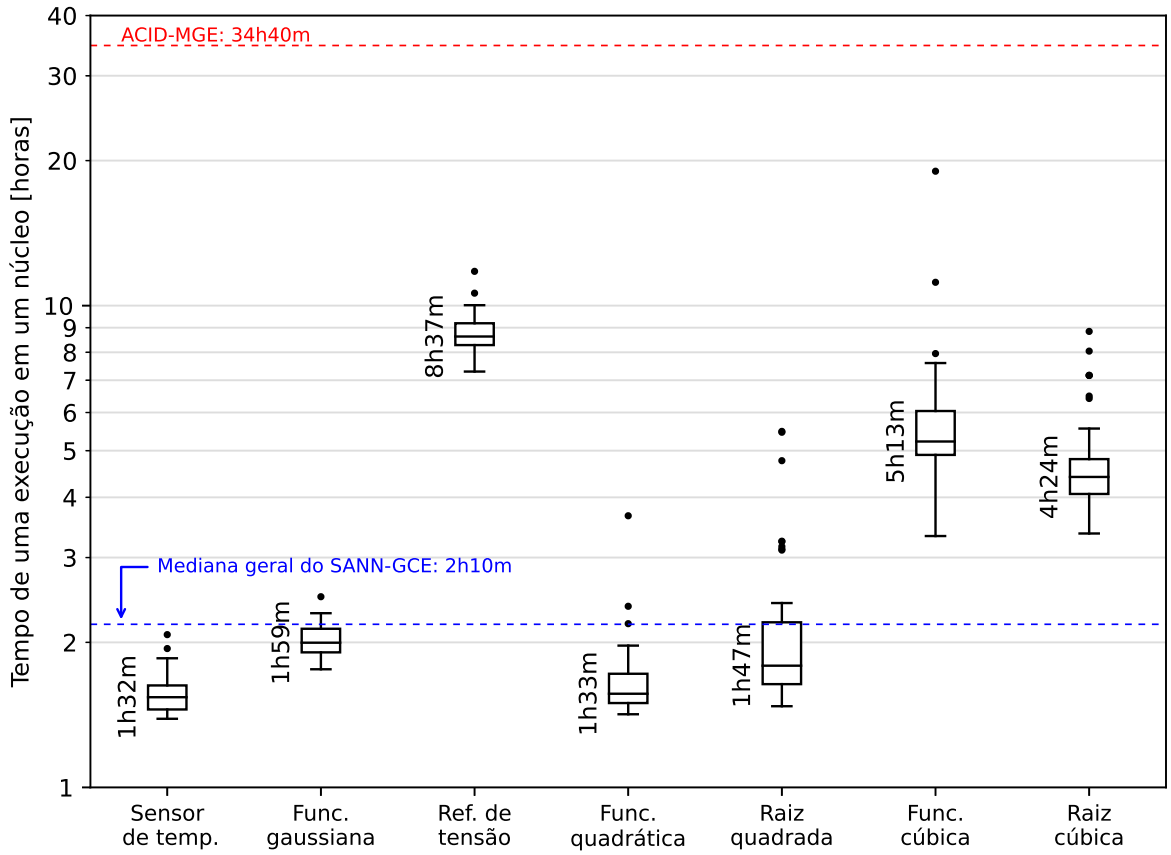
Fonte: o autor (2024), com dados obtidos de Castejón e Carmona (2020)

Para aproveitar toda a capacidade de processamento dessa máquina virtual, o *Queue Manager* (detalhado na seção 5.3.1) foi utilizado para distribuir a carga de até 64 execuções simultâneas do SANN-GCE pelos 64 núcleos da máquina virtual. Esta configuração permite que o tempo de execução de até 64 rodadas do SANN-GCE seja equivalente ao tempo gasto em única execução isolada. Isto ocorre porque o SANN-GCE utiliza apena um núcleo, portanto cada execução é independente e cada núcleo fica ocupado apenas com uma tarefa. Assim, o paralelismo do SANN-GCE é obtido por meio de suas múltiplas execuções através do *Queue Manager*, e não ao nível de algoritmo.

O tempo de cada uma das 50 execuções do SANN-GCE no hardware supracitado, em cada um dos sete casos de uso, foi registrado nos seus arquivos de saída no campo `vs_rodadas.tempo_gasto_segundos`, conforme explicado na seção 2.9. A compilação destes tempos é apresentada na Figura 33, através de um diagrama de caixas. Nessa figura, cada caixa apresenta a distribuição dos 50 tempos de execução de um caso de uso. A interpretação dos elementos das caixas dessa figura, incluindo suas indicações de mediana, quartis, LTI, LTS e *outliers*, é a mesma da apresentada na seção 7.2 para a Figura 24.

Ainda na Figura 33, é possível observar que alguns casos de uso exigem um tempo de execução consideravelmente maior. Por exemplo, os problemas *referência de tensão*, *função cúbica* e *raiz cúbica* possuíram medianas de 8h37, 5h13m e 4h24m, respectivamente, enquanto os demais 4 casos de uso demandaram menos de 2h. No caso da função cúbica e da raiz cúbica, este tempo de execução mais dilatado pode ser explicado pelo uso de uma placa maior (5×5 na primeira e 6×6 na segunda), ao passo que os demais casos de uso utilizaram uma placa de dimensões 4×4, de acordo com a Tabela 6. Já no problema referência de tensão, este tempo a maior pode ser atribuído à complexidade inerente do

Figura 33 – Diagrama de caixas do tempo de uma execução em um núcleo do SANN-GCE em cada caso de uso, utilizando os parâmetros ajustados



Fonte: o autor (2024)

problema, pois sua simulação SPICE envolve a variação de duas dimensões para avaliar seu desempenho: tensão de entrada e a temperatura.

Considerando todas as 350 execuções do SANN-GCE nas configurações supracitadas (50 rodadas × 7 casos de uso), a mediana[6] do tempo de execução foi de 2h10m. É salutar enfatizar que esta duração representa o tempo gasto para executar o SANN-GCE uma vez em apenas um núcleo do processador. Esse tempo é 15,9× inferior quando comparado com o tempo de execução normalizado do ACID-MGE (de 34h40m, caso fosse executado em apenas um núcleo). Ambos os tempos de execução agregados são mostrados na Figura 33 através de linhas horizontais.

Este bom resultado do SANN-GCE deve ser observado com cautela devido às diferenças inerentes de hardware e software entre os algoritmos analisados, algumas evidentes, outras incertas, pois o algoritmo comparado não forneceu dados o suficiente sobre a sua implementação. Ainda assim, é possível enumerar alguns fatores que podem ter

[6] Foi utilizada a mediana em vez da média porque os dados sugerem uma distribuição assimétrica, com a presença de *outliers*

produzido contribuições significativas para o desempenho positivo no tempo de execução do SANN-GCE:

1. Forte integração do Ngspice com o `circ_autoproj`: O Ngspice é utilizado diretamente através de chamadas de funções disponibilizadas pela sua *biblioteca* `libngspice.so`, conforme detalhado na seção 2.5. Esse método requer uma única inicialização e finalização do Ngspice junto com o `circ_autoproj`, o que traz eficiência em seu uso. Isso se contrapõe com a chamada ingênua do *executável* do Ngspice a cada necessidade de se avaliar a aptidão de uma solução candidata, que normalmente incorreria em um *overhead* significativo, pois, o sistema operacional precisaria lançar (e posteriormente finalizar) o *processo* do Ngspice centenas de vezes a cada segundo[7];

2. Linguagem de programação compilada e eficiente: O framework `circ_autoproj` e o SANN-GCE foram desenvolvidos em C++, uma linguagem que reconhecidamente gera código de máquina eficiente, às custas de uma maior complexidade em seu desenvolvimento. Isso pode ser colocado em contraste com linguagens mais ágeis, como Python e Matlab, que normalmente apresentam um *overhead* maior;

3. Simplicidade do algoritmo de busca: O recozimento simulado, utilizado neste trabalho pelo SAAN-GCE, é um algoritmo muito mais simples do que os algoritmos genéticos, utilizados tanto no ACID-GE quanto no ACID-MGE. O recozimento simulado trabalha apenas com uma solução por vez e utiliza unicamente a operação de mutação para alterar uma solução candidata, ao passo que o ACID-GE e o ACID-MGE utilizaram uma população de 1000 soluções candidatas e fazem uso tanto da mutação quanto do operador *recombinação*. Esta diferença de complexidade entre os algoritmos pode ter sido um fator importante para a obtenção desta vantagem no tempo de execução do SANN-GCE.

Por outro lado, há outros fatores difíceis ou impossíveis de se avaliar que poderiam reduzir esta vantagem aparente do SANN-GCE:

1. Velocidade de um núcleo do processador: Mesmo possuindo frequências de *clock* semelhantes, o processador Intel Xeon utilizado pelo SANN-GCE é mais moderno e poderoso que os processadores utilizados no *cluster* do ACID-MGE (Core i5 e Core 2 Quad) e seu desempenho por núcleo é sabidamente melhor. Entretanto, é improvável este fator seja o responsável por uma parcela significativa da vantagem de 15,9× obtida pelo SANN-GCE;

[7] Considerando uma rodada com critério de parada de 3 milhões de avaliações da função de aptidão, com duração de 2h (dados típicos de configurações do SANN-GCE empregadas neste trabalho), haveria cerca de 416 chamadas do executável Ngspice por segundo

2. Uso do modelo linear para inferir o tempo de execução em um núcleo do ACID-MGE: É provável que o tempo e execução do ACID-MGE obedeça a um modelo razoavelmente distante do modelo linear adotado neste trabalho para sua análise. Neste caso, o tempo de execução do ACID-MGE em apenas um núcleo seria menor que as 34h40m utilizadas neste trabalho. O uso do modelo linear se justifica pela ausência de informações detalhadas a respeito da implementação computacional do ACID-MGE.

Considerando este tempo de execução expressivamente menor em favor do SANN-GCE, seria possível testar mais soluções caso o critério de parada fosse estendido para ocupar todo o tempo utilizado pelo ACID-MGE. Isso provavelmente possibilitaria a obtenção de soluções (circuitos) com aptidões (qualidades) ainda melhores.

Capítulo 8

Conclusão

Neste trabalho foram desenvolvidos e avaliados o algoritmo para síntese de circuitos eletrônicos analógicos, denominado SANN-GCE, e o framework que o suporta, intitulado de `circ_autoproj`. O SANN-GCE é composto pelo algoritmo de busca *Recozimento Simulado* em conjunto com a representação de solução *Evolução Geométrica de Circuitos - GCE*.

O framework `circ_autoproj` foi apresentado, incluindo sua arquitetura modular, detalhes de seus componentes e de seu funcionamento que permite a realização da síntese de circuitos eletrônicos. Seus *plugins* SA e SR, responsáveis respectivamente pela condução da otimização e pela codificação das soluções candidatas foram abordados. O *problema SPICE* também foi apresentado, um *plugin* responsável por acomodar as especificações do circuito a ser sintetizado, incluindo a função objetivo do problema. A integração do simulador de circuitos Ngspice com o framework, fundamental para avaliar as soluções candidatas durante o processo de síntese e otimização, também foi explicada. Foram discutidos o formato e a estrutura dos arquivos de entrada e saída do framework, que permitiram estabelecer configurações e resultados padronizados, facilitando sua posterior análise.

Foram apresentados os principais conceitos e o funcionamento da simulação computacional de circuitos eletrônicos utilizando o simulador Ngspice. Os principais tipos de análises, a sintaxe da netlist para a especificação de elementos, o uso de modelos de componentes e alguns de seus comandos importantes foram abordados.

O SANN-GCE foi apresentado, explicitando a sua bipartição entre o algoritmo de busca e a representação da solução, imposta pelo `circ_autoproj`. O algoritmo de busca recozimento simulado e sua interface com o GCE foram explicados, incluindo o seu algoritmo de mutação, primordial para o bom desempenho da otimização. Também foi abordado o algoritmo de tradução, que decodifica uma solução candidata em uma netlist, permitindo que ela seja encaminhada para o simulador de circuitos e tenha a aptidão calculada. A representação de solução GCE foi introduzida em conjunto com

seus principais conceitos, como a sua hierarquia de objetos e os seus *graus de liberdade* categorizados, que permitem que diferentes características de um circuito sejam alteradas com probabilidades distintas durante o processo de construção da solução.

Também foram introduzidos os conceitos da computação em nuvem e seu uso como ferramenta para acelerar a execução do `circ_autoproj` através de seus recursos de escalabilidade e desempenho computacional. Isto permitiu a execução de centenas de sínteses de circuitos em tempo hábil, gerando um grande conjunto de dados que possibilitou a avaliação do SANN-GCE sob diferentes casos de uso e configurações.

O framework e o algoritmo SANN-GCE foram avaliados por meio de sete diferentes casos de uso: circuito sensor de temperatura, função Gaussiana, referência de tensão, função quadrática, raiz quadrada, função cúbica e raiz cúbica, primeiramente utilizando *parâmetros iniciais*, não otimizados. Em cada um dos casos, foram executadas 50 rodadas com o objetivo de se obter resultados estatisticamente representativos que permitam comparações posteriores com maior fidelidade.

Os resultados obtidos foram avaliados por meio de 12 métricas de desempenho e comparados com os algoritmos ACID-GE e ACID-MGE. As métricas utilizadas foram: taxa de sucesso (SR%), erro médio absoluto (mínimo – $\text{MAE}_{\text{mín}}$, média – $\text{MAE}_{\text{média}}$ e desvio padrão – MAE_{DP}), melhor aptidão (mínima – $\text{BF}_{\text{mín}}$, média – $\text{BF}_{\text{média}}$ e desvio padrão BF_{DP}), porcentagem de êxito (média – $\text{Hits}_{\text{média}}$ e desvio padrão – Hits_{DP}), número de gerações para se obter sucesso (média – $\text{Gen}_{\text{média}}$ e desvio padrão – Gen_{DP}) e número de componentes do melhor circuito (NCBC). Além destas métricas de desempenho da solução, em cada rodada do SANN-GCE seu tempo gasto foi coletado, o que permitiu uma análise pormenorizada da sua velocidade de execução e posterior comparação com o ACID-MGE.

Depois da avaliação com os parâmetros iniciais, foram executados ajustes nos parâmetros do SANN-GCE com o objetivo de otimizar os seus resultados. O ajuste de parâmetros consistiu na alteração de sete parâmetros selecionados em 41 configurações distintas. O SANN-GCE foi testado com essas novas configurações em cada um dos casos de uso, e a significância estatística de seus resultados, quando comparados com os obtidos usando os parâmetros iniciais, foram validados através do Teste de Permutação. Os resultados obtidos com as melhores configurações encontradas para cada caso de uso, após o ajuste de parâmetros, foram então utilizados em uma nova rodada de comparações do SANN-GCE com o ACID-GE e ACID-MGE.

Através dos resultados obtidos, foi evidenciado por meio da comparação da mediana de cada métrica (calculada ao longo de todos os casos de uso) que o SANN-GCE apresentou melhores resultados em 11 das 12 métricas quando comparado ao ACID-MGE. Dentre os resultados obtidos, destacam-se as seguintes métricas: foi obtida uma mediana 1,52× melhor (maior) para a métrica SR%, 13,94× melhor (menor) para a $\text{BF}_{\text{média}}$, 64,75× melhor

(menor) na métrica $\mathrm{BF_{DP}}$, e 7,83× melhor (maior) para a $\mathrm{Hits_{DP}}$. Ainda, a mediana do tempo de execução do SANN-GCE foi 15,9× menor quando comparada com o tempo de execução normalizado do ACID-MGE.

A obtenção destes resultados expressivos, especialmente nas métricas $\mathrm{BF_{média}}$, $\mathrm{BF_{DP}}$ e $\mathrm{Hits_{DP}}$, indicam que o SANN-GCE foi capaz de sintetizar circuitos eletrônicos com um desempenho melhor e com menor variabilidade quando comparado ao ACID-MGE. Em uso real, isso significa que é possível obter circuitos eletrônicos mais precisos utilizando menos execuções e, considerando o seu tempo de execução muito menor, estas soluções podem ser encontradas em um intervalo de tempo muito mais reduzido.

O ajuste de parâmetros, por sua vez, revelou 19 resultados melhores e estatisticamente significativos quando comparados com os resultados obtidos utilizando os parâmetros iniciais. As melhorias abrangeram todos os sete circuitos de teste com valor p médio de 0,00993 – cerca de cinco vezes inferior ao valor utilizado habitualmente de 0,05. Dos sete parâmetros avaliados, apenas 2 foram responsáveis pelos resultados positivos encontrados: 1. n_{min}, n_{max}; e 2. probabilidade de tipos de componentes. O primeiro parâmetro especifica a quantidade mínima e máxima de componentes admitidos em um circuito, enquanto o segundo indica as probabilidades relativas de síntese de cada tipo de componente em uma solução. Estes dois parâmetros têm em comum a categoria de seus DOFs subordinados: ambos são do tipo *estrutura*.

Os testes abrangentes realizados demonstraram a bom desempenho do SANN-GCE sob diversas métricas. Esses resultados revelaram a eficácia do algoritmo e podem abrir caminho para futuras pesquisas e aplicações práticas. Espera-se que os desenvolvimentos relatados neste trabalho possam contribuir com avanços na área da engenharia elétrica, apresentando um caminho para aprimorar o projeto de circuitos eletrônicos analógicos, tornando-o mais eficiente e preciso.

Apesar destes resultados positivos, é de suma importância apontar e discutir as limitações desse estudo. Primeiramente, o algoritmo de busca utilizado permite a otimização de apenas um objetivo. Isto limita a sua aplicação, não permitindo, por exemplo, que seja otimizado o tamanho do circuito e sua precisão ao mesmo tempo. Outra limitação do SANN-GCE é a ausência de um pós-processamento da netlist. Isto eventualmente resulta em circuitos com redundâncias, como a associação de alguns resistores em série ou em paralelo. Há também casos de geração de componentes sem função, como transistores conectados ao circuito que operam sempre em corte ou saturação. Além disso, os circuitos criados pelo algoritmo são apresentados ao usuário apenas através de sua netlist, não havendo integração com um software tradicional de automação de projeto eletrônicos.

Com a finalidade de aperfeiçoar o desenvolvimento exposto neste trabalho e suprimir algumas de suas limitações, são elencadas as seguintes atividades como trabalhos futuros:

1. Emprego de um otimizador multiobjetivo que permita, por exemplo, otimizar o consumo de energia e a precisão do circuito simultaneamente;

2. Aplicação de um pós-processamento na netlist, para simplificar ligações que associam em série ou em paralelo alguns tipos de componentes, como resistores, e também remover componentes sem função, como transistores conectados ao circuito que operam sempre em corte ou saturação;

3. Investigação do motivo da alta sensibilidade dos parâmetros n_{min}, n_{max} e *probabilidade de tipos de componentes* e proposição de melhorias no algoritmo baseadas neste achado;

4. Avaliação de outros algoritmos de busca no lugar do Recozimento Simulado, para trabalhar em conjunto com o GCE;

5. Desenvolvimento de um software que faça o desenho do esquemático do circuito gerado a partir de sua netlist e o integre a um software EDA tradicional – por exemplo, ao *KiCad*.

6. Avaliação da influência dos parâmetros não abordados neste trabalho: tamanho da placa ($M \times N$) e o mapa dos nós externos;

7. Uso de uma estratégia automatizada para o ajuste de parâmetros que permita que diversos parâmetros sejam alterados simultaneamente, reduzindo assim o tempo gasto para otimizar o algoritmo;

8. Análise da viabilidade e possível implementação do ajuste de parâmetros *online*, onde os parâmetros são atualizados dinamicamente durante a execução;

Ao finalizar este trabalho, é inevitável ponderar sobre todo o percurso percorrido desde o planejamento até a sua conclusão. O desenvolvimento desta tese não se restringiu apenas a uma busca por novos saberes, mas também representou uma chance de desenvolvimento pessoal e profissional.

O desenvolvimento do algoritmo SANN-GCE, de seu framework `circ_autoproj` e de inúmeras ferramentas auxiliares não relatadas aqui, foi um projeto desafiador e enriquecedor, que proporcionou uma imersão nos conceitos de otimização e de projetos de circuitos analógicos. Durante essa jornada, houve inúmeros desafios que exigiram resiliência e capacidade criativa. Em certos momentos, os resultados obtidos não estavam de acordo com as expectativas, o que exigiu aprimoramentos e, em alguns casos, mudanças profundas nos métodos.

Após escrever, revisar, reescrever, programar, corrigir, executar, e, enfim, chegar ao final desta jornada, surge em mim um sentimento genuíno de agradecimento pela chance

de colaborar com o progresso do saber e pela oportunidade de criar respostas que talvez um dia tragam benefícios para a sociedade de formas ainda desconhecidas. Este trabalho é fruto de uma paixão profunda pela eletrônica e computação, assim almejo que ele estimule estudos vindouros que sigam impulsionando a fronteira do possível.

Referências

AMAZON. *Amazon EC2: Now in Unlimited Beta and Launching New Instance Types*. 2007. Disponível em: <https://aws.amazon.com/about-aws/whats-new/2007/10/22/amazon-ec2-now-in-unlimited-beta-and-launching-new-instance-types/>. Acesso em: 6 nov. 2023.

AMAZON. *Amazon EC2 Exits Beta and Now Offers a Service Level Agreement*. 2008. Disponível em: <https://aws.amazon.com/about-aws/whats-new/2008/10/23/amazon-ec2-exits-beta-and-now-offers-a-service-level-agreement/>. Acesso em: 6 nov. 2023.

AMAZON. *Amazon Compute Service Level Agreement*. 2022. Disponível em: <https://aws.amazon.com/compute/sla/>. Acesso em: 8 nov. 2023.

AMAZON. *AWS Cloud Credit for Research*. 2023. Disponível em: <https://aws.amazon.com/government-education/research-and-technical-computing/cloud-credit-for-research/>. Acesso em: 17 set. 2023.

AMAZON. *AWS Free Tier*. 2023. Disponível em: <https://aws.amazon.com/free>. Acesso em: 2 ago. 2023.

AMAZON. *Compute - Amazon EC2 Instance Types - AWS*. 2023. Disponível em: <https://aws.amazon.com/ec2/instance-types/>. Acesso em: 17 set. 2023.

AMAZON. *EC2 On-Demand Instance Pricing - Amazon Web Services*. 2023. Disponível em: <https://aws.amazon.com/ec2/pricing/on-demand/>. Acesso em: 17 set. 2023.

BALOGH, A. *Google Cloud Platform Blog: Google Compute Engine is now Generally Available with expanded OS support, transparent maintenance, and lower prices*. 2013. Disponível em: <https://cloudplatform.googleblog.com/2013/12/google-compute-engine-is-now-generally-available.html>. Acesso em: 6 nov. 2023.

BARR, J. *Amazon EC2 Beta | AWS News Blog*. 2006. Disponível em: <https://aws.amazon.com/blogs/aws/amazon_ec2_beta/>. Acesso em: 6 nov. 2023.

BRAY, T. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC Editor, 2017. RFC 8259. (Request for Comments, 8259). Disponível em: <https://www.rfc-editor.org/info/rfc8259>. Acesso em: 15 jun. 2022. DOI: https://doi.org/10.17487/RFC8259.

CASTEJÓN, F.; CARMONA, E. J. Automatic design of analog electronic circuits using grammatical evolution. *Applied Soft Computing*, Elsevier BV, v. 62, p. 1003–1018, jan. 2018. DOI: https://doi.org/10.1016/j.asoc.2017.09.036.

CASTEJÓN, F.; CARMONA, E. J. Introducing modularity and homology in grammatical evolution to address the analog electronic circuit design problem. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 137275–137292, ago. 2020. DOI: https://doi.org/10.1109/access.2020.3011641.

DELAHAYE, D.; CHAIMATANAN, S.; MONGEAU, M. Simulated annealing: From basics to applications. In: _____. *Handbook of Metaheuristics*. Cham: Springer International Publishing, 2019. p. 1–35. ISBN 978-3-319-91086-4. DOI: https://doi.org/10.1007/978-3-319-91086-4_1.

Engineering and Technology History Wiki. *Milestones: SPICE Circuit Simulation Program, 1970*. 2011. Disponível em: <https://ethw.org/Milestones:SPICE_Circuit_Simulation_Program,_1970>. Acesso em: 8 mar. 2022.

ERL, T.; MONROY, E. B. *Cloud Computing: Concepts, Technology, Security, and Architecture*. 2. ed. Upper Saddle River, NJ: Pearson, 2023. (The Pearson Digital Enterprise Series from Thomas Erl).

GOOD, P. I. *Permutation, Parametric, and Bootstrap Tests of Hypotheses*. New York, NY: Springer, 2004. 316 p. (Springer Series in Statistics). ISBN 978-0387202792.

GOOGLE. *Google App Engine Blog: App Engine 1.6.0 Out of Preview Release*. 2011. Disponível em: <https://googleappengine.blogspot.com/2011/11/app-engine-160-out-of-preview-release.html>. Acesso em: 6 nov. 2023.

GOOGLE. *Compute Engine Service Level Agreement (SLA) | Google Cloud*. 2023. Disponível em: <https://cloud.google.com/compute/sla>. Acesso em: 8 nov. 2023.

GOOGLE. *Free cloud features and trial offer*. 2023. Disponível em: <https://cloud.google.com/free/docs/free-cloud-features>. Acesso em: 21 set. 2023.

GOOGLE. *VM instance pricing | Compute Engine: Virtual Machines (VMs) | Google Cloud*. 2023. Disponível em: <https://cloud.google.com/compute/vm-instance-pricing>. Acesso em: 9 nov. 2023.

HARJANI, R.; RUTENBAR, R.; CARLEY, L. OASYS: a framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 8, n. 12, p. 1247–1266, 1989. DOI: https://doi.org/10.1109/43.44506.

JIAO, F.; DOBOLI, A. Three learning methods for reasoning-based synthesis of novel analog circuits. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. [S.l.]: IEEE Press, 2016. p. 2411–2414. DOI: https://doi.org/10.1109/ISCAS.2016.7539078.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. ISSN 00368075, 10959203. DOI: https://doi.org/10.1126/science.220.4598.671.

KOH, H.; SEQUIN, C.; GRAY, P. OPASYN: a compiler for CMOS operational amplifiers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 9, n. 2, p. 113–125, 1990. DOI: https://doi.org/10.1109/43.46777.

KOZA, J. et al. Synthesis of topology and sizing of analog electrical circuits by means of genetic programming. *Computer Methods in Applied Mechanics and Engineering*, v. 186, n. 2, p. 459–482, 2000. ISSN 0045-7825. DOI: https://doi.org/10.1016/S0045-7825(99)00397-7.

KOZA, J. R. et al. *Genetic programming III: Darwinian invention and problem solving*. San Francisco: Morgan Kaufmann, 1999. v. 3. ISBN 1558605436. DOI: https://doi.org/10.1109/TEVC.1999.788530.

LAARHOVEN, P. J. M. van; AARTS, E. H. L. Simulated annealing. In: _____. *Simulated Annealing: Theory and Applications*. Dordrecht: Springer Netherlands, 1987. p. 7–15. ISBN 978-94-015-7744-1. DOI: https://doi.org/10.1007/978-94-015-7744-1_2.

LIU, F. et al. *NIST Cloud Computing Reference Architecture*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2011. Disponível em: <https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=909505>. Acesso em: 9 nov. 2023. DOI: https://doi.org/10.6028/NIST.SP.500-292.

MAGNUSSON, P. S. *Google Cloud Platform Blog: Google Compute Engine launches, expanding Google's cloud offerings*. 2012. Disponível em: <https://cloudplatform.googleblog.com/2012/06/google-compute-engine-launches.html>. Acesso em: 6 nov. 2023.

MATTIUSSI, C.; FLOREANO, D. Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on Evolutionary Computation*, v. 11, n. 5, p. 596–607, 2007. DOI: https://doi.org/10.1109/TEVC.2006.886801.

MCCONAGHY, T. et al. Trustworthy genetic programming-based synthesis of analog circuit topologies using hierarchical domain-specific building blocks. *IEEE Transactions on Evolutionary Computation*, v. 15, n. 4, p. 557–570, 2011. DOI: https://doi.org/10.1109/TEVC.2010.2093581.

MCDONALD, P. *Google App Engine Blog: Introducing Google App Engine + our new blog*. 2008. Disponível em: <https://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html>. Acesso em: 6 nov. 2023.

MEISSNER, M.; HEDRICH, L. FEATS: Framework for explorative analog topology synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 34, n. 2, p. 213 – 226, 2015. DOI: https://doi.org/10.1109/TCAD.2014.2376987.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD, 2011. Disponível em: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf>. Acesso em: 8 nov. 2023. DOI: https://doi.org/10.6028/NIST.SP.800-145.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, v. 21, n. 6, p. 1087–1092, 1953. DOI: https://doi.org/10.1063/1.1699114.

MUTTONI, L.; VEIGA, A. C. P. Analog electronic circuit synthesis using simulated annealing and geometric circuit evolution. *Journal of Microwaves, Optoelectronics and Electromagnetic Applications*, Sociedade Brasileira de Microondas e Optoeletrônica

e Sociedade Brasileira de Eletromagnetismo, v. 22, n. 1, p. 13–32, mar. 2023. ISSN 2179-1074. DOI: https://doi.org/10.1590/2179-10742023v22i1265775.

NAGEL, L. W. *The History of SPICE*. 2013. Disponível em: <http://www.omega-enterprises.net/TheHistoryofSPICE.pdf>. Acesso em: 8 mar. 2022.

NGSPICE. *Ngspice, the open source Spice circuit simulator*. 2021. Disponível em: <http://ngspice.sourceforge.net/>. Acesso em: 23 nov. 2021.

O'NEILL, M.; RYAN, C. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, v. 5, n. 4, p. 349–358, ago. 2001. ISSN 1089-778X. DOI: https://doi.org/10.1109/4235.942529.

PHAM, D. T.; KARABOGA, D. *Intelligent optimisation techniques : genetic algorithms, tabu search, simulated annealing and neural networks*. [S.l.]: Springer London, 2000. ISBN 978-1-4471-0721-7. DOI: https://doi.org/10.1007/978-1-4471-0721-7.

ROJEC, Z.; BURMEN, A.; FAJFAR, I. Analog circuit topology synthesis by means of evolutionary computation. *ENGINEERING APPLICATIONS OF ARTIFICIAL INTELLIGENCE*, 80, p. 48–65, abr. 2019. ISSN 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2019.01.012.

RUPARELIA, N. B. *Cloud Computing*. Cambridge, MA: The MIT Press, 2016. (The MIT Press Essential Knowledge series).

SORKHABI, S. E.; ZHANG, L. Automated topology synthesis of analog and RF integrated circuits: A survey. *INTEGRATION-THE VLSI JOURNAL*, 56, p. 128–138, jan. 2017. ISSN 0167-9260. DOI: https://doi.org/10.1016/j.vlsi.2016.10.017.

SPANOS, C. *UCB Letterhead TT*. 2010. Disponível em: <https://ethw.org/w/images/4/46/IEEE.Plaque.SPICE.pdf>. Acesso em: 8 mar. 2022.

Synergy Research Group. *AI Helps to Stabilize Quarterly Cloud Market Growth Rate; Microsoft Market Share Nudges Up Again*. 2023. Disponível em: <https://www.srgresearch.com/articles/ai-helps-to-stabilize-quarterly-cloud-market-growth-rate-microsoft-market-share-nudges-up-again>. Acesso em: 3 nov. 2023.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Editora: Independente, 2020.

VOGT, H. et al. *Ngspice user manual*. 2022. Disponível em: <http://ngspice.sourceforge.net/docs/ngspice-manual.pdf>. Acesso em: 9 abr. 2022.

Apêndices

APÊNDICE A

Exemplo de arquivo de configuração

Este apêndice apresenta, a título de exemplo, o conteúdo de um arquivo de configuração no formato *JSON* utilizado pelo `circ_autoproj`.

Função quadrática

```
{
  "registrar_props_melhor_ind" : ["mae", "erro_total", "n_comp", "hits_ratio",
  ↪  "rmse"],
  "sr" : "plugins/sr/libplugin_sr-spice-gce.so",
  "params_sr": {
    "problema" :
    ↪  "plugins/problemas-spice/libplugin_problema-spice-squaring2.so",
    "board_params": {
      "tam_board": [4,4],
      "gen_spec_num_componentes": {
        "tipo": "RND_INT",
        "gen_params": [7,42]
      },
      "first_node": 100,
      "prob_regen_tipos": [
        ["TERMINAL",  1.0],
        ["CONTINUO",  1.0],
        ["DISCRETO",  1.0],
        ["ESTRUTURA", 0.25]
      ],
      "fixed_nodes": [
        [[0,1], 0],
        [[3,2], 0],
        [[0,2], 1],
        [[3,1], 2],
```

```
        [[1,0], 3],
        [[2,3], 4]
      ],
      "v_comp_spec": [
        {
          "tipo": "R",
          "prob": 70.0,
          "v_gen_spec":[{
            "tipo": "RND_STEPBOARDPOS",
            "gen_params": ["@tam_board"]
          },{
            "tipo": "RND_RESISTENCIA",
            "gen_params": []
          }]
        },{
          "tipo": "Q",
          "prob": 30.0,
          "v_gen_spec":[{
            "tipo": "RND_STEPBOARDPOS",
            "gen_params": ["@tam_board"]
          },{
            "tipo": "RND_ALTSTRING",
            "gen_params": [["2N3904", "2N3906"]]
          }]
        }
      ]
    },
    "path_libngspice": "libngspice.so",
    "max_cputime_sim_spice_us": 20000,
    "timeout_sim_spice_us": 1000000
  },
  "sa" : "plugins/sa/libplugin_sa-sann.so",
  "params_sa": {
    "T0": 100,
    "alfa": 0.9999,
    "pm": 0.015,
    "Treaq": 0.000000001
  },
  "criterio_parada": "NUM_AVS >= 3000000",
  "criterio_registro": "DIF_NUM_AVS >= 100000"
}
```

APÊNDICE B

Exemplo de arquivo de resultado

Este apêndice apresenta, a título de exemplo, o conteúdo de um arquivo de resultado no formato *JSON* gerado pelo `circ_autoproj`. Alguns campos desta listagem foram omitidos (com a devida indicação no texto) devido a sua grande extensão.

out.json

```
{
"appconfig": {
  "criterio_parada": "NUM_AVS >= 3000000",
  "criterio_registro": "DIF_NUM_AVS >= 100000",
  "params_sa": {"T0": 100, "Treaq": 1e-09, "alfa": 0.9999, "pm": 0.015},
  "params_sr": {
    "board_params": {
      "first_node": 100,
      "fixed_nodes":[[[0,1],0],[[0,2],1],[[2,3],3],[[3,1],2],[[3,2],0]],
      "gen_spec_num_componentes": {"gen_params": [7, 42],"tipo": "RND_INT"},
      "prob_regen_tipos":
      ↪  [["TERMINAL",1],["CONTINUO",1],["DISCRETO",1],["ESTRUTURA",0.25]],
      "tam_board": [4,4],
      "v_comp_spec": [{
          "prob": 70,
          "tipo": "R",
          "v_gen_spec": [
            {"gen_params": ["@tam_board"],"tipo": "RND_STEPBOARDPOS"},
            {"gen_params": [],"tipo": "RND_RESISTENCIA"}
          ]
        },{
          "prob": 30,
          "tipo": "Q",
          "v_gen_spec": [
            {"gen_params": ["@tam_board"],"tipo": "RND_STEPBOARDPOS"},
            {"gen_params": [["2N3904","2N3906"]],"tipo": "RND_ALTSTRING"}
          ]
```

```
        }]
    },
    "path_libngspice": "libngspice.so",
    "problema": "plugins/problemas-spice/libplugin_problema-spice-temperatura.so",
    "timeout_sim_spice_us": 1000000
  },
  "registrar_props_melhor_ind": ["mae","erro_total","n_comp","hits_ratio","rmse"],
  "sa": "plugins/sa/libplugin_sa-sann.so",
  "sr": "plugins/sr/libplugin_sr-spice-gce.so"
},
"file_format": "circ_autoproj_cpp",
"file_version": 2,
"host_info": {
  "argv": ["bin/circ_autoproj","-c","config-SANN-GCE-Temperatura-MIN-03.json","-o","r⌋
  ↪  esultado-SANN-GCE-Temperatura-MIN-03.json"],
  "cpu_info": (Suprimido devido à extensão),
  "hostname": "example.net",
  "path_exe": "/home/muttoni/circ_autoproj_cpp/.build_vscode/bin/circ_autoproj",
  "username": "muttoni",
  "version": "d052ada255f1826cb7ebc5eead7ea0b3383497ec"
},
"rnd": {
  "engine_name": "std::mt19937_64",
  "seed": 56477962
},
"rodada": 1,
"run_name": "SAnn-GCE-Temperatura",
"sa": {
  "nome": "SAnn",
  "num_reaq": 15,
  "objetivo": "MIN"
},
"sr": {
  "netlist_comandos": ".include ~/ngspice/2N3904.sp3\n.include
  ↪  ~/ngspice/2N3906.sp3\n.dc temp 0 100 5\n.save v(3)\n.end\n",
  "netlist_dt": "V1 1 0 dc 15\nV2 0 2 dc 15\nRs 3 0 1000\n",
  "ngspice_version_banner": "stdout ******\nstdout ** ngspice-40 : Circuit level
  ↪  simulation program\nstdout ** The U. C. Berkeley CAD Group\nstdout ** Copyright
  ↪  1985-1994, Regents of the University of California.\nstdout ** Copyright
  ↪  2001-2023, The ngspice team.\nstdout ** Please get your ngspice manual from
  ↪  https://ngspice.sourceforge.io/docs.html\nstdout ** Please file your
  ↪  bug-reports at http://ngspice.sourceforge.net/bugrep.html\nstdout ** Creation
  ↪  Date: Fri Apr 14 19:10:16 UTC 2023\nstdout **\nstdout ** CIDER 1.b1 (CODECS
  ↪  simulator) included\nstdout ** XSPICE extensions included\nstdout ** Relevant
  ↪  compilation options (refer to user's manual):\nstdout ** OpenMP multithreading
  ↪  for BSIM3, BSIM4 enabled\nstdout ** X11 interface not compiled into
  ↪  ngspice\nstdout **\nstdout ** ngspice shared library.\nstdout ******\n",
```

```
  "nome": "GCE-Temperatura",
  "num_avs": 3000000,
  "referencia": [
    {
      "legenda": "referência v(3)",
      "plot_hint": {
        "idx_x": [1],
        "idx_y": [0],
        "plot_type": "xy"
      },
      "tipo_plot": "DC transfer characteristic",
      "values": [[0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5,5.5,6,6.5,7,7.5,8,8.5,9,9.5,10],[0,
      ↪  5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]]
    }
  ]
},
"vs_rodadas": [
  {
    "datahora_final_rodada": "2023-12-01T00:55:36-0300",
    "eval": 3000000,
    "geracao": 3902361,
    "melhor_ind": {
      "aptidao": 0.2888948321342468,
      "cromossomo": (Suprimido devido à extensão),
      "erro_total": 0.2888948321342468,
      "hits_ratio": 1,
      "mae": 0.013756897300481796,
      "n_comp": 24,
      "netlist_ce": "Q1 102 101 100 2N3904\nR1 104 103 4915.251407\nQ2 0 106 105
      ↪  2N3906\nR2 106 100 5097.969528\nR3 102 107 5996.428012\nR4 0 108
      ↪  6575.036582\nR5 109 107 1770.142355\nR6 110 2 1854.458094\nR7 101 2
      ↪  248.236300\nR8 0 100 79.048918\nQ3 101 106 3 2N3906\nR9 105 2 48.435204\nQ4
      ↪  1 105 101 2N3906\nR10 101 0 52.562350\nR11 107 100 3901.380737\nR12 107 108
      ↪  307.922412\nR13 103 106 96864.474877\nQ5 106 106 108 2N3904\nR14 0 105
      ↪  3434.195471\nR15 108 105 7088.866762\nR16 2 110 51407.756587\nQ6 110 104
      ↪  109 2N3904\nR17 100 108 27423.562033\nR18 1 104 629.487089\n",
      "rmse": 0.019397325813770294,
      "spice_response": [
        {
          "name": "DC transfer characteristic",
          "title": "* dispositivo de teste",
          "type": "dc1",
          "values": [
```

```
        [0.009617027249966261,0.5326694666905191,0.9801568797241078,1.43735993943
        ↪  62289,1.9803379582736294,2.5233929926024588,3.000748113408695,3.49140
        ↪  907140021,3.9925526567844307,4.501754339663535,5.0115773014915685,5.5
        ↪  170187889101445,6.016184817086908,6.5096252861639,6.999859175600335,7
        ↪  .490517785296888,7.985142999095608,8.485808816674492,8.99186809224615
        ↪  6,9.499232864689521,10.000552925073022],
        [0,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]
      ],
      "variables": ["V(3)","temp-sweep"]
    }
  ],
  "tempo_simulacao_spice_us": 1107,
  "tipo_ce": "VIAVEL",
  "to_string_resumo": "aptidao: 0.288895; tipo_ce: VIAVEL; n_comp: 24 [(R, 18),
  ↪  (Q, 6)]; erro_total: 0.288895; hits_ratio: 1; mae: 0.0137569; rmse:
  ↪  0.0193973"
},
"tempo_gasto_segundos": 8566.200563999999,
"v_evals": [2,100000,200000,300000,400000,500000,600000,700000,800000,900000,1000
↪  000,1100000,1200000,1300000,1400000,1500000,1600000,1700000,1800000,1900000,2
↪  000000,2100000,2200000,2300000,2400000,2500000,2600000,2700000,2800000,290000
↪  0,3000000],
"v_geracoes": [0,123242,242056,360014,477110,609790,734084,862823,984250,1146550,
↪  1301865,1422998,1545305,1665128,1786006,1904050,2023674,2149471,2277245,24129
↪  98,2555294,2726989,2863269,2984597,3134507,3298610,3415366,3529387,3653155,37
↪  78302,3902361],
"v_media_fit": [12502005,5.318207740783691,0.5555732250213623,5.0599541664123535,
↪  4.94331693649292,5.595841884613037,5.575471878051758,10.893487930297852,10.87
↪  924861907959,3.9974350929260254,12.970705032348633,0.5665915012359619,21.5466
↪  0987854004,0.5323211550712585,65.09066772460938,1.875274658203125,0.762701451
↪  7784119,1.410942554473877,0.5811207890510559,0.30328768491744995,50.398311614
↪  990234,9.35307502746582,3.188880681991577,3.0845863819122314,7.56668376922607
↪  4,268.7163391113281,2.100984811782837,2.0315868854522705,0.34757113456726074,
↪  0.30282726883888245,3.8848230838775635],
"v_melhor_fit": [4010.236083984375,5.317505359649658,0.5555732250213623,0.5555710
↪  196495056,0.5555710196495056,0.5555710196495056,0.5555710196495056,0.55557101
↪  96495056,0.5555710196495056,0.5555710196495056,0.5555710196495056,0.555571019
↪  6495056,0.5219978094100952,0.5219978094100952,0.5219978094100952,0.5219978094
↪  100952,0.5219978094100952,0.5219978094100952,0.5219978094100952,0.30299529433
↪  25043,0.2888948321342468,0.2888948321342468,0.2888948321342468,0.288894832134
↪  2468,0.2888948321342468,0.2888948321342468,0.2888948321342468,0.2888948321342
↪  468,0.2888948321342468,0.2888948321342468,0.2888948321342468],
"v_melhor_props": {
```

```
      "erro_total": [4010.236083984375,5.317505359649658,0.5555732250213623,0.5555710
      ↪  196495056,0.5555710196495056,0.5555710196495056,0.5555710196495056,0.555571
      ↪  0196495056,0.5555710196495056,0.5555710196495056,0.5555710196495056,0.55557
      ↪  10196495056,0.5219978094100952,0.5219978094100952,0.5219978094100952,0.5219
      ↪  978094100952,0.5219978094100952,0.5219978094100952,0.5219978094100952,0.302
      ↪  9952943325043,0.2888948321342468,0.2888948321342468,0.2888948321342468,0.28
      ↪  88948321342468,0.2888948321342468,0.2888948321342468,0.2888948321342468,0.2
      ↪  888948321342468,0.2888948321342468,0.2888948321342468,0.2888948321342468],
      "hits_ratio": [0,0.9047619104385376,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
      ↪  ,1,1,1,1,1,1,1],
      "mae":[19.096363067626953,0.06549278646707535,0.026455868035554886,0.0264557637
      ↪  2742653,0.02645576372742653,0.02645576372742653,0.02645576372742653,0.02645
      ↪  576372742653,0.02645576372742653,0.02645576372742653,0.02645576372742653,0.
      ↪  02645576372742653,0.024857038632035255,0.024857038632035255,0.0248570386320
      ↪  35255,0.024857038632035255,0.024857038632035255,0.024857038632035255,0.0248
      ↪  57038632035255,0.01442834734916687,0.013756897300481796,0.01375689730048179
      ↪  6,0.013756897300481796,0.013756897300481796,0.013756897300481796,0.01375689
      ↪  7300481796,0.013756897300481796,0.013756897300481796,0.013756897300481796,0
      ↪  .013756897300481796,0.013756897300481796],
      "n_comp": [17,32,33,33,33,33,33,33,33,33,33,33,31,31,31,31,31,31,31,24,24,24,24
      ↪  ,24,24,24,24,24,24,24,24],
      "rmse": [19.33884620666504,0.09215685725212097,0.0322280153632164,0.03222839161
      ↪  7536545,0.032228391617536545,0.032228391617536545,0.032228391617536545,0.03
      ↪  2228391617536545,0.032228391617536545,0.032228391617536545,0.03222839161753
      ↪  6545,0.032228391617536545,0.03626997768878937,0.03626997768878937,0.0362699
      ↪  7768878937,0.03626997768878937,0.03626997768878937,0.03626997768878937,0.03
      ↪  626997768878937,0.019697073847055435,0.019397325813770294,0.019397325813770
      ↪  294,0.019397325813770294,0.019397325813770294,0.019397325813770294,0.019397
      ↪  325813770294,0.019397325813770294,0.019397325813770294,0.019397325813770294
      ↪  ,0.019397325813770294,0.019397325813770294]
    }
  }
]
}
```

APÊNDICE C

Modelos SPICE

Este apêndice apresenta a listagem dos dois modelos SPICE de transistores bipolares utilizados neste trabalho, o 2N3904 (NPN) e o 2N3906 (PNP).

2N3904.sp3

```
**************************************
*      Model Generated by MODPEX     *
*Copyright(c) Symmetry Design Systems*
*         All Rights Reserved        *
*    UNPUBLISHED LICENSED SOFTWARE   *
*   Contains Proprietary Information *
*      Which is The Property of      *
*     SYMMETRY OR ITS LICENSORS      *
*Commercial Use or Resale Restricted *
*   by Symmetry License Agreement    *
**************************************
* Model generated on Aug 07, 01
* MODEL FORMAT: SPICE3
.MODEL 2n3904 npn
+IS=1.26532e-10 BF=206.302 NF=1.5 VAF=1000
+IKF=0.0272221 ISE=2.30771e-09 NE=3.31052 BR=20.6302
+NR=2.89609 VAR=9.39809 IKR=0.272221 ISC=2.30771e-09
+NC=1.9876 RB=5.8376 IRB=50.3624 RBM=0.634251
+RE=0.0001 RC=2.65711 XTB=0.1 XTI=1
+EG=1.05 CJE=4.64214e-12 VJE=0.4 MJE=0.256227
+TF=4.19578e-10 XTF=0.906167 VTF=8.75418 ITF=0.0105823
+CJC=3.76961e-12 VJC=0.4 MJC=0.238109 XCJC=0.8
+FC=0.512134 CJS=0 VJS=0.75 MJS=0.5
+TR=6.82023e-08 PTF=0 KF=0 AF=1
```

2N3906.sp3

```
**************************************
*      Model Generated by MODPEX     *
*Copyright(c) Symmetry Design Systems*
*         All Rights Reserved        *
*    UNPUBLISHED LICENSED SOFTWARE   *
*   Contains Proprietary Information *
*      Which is The Property of      *
*     SYMMETRY OR ITS LICENSORS      *
*Commercial Use or Resale Restricted *
*   by Symmetry License Agreement    *
**************************************
* Model generated on Aug 07, 01
* MODEL FORMAT: SPICE3
.MODEL 2n3906 pnp
+IS=7.75521e-12 BF=194.093 NF=1.35509 VAF=156.436
+IKF=0.0660057 ISE=1.88546e-12 NE=1.81673 BR=3.41317
+NR=1.5 VAR=5.86061 IKR=1.70599 ISC=7.64281e-10
+NC=1.92376 RB=6.48961 IRB=0.1 RBM=0.1
+RE=0.0001 RC=2.45044 XTB=0.1 XTI=1
+EG=1.05 CJE=6.11928e-12 VJE=0.4 MJE=0.248812
+TF=5.21843e-10 XTF=0.932702 VTF=9.1046 ITF=0.0106472
+CJC=6.85007e-12 VJC=0.4 MJC=0.279018 XCJC=0.9
+FC=0.478887 CJS=0 VJS=0.75 MJS=0.5
+TR=4.30605e-07 PTF=0 KF=0 AF=1
```

APÊNDICE D

Melhores Circuitos

Este apêndice apresenta a netlist dos melhores circuitos encontrados em cada caso de uso. Este conteúdo, em conjunto com os modelos SPICE listados no apêndice C, permite a reprodução e comprovação do desempenho dos circuitos sintetizados pelo SANN-GCE.

Sensor de temperatura

```
* Circuito SAnn-GCE-Temperatura (gerado automaticamente). Aptidão: 0.0587846
* Dispositivo de teste (DT):
V1 1 0 dc 15
V2 0 2 dc 15
Rs 3 0 1000

* Circuito evolutivo (CE):
R1 101 100 538.572270
Q1 3 100 102 2N3906
R2 104 103 9613.219561
R3 106 105 69235.814548
R4 2 3 723.971369
Q2 3 100 2 2N3906
R5 3 104 34096.309675
R6 105 104 9915846.296330
R7 107 101 9910.191776
R8 102 108 208455.208557
Q3 3 100 107 2N3906
R9 107 109 4853.910134
R10 103 101 2750.896971
Q4 3 100 108 2N3906
R11 109 100 4540.303641
Q5 102 1 104 2N3904
Q6 104 110 0 2N3904
```

```
R12 103 0 388.160888
Q7 102 104 1 2N3904
R13 102 101 6563.292101
R14 101 106 190.340032
Q8 3 100 107 2N3906
R15 100 108 5292.680564
Q9 3 100 107 2N3906
Q10 0 100 100 2N3906
Q11 105 3 0 2N3906
R16 108 101 539.890538
R17 1 106 5760.267337
R18 102 107 203.791383
R19 101 2 544.209876
Q12 100 0 0 2N3904
Q13 3 100 2 2N3906
R20 109 106 839.690367
R21 100 0 85563.042198
R22 100 1 78.304910
Q14 100 3 100 2N3904
* resistores para os nós pendentes
Rpend1 0 110 1e9

* Comandos:
.include ~/ngspice/2N3904.sp3
.include ~/ngspice/2N3906.sp3
.dc temp 0 100 5
.save v(3)
.end
```

Função gaussiana

```
* Circuito SAnn-GCE-Gauss (gerado automaticamente). Aptidão: 0.0223713
* Dispositivo de teste (DT):
Vcc 1 0 dc 5
Vin 50 0 dc 0
Rs 50 2 1
Visense 3 0 2.5

* Circuito evolutivo (CE):
M1 101 101 100 1 PMOS1 L=10u W=154u
M2 2 103 102 0 NMOS1 L=10u W=80u
R1 1 104 984.232345
M3 1 104 102 1 PMOS1 L=10u W=107u
M4 102 100 101 1 PMOS1 L=10u W=146u
M5 104 105 103 1 PMOS1 L=10u W=194u
M6 106 103 3 1 PMOS1 L=10u W=74u
M7 105 1 102 0 NMOS1 L=10u W=93u
R2 104 100 64.170843
M8 101 1 104 0 NMOS1 L=10u W=67u
M9 107 0 107 0 NMOS1 L=10u W=48u
M10 0 104 2 0 NMOS1 L=10u W=197u
M11 1 102 101 0 NMOS1 L=10u W=122u
R3 109 108 50119.159869
R4 101 2 5480.404753
R5 2 109 9505953.006908
M12 103 2 1 1 PMOS1 L=10u W=195u
M13 109 102 0 1 PMOS1 L=10u W=84u
M14 1 107 3 1 PMOS1 L=10u W=89u
M15 101 100 2 1 PMOS1 L=10u W=199u
M16 102 104 2 0 NMOS1 L=10u W=118u
M17 1 106 100 0 NMOS1 L=10u W=199u
M18 1 105 101 0 NMOS1 L=10u W=183u
M19 101 100 108 0 NMOS1 L=10u W=107u
M20 101 101 102 1 PMOS1 L=10u W=87u
M21 101 102 102 0 NMOS1 L=10u W=160u
M22 109 1 104 0 NMOS1 L=10u W=13u
M23 1 102 101 0 NMOS1 L=10u W=197u
M24 100 102 104 1 PMOS1 L=10u W=186u
M25 1 106 107 1 PMOS1 L=10u W=46u
M26 101 102 102 0 NMOS1 L=10u W=196u
M27 108 100 1 0 NMOS1 L=10u W=14u
```

```
M28 109 100 102 0 NMOS1 L=10u W=75u
M29 103 106 2 0 NMOS1 L=10u W=115u
M30 101 3 2 1 PMOS1 L=10u W=162u
M31 100 0 0 1 PMOS1 L=10u W=115u
R6 106 109 588624.312786
M32 101 106 107 0 NMOS1 L=10u W=197u
M33 100 107 1 1 PMOS1 L=10u W=199u
R7 105 103 209.429824
M34 0 101 3 1 PMOS1 L=10u W=24u
M35 3 1 3 1 PMOS1 L=10u W=104u

* Comandos:
.model NMOS1 NMOS
.model PMOS1 PMOS
.dc Vin 2 3 0.01
.save i(Visense)
.end
```

Referência de tensão

```
* Circuito SAnn-GCE-Vref (gerado automaticamente). Aptidão: 0.0481201
* Dispositivo de teste (DT):
Vin 50 0 dc 0
Rs 50 1 1000
Rl 2 0 10000

* Circuito evolutivo (CE):
R1 101 100 4646.211972
Q1 104 103 102 2N3904
Q2 106 105 105 2N3906
Q3 107 0 0 2N3904
R2 103 106 3021657.556991
R3 104 103 6516496.981521
R4 0 108 76159.527646
R5 106 109 4046.378927
Q4 105 106 101 2N3906
Q5 109 1 102 2N3906
Q6 2 107 101 2N3906
Q7 110 108 104 2N3906
Q8 103 101 111 2N3904
R6 109 0 6196.342682
R7 109 105 68030.724699
R8 104 109 4699.559848
R9 104 108 6124.122746
Q9 102 101 106 2N3904
Q10 2 105 100 2N3906
R10 103 1 863018.387160
R11 111 1 23.756658
R12 101 109 315959.325750
Q11 1 102 111 2N3904
R13 0 108 8817.402942
Q12 0 100 111 2N3906
Q13 107 2 1 2N3904
Q14 100 102 1 2N3906
R14 105 1 7586.482714
Q15 109 2 105 2N3906
Q16 105 107 111 2N3906
R15 103 109 4862.088635
Q17 109 101 101 2N3906
R16 103 106 3188.694116
```

```
Q18 101 109 1 2N3904
R17 111 108 51825.320499
R18 103 105 9546.454632
R19 105 103 71506.727061
Q19 104 108 0 2N3906
Q20 0 103 111 2N3904
R20 105 0 7831.407676
Q21 102 104 101 2N3904
* resistores para os nós pendentes
Rpend1 0 110 1e9

* Comandos:
.include ~/ngspice/2N3904.sp3
.include ~/ngspice/2N3906.sp3
.dc TEMP 0 100 25 Vin 4 6 0.1
.save v(50) v(2)
.end
```

Função quadrática

```
* Circuito SAnn-GCE-Squaring2 (gerado automaticamente). Aptidão: 0.00303819
* Dispositivo de teste (DT):
Vcc1 1 0 dc 15
Vcc2 0 2 dc 15
Vs 50 0 PWL(0 -0.25 0.2 0.25)
Rs 50 3 1000
Rl 4 0 1000

* Circuito evolutivo (CE):
R1 101 100 8092.160623
Q1 104 103 102 2N3906
Q2 1 100 102 2N3904
R2 105 2 62802.491069
R3 103 4 1073.230507
R4 106 101 79.390647
Q3 105 4 104 2N3906
R5 107 3 9408.650568
R6 4 103 339.805370
R7 108 2 4144.408978
R8 107 1 42126.556825
Q4 102 3 103 2N3904
R9 2 109 415332.094219
Q5 107 105 2 2N3904
Q6 0 103 108 2N3904
R10 4 103 344.734308
Q7 103 100 106 2N3906
R11 104 105 3418915.998907
R12 109 106 72.710677
R13 100 3 393.023558
R14 0 100 455821.286445
R15 101 1 9240.732994
Q8 106 3 102 2N3906
Q9 109 3 103 2N3904
R16 104 4 388.257322
R17 0 109 46985.175637
Q10 104 105 2 2N3904
R18 2 101 334481.264030
Q11 109 0 109 2N3906
Q12 104 2 4 2N3904
R19 0 108 60.718583
```

```
R20 0 3 646056.022464
R21 101 106 80412.986094

* Comandos:
.include ~/ngspice/2N3904.sp3
.include ~/ngspice/2N3906.sp3
.options interp
.tran 0.01 0.2 0 0.01
.save v(50) v(4)
.end
```

Raiz quadrada

```
* Circuito SAnn-GCE-SqRoot2 (gerado automaticamente). Aptidão: 0.00691906
* Dispositivo de teste (DT):
Vcc1 1 0 dc 15
Vcc2 0 2 dc 15
Vs 50 0 PWL(0 0 0.2 0.5)
Rs 50 3 1000
Rl 4 0 1000

* Circuito evolutivo (CE):
R1 101 100 9007845.466328
R2 0 102 709624.741782
R3 2 103 86431.148393
Q1 102 105 104 2N3904
Q2 107 106 102 2N3906
R4 108 0 840.700240
R5 100 109 1318.401437
R6 4 107 50584.482049
Q3 106 0 107 2N3904
R7 1 102 644.950711
R8 108 101 9556.863483
Q4 104 100 102 2N3906
Q5 2 3 102 2N3906
Q6 108 106 0 2N3904
Q7 104 101 106 2N3904
R9 109 100 78706.054625
R10 0 104 52.765438
R11 0 102 5053.294756
Q8 1 103 108 2N3904
Q9 105 109 108 2N3904
Q10 103 109 102 2N3906
Q11 100 107 4 2N3904
Q12 1 100 108 2N3904
R12 0 100 8153.677088
R13 108 0 702832.516050
Q13 102 105 0 2N3904
Q14 108 101 100 2N3906
R14 101 106 595.481360
Q15 0 101 100 2N3906
Q16 102 108 0 2N3904
Q17 3 105 103 2N3904
```

```
Q18 101 106 3 2N3904
R15 0 108 4694.102188
R16 107 103 81.518136
R17 109 102 2.277890

* Comandos:
.include ~/ngspice/2N3904.sp3
.include ~/ngspice/2N3906.sp3
.options interp
.tran 0.01 0.2 0 0.01
.save v(50) v(4)
.end
```

Função cúbica

```
* Circuito SAnn-GCE-Cubing2 (gerado automaticamente). Aptidão: 0.000811866
* Dispositivo de teste (DT):
Vcc1 1 0 dc 15
Vcc2 0 2 dc 15
Vs 50 0 PWL(0 -0.25 0.2 0.25)
Rs 50 3 1000
Rl 4 0 1000

* Circuito evolutivo (CE):
Q1 2 101 100 2N3904
Q2 103 2 102 2N3904
Q3 104 100 102 2N3904
Q4 106 102 105 2N3906
Q5 109 108 107 2N3906
Q6 110 110 109 2N3906
Q7 108 106 107 2N3906
R1 4 108 95.625909
Q8 112 111 111 2N3904
R2 1 113 91630.744854
R3 102 109 5332.307857
R4 112 114 31726.914304
R5 111 115 4846.916690
R6 112 3 37.117273
Q9 102 108 3 2N3904
Q10 102 104 115 2N3906
Q11 115 3 100 2N3906
R7 114 116 893.136305
Q12 108 117 101 2N3904
R8 113 107 3.377158
Q13 104 108 108 2N3904
R9 111 107 182.424679
R10 101 116 38.390352
Q14 115 0 116 2N3906
Q15 109 103 111 2N3906
Q16 115 108 112 2N3906
Q17 115 4 102 2N3904
Q18 102 108 4 2N3906
Q19 108 3 107 2N3904
Q20 117 105 110 2N3906
Q21 117 102 100 2N3906
```

```
Q22 0 118 105 2N3906
Q23 108 110 1 2N3904
* resistores para os nós pendentes
Rpend1 0 118 1e9

* Comandos:
.include ~/ngspice/2N3904.sp3
.include ~/ngspice/2N3906.sp3
.options interp
.tran 0.01 0.2 0 0.01
.save v(50) v(4)
.end
```

Raiz cúbica

```
* Circuito SAnn-GCE-CbRoot2 (gerado automaticamente). Aptidão: 0.0242188
* Dispositivo de teste (DT):
Vcc1 1 0 dc 15
Vcc2 0 2 dc 15
Vs 50 0 PWL(0 -0.25 0.2 0.25)
Rs 50 3 1000
Rl 4 0 1000

* Circuito evolutivo (CE):
R1 101 100 176.391976
R2 103 102 66.896753
Q1 105 103 104 2N3904
R3 106 3 6.419166
Q2 109 108 107 2N3904
R4 106 110 464.788810
Q3 0 110 110 2N3906
Q4 0 109 111 2N3904
R5 100 4 66.779910
Q5 100 105 111 2N3904
Q6 113 112 110 2N3904
R6 115 114 8955.690557
R7 102 101 750.262968
R8 115 109 906507.163298
R9 114 105 436.240086
R10 116 112 2.108872
R11 118 117 96.047220
R12 120 119 366.998434
R13 120 107 7922.031648
R14 109 110 15.135736
Q7 2 117 105 2N3906
Q8 110 102 119 2N3906
R15 116 3 778464.498364
R16 114 121 5896.692688
R17 119 113 96.171813
R18 122 105 4.230913
R19 0 121 35.112853
R20 118 101 1835.213119
Q9 123 112 121 2N3906
R21 115 0 436.919455
Q10 114 108 119 2N3904
```

```
Q11 102 106 112 2N3904
Q12 122 107 104 2N3904
Q13 101 103 109 2N3906
R22 4 116 203.745063
R23 2 120 809184.096063
R24 114 104 951.558258
R25 124 118 556838.021322
R26 100 1 9533.720435
Q14 124 109 0 2N3904
R27 111 2 3491.258190
Q15 116 0 121 2N3906
R28 0 3 54189.578587
Q16 114 118 104 2N3904
R29 113 117 2242.915927
R30 108 121 4.293934
Q17 116 117 114 2N3906
R31 100 110 32797.881495
* resistores para os nós pendentes
Rpend1 0 123 1e9

* Comandos:
.include ~/ngspice/2N3904.sp3
.include ~/ngspice/2N3906.sp3
.options interp
.tran 0.01 0.2 0 0.01
.save v(50) v(4)
.end
```

APÊNDICE E

Tutoriais para criação de máquinas virtuais na nuvem

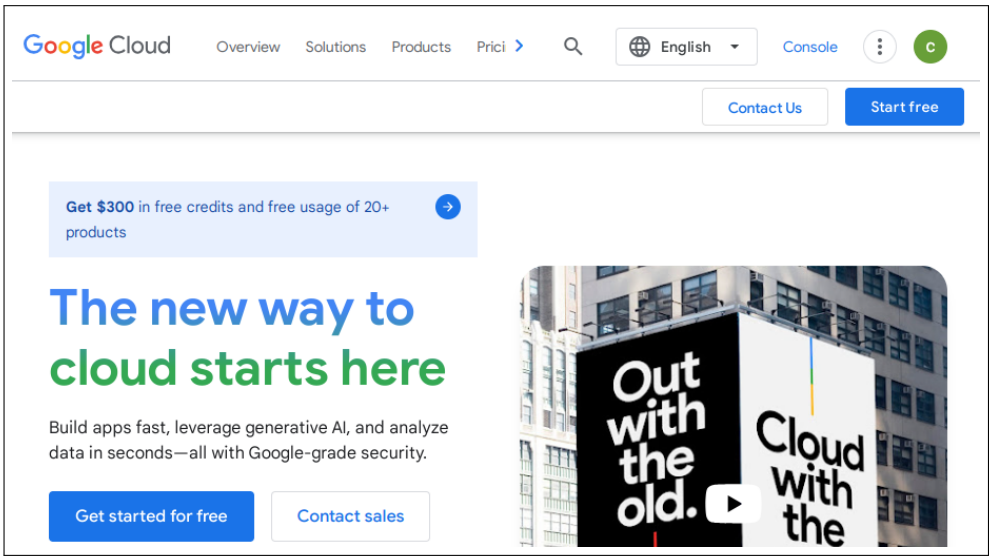
Este apêndice apresenta dois tutoriais para a criação de máquinas virtuais na nuvem, um para o *Google Cloud* e outro para o *Amazon Web Services - AWS*.

E.1 Google Cloud

Para criar uma máquina virtual no Google Cloud, siga os seguintes passos:

1. Acesse <https://cloud.google.com/>. Será exibida a tela inicial do Google Cloud, ilustrada na Figura 34;

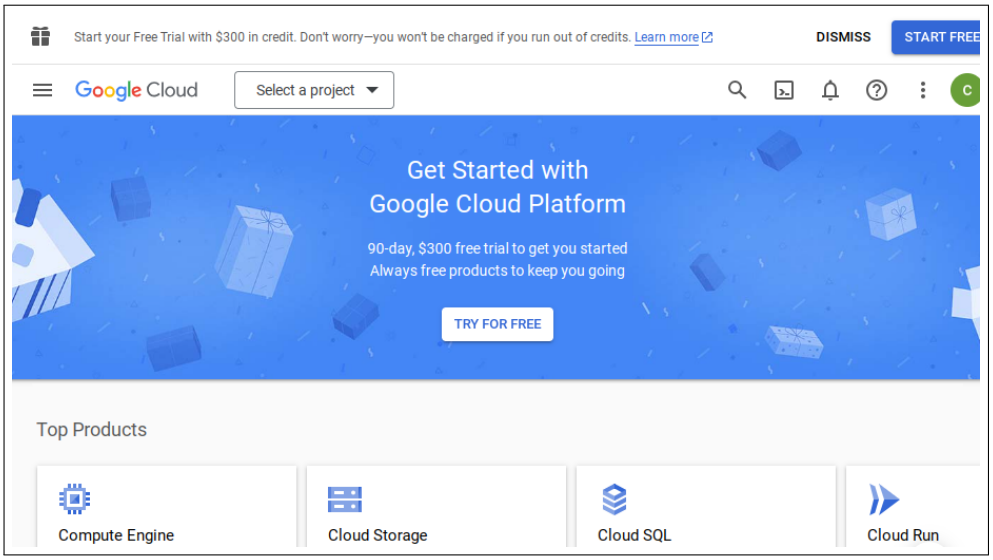
Figura 34 – Tela inicial do Google Cloud



Fonte: o autor (2023)

2. No canto superior direito, clique em *Console*. A página da Figura 35 será exibida;

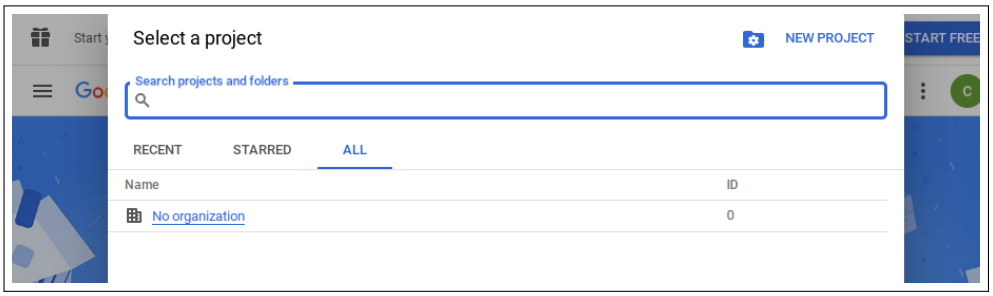
Figura 35 – Página inicial do console do Google Cloud



Fonte: o autor (2023)

3. Clicar na caixa de seleção *Select a project*. Será exibida uma tela semelhante à da Figura 36;

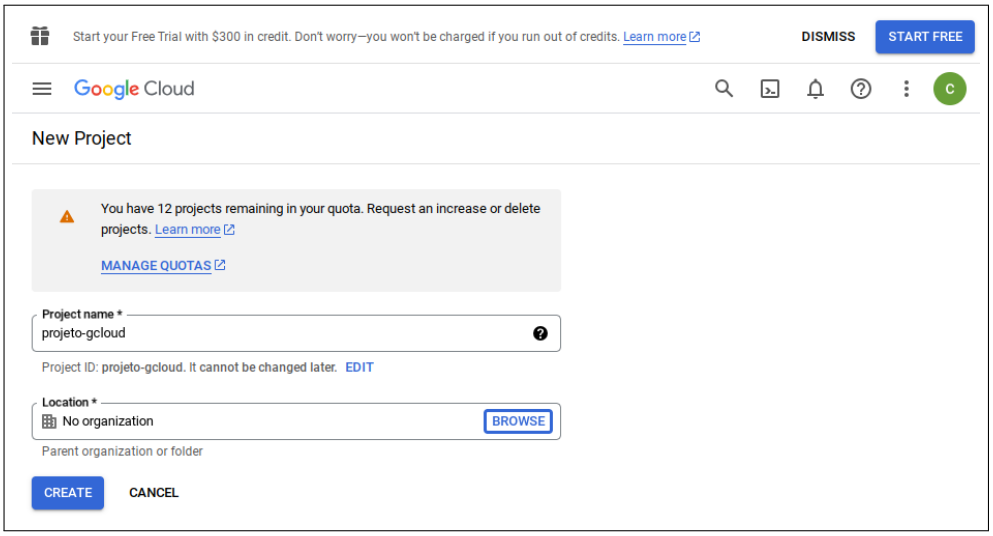
Figura 36 – Tela de seleção ou criação de projetos



Fonte: o autor (2023)

4. Clicar no botão *New Project*. Será exibida a tela de criação de projeto, conforme a Figura 37. No campo *Project name* digite um nome para o projeto e clique no botão *Create*;

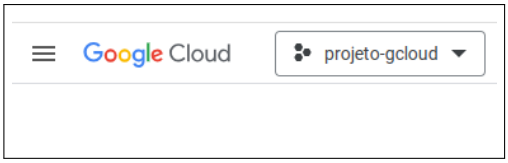
Figura 37 – Tela de criação de projetos



Fonte: o autor (2023)

5. No canto superior esquerdo da página, certifique-se que o projeto selecionado é o projeto recém criado (como na Figura 38). Se o projeto não aparecer, talvez seja necessário recarregar a página;

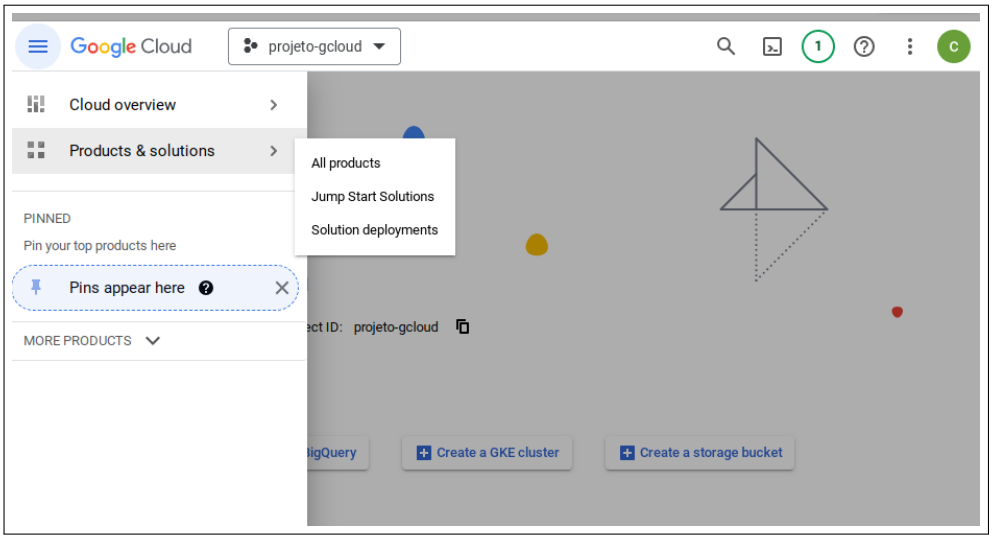
Figura 38 – Projeto do Google Cloud selecionado (recorte da tela)



Fonte: o autor (2023)

6. Acesse o menu do Google Cloud clicando no *botão hamburger* (ícone ≡, localizado no canto superior esquerdo da página). No menu selecione *Products & Solutions*, depois *All products* (veja a Figura 39);

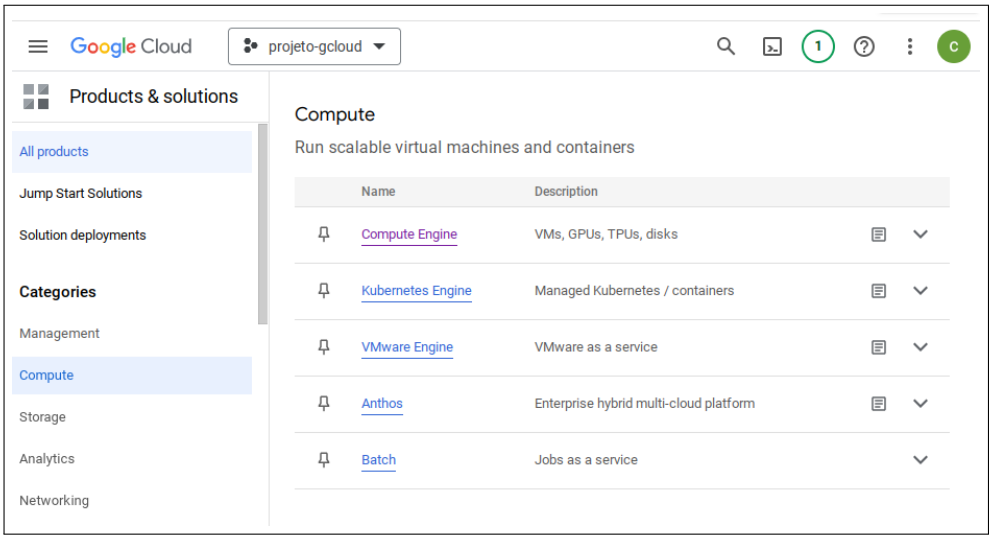
Figura 39 – Menu do Google Cloud



Fonte: o autor (2023)

7. Na lista de opções, clique em *Compute Engine*, conforme mostrado na Figura 40;

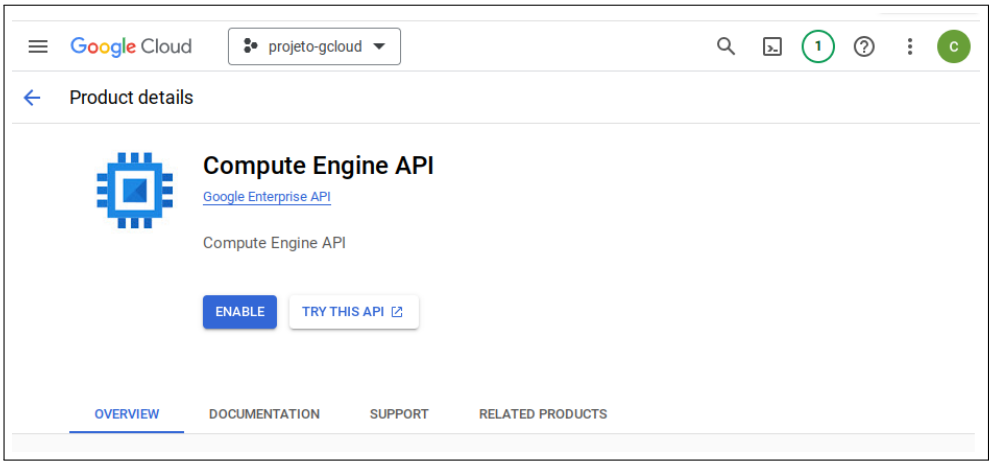
Figura 40 – Lista de produtos do Google Cloud, em destaque o *Compute Engine*



Fonte: o autor (2023)

8. Na próxima tela, ilustrada pela Figura 41, clique em *Enable*;

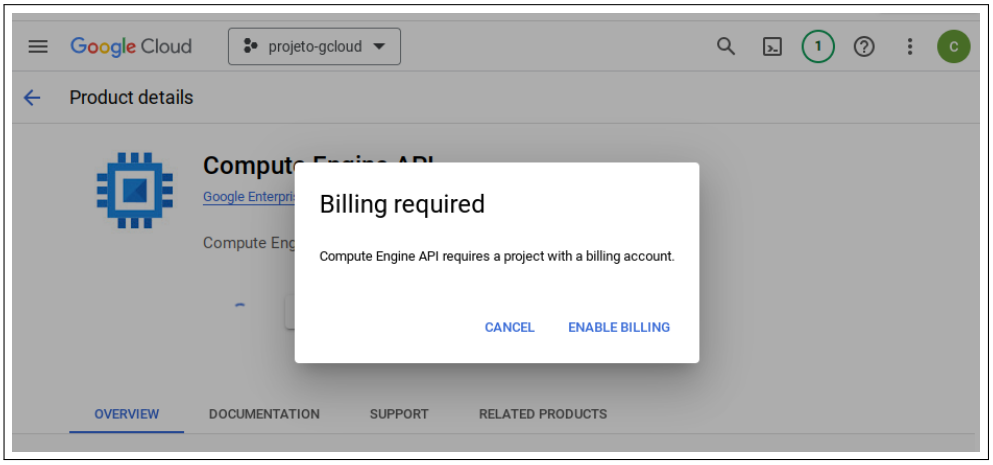
Figura 41 – Detalhe do produto *Compute Engine*



Fonte: o autor (2023)

9. Caso não haja uma conta de faturamento associada ao projeto do Google Cloud, aparecerá a mensagem *Compute Engine API requires a project with a billing account* (Figura 42). Selecione *Enable Billing*;

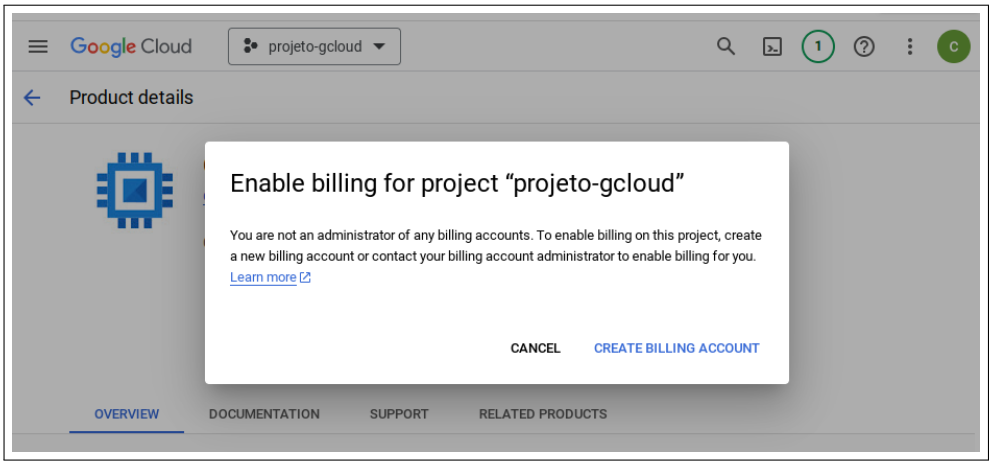
Figura 42 – Tela de aviso da obrigatoriedade de se habilitar uma conta de faturamento



Fonte: o autor (2023)

10. Em seguida aparecerá a mensagem *You are not an administrator of any billing accounts. To enable billing on this project, create a new billing account or contact your billing account administrator to enable billing for you* (Figura 43). Clique em *CREATE BILLING ACCOUNT*;

Figura 43 – Tela de aviso da obrigatoriedade de se criar uma conta de faturamento

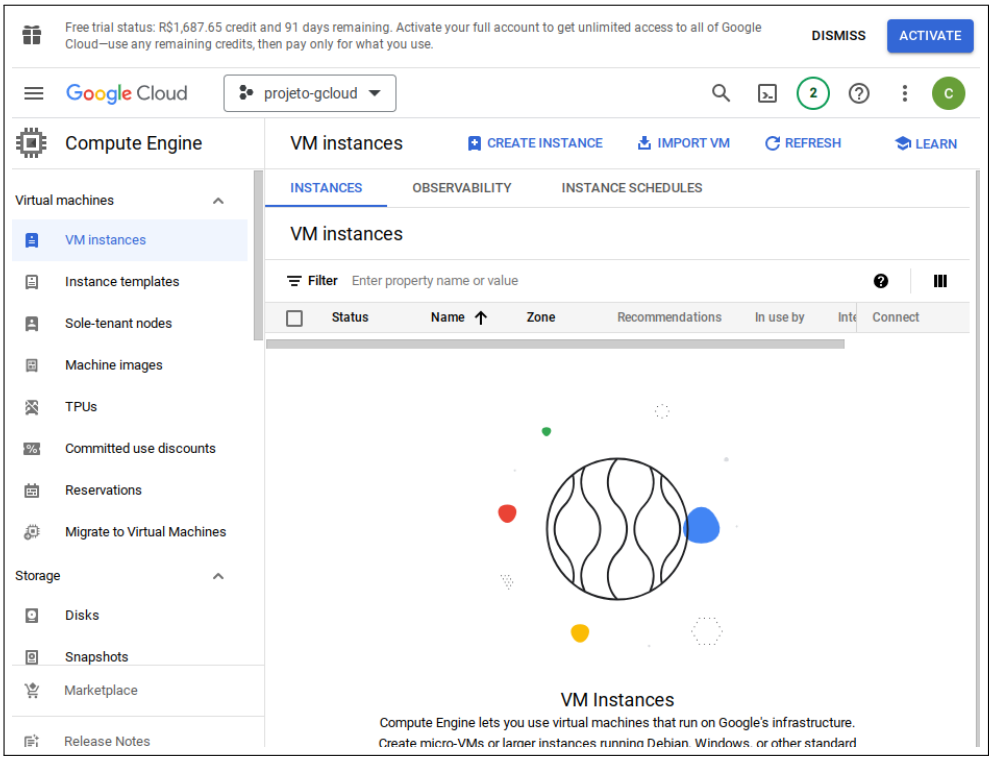


Fonte: o autor (2023)

11. Na tela seguinte, preencha os dados solicitados: CPF, endereço, dados de cobrança do cartão de crédito e siga com as instruções apresentadas. Se a criação da conta de faturamento ocorrer com sucesso, a tela *Compute Engine API* aparecerá novamente (ver Figura 41). Clique em *Enable* novamente e aguarde. Esta etapa pode levar um tempo considerável.

12. Após a habilitação do Compute Engine, o browser será redirecionado para o endereço <https://console.cloud.google.com/compute/instances>, cuja tela é reproduzida na Figura 44;

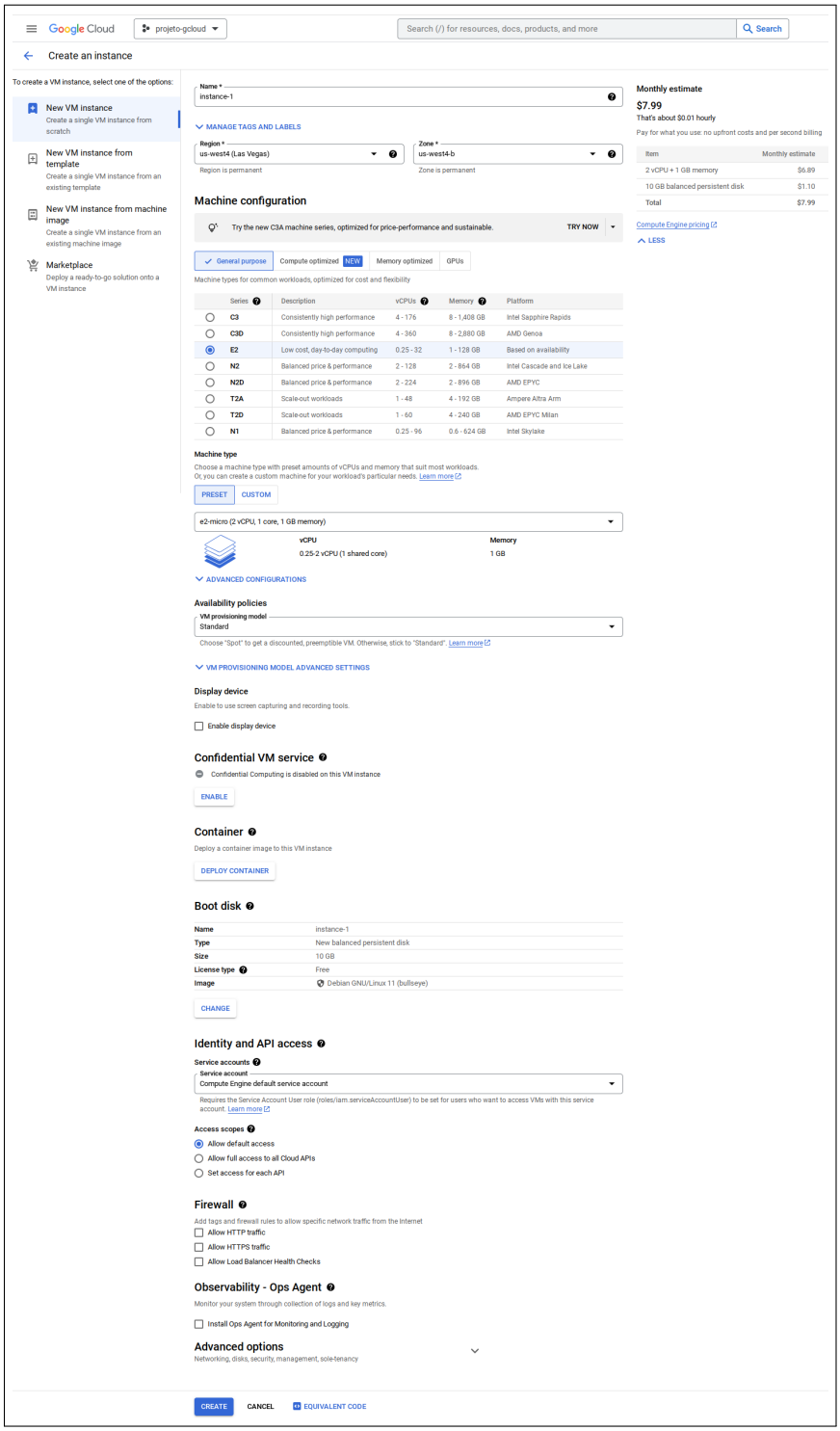
Figura 44 – Tela do Compute Engine



Fonte: o autor (2023)

13. Clique no botão *Create instance*. Aparecerá uma página com as configurações da máquina virtual a ser criada (Figura 45). Ajuste as configurações pertinentes e depois clique em *Create* para criar a instância;

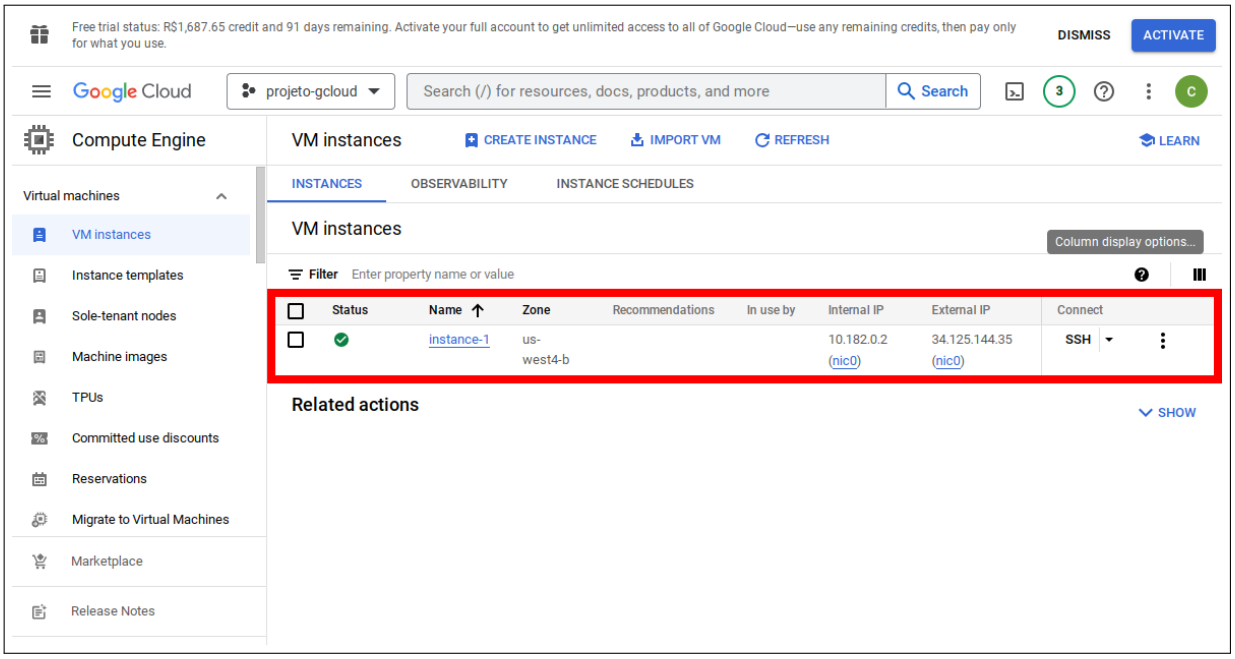
Figura 45 – Tela de criação de uma instância do Compute Engine



Fonte: o autor (2023)

14. Logo após a criação da instância, será exibida a página com a lista de instâncias, contendo a máquina virtual recém criada. Observe o ícone da coluna *status* em verde, que indica que a instância já foi iniciada, portanto, a tarifação já está em vigor. (Figura 46);

Figura 46 – Lista de instâncias

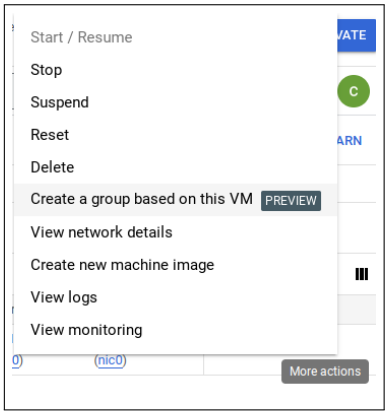


Fonte: o autor (2023)

Nota: Em destaque, a instância recém criada e seu status *em execução*, denotado pelo ícone verde

15. Ainda na tela da Figura 46, no final da linha da instância criada, ao se selecionar o ícone de três pontos verticais ⋮ (menu *more options*), haverá algumas ações importantes, como *Start/Resume* e *Stop* (veja Figura 47), utilizadas para iniciar e parar a instância, respectivamente.

Figura 47 – Menu *more options* da instância



Fonte: o autor (2023)

Ao fim destes passos, a máquina virtual estará criada e sendo executada. Para acessá-la via SSH, há três métodos possíveis:

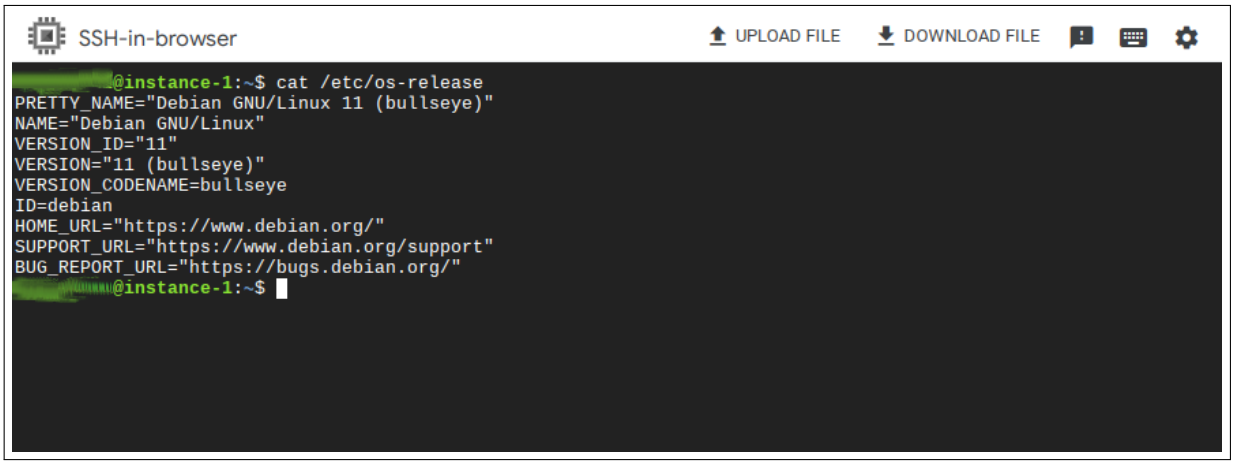
Método 1 : Acesso indireto através do *SSH-in-browser*;

Método 2 : Acesso direto através do comando `gcloud`, instalado no computador local. Requer a instalação do google-cloud-sdk;

Método 3 : Acesso direto através do comando `ssh` do Linux local.

O Método 1 é o mais simples. Na lista de instâncias (Figura 46), basta clicar no botão *SSH* na coluna *Connect*. Será aberta uma janela do browser contendo um emulador de terminal com acesso à maquina virtual remota, conforme ilustrado pela Figura 48.

Figura 48 – Janela do SSH-in-browser



Fonte: o autor (2023)

O Método 2 requer a instalação do google-cloud-sdk. Isto pode ser feito seguindo as instruções contidas em <https://cloud.google.com/sdk/docs/install>. Depois, a configuração deve ser feita conforme as explicações da página <https://cloud.google.com/compute/docs/connect/standard-ssh?cloudshell=false#gcloud>.

O Método 3 é o mais versátil, pois permite o uso das ferramentas locais de acesso `ssh` e `scp` comuns a todos os Linux. O acesso via comando ssh local não necessita de senha, mas requer a transferência da chave pública do ssh da máquina local para a máquina remota uma única vez. Isto pode ser feito através dos seguintes passos:

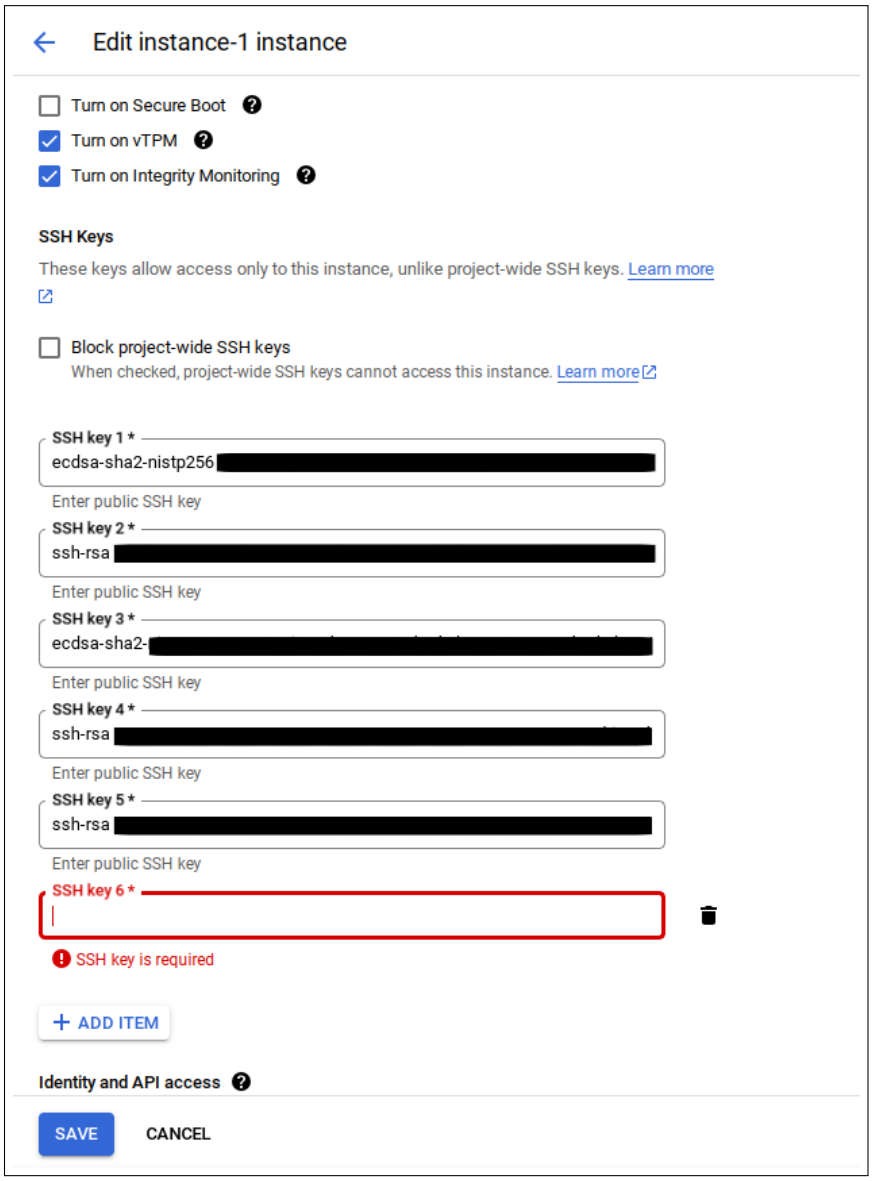
1. Obtenha a chave pública do cliente ssh de sua máquia local. Em sistemas Linux, a chave pública é normalmente armazenada em `$HOME/.ssh/id_rsa.pub`. Este arquivo pode ser aberto por um editor de textos comum e seu conteúdo copiado para a área de transferência (CTRL+C);

2. Caso a instância esteja em execução, pare-a;

3. Edite a instância: na lista de instâncias, clique no nome dela. Na próxima tela clique no ícone *Edit*. Na página seguinte que irá abrir, vá na seção *SSH Keys* (veja Figura 49). No final da lista de chaves SSH clique em *Add Item* e cole (CTRL+V) a

chave pública no campo apropriado. Depois clique em *Save* para finalizar a edição da instância.

Figura 49 – Tela de edição da instância, com destaque para a seção de chaves SSH



Fonte: o autor (2023)

4. Inicie a instância;

Após a transferência da chave pública para a máquina remota e sua inicialização, é possível acessá-la sem senha através do comando ssh:

```
ssh <usuario>@<end_ip>
```

No comando acima, `<usuario>` é normalmente o nome de usuário da máquina local de onde a chave pública foi obtida. O `<end_ip>` é o endereço IP da máquina remota.

Este endereço pode ser obtido na página que lista as instâncias, na coluna *External IP* (veja Figura 46).

Para copiar arquivos entre a máquina virtual e máquina local, o comando `scp` pode ser utilizado com uma das seguintes sintaxes, conforme a direção de cópia desejada:

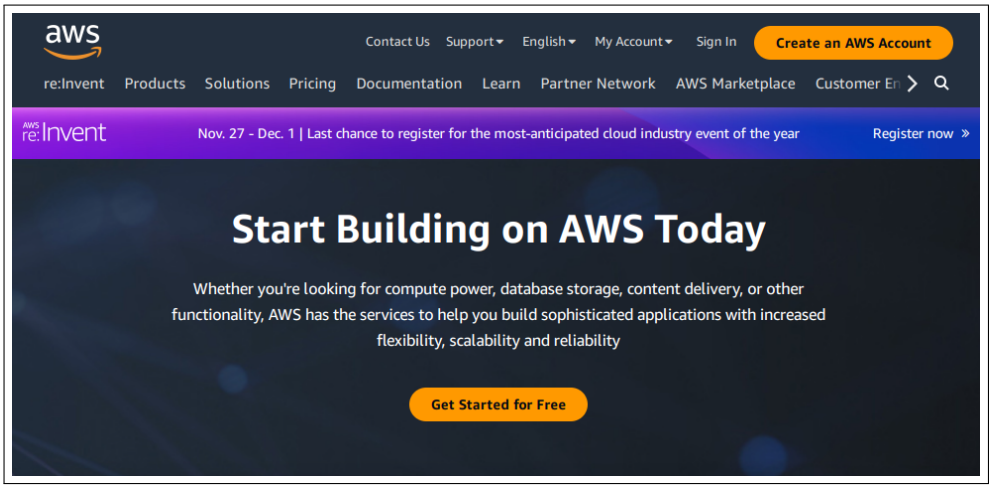
```
scp <arq_local_origem> <usuario>@<end_ip>:<dir_remoto_destino>

scp <usuario>@<end_ip>:<arq_remoto_origem> <dir_local_destino>
```

E.2 Amazon Web Services

Esta seção descreve em detalhes o processo de criação e acesso de uma máquina virtual na AWS. Isto pode ser feito através dos seguintes passos:

1. Acesse <https://aws.amazon.com/>. Será exibida a tela inicial da AWS, ilustrada na Figura 50;

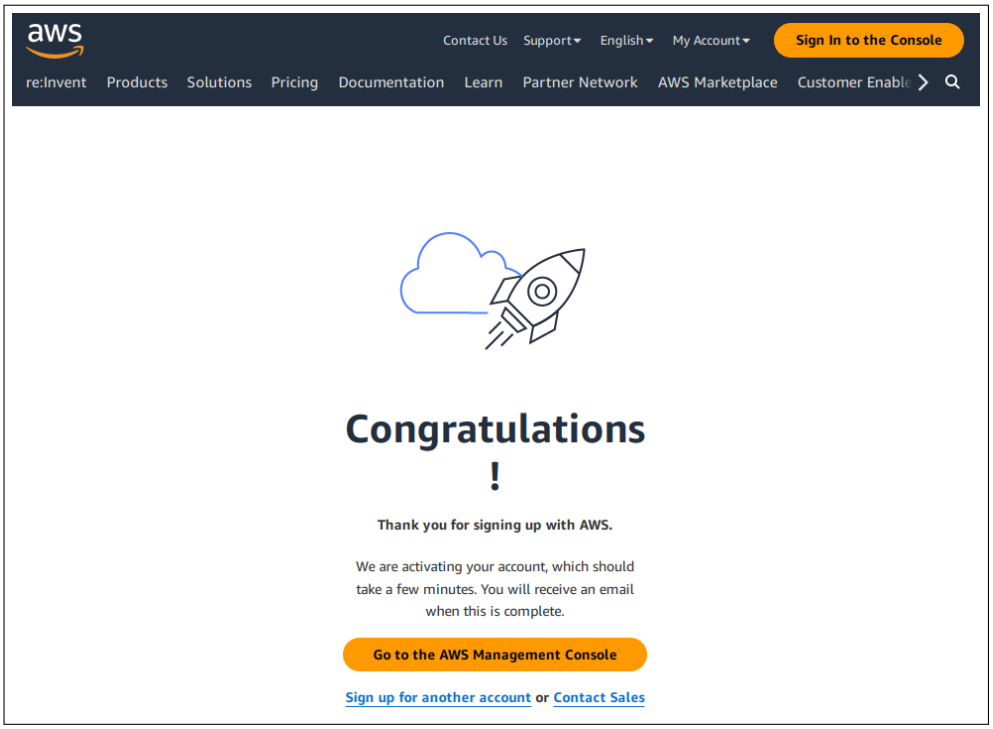
Figura 50 – Tela inicial da AWS



Fonte: o autor (2023)

2. Clique no botão *Create an AWS Account* e preencha as informações solicitadas para a criação de uma conta AWS. Isto é feito em cinco etapas e requer validação de email, nome da conta, senha raiz, informações pessoais, cartão de crédito. Ao final será perguntado o plano de suporte desejado. A opção *Basic Support - Free* é suficiente na maioria dos casos. Após a criação da conta com sucesso, será exibida a tela da Figura 51;

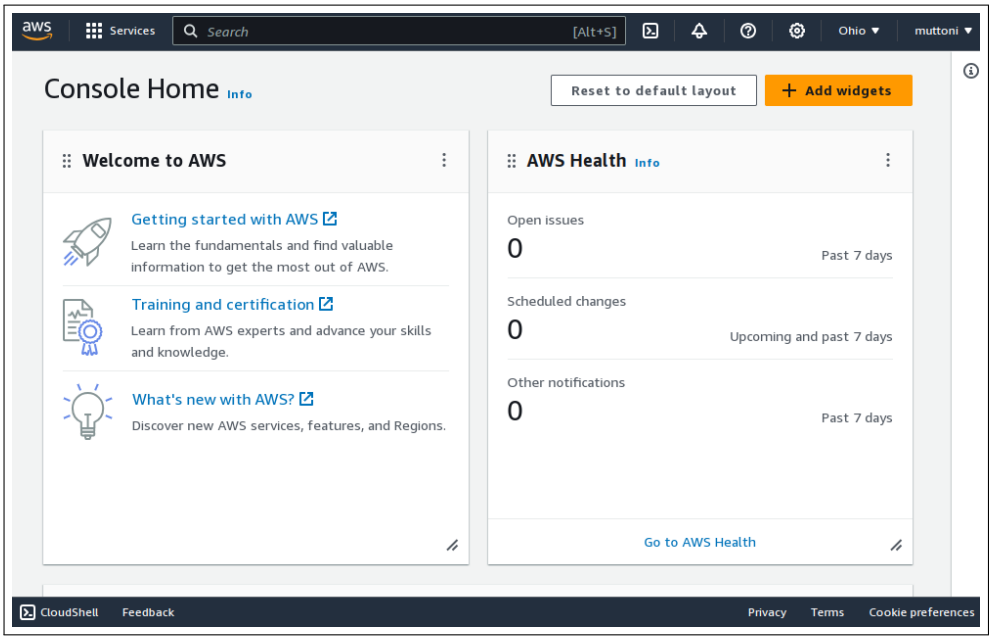
Figura 51 – Tela exibida logo após a criação de uma conta AWS



Fonte: o autor (2023)

3. Acesse o *console AWS*. Para isso, clique no botão *Go to the AWS Management Console* ou acesse diretamente o endereço <https://console.aws.amazon.com/>. Será exibida a tela *Console Home*, mostrada na Figura 52;

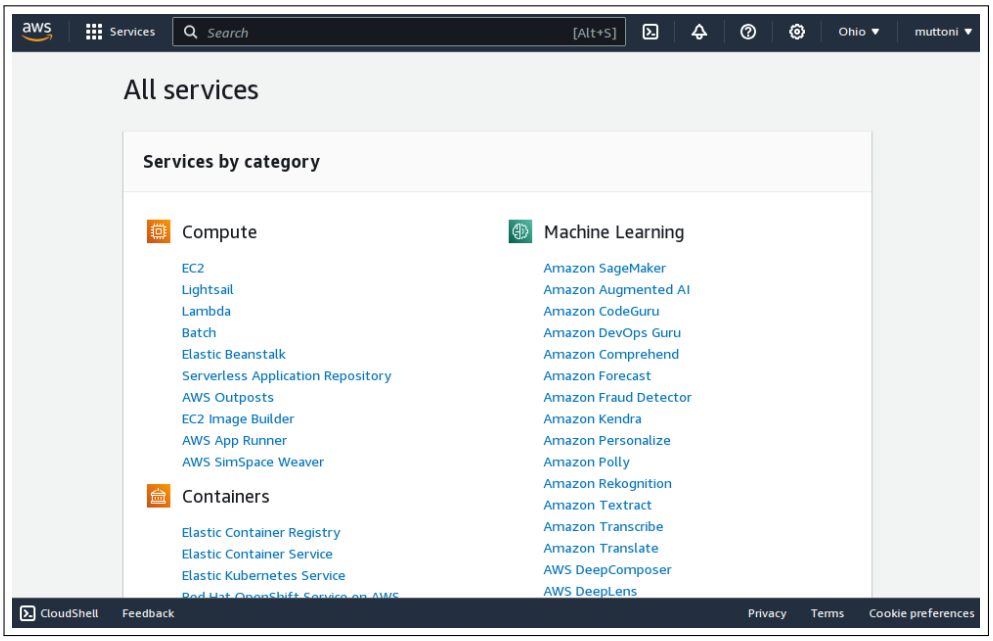
Figura 52 – Tela *Console Home*



Fonte: o autor (2023)

4. Clique no botão *Services* no canto superior esquerdo da página. Depois acesse o item de menu *All services*, seguido por *View all services*. Serão exibidos todos os serviços da AWS disponíveis, conforme a Figura 53;

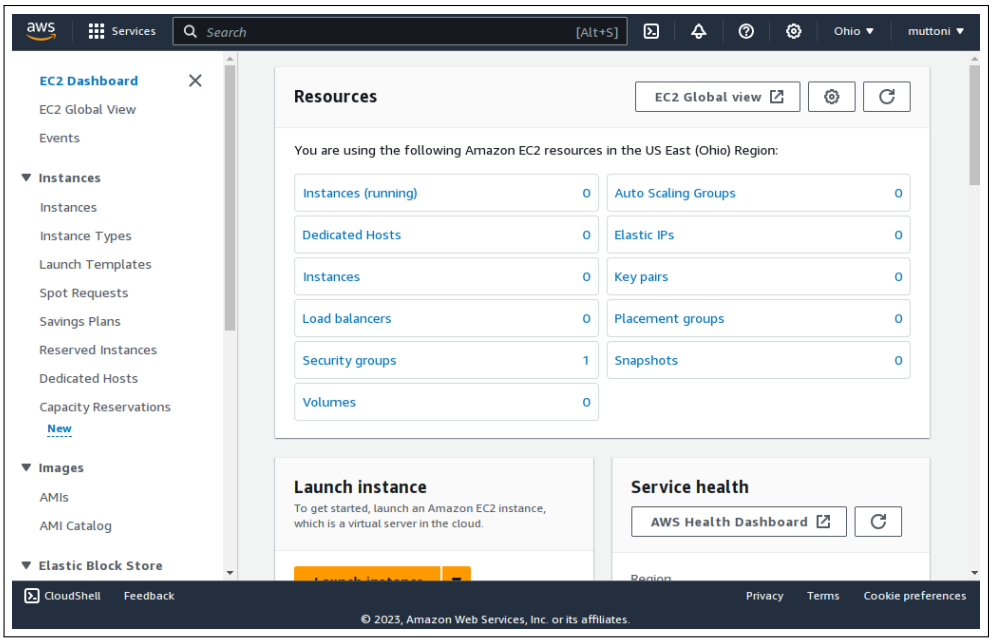
Figura 53 – Lista de todos os serviços da AWS disponíveis



Fonte: o autor (2023)

5. Clique no serviço EC2, sob a categoria *Compute*. A página *EC2 Dashboard* será exibida, de acordo com a Figura 54;

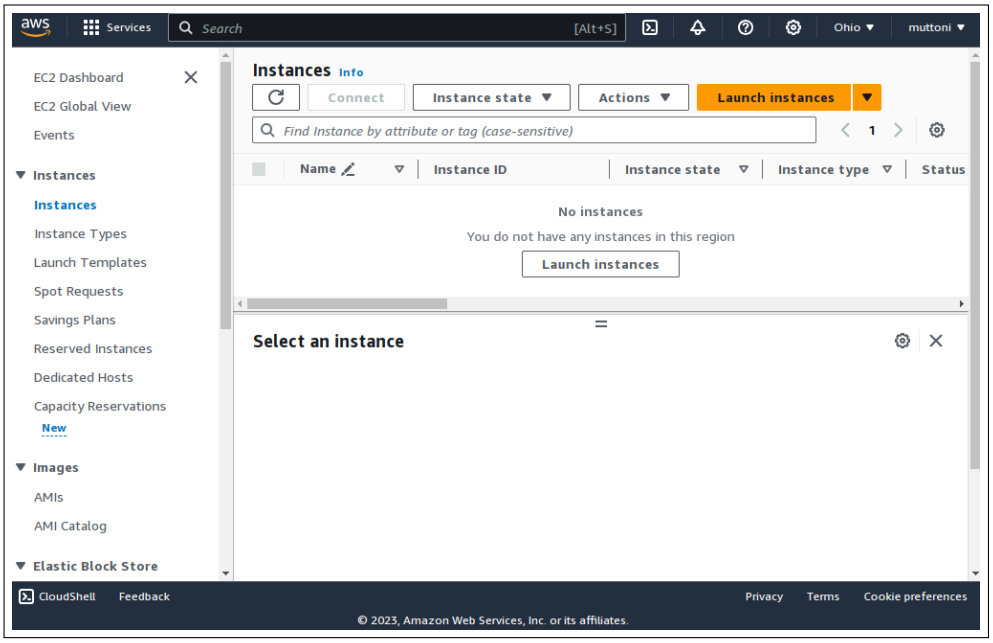
Figura 54 – Página *EC2 Dashboard*



Fonte: o autor (2023)

6. No menu exibido na lateral esquerda da página, acesse o item *Instances*. Será exibida a página da Figura 55;

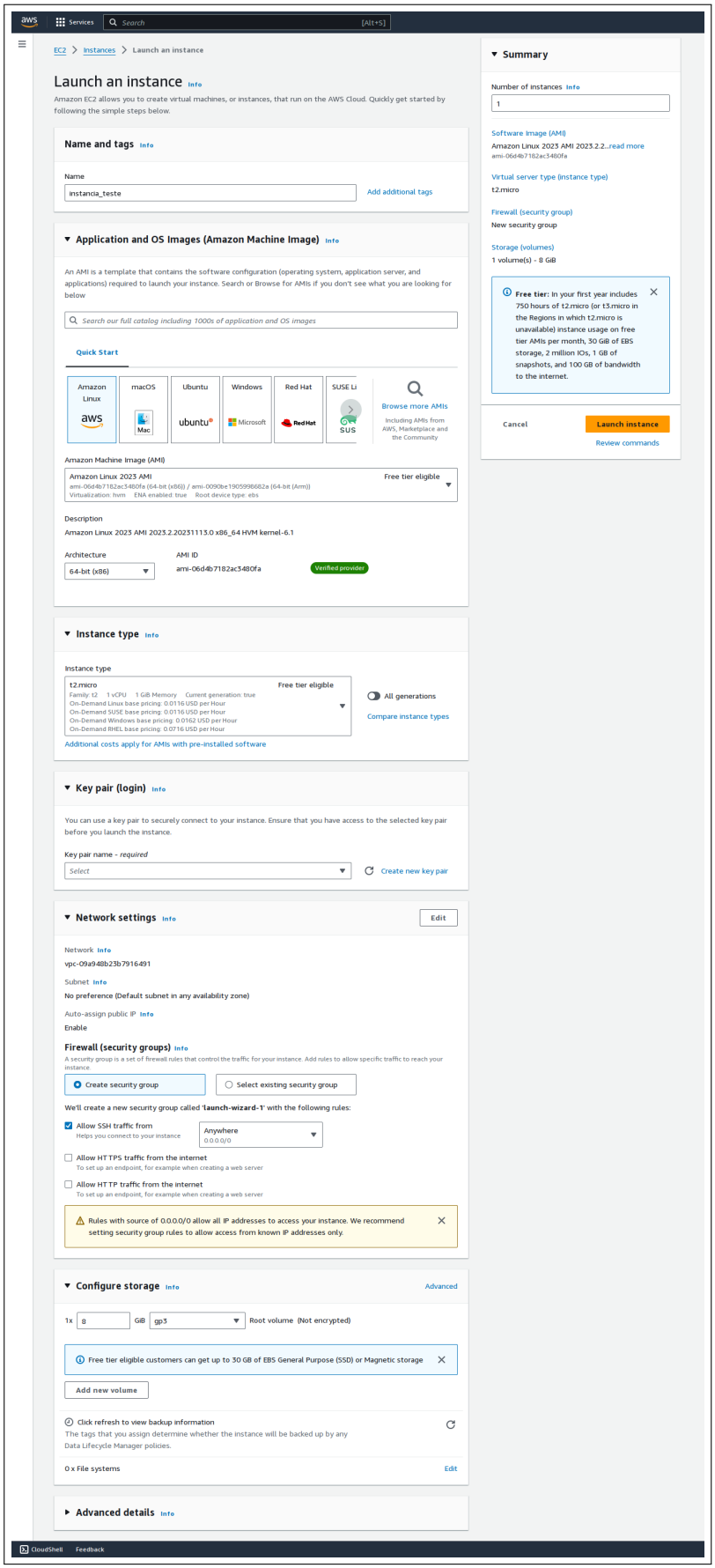
Figura 55 – Página *instances*



Fonte: o autor (2023)

7. Clique no botão *Launch instances*. Será exibida a página de lançamento (criação) da máquina virtual, conforme a Figura 56;

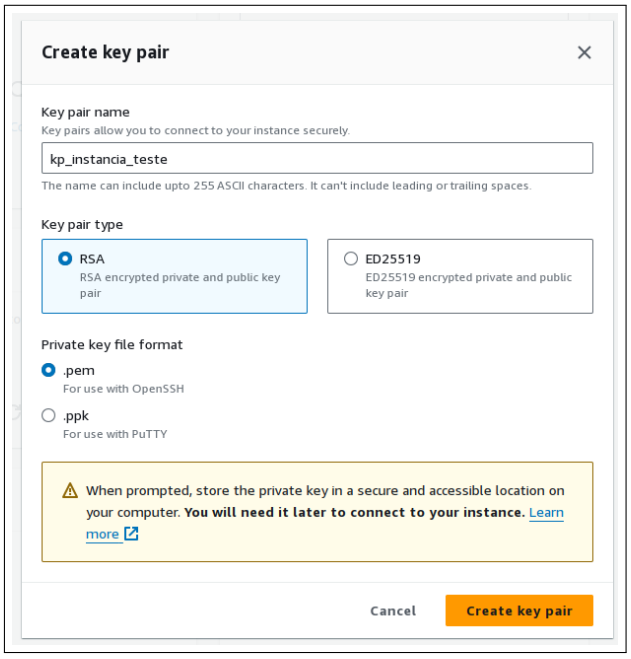
Figura 56 – Página com as configurações para a criação de uma máquina virtual na AWS



Fonte: o autor (2023)

8. Na seção *Key pair (login)*, clique em *create new key pair* para criar um par de chaves criptográficas para posterior acesso à máquina virtual. Uma janela será exibida, conforme a Figura 57. No campo *Key pair name* digite um nome para o par de chaves a ser criada. Se necessário, ajuste as demais opções. Clique em *Create key pair*. Neste momento será feito o download da chave recém criada (arquivo com extensão `.pem`). Salve este arquivo em um local seguro;

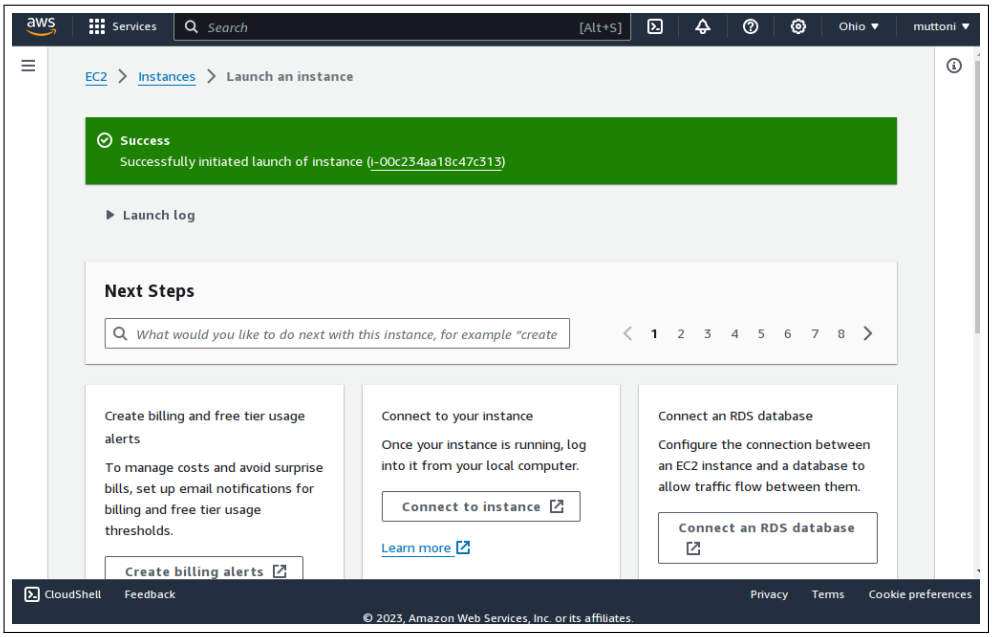
Figura 57 – Janela para criação de um par de chaves criptográficas



Fonte: o autor (2023)

9. Após a criação e download do par de chaves, assegure que a caixa de seleção *Key pair name* (ainda na tela da Figura 56) está apontando para o nome do par de chaves recém criado. Após ajustar as configurações desejadas para a máquina virtual, clique no botão laranja *Launch instance* (à direita da página). A máquina virtual será criada e uma tela semelhante à da Figura 58 será exibida. É importante notar que, logo após esta etapa de criação da instância, a máquina virtual será iniciada pela primeira vez automaticamente e a tarifação já está em vigor;

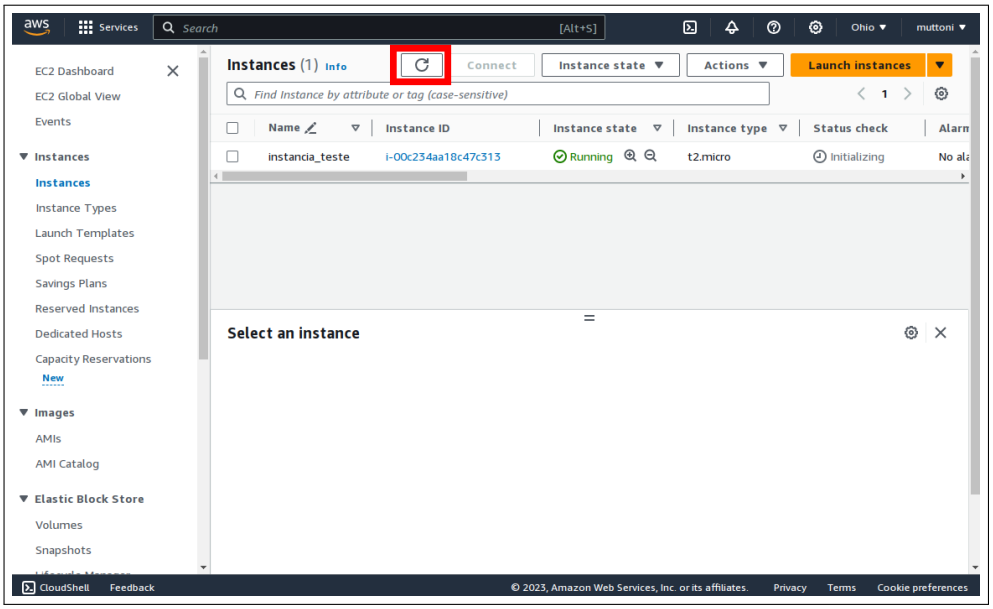
Figura 58 – Página exibida logo após a criação de uma instância EC2 na AWS



Fonte: o autor (2023)

10. Acesse o menu do EC2 clicando no *botão hamburger* (ícone ≡, localizado no canto superior esquerdo da página). No menu selecione *Instances*. Será exibida a lista de máquinas virtuais. Eventualmente será necessário clicar no botão *refresh instances* para que a máquina recém criada seja exibida na lista. Esta tela é reproduzida na Figura 59. Observe a coluna *Instance state* com o texto *Running*, que indica que a instância está em execução;

Figura 59 – Lista de instâncias EC2 logo após a criação de uma máquina virtual

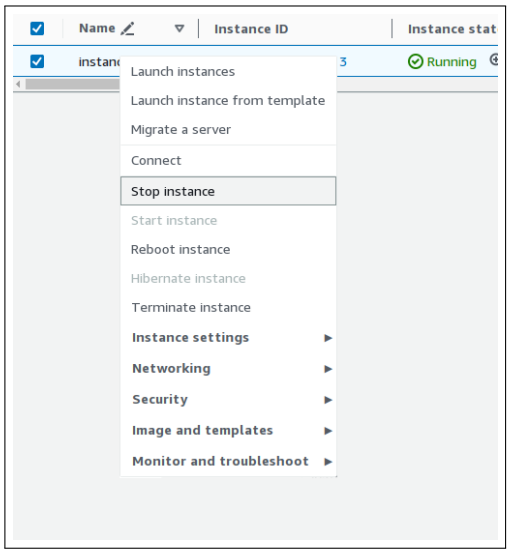


Fonte: o autor (2023)

Nota: Em destaque (retângulo vermelho) o botão *refresh instances*.

11. Ao clicar com o botão direito do mouse sobre o nome da instância, será exibido um menu de contexto (ver Figura 60) contendo alguns comandos úteis, como *Stop instance* e *Start instance*, utilizados para parar e iniciar a máquina virtual, respectivamente.

Figura 60 – Menu de contexto da instância



Fonte: o autor (2023)

Completados os passos descritos acima para a criação da máquina virtual na nuvem AWS, a instância estará configurada e em execução. O seu acesso via SSH pode ser feito através dos seguintes métodos:

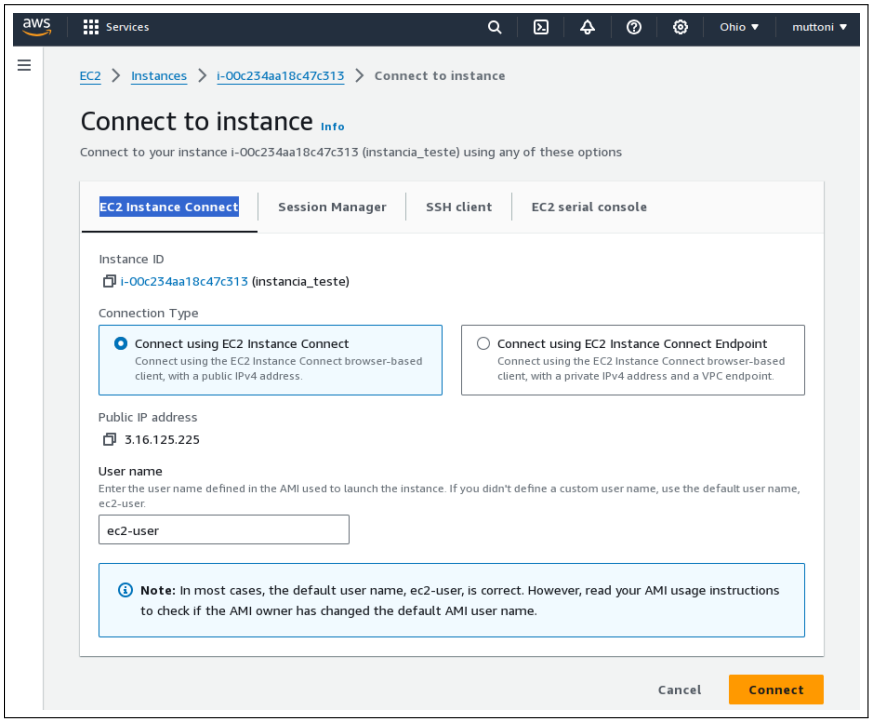
Método 1 : Acesso via *EC2 Instance Connect* utilizando apenas o navegador;

Método 2 : Acesso pelo comando `aws`. Isto requer a instalação do *AWS Command Line Interface* (AWS CLI);

Método 3 : Acesso direto usando o comando `ssh` do Linux local.

O Método 1 é o mais simples pois requer apenas o uso do navegador web. Na lista de instâncias (Figura 59), clique com o botão direito do mouse no nome da instância e selecione o comando *Connect* (Figura 60). Isto abrirá a página *Connect to instance*, que apresentará 4 métodos de conexão. O primeiro deles, selecionado por padrão, é o *EC2 Instance Connect*, conforme ilustrado pela Figura 61.

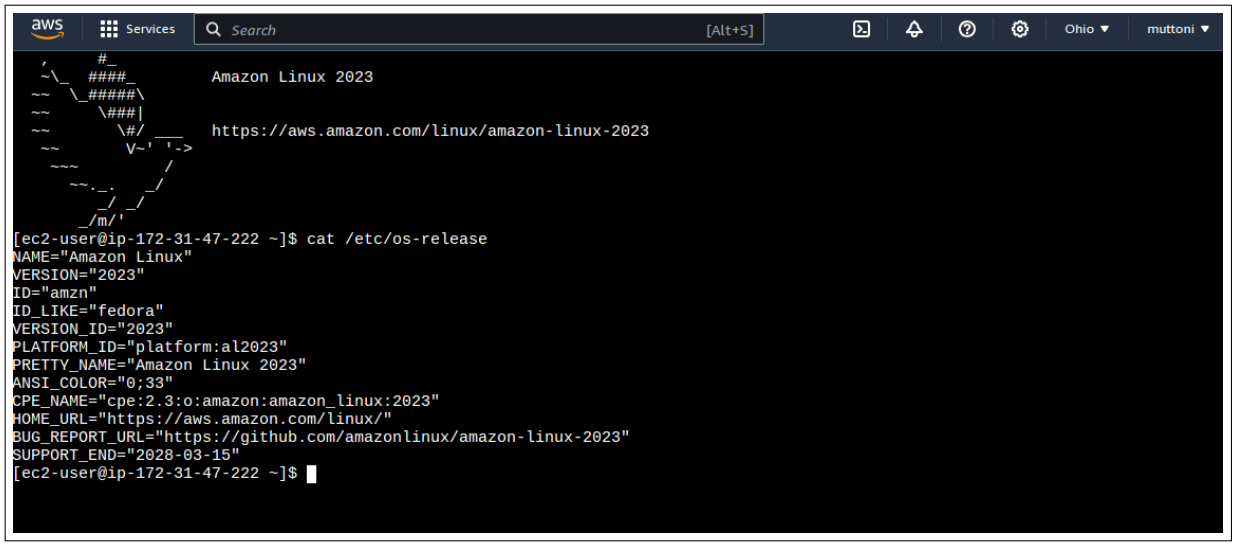
Figura 61 – Página *Connect to instance*



Fonte: o autor (2023)

Em seguida, clique no botão laranja *Connect*. Será aberta uma nova janela ou aba do navegador contendo um emulador de terminal com acesso à maquina virtual remota, conforme ilustrado pela Figura 62.

Figura 62 – Janela do EC2 Instance Connect



Fonte: o autor (2023)

O Método 2 oferece a maior quantidade de recursos, mas requer a instalação do *AWS Command Line Interface*. Para isto, deve-se seguir os passos contidos em <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>. A configuração pós-instalação pode ser executada conforme o guia rápido disponível em <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-quickstart.html>.

O Método 3 faz o uso dos comandos `ssh` e `scp` presentes em sistemas Unix/Linux e derivados. As instruções para executar este método estão listadas na opção *SSH Client* da página *Connect to instance* (vide Figura 61), e estão reproduzidas nos parágrafos seguintes com modificações para facilitar o entendimento.

Esta forma de acesso requer o arquivo `.pem` contendo a chave privada gerada durante a criação da máquina virtual. O seu par (a chave pública) já foi copiado para o arquivo `/home/ec2-user/.ssh/authorized_keys` da instância no momento da criação da chave privada. Suponhamos, a partir daqui, que o referido arquivo tenha sido nomeado `chave_privada_aws.pem`. Por questões de segurança, a chave privada precisa ter acesso restrito, com permissão de leitura apenas para o dono do arquivo. Isto pode ser feito através do seguinte comando:

```
chmod 400 chave_privada_aws.pem
```

Com a permissão correta da chave privada, o acesso via `ssh` poderá ser feito através do seguinte comando:

```
ssh -i chave_privada_aws.pem <usuario>@<end_ip>
```

No comando acima, `<usuario>` é o nome de usuário padrão do sistema operacional utilizado na máquina virtual da AWS, normalmente `ec2-user`. O `<end_ip>` é o endereço IP da máquina virtual que pode ser obtido na página que lista as instâncias (Figura 59), na coluna *Public IPv4 address* (ou *IPv6 IPs*, se aplicável).

O comando `scp` pode ser usado para copiar arquivos entre as máquinas local e remota. A sintaxe é uma das seguintes opções, dependendo da origem ou destino escolhidos:

```
scp -i chave_privada_aws.pem <arq_local_origem>
↪  <usuario>@<end_ip>:<dir_remoto_destino>

scp -i chave_privada_aws.pem <usuario>@<end_ip>:<arq_remoto_origem>
↪  <dir_local_destino>
```