TESIS DOCTORAL

2020

DISEÑO AUTOMÁTICO DE CIRCUITOS ELECTRÓNICOS ANALÓGICOS MEDIANTE ALGORITMOS EVOLUTIVOS

FEDERICO CASTEJÓN LAPEYRA

PROGRAMA DE DOCTORADO EN SISTEMAS INTELIGENTES

DIRECTOR: ENRIQUE J. CARMONA SUÁREZ

Profesor Titular en el Departamento de Inteligencia Artificial

Universidad Nacional de Educación a Distancia

Universidad Nacional de Educación a Distancia
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática

DISEÑO AUTOMÁTICO DE CIRCUITOS ELECTRÓNICOS ANALÓGICOS MEDIANTE ALGORITMOS EVOLUTIVOS

FEDERICO CASTEJÓN LAPEYRA

Ingeniero Superior de Telecomunicación por la Universidad Politécnica de Madrid
Máster en Inteligencia Artificial Avanzada por la UNED

PROGRAMA DE DOCTORADO EN SISTEMAS INTELIGENTES

Director
Enrique J. Carmona Suárez
Profesor Titular en el Departamento de Inteligencia Artificial
en la Universidad Nacional de Educación a Distancia

Este documento ha sido generado
utilizando *Lyx*, LaTeXy otras
herramientas *open-source*.

«*La mutación es aleatoria,*
la selección natural es todo lo contrario de aleatorio.»
Richard Dawkins

A Sara y Susana

Agradecimientos

Este momento representa el final de una etapa. Finalizar un trabajo, como es una tesis doctoral, con un largo recorrido de desarrollo y con un importante crecimiento personal, conlleva un esfuerzo considerable y, por ello, quiero mostrar mi agradecimiento a las personas que me han acompañado en este viaje y me han ayudado a hacerlo posible.

En primer lugar, quiero mostrar mi agradecimiento a mi director de tesis, Enrique, por haberme motivado a abordar el doctorado, así como por todo el apoyo, atención y dedicación que me ha dado durante todo este tiempo.

Quiero mostrar también mi agradecimiento, muy especialmente, a mi mujer, Sara, y a mi hija, Susana, sin cuyo apoyo, comprensión y paciencia, a lo largo de todo este tiempo, este camino no habría sido posible en modo alguno.

Resumen

Los circuitos electrónicos analógicos se caracterizan por utilizar un rango continuo de valores de una magnitud, tanto en su entrada como en su salida, a diferencia de los circuitos electrónicos digitales que se caracterizan por utilizar sólo dos valores discretos. Desde los años 70, los circuitos digitales han venido reemplazando a los circuitos electrónicos analógicos de forma general. Sin embargo, hay áreas en las que los circuitos analógicos o de señal mixta siguen en uso pues hay funciones que sólo pueden permanecer en el ámbito analógico. En el diseño de circuitos electrónicos analógicos no existe una metodología general ni existen herramientas automatizadas tan avanzadas como en el ámbito digital. Esto es así porque el diseño analógico tiene una mayor complejidad que su contrapartida digital, requiriendo todavía, la intervención de personal muy experto. Esta situación se ha llamado el «dilema analógico» y, por esta razón, hay un gran interés en disponer de herramientas automáticas para facilitar el diseño analógico.

En el ámbito de la electrónica evolutiva existen trabajos previos basados en el uso de diferentes paradigmas evolutivos. En estos trabajos se han utilizado diferentes formas de representar las soluciones (cromosomas), así como diferentes operadores de selección (de padres y de supervivientes) y de variación (cruce y mutación). Básicamente, el problema de diseño de circuitos electrónicos analógicos consiste en sintetizar un circuito electrónico analógico que cumpla un conjunto de requisitos y consta de dos tareas: selección de la topología del circuito (qué componentes se utilizan y cómo se interconectan) y dimensionamiento (cuál es el valor de cada uno de los parámetros que definen cada componente de circuito utilizado).

En esta tesis se busca obtener resultados competitivos en el problema del diseño automático de circuitos electrónicos analógicos, en relación con resultados previos obtenidos en la literatura relacionada. En particular, en esta tesis se han propuesto dos algoritmos para abordar el mencionado problema. El primero está basado en un paradigma evolutivo, ya existente en la literatura, denominado evolución gramatical

(EG) y, el segundo, en una nueva variante de dicho paradigma, desarrollada en esta tesis, y que hemos denominado evolución multigramatical (EMG). Por un lado, EG es una variante de programación genética, capaz de generar código en cualquier lenguaje de programación, basada en el uso de cromosomas formados por cadenas binarias de longitud variable y en la decodificación del cromosoma mediante una gramática libre de contexto, normalmente especificada mediante notación Backus-Naur (BNF). Por otro lado, EMG es una variante de EG en el que se utiliza un enfoque modular, dividiendo el problema en subproblemas, cada uno de los cuales se aborda mediante una gramática específica. En concreto, para resolver el problema planteado, se utiliza una gramática para cada tarea del diseño de circuitos: una para la selección de la topología y otra para el dimensionamiento. En EMG se produce una división del cromosoma en particiones que, utilizadas convenientemente por operadores de cruce adecuados (operadores de cruce homólogos), permitirá mitigar el potencial efecto destructivo de los operadores de cruce tradicionales en EG.

Se han desarrollado varias gramáticas para la generación directa de *netlists* y diferentes operadores de cruce que sacan partido del tipo de representación utilizado en cada algoritmo propuesto. Se presenta también un nuevo mecanismo de aprendizaje en EMG que permite aprender evolutivamente los parámetros deseados de las gramáticas utilizadas. En particular, dicho mecanismo se utiliza en esta tesis para aprender un parámetro importante de diseño: el máximo número de nodos (MNN) del circuito que evoluciona. Adicionalmente, se ha aplicado un mecanismo de manejo de restricciones, denominado parsimonia simple, que permite evolucionar el circuito para conseguir simultáneamente que se cumplan las especificaciones de diseño y, además, que el circuito sea lo más simple posible.

Los experimentos realizados en esta tesis para validar los algoritmos propuestos se basan en la síntesis de siete circuitos de *benchmarking* utilizados en artículos relevantes de la bibliografía relacionada. Por un lado, los resultados con EG sobre dichos circuitos son competitivos, mejorando los resultados obtenidos por algoritmos anteriormente propuestos en la literatura. Por otro lado, los resultados obtenidos con EMG mejoran, de forma importante, los resultados ya obtenidos con EG. Adicionalmente, el mecanismo de aprendizaje del parámetro MNN ha demostrado realizar un buen ajuste de dicho parámetro durante la ejecución del algoritmo, sin necesidad de ser fijado *a priori* por el usuario. Finalmente, el mecanismo de parsimonia simple se ha mostrado eficaz en el compromiso de reducir el número de componentes del circuito sin que éste deje de cumplir las especificaciones de diseño.

Summary

Analog electronic circuits use a continuous range of values of a magnitude, input or output, differently from digital circuits, which use just two discrete values. Since 70s, digital circuits have been replacing analog ones in general. However, analog circuits are still used in some domains because some functions still have to remain analog. In analog design, there is no general methodology nor advanced automated tools as there are in digital design. This is due to the fact that analog design is much more complex than digital design is, and still needs to be done by experts. This situation has been called the «analog dilemma» and is the reason why there is much interest in the development of automated tools to ease analog design.

There are previous works in the evolutionary electronics literature, based on different evolutionary paradigms. These works used different chromosome encoding schemes as different selection operators (parents and survivors) and variation operators (crossover and mutation). Basically, the problem of analog electronic circuit design consists in designing an analog electronic circuit which meets a set of requirements and can be divided into two tasks: topology selection (determining components and their interconnections) and sizing (finding the optimal component values for them).

the objective of this thesis is to get competitive results in the problem of analog electronic circuit design, compared to previous results in the related literature. In particular, this thesis proposes two algorithms to tackle this problem. The first algorithm is based on an existing evolutionary paradigm, which is grammatical evolution (GE), and the second algorithm, is based on a new GE variant, proposed in this thesis, called multigrammatical evolution (MGE). GE is a variant of genetic programming which can generate code in any programming language, and uses variable length linear binary chromosomes and uses a decoding process using a context-free grammar, normally described by its Backus-Naur normal form (BNF). MGE, is a GE variant, which uses a modular approach, dividing the design problem into sub-

problems, which are tackled by a specific grammar. In the problem of analog circuit design, we are using a grammar for each aforementioned tasks: one for topology selection and another one for sizing. MGE induces a chromosome partition which can be taken advantage of by using specially suited crossover operators (homologous crossover operators) with the objective of reducing the potentially destructive effect associated to standard crossover operators when used in GE.

Several grammars, for direct netlist generation, and several crossover operators, which take into advantage of the specific encoding used, have been developed. A new grammar parameter learning mechanism is proposed for MGE, which allows us to learn an important design parameter: maximum node number (MNN) for the circuit being evolved. Additionally, a new penalizing mechanism is proposed, called simple parsimony, which allows us to get a circuit which meets specification requirements while using the simplest possible circuit.

The experiments done to validate the proposed algorithms are based on the synthesis of Seven benchmarking circuits from the related literature. On the one hand, the results obtained using GE, over the seven benchmark circuits, are competitive, improving the results obtained by earlier algorithms proposed in the literature. On the other hand, the results obtained by MGE improve importantly, the results obtained using GE. Additionally, the parameter learning mechanism showed a good adjustment of the MNN parameter during the algorithm execution, without the need for the user to set it beforehand. Finally, the simple parsimony mechanism was able to reduce the component number of the circuits obtained while still meeting design specifications.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

AE	Algoritmo evolutivo.
AED	Algoritmo de estimación de la distribución.
AES	Average number of evaluations to a solution.
AG	Algoritmo genético.
AGE	Analog genetic encoding.
AIS	Sistema inmune artificial, por sus siglas en inglés.
BDD	Diagrama de decisión binario, por sus siglas en inglés.
BG-MHX	Operador de cruce homólogo multipartición basado en gramáticas de bloques, por sus siglas en inglés.
BNF	Backus-Naur form.
BPSO	Optimización mediante enjambre de partículas binario, por sus siglas en inglés.
CAD	Diseño asistido por ordenador, por sus siglas en inglés.
CGP	Programación genética cartesiana, por sus siglas en inglés.
CI	Circuito integrado.
DSP	Procesador digital de señal, por sus siglas en inglés.
EBNF	Extended BNF.
ED	Evolución diferencial.

EDA	Diseño electrónico automatizado, por sus siglas en inglés.
EE	Estrategia evolutiva.
EG	Evolución gramatical.
EHW	Hardware evolutivo, por sus siglas en inglés.
EMG	Evolución multigramatical.
ESL	Electronic-system level.
FPAA	Matriz de componentes analógicos programables, por sus siglas en inglés.
FPGA	Matriz de puertas lógicas programable, por sus siglas en inglés.
FPTA	Matriz de transistores programable, por sus siglas en inglés.
GPL	Licencia pública general de GNU, por sus siglas en inglés.
GRN	Red de regulación genética, por sus siglas en inglés.
HDL	Lenguaje descriptivo del hardware, por sus siglas en inglés.
IP	Intellectual property.
MAE	Error medio absoluto, por sus siglas en inglés.
MBF	Mean best fitness.
minBF	Minimum best fitness.
MOGA	Algoritmo genético multiobjetivo, por sus siglas en inglés.
MOSFET	Transistor de efecto de campo metal-óxido-semiconductor, por sus siglas en inglés.
PCB	Circuito impreso, por sus siglas en inglés.
PG	Programación genética.
RTL	Register transfer level.

SPICE	Programa de simulación con énfasis en circuitos integrados, por sus siglas en inglés.
SR	Success rate.
USB	Universal serial bus.

1. Introducción

Esta tesis tiene como objeto de investigación el diseño automático de circuitos electrónicos analógicos mediante algoritmos evolutivos (AE), por lo que se encuadra en el campo de la electrónica evolutiva.

En este capítulo se incluye la motivación de la tesis en la sección 1.1. La sección 1.2 contiene la descripción del problema. A continuación, se establecen la hipótesis y objetivos en la sección 1.3. En la sección 1.4 se describe la metodología utilizada. En la sección 1.5 se describen las limitaciones de la tesis. La sección 1.6 describe los materiales utilizados. Finalmente, la estructura de la tesis se indica en la sección 1.7.

1.1. Motivación

Los circuitos electrónicos analógicos se caracterizan por utilizar un rango continuo de valores de una magnitud eléctrica, como por ejemplo el voltaje, a diferencia de los circuitos electrónicos digitales que se caracterizan por utilizar sólo dos valores discretos.

Desde los años 70, los circuitos digitales han venido reemplazando a los circuitos electrónicos analógicos de forma general. Sin embargo, hay ámbitos en los que los circuitos analógicos o de señal mixta siguen en uso pues hay funciones que sólo pueden permanecer en el ámbito analógico (Gielen & Rutenbar, 2002; Camenzind, 2005).

En el diseño de circuitos electrónicos analógicos no existe una metodología general ni existen herramientas automatizadas tan avanzadas como en el ámbito digital. Esto es así porque el diseño analógico tiene una mayor complejidad que su contrapartida digital, incluso para problemas pequeños (Scheible & Lienig, 2015). Por este motivo, el diseño de circuitos analógicos todavía requiere la intervención de personal muy

experto. Esta situación se ha llamado el «dilema analógico» (Aaserud & Nielsen, 1995) y, por esta razón, hay un gran interés en disponer de herramientas automáticas para facilitar el diseño analógico.

En el ámbito de la electrónica evolutiva existen trabajos previos basados en el uso de diferentes paradigmas evolutivos tales como programación genética (PG) (Koza et al., 1999a), estrategias evolutivas (EE) (Sapargaliyev & Kalganova, 2012; Lanchares et al., 2013), o algoritmos genéticos (AG) (Grimbleby, 1995; Aggarwal, 2003), entre otros. En estos trabajos se ha realizado el diseño de circuitos electrónicos mediante el uso de un AE y utilizando diferentes formas de codificación de los circuitos a evolucionar y diversos operadores de variación especializados. Aunque los resultados han sido prometedores, pensamos que aún es posible probar el uso de otros algoritmos evolutivos en este contexto y analizar y evaluar los resultados obtenidos comparándolos con resultados previos.

1.2. Descripción del problema

El problema de diseño de circuitos electrónicos analógicos consiste en sintetizar un circuito electrónico analógico que cumpla un conjunto de requisitos. Estos requisitos definen el comportamiento del circuito y las condiciones en las que opera. Los requisitos de comportamiento del circuito definen las salidas que deben obtenerse del mismo en función de las entradas que recibe y de su estado interno. En los circuitos abordados en esta tesis se contemplará una única salida que será función directa de la entrada considerada. Adicionalmente, las especificaciones también introducen las restricciones bajo las que el circuito tendrá que operar. Estas restricciones contemplan variables externas como la temperatura, o bien, componentes externos que proveen al circuito de alimentación, señal de entrada y carga para la salida. Estos componentes externos se agrupan en una estructura llamada parte fija o estructura de test (Koza et al., 1999a) y es donde se alojará el circuito objetivo. La parte fija no se modifica durante la ejecución del algoritmo. En la figura 1.1 se muestra un ejemplo de parte fija donde se puede observar un recuadro marcado con línea discontinua que albergará el futuro circuito objetivo debidamente conectado a los nodos representados.

Una vez se disponga de un circuito candidato conectado a la parte fija, se podrá realizar la evaluación del mismo. En el caso de los circuitos abordados en esta tesis,

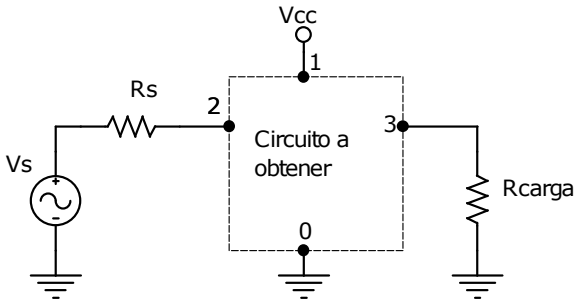


Figura 1.1. – Ejemplo de parte fija o estructura de test formada por todos los componentes de circuito situados fuera del recuadro marcado con línea discontinua.

esta evaluación consistirá en introducir la señal de entrada y en medir la salida obtenida en un conjunto de puntos de ajuste. Las diferencias entre la salida obtenida y la salida esperada se introducirán en una función de error que devolverá un valor de mérito, el cuál permitirá evaluar el grado de bondad del circuito candidato. De esta manera, el problema de diseño se puede transformar en un problema de optimización cuyo objetivo consistirá en encontrar un circuito que minimice el valor devuelto por la función de error utilizada, tal como se muestra en (1.1), donde $\boldsymbol{O}$ es la salida esperada para la señal de entrada, $\boldsymbol{I}$, y $\widetilde{\boldsymbol{O}}_j$ es la salida obtenida para el circuito j-ésimo ante la misma entrada. Por tanto, el problema de diseño de circuitos puede ser abordado como un problema de minimización.

$$\underset{j}{minimizar} \quad error(\boldsymbol{O} - \widetilde{\boldsymbol{O}}_j) \tag{1.1}$$

El diseño de un circuito electrónico analógico comprende dos tareas: selección de la topología y dimensionamiento. La selección de la topología consiste en determinar el conjunto de componentes electrónicos que compondrán el circuito y las conexiones entre los mismos. El dimensionamiento consiste en la asignación de valores a cada componente que forma parte del circuito. Ambas tareas deben dar lugar a un circuito que cumpla las especificaciones del mismo.

Para atender el problema de diseño de circuitos mediante un AE será necesario abordar ambas tareas y, además, será necesario atender las siguientes consideraciones: establecer una forma de representar un circuito mediante un cromosoma y definir un conjunto de operadores de variación adecuados que permitan introducir variedad en los individuos que forman la población. Finalmente, el proceso evolutivo se

encargará de encontrar la mejor solución.

En nuestro caso, cada circuito vendrá descrito por una *netlist*, es decir, por una lista de los componentes del circuito, sus conexiones y los valores de los mismos. El formato de una *netlist* viene fijado por el simulador utilizado para evaluar cada individuo (circuito).

1.3. Hipótesis y objetivos de investigación

Existe evidencia de que, en los últimos años, los denominados paradigmas evolutivos guiados por gramáticas han sido utilizado con éxito en diferentes problemas de diseño. Un ejemplo de este tipo de paradigma es el denominado Evolución Gramatical (EG), que ha sido aplicado con éxito en el diseño automático de circuitos electrónicos digitales. Sin embargo, no existe constancia que EG haya sido aplicado al diseño de circuitos analógicos. Dado el potencial de este paradigma pensamos que merece la pena estudiar la posibilidad de que también puede ser aplicado con éxito al diseño de este último tipo de circuitos. En este contexto, la hipótesis principal de esta tesis asume que **es posible diseñar e implementar algoritmos basados en EG para abordar el problema del diseño automático de circuitos analógicos de forma eficaz y eficiente**.

Por tanto, el objetivo principal de la tesis es **mostrar evidencia de la validez de la hipótesis planteada.** El objetivo principal se puede descomponer en los siguientes subobjetivos:

1. Elección de un conjunto de circuitos *benchmark*.
2. Desarrollo de gramáticas adecuadas para el diseño de circuitos electrónicos analógicos en el marco de EG.
3. Desarrollo de operadores de cruce que puedan incorporar conocimiento del dominio para mejorar sus prestaciones.
4. Implementación de un algoritmo basado en EG que permita el diseño automático de circuitos electrónicos analógicos: ACID-GE (*Analog Circuit Design based on Grammatical Evolution*).
5. Evaluación del algoritmo ACID-GE, sobre los circuitos *benchmark* incluyendo un estudio estadístico del comportamiento de diferentes parámetros sobre el rendimiento del algoritmo evolutivo diseñado.

6. Comparativa de los resultados obtenidos por ACID-GE frente a trabajos previos en la literatura.
7. Estudio de nuevas variantes de EG que puedan acomodarse mejor al problema del diseño de circuitos analógicos.
8. Desarrollo de nuevas gramáticas para el diseño de circuitos electrónicos en el marco de la nueva variante de EG propuesta.
9. Diseño de operadores de cruce en el marco de la nueva variante de EG propuesta.
10. Implementación de un algoritmo basado en la nueva variante propuesta para abordar el problema de diseño de circuitos analógicos.
11. Evaluación del nuevo algoritmo propuesto sobre los circuitos *benchmark*, incluyendo un estudio estadístico del mismo, que permita realizar una comparativa con los resultados obtenidos por ACID-GE.
12. Estudio, diseño e implementación de un mecanismo de aprendizaje de parámetros de gramática con el objeto de liberar al usuario de la tarea de tener que especificar manualmente el valor de determinados parámetros que pueden ser críticos en las prestaciones del algoritmo. Este mecanismo debería incorporarse sólo al mejor de los dos algoritmos implementados.
13. Evaluación del mecanismo de aprendizaje automático de parámetros sobre los circuitos *benchmark*, incluyendo un estudio estadístico del mismo, que permita realizar una comparativa con los resultados obtenidos sin utilizar dicho mecanismo.
14. Estudio, diseño e implementación de un mecanismo de manejo de restricciones que, incorporado al mejor de los dos algoritmos implementados, permita obtener circuitos que sigan cumpliendo las especificaciones de diseño, pero con el mínimo número de componentes.
15. Evaluación del mecanismo de manejo de restricciones sobre los circuitos *benchmark*, incluyendo un estudio estadístico del mismo, que permita realizar una comparativa con los resultados obtenidos sin utilizar dicho mecanismo.

1.4. Metodología

Existen varios trabajos previos en electrónica evolutiva en el ámbito de diseño de circuitos electrónicos analógicos. En particular, esta tesis utilizará tres de ellos como referencia por su relevancia en la literatura relacionada y por trabajar sobre un conjunto de problemas comunes, facilitando así la evaluación y comparación de resultados. Entre estos trabajos, el más importante corresponde al grupo de Koza, basado en el uso de programación genética (PG) (Koza et al., 1999a). Aunque es un trabajo de hace veinte años, todavía sigue siendo muy relevante debido al amplio conjunto de circuitos sintetizados con su enfoque. Muchos de los circuitos estudiados por Koza se han establecido como circuitos de referencia o *benchmark* y han sido utilizados en trabajos sucesivos. Por otro lado, el enfoque de Mattiussi & Floreano (Mattiussi & Floreano, 2007) se caracteriza principalmente por usar una representación de circuito novedosa denominada codificación genética analógica (AGE, por sus siglas en inglés) y, el de Sapargaliyev & Kalganova (Sapargaliyev & Kalganova, 2012), por el uso de un enfoque abierto basado en estrategias evolutivas (EE). Ambos trabajos utilizaron diferentes problemas *benchmark* de Koza, sobre los que consiguieron mejorar sus resultados.

Los tres trabajos previos anteriormente indicados se basan en el uso de una representación especializada y compleja de los circuitos analógicos. Adicionalmente, los enfoques de Koza *et al.* y Sapargaliyev & Kalganova requieren el uso de funciones de transformación de circuitos para mantener su viabilidad. Estas funciones se incorporan en el proceso de decodificación, caso del primer enfoque, o en el operador de mutación, caso del segundo enfoque. Finalmente, AGE utiliza una representación y un paradigma evolutivo diferente de los dos anteriores, que también implican el uso de una representación y operadores complejos.

En esta tesis se propone el uso de evolución gramatical (EG) como paradigma evolutivo base que permitirá el uso de una representación más sencilla del circuito. Hay que tener en cuenta que en EG se utilizan cadenas binarias de longitud variable y, además, hace uso el uso de operadores de variación estándar o con adaptaciones simples, no requiriendo de ningún mecanismo para garantizar la propiedad de clausura (lo que sí es necesario en el caso de PG). Dicha propiedad queda asegurada por el proceso de decodificación de EG basado en el uso de una gramática. Además, pensamos que el uso de una gramática tiene una ventaja añadida y es la de permitir incorporar cierto tipo de conocimiento del dominio en el diseño del algoritmo evo-

lutivo, reduciendo de esta manera el tamaño del espacio de búsqueda y facilitando, a su vez, la búsqueda de la solución.

Por otro lado, actualmente, existe un apasionante debate en la literatura relacionada sobre el potencial efecto destructivo asociado al operador de cruce estándar en EG. Por este motivo, en esta tesis, se estudiará la posibilidad de estudiar y diseñar operadores de cruce en EG más eficaces y que puedan incorporar un cierto conocimiento del dominio en su diseño. Otra alternativa es estudiar la posibilidad de diseñar otras variantes de EG que se adapten mejor al problema del diseño de circuitos analógicos, y que, de esta manera, seamos capaces de obtener resultados más competitivos.

Por último, existe la posibilidad de plantear un reto más exigente consistente en diseñar un circuito que cumpla con las especificaciones y que, simultáneamente, sea lo más simple posible. En este caso se analizarán distintas técnicas en el contexto de algoritmos evolutivos que abordan problemas de optimización con restricciones y, además, se estudiará la posibilidad de emplearlas o adaptarlas para superar el reto planteado.

1.5. Limitaciones

En esta tesis sólo se abordará el diseño automático de circuitos electrónicos analógicos, no contemplando el diseño de circuitos digitales. Adicionalmente, dentro del diseño analógico, esta tesis se centrará en el estudio, uso y adaptación de diferentes técnicas basadas en algoritmos evolutivos.

Los algoritmos implementados en esta tesis se basarán en la generación directa de la *netlist* del circuito objetivo (véase la sección 2.3.1 para una clasificación de los algoritmos).

Además de las dos tareas básicas asociadas al diseño de circuitos, selección de la topología y dimensionamiento, existe una tarea adicional, que consiste en determinar la disposición espacial o *layout* del circuito. En esta tesis, se abordará el diseño analógico, teniendo en cuenta únicamente la selección de topología y dimensionamiento, por lo que la fase de disposición espacial o *layout* no será abordada.

1.6. Materiales

Una herramienta fundamental en el diseño de circuitos electrónicos analógicos es el simulador de circuitos. En el caso de esta tesis, esta herramienta se utilizará para obtener el valor de adaptación de los circuitos candidatos. El simulador SPICE es el estándar de facto de la industria electrónica para circuitos analógicos (McConaghy & Gielen, 2006a). Este simulador se desarrolló en la Universidad de California, Berkeley en 1973 (Nagel & Pederson, 1973) y, a partir del mismo, se han derivado nuevas implementaciones, bien comerciales o bien de código abierto. En esta tesis se utilizará el simulador NGSpice (Nenzi & Vogt, 2011) que es una implementación de código abierto derivada de SPICE3.

Para la evaluación de los diferentes algoritmos propuestos en la tesis y su comparación con los trabajos previos anteriormente indicados, se utilizarán varios problemas basados en los trabajos de Koza y que, a lo largo de los años, han sido usados por investigadores en este campo como problemas de referencia o *benchmark*. En concreto, se utilizarán un circuito sensor de temperatura, un circuito de referencia de voltaje, un circuito generador de función gaussiana y cuatro circuitos computacionales dedicados a implementar las siguientes funciones matemáticas: potencia al cuadrado, raíz cuadrada, potencia al cubo y raíz cúbica. Todos ellos deben resolverse utilizando tanto componentes pasivos, tales como resistencias y condensadores, como activos, tales como transistores bipolares o MOSFET. La complejidad de cada uno de estos circuitos es tal que requiere de conocimiento experto para su diseño (Koza et al., 1999a).

1.7. Estructura de la tesis

A continuación, se indica cómo está constituida la estructura del resto de la tesis. El capítulo 2 contiene una revisión del estado del arte relacionado con el problema del diseño automático de circuitos analógicos, asumiendo que es abordado desde la perspectiva de la computación evolutiva. El capítulo 3 contiene la descripción de las diferentes aproximaciones planteadas en esta tesis para abordar el problema planteado. El capítulo 4 describe las especificaciones de los circuitos propuestos como casos de estudio. En capítulo 5, se muestran los resultados obtenidos y el análisis de los mismos, mostrando también los mejores circuitos sintetizados. Finalmente, el

capítulo 6 contiene las conclusiones.

Adicionalmente, la tesis contiene una relación de apéndices. El apéndice A muestra las publicaciones realizadas como producto de esta tesis. El apéndice B contiene los resultados del denominado problema de regresión simbólica planteado en la sección 3.3.3. El apéndice C recopila las *netlists* correspondientes a los mejores circuitos sintetizados. El apéndice D muestra las diferentes gramáticas utilizadas en cada aproximación y para cada uno de los problemas específicos abordados en esta tesis. El apéndice E hace un análisis de la potencia estadística de los parámetros de los experimentos y de los tests de hipótesis utilizados. El apéndice F muestra información adicional relacionada con los mecanismos de *wrapping* y caché de los experimentos. Finalmente, el apéndice G incluye la documentación relacionada con los dos algoritmos implementados.

Está prevista la liberación de los algoritmos implementados en esta tesis bajo la licencia pública general (GPL, por sus siglas en inglés) de GNU en su versión 3, según se describe en el apéndice G.

2. Estado del arte

La electrónica puede definirse como la ciencia y tecnología que estudia el movimiento de partículas cargadas a través de un gas, el vacío o de un semiconductor (Millman & Valls, 1981). Se inicia con la invención de la válvula de vacío en 1904 por J. A. Fleming. Las válvulas de vacío permitieron una primera etapa de la electrónica, pero presentaban problemas, tales como una gran generación de calor y fragilidad. A partir de la invención del transistor bipolar en 1948 por W. B. Shockley, W. H. Brattain y J. Bardeen, se produjo la introducción de los semiconductores, lo que permitió el uso de componentes de estado sólido. La posterior introducción del circuito integrado (CI), a partir de los trabajos de J.S. Kilby (1958) y R. Noyce (1959), dio lugar a la aparición de la microelectrónica.

Por otro lado, el trabajo de Shannon enlazando el álgebra de Boole con la conmutación de circuitos, del ámbito de la telefonía, sentó las bases para la aparición de la electrónica digital (Shannon, 1938). La electrónica digital se basa en el uso de dos niveles de voltaje que se corresponden con los niveles lógicos utilizados en el álgebra de Boole.

El avance de los CI permitió la integración de puertas lógicas completas en un único chip y posteriormente circuitos digitales completos. Este rápido avance ha permitido su producción masiva y reducción de costes. Todo ello, junto con la aparición del ordenador digital y su posterior confluencia con las tecnologías de las comunicaciones, han constituido la revolución digital, también llamada tercera revolución industrial, en la que se ha venido produciendo una sustitución de las tecnologías mecánicas y analógicas por la tecnología digital. Este proceso transformador de la revolución digital todavía sigue en curso y tiene un alcance mayor del ámbito de esta tesis.

En el ámbito de los circuitos electrónicos analógicos, se ha venido produciendo una sustitución gradual de estos por circuitos digitales en casi todos los campos de aplicación. Actualmente, es muy habitual el uso de procesadores digitales de señal (DSP, por sus siglas en inglés) en lugar de filtros analógicos. Este avance del dominio di-

gital ha supuesto una reducción de los campos donde anteriormente sólo existía la alternativa analógica. Tanto es así, que de forma continuada se viene prediciendo el abandono de la tecnología analógica. Sin embargo, han surgido opiniones en contra (Rutenbar, 1993; Gielen & Rutenbar, 2002; Gilbert, 2001; Camenzind, 2005; Soderstrand, 2012), ya que hay funcionalidades que no pueden ser digitalizadas y deben permanecer en el campo de las tecnologías analógicas. Una de estas funcionalidades es el caso de los subsistemas de entrada y salida, puesto que los sensores y transductores son analógicos. Por ejemplo, micrófonos, detectores de infrarrojos, antenas, altavoces o motores. En este sentido, se ha llegado a decir que «el mundo es analógico» (Camenzind, 2005) y que «la revolución digital está construida sobre una realidad analógica» (Camenzind, 2005). Otras aplicaciones que deben permanecer como tecnologías analógicas son las comunicaciones de radiofrecuencia (Gielen & Rutenbar, 2002) y las aplicaciones de muy baja potencia (Aaserud & Nielsen, 1995).

Aunque la preeminencia de lo digital y los rápidos e impactantes avances que se producen en este campo ensombrecen el ámbito analógico, actualmente, el mercado de CI analógicos sigue desempeñando un papel muy firme en la industria electrónica. La tasa prevista de crecimiento anual compuesta del mercado de CI analógicos es de un 6,6 %, frente a un 5,1 % del total del mercado de CI incluyendo analógicos y digitales. Adicionalmente, dicha previsión porcentual supone un crecimiento del mercado de circuitos analógicos desde 54.500 millones de dólares en 2017, hasta 74.800 millones de dólares en 2022 (D&R, 2018; McClean, 2018).

A lo largo de este capítulo se realiza un análisis del estado del arte relacionado con el diseño de circuitos analógicos en general y también, de una forma más específica, desde el punto de vista de la electrónica evolutiva, siendo este último el campo en el que se encuadra esta tesis. También será necesario comparar con aspectos relevantes del diseño de circuitos digitales. Así, en la sección 2.1, se indican las diferencias en el diseño de circuitos analógicos y digitales. En la sección 2.2, se detallan las fases en que se divide el diseño de circuitos electrónicos. La sección 2.3 introduce la electrónica evolutiva y la problemática que aborda. La sección 2.4 muestra los trabajos previos más relevantes en electrónica evolutiva aplicada al diseño de circuitos analógicos. Finalmente, la sección 2.5 describe algunos trabajos en el ámbito del diseño de circuitos digitales.

2.1. Diferencias en el diseño de circuitos analógicos y digitales

El diseño de los circuitos electrónicos se realiza utilizando herramientas de diseño electrónico automatizado (EDA, por sus siglas en inglés) que se encuadran dentro del diseño asistido por ordenador (CAD, por sus siglas en inglés). Atendiendo al tipo y potencia de estas herramientas, nos encontramos con que los circuitos electrónicos digitales disponen de herramientas EDA muy avanzadas y en las que muchos aspectos están automatizados de forma completa.

El diseño digital comienza a partir de una descripción funcional o de alto nivel del circuito que se desea implementar *(electronic-system level* en inglés, ESL). Una herramienta automatizada genera una traducción al nivel de transferencia de registros o *register transfer level* (RTL) en inglés, donde se describe la funcionalidad del circuito objetivo indicando la transferencia de datos entre registros, sus operaciones lógicas y la sincronización. Esta descripción se realiza en lenguajes descriptivos del hardware (HDL, por sus siglas en inglés), como son VHDL o Verilog (Xiu, 2007). A partir de una descripción en un lenguaje HDL se puede realizar su traducción a nivel de puertas lógicas de forma automatizada, utilizando una librería de celdas lógicas apropiada para la tecnología objetivo (Xiu, 2007). Adicionalmente, existen librerías de bloques de subsistemas completos como procesadores, memoria, interfaces USB, etc... (Xiu, 2007). Estas librerías se venden bajo licencia y se denominan IP (propiedad intelectual, por sus siglas en inglés).

En cambio, en el caso de los circuitos electrónicos analógicos, no existe una automatización tan completa para su diseño. En particular, se han llevado a cabo importantes avances para obtener herramientas que asistan al diseñador en esta tarea (Johnson, 2015), tanto en el caso de diseño de circuitos integrados analógicos como de señal mixta. Por ejemplo, la disponibilidad de librerías de celdas o el uso de sistemas expertos basados en reglas (Gielen & Rutenbar, 2002) van en este sentido. En relación con los HDL, existen extensiones para los circuitos analógicos y de señal mixta, como son VHDL-AMS y Verilog-AMS (Pêcheux et al., 2005), así como también existen librerías IP analógicas (Xiu, 2007).

A pesar de los importantes avances en el campo del diseño analógico, las herramientas EDA no están tan avanzadas como en el campo digital. Esto es así porque el diseño analógico tiene una mayor complejidad que su contrapartida digital, incluso

para problemas pequeños (Scheible & Lienig, 2015). A diferencia de los sistemas puramente digitales, en el diseño de los sistemas de señal mixta, típicamente, es necesario optimizar docenas de especificaciones opuestas y basadas en valores continuos, todo ello dependiendo de la habilidad del diseñador para utilizar un rango de comportamientos no lineales a lo largo de varios niveles de abstracción, desde el nivel de dispositivos al nivel de sistema (Rutenbar et al., 2007).

El diseño analógico comprende un número mayor de clases de circuitos, requiere un enfoque de diseño ajustado para cada circuito y es más susceptible al ruido (Scheible & Lienig, 2015). La descomposición en subsistemas no es tan limpia como en el caso digital, pues los circuitos analógicos se comportan de forma bidireccional (Gilbert, 2001), afectando al resto del diseño.

Esta mayor complejidad en el caso analógico da lugar a que sea más costoso el diseño de la parte analógica de los CI de señal mixta, aun cuando comprende un número menor de dispositivos que la parte digital, pudiendo convertirse en el cuello de botella del diseño total de este tipo de CI (Gielen & Rutenbar, 2002; Scheible & Lienig, 2015).

En relación con el diseño analógico, también se ha dicho que es menos sistemático y más intensivo en conocimiento (Harjani et al., 1987; Gielen & Rutenbar, 2002), requiriendo el diseño por parte de personal muy experto en el campo. En este sentido también se ha indicado que el conocimiento experto analógico no puede traducirse a expresiones formales de alto nivel de abstracción (Scheible & Lienig, 2015). Esta necesidad de conocimiento experto ha chocado con una escasez de dichos profesionales que dominen dichos conocimientos, debido principalmente al predominio del diseño digital (Camenzind, 2005) y a las dificultades propias del ámbito analógico, pues el aprendizaje del diseño analógico es un proceso que lleva años para comenzar y décadas para dominar (McConaghy et al., 2009). Todo ello ha dado lugar a lo que se ha denominado *dilema analógico* (Aaserud & Nielsen, 1995).

2.2. Fases de diseño de los circuitos electrónicos

En una primera aproximación, el diseño de circuitos consta de dos tareas: la selección de la topología del circuito y el dimensionamiento. La selección de la topología del circuito consiste en determinar el conjunto de componentes y sus conexiones, mientras que el dimensionamiento consiste en la asignación de los valores de los

parámetros que definen cada componente. Ambas tareas pueden realizarse de forma separada o en paralelo (Martens & Gielen, 2008) y su resultado debe ser un circuito que cumpla la funcionalidad requerida.

En el caso de abordar las tareas indicadas de forma separada, su ejecución es secuencial, por lo que se comienza por la selección de la topología, abordando el dimensionamiento posteriormente. Sin embargo, también es posible trabajar sobre un conjunto de topologías candidatas, proceder a su dimensionamiento, y hacer la selección final de la topología posteriormente (Martens & Gielen, 2008). En el caso de ejecución paralela de ambas tareas, se pueden considerar varias opciones de topología: para la arquitectura completa, o para subbloques, permitiendo la combinación de dichas opciones y la selección entre ellas una vez hecho su dimensionamiento (Martens & Gielen, 2008).

Además de los enfoques mencionados, existen los enfoques descendente o *top-down* y ascendente o *bottom-up* (Sorkhabi & Zhang, 2017). En el enfoque descendente se comienza a partir de una descripción funcional del circuito a alto nivel de abstracción, normalmente en un HDL, y se va descomponiendo en pasos sucesivos hasta llegar a un circuito final (Martens & Gielen, 2008). En el enfoque ascendente se comienza a bajo nivel de abstracción, bien a nivel de componentes o bien de bloques, como las celdas analógicas, cuya agregación permite alcanzar una funcionalidad de más alto nivel (Sorkhabi & Zhang, 2017). El enfoque descendente presenta problemas al llegar a bajo nivel, pues aparecen problemas como el acoplamiento entre bloques (Martens & Gielen, 2008), por lo que se ha propuesto un enfoque intermedio entre los enfoques ascendente y descendente que podría llamarse de «encuentro en un punto medio» (*«meet-in-the-middle»* (Martens & Gielen, 2008) *o «Bottom-up meets top-down»* (Scheible & Lienig, 2015), en inglés), mediante el que se utilizan las fortalezas de ambos enfoques: la descomposición de un problema en subproblemas del enfoque descendente y la capacidad de reutilización de celdas del enfoque ascendente (Scheible & Lienig, 2015).

De un lado, la selección de la topología puede basarse en la experiencia del diseñador o bien en la reutilización de una celda o un bloque IP contenidos en una librería de topologías. También existen herramientas EDA que pueden asistir en la selección de la topología como sistemas expertos basados en reglas (Martens & Gielen, 2008) o bien, herramientas basadas en AE (Sorkhabi & Zhang, 2017) (véase la sección 2.3). Por otro lado, en el caso del dimensionamiento, existe múltiples algoritmos de

optimización como podrían ser *simulated annealing* (Gielen et al., 1990) o, de nuevo, los AE (Zebulum et al., 1998b; Sabat et al., 2009; El Dor et al., 2016; Povoa et al., 2016).

Una herramienta fundamental en el diseño analógico es el simulador de circuitos electrónicos, siendo SPICE (programa de simulación con énfasis en circuitos integrados, por sus siglas en inglés) el estándar de facto de la industria para circuitos analógicos. Este simulador fue desarrollado por Nagel y Pederson en la Universidad de California Berkeley (Nagel & Pederson, 1973). La distribución abierta del mismo contribuyó a su expansión, siendo también la base para desarrollos propietarios comerciales. La última versión de Berkeley es *Spice3f4*.

Además de las dos tareas vistas hasta ahora en el diseño de circuitos electrónicos, existe una tarea adicional, que consiste en determinar la disposición espacial o *layout* del circuito. En el caso de los circuitos integrados, esta tarea busca una disposición eficiente, que permita su integración en un chip. En el caso de componentes discretos, la disposición espacial daría lugar a la obtención del esquema de circuito impreso (PCB, por sus siglas en inglés).

2.3. Electrónica evolutiva

La electrónica evolutiva surge en 1998 (Zebulum et al., 1998a) y busca conseguir la síntesis automática de circuitos electrónicos mediante el uso de AE. Esta síntesis puede abordar ambas tareas del diseño de circuitos electrónicos: la selección de la topología y el dimensionamiento. Sin embargo, hay trabajos que sólo utilizan un AE como herramienta de optimización numérica para asistir a un diseñador humano en la tarea de dimensionamiento, una vez la topología ya ha sido seleccionada (Puhan et al., 1999; Varios, 2013). Adicionalmente, se buscan soluciones que no sólo cumplan los requisitos de diseño, sino que además sean óptimas en términos de número de componentes, velocidad o consumo.

Los AE se inspiran en la selección natural y pueden obtener soluciones a problemas complejos. Estos algoritmos utilizan una población de soluciones tentativas que se seleccionan de acuerdo a su nivel de adaptación. Las mejores soluciones se cruzan y se mutan dando lugar a una nueva generación de soluciones tentativas. Mediante generaciones sucesivas el algoritmo mejora la población de soluciones en general y la mejor solución en particular. Los AE aplicados a tareas de diseño pueden trabajar

sin reglas de diseño ni conocimiento experto en el diseño (Grimbleby, 2000). Por este motivo, podrían dar lugar a soluciones poco convencionales que desafiarían la intuición de un diseñador humano (Eiben & Smith, 2003). Aún más, los circuitos obtenidos pueden ser difíciles de analizar por diseñadores humanos y, en estos casos, incluso ser objeto de rechazo (McConaghy & Gielen, 2006b).

En relación con la electrónica evolutiva nos encontramos también con el campo del hardware evolutivo (EHW, por sus siglas en inglés), que se caracteriza por realizar la evaluación de los circuitos candidatos en el propio hardware, en cuyo caso la evaluación se denomina evaluación intrínseca. En este ámbito es muy habitual el uso de sistemas basados en una matriz de puertas lógicas programables (FPGA, por sus siglas en inglés) o en una matriz de componentes analógicos programables (FPAA, por sus siglas en inglés), que son unos tipos de CI que permiten su programación con la configuración del circuito candidato a evaluar (Higuchi et al., 1996; Hidalgo et al., 2003).

En oposición al caso anterior, se denomina evaluación extrínseca a la evaluación de los circuitos candidatos mediante un simulador de circuitos, como es el caso de SPICE (véase la sección 2.2).

En relación con trabajos previos en la literatura sobre electrónica evolutiva, nos encontramos con que, debido a la falta de métodos completamente automatizados para la síntesis de los circuitos analógicos, existe un mayor interés en el diseño analógico que en el digital (véase la sección 2.4), si bien también se pueden encontrar trabajos previos en el campo de los circuitos digitales (véase la sección 2.5).

2.3.1. Codificación de circuitos en un cromosoma

Uno de los principales problemas que aparecen en la electrónica evolutiva es la elección de la representación o codificación de un circuito electrónico en un cromosoma. Esta elección está fuertemente relacionada con el tipo de AE utilizado. A continuación, se verán los principales esquemas de codificación que han ido surgiendo en electrónica evolutiva.

El primer esquema de codificación provenía de la representación de un circuito electrónico en forma de lista de componentes o *netlist,* que es un formato utilizado como entrada a los simuladores de circuitos, como es el caso de SPICE. En cada línea de una lista de componentes se codifica un componente individual junto con los nodos

a los que está conectado y los valores que definen los parámetros de dicho componente. Esta forma de lista facilita su codificación en forma lineal, dando lugar a los denominados como esquemas de *codificación directa*, basados en la codificación de los componentes y sus conexiones, de forma muy similar a una *netlist*. Este esquema de codificación es muy apropiado para AE que utilizan cromosomas lineales.

Un segundo esquema de codificación de circuitos surgió en el ámbito de PG donde la representación se basa en árboles de parseado. Este enfoque, conocido como *codificación de desarrollo,* codifica una expresión de desarrollo o programa de transformación. Dicha expresión de desarrollo define una sucesión de transformaciones que se aplican sobre un circuito básico, denominado embrión, que permite dar lugar a un circuito final.

Por otro lado, el enfoque denominado como *codificación indirecta*, permite codificar parte de los componentes de forma directa, mientras que utiliza el concepto de fortaleza de conexiones entre dichos componentes para codificar, de forma indirecta, las resistencias.

Adicionalmente, dado que un circuito electrónico puede verse como un grafo cerrado en el que los arcos corresponden a componentes y los nodos del grafo corresponden a los nodos del circuito, nos encontramos con el enfoque de *codificación basada en grafo*, que se basa en codificar directamente el grafo que representa el circuito.

Finalmente, en el caso particular de que se abordara únicamente el dimensionamiento de un circuito mediante un AE, sólo sería necesario codificar los valores de los componentes de un circuito cuya topología se habría seleccionado previamente, dando lugar a la denominada *codificación de parámetros de dimensionamiento.*

En resumen, los esquemas de representación de circuitos electrónicos para su codificación en un cromosoma utilizados en la literatura se pueden clasificar de la siguiente manera:

1. Codificación directa: este esquema codifica los componentes del circuito, sus valores y sus conexiones en un cromosoma que suele ser lineal. Su transformación a una *netlist* es directa.

2. Codificación de desarrollo: este esquema codifica un programa que define una sucesión de transformaciones, tales como la incorporación o borrado de diferentes tipos de componentes en el circuito, que permiten el desarrollo desde un circuito inicial o embrión, hasta producir un circuito final. Su transforma-

ción final en una *netlist* requiere la ejecución del programa codificado en el cromosoma.

3. Codificación indirecta: este esquema codifica sólo aquellos componentes que no son resistencias, como son los transistores y condensadores, y codifica de forma indirecta las conexiones entre los mismos utilizando el concepto de fortaleza de las conexiones. La fortaleza de la conexión entre dos dispositivos dará finalmente lugar a una resistencia conectada entre ambos.
4. Codificación basada en grafo: este esquema se basa en la representación de un circuito como grafo cerrado y suele codificar en el cromosoma una matriz de conexiones entre los nodos del circuito.
5. Codificación de parámetros de dimensionamiento: este esquema se utiliza en el caso de realizar sólo el dimensionamiento de un circuito para el que ya se ha determinado previamente su topología.

2.3.2. Operadores de variación y tratamiento de circuitos inviables

Otro problema que surge al abordar la generación de circuitos mediante un AE es cómo definir operadores de variación que permitan modificar un circuito válido para obtener un nuevo circuito candidato que también sea válido. A lo largo de este capítulo, consideraremos *circuitos válidos* a aquellos circuitos que se puedan simular sin error, mientras que serán *inviables* o *no válidos* en caso contrario. En el capítulo 3, se refinará esta definición (véase la sección 3.2.3).

Una primera aproximación para abordar este problema es evitar la aparición de circuitos no válidos en el algoritmo, lo que se puede conseguir mediante el uso de operadores de variación que garanticen la viabilidad de los circuitos resultantes, es decir, dichos operadores deberán actuar sobre cromosomas que representen circuitos válidos debiendo dar lugar siempre a circuitos válidos. Para este fin, en la literatura aparecen propuestas que se basan en el uso de funciones de transformación que operan sobre un circuito válido dando lugar a otro circuito válido. Estas funciones se pueden incluir, por ejemplo, en el operador de mutación (Grimbleby, 2000; Sapargaliyev & Kalganova, 2012), aunque la validez del circuito también podría estar garantizada en algunos casos de representación basada en codificación de desarrollo (Lohn & Colombano, 1999; Koza et al., 1999a).

Alternativamente, existe la opción de permitir que el operador de variación pueda generar circuitos inviables o no válidos y dejar que dichos circuitos desaparezcan por la propia presión selectiva asignando, por ejemplo, un valor de adaptación pésimo a este tipo de circuitos. En estos casos, no se limitan las transformaciones posibles y se permite el uso de operadores de variación estándar (Zebulum et al., 1998a; Zebulum et al., 1998b; Zebulum et al., 1999; Zebulum et al., 2000; Ando & Iba, 2000; Aggarwal, 2003; Tlelo-Cuautle & Duarte-Villaseñor, 2008; Wang et al., 2008).

De esta manera, los enfoques existentes en la literatura para garantizar la viabilidad de los nuevos circuitos obtenidos durante el proceso evolutivo son los siguientes:

1. Uso de operadores de variación que garantizan la viabilidad del nuevo circuito. Estos operadores permiten transformar un circuito válido en un nuevo circuito válido, como por ejemplo, cambiando un componente por otro, o conectando un nuevo componente en paralelo con uno existente.
2. Uso de operadores de variación que no garantizan la viabilidad del nuevo circuito. En este caso, se permite la aparición de circuitos no válidos. Este enfoque requiere de algún mecanismo adicional para la eliminación de estos circuitos mediante presión selectiva, si bien, puede utilizarse algún otro mecanismo para reparar los circuitos inviables generados.

Uno de los problemas que suele estar asociado a la aparición de un circuito inviable es la aparición de componentes con terminales no conectados. A continuación, se muestran algunas de las medidas adoptadas en la literatura para abordar este tipo de problemas:

1. Eliminar directamente el componente con terminales no conectados (Wang et al., 2007; Koza et al., 1999a).
2. Conectar el terminal no conectado a un nodo del circuito predeterminado, como por ejemplo, masa o alimentación del circuito (Lohn & Colombano, 1999; Slezák & Petržela, 2014).
3. Conectar el terminal no conectado a un nodo elegido aleatoriamente del circuito (Sapargaliyev, 2011).
4. Insertar una resistencia de alto valor entre el terminal no conectado y masa (Mattiussi & Floreano, 2007).

Finalmente, también es muy habitual la eliminación de aquellos componentes que tengan sus terminales cortocircuitados, dado que no aportan nada a la funcionalidad

del circuito (Sripramong & Toumazou, 2002).

2.3.3. Evaluación de los circuitos

La evaluación de los circuitos se puede realizar mediante diferentes tipos de análisis, dependiendo normalmente de la funcionalidad asociada al circuito. Por ejemplo, en el caso de filtros se suele utilizar el análisis en frecuencia, midiendo la respuesta del filtro en un número de frecuencias y acumulando la desviación obtenida en cada una respecto de la deseada. En otros circuitos se suele utilizar un análisis en continua, evaluando la salida obtenida mediante un barrido de valores de voltaje de entrada (Sapargaliyev & Kalganova, 2012; Mattiussi & Floreano, 2007). En otros casos, la evaluación se realiza en el dominio del tiempo, utilizando como especificación la respuesta a una señal de escalón unidad (Grimbleby, 2000), o mediante una rampa de voltaje. En este último caso, existe evidencia en la literatura relacionada de que el uso de un análisis en el dominio del tiempo parece producir circuitos más robustos que los obtenidos mediante análisis en condiciones estacionarias (Sapargaliyev & Kalganova, 2012; Mydlowec & Koza, 2000).

Las funciones de adaptación a optimizar se diseñan en general para cumplir la funcionalidad del circuito mediante un único objetivo, sin embargo, también aparecen trabajos que plantean un enfoque multiobjetivo, bien mediante una función de adaptación con una suma ponderada de subobjetivos (Zebulum et al., 1998b; Trefzer, 2006; Castejón & Carmona, 2013), o bien, mediante un AE basado en el concepto de dominancia de Pareto, como NSGA-II (McConaghy et al., 2007).

2.3.4. Enfoque abierto frente a confiabilidad de los circuitos

En la industria electrónica nos encontramos con que la introducción de una nueva topología sobre la que no hay experiencia previa aumenta significativamente la posibilidad de encontrar un error en una fase avanzada de la fabricación y tener que volver a la fase de diseño inicial, lo que explica que la adopción de una nueva herramienta de diseño puede ser una propuesta arriesgada y costosa (McConaghy & Gielen, 2006b). Por este motivo, aparece el concepto de confiabilidad. La confiabilidad es la propiedad de un circuito que le permite ser comprendido completamente por un diseñador (Sapargaliyev, 2011). De una manera más práctica, también se ha dicho que una topología determinada se considera confiable si se ha diseñado por

diseñadores expertos y se ha fabricado y verificado en múltiples implementaciones (McConaghy & Gielen, 2006b). Dado el alto coste en el que se puede incurrir por una vuelta a la fase de diseño, también se ha dicho que un diseño se considera confiable si el diseñador siente suficiente confianza para proceder a su implementación en silicio (McConaghy et al., 2009).

En este sentido, la confiabilidad choca con los diseños provistos por enfoques evolutivos ya que pueden obtener soluciones no convencionales. Los diseños convencionales pueden estar limitados por la propia experiencia del diseñador o por otras preferencias geométricas humanas de diseño como puede ser la simetría. En cambio, los AE, al estar dirigidos sólo por su nivel de adaptación, pueden encontrar soluciones fuera de las limitaciones del pensamiento humano (Eiben & Smith, 2003). Un AE recibe un juego de componentes que se pueden conectar arbitrariamente, sin reglas ni conocimiento experto, donde los bloques son inventados o reinventados desde cero y esto es lo que le da su naturaleza de diseño abierta (McConaghy et al., 2009). El enfoque abierto u *open-ended*, además de encontrar soluciones no convencionales, también consigue diseños altamente compactos (Sapargaliyev & Kalganova, 2012), y consigue resultados competitivos con los resultados creados por diseñadores humanos (Koza et al., 1999a) siendo, en algunos casos, comparables a circuitos patentados previamente por un diseñador humano (Koza et al., 2003; Koza et al., 2008).

Sin embargo, la mayoría de los diseñadores humanos han considerado los diseños obtenidos por AE como no confiables en muchos de los casos por tener una apariencia extraña o por carecer de una lógica aparente en los mismos (McConaghy et al., 2009). Posiblemente, éste puede ser el problema más serio que se está encontrando el enfoque abierto en electrónica evolutiva: el paso de diseños experimentales obtenidos evolutivamente a diseños industriales (Stoica et al., 2004), por ser etiquetados como soluciones EDA de «baja eficiencia» (Sorkhabi & Zhang, 2017).

Otra crítica al enfoque abierto se debe a que las topologías no convencionales pueden apoyarse en el uso de transistores operando en regiones que no se utilizan habitualmente en el diseño convencional, como el uso de polarización en inversa (Zebulum et al., 2000), por lo que el simulador de circuitos podría no ser preciso en estos casos (McConaghy & Gielen, 2006b). Siguiendo en esta línea, se observa que, a partir de 2013, los trabajos en síntesis de circuitos electrónicos analógicos utilizando un enfoque abierto se van haciendo menos numerosos (véase la tabla 2.1), mostrando que se centran más en resolver sólo el problema de dimensionamiento, partiendo de

topologías previamente diseñadas, o bien, se basan en el uso de librerías de soluciones confiables donde un AE podrá escoger bloques predefinidos para componer un circuito completo (McConaghy et al., 2009).

2.3.5. El problema de la escalabilidad

El problema de la escalabilidad ocurre al aumentar la complejidad de los problemas, de problemas pequeños a problemas medianos o grandes, y su efecto, desde el punto de vista evolutivo es, bien la necesidad de aumentar drásticamente el número de generaciones, o bien, que el AE no sea capaz de encontrar una solución, dando lugar también el efecto de estancamiento o *stalling effect* (Kalganova, 2000). El origen de este problema proviene del aumento del espacio de búsqueda debido a la explosión combinatoria de las soluciones y los elevados tiempos de simulación de circuitos grandes o muy grandes (Stoica et al., 2004).

Para atender este tema, todavía abierto, por un lado se ha explorado la vía de la codificación de desarrollo, sobre la que hay evidencias de que puede mejorar el problema de la escalabilidad (Gordon & Bentley, 2005) y, por otro lado, nos encontramos con lo que se conoce como *evolución incremental,* que permite abordar un circuito de mayor tamaño mediante la partición en varios subcircuitos más pequeños que deben ser abordados en orden y cuya composición final da lugar al circuito buscado (Sapargaliyev & Kalganova, 2012).

2.4. Trabajos previos en síntesis de circuitos electrónicos analógicos

En esta sección se comentan los trabajos previos más relevantes en la literatura dentro del ámbito de la electrónica evolutiva aplicada a los circuitos electrónicos analógicos. En la tabla 2.1 se muestran dichos trabajos indicando el nombre del autor, el tipo de representación utilizada en el cromosoma, el tipo de codificación, el uso o no de operadores de variación que preservan la viabilidad, el tipo de AE, el uso de cromosomas de longitud fija o variable, el tipo de circuitos sintetizados, el año de publicación y el simulador utilizado para la evaluación de los circuitos.

2.4.1. Codificación directa

En este tipo de representación, se codifican directamente los componentes del circuito y sus conexiones en un cromosoma lineal (véase la sección 2.3.1). En general, también es común almacenar los valores de los componentes en el cromosoma para, de esta forma, permitir la evolución conjunta de la topología y el dimensionamiento. Sin embargo, en algunos casos no se almacenan los valores de los componentes, realizando la tarea de dimensionamiento mediante un optimizador numérico, que puede aplicarse posteriormente a la determinación de la topología (Ando & Iba, 2000), o incorporarlo dentro de la evaluación de cada individuo (Grimbleby, 1995; Grimbleby, 2000). El optimizador puede ser también un AE (Ando & Iba, 2000).

En general, la longitud de los cromosomas suele ser variable con el fin de poder representar circuitos de diferentes tamaños. Sin embargo, también hay casos en los que se utilizan cromosomas de tamaño fijo, pero permitiendo representar circuitos de diferentes tamaños. En estos casos, la forma de permitir circuitos más pequeños que lo que permitiría el tamaño fijo del cromosoma se realiza bien mediante la posibilidad de utilizar componentes nulos (Grimbleby, 1995; Grimbleby, 2000), o bien mediante la activación o desactivación de genes, para lo que se usa una cadena de valores de activación/desactivación de los genes como parte del cromosoma (Zebulum et al., 1998a; Zebulum et al., 2000). Finalmente, también hay casos en los que se utiliza un cromosoma de tamaño fijo permitiendo un número fijo de componentes (Tlelo-Cuautle & Duarte-Villaseñor, 2008).

En los casos en los que se usa longitud variable, se observa la aparición del efecto engorde o *bloat* (Eiben & Smith, 2003), cuyo efecto es el aumento de tamaño de los cromosomas a medida que aumenta el número de generaciones. El efecto engorde tiene un impacto negativo en el rendimiento de un AE, incrementando el uso de memoria y el tiempo de ejecución, al considerar soluciones innecesariamente complejas (Ando & Iba, 2000; Sapargaliyev, 2011). Por este motivo, es necesario utilizar mecanismos, denominados de parsimonia, que limiten el crecimiento excesivo de los cromosomas y así obtener soluciones eficientes en el número de componentes de los circuitos sintetizados (Zebulum et al., 2000). Incluso en los casos de longitud de cromosoma fija que, como se ha indicado previamente, son realmente representaciones de longitud variable con un límite máximo, se menciona la introducción de mecanismos de parsimonia (Grimbleby, 1995).

El tipo de AE más habitualmente utilizado suele ser un algoritmo genético, o bien

una variación del mismo, que además puede haberse modificado para acoplar un optimizador numérico (Grimbleby, 2000; Ando & Iba, 2000), cuya función es realizar el dimensionamiento de cada circuito candidato. El tamaño de población utilizado suele ser bajo, entre 30 y 200 individuos, aunque también se han usado tamaños más grandes, entre 500 y 2 000 (Ando & Iba, 2000), llegando a 30 000 individuos en el caso de uso de una EE (Sapargaliyev & Kalganova, 2012).

En los trabajos iniciales en la literatura, los circuitos abordados fueron filtros pasivos (Grimbleby, 1995; Zebulum et al., 1998a; Ando & Iba, 2000; Grimbleby, 2000) y posteriormente se abordaron circuitos con componentes activos (Zebulum et al., 1999; Zebulum et al., 2000; Aggarwal, 2003; Trefzer, 2006; Mattiussi & Floreano, 2007; Sapargaliyev & Kalganova, 2012). Esta evolución está relacionada con la carga computacional que conlleva la simulación de los circuitos, que es la tarea más costosa computacionalmente en electrónica evolutiva. Los trabajos iniciales con filtros pasivos permiten su evaluación mediante análisis simbólico, que resulta menos costosa computacionalmente. Los trabajos posteriores con elementos activos fueron introduciendo simuladores de circuitos, siendo SPICE el más utilizado (véase la sección 2.2).

Aunque en la literatura predominan los trabajos que realizan evolución extrínseca, es decir, donde los circuitos son simulados para evaluar su grado de adaptación, también hay algunos trabajos que usan evolución intrínseca, es decir, donde los circuitos se evalúan usando hardware real. En este último caso se utilizan circuitos integrados programables como las FPGA (Thompson, 1997) o matrices de transistores programables (FPTA, por sus siglas en inglés) (Stoica et al., 2003; Trefzer, 2006). El caso de uso de una FPGA para diseño analógico resulta muy llamativo ya que, a pesar de ser una estructura hardware diseñada para uso digital, también se ha utilizado como base para diseño de un circuito analógico (Thompson, 1997), que además se ha puesto como ejemplo de solución no convencional (Sapargaliyev, 2011).

Finalmente, también hay un trabajo que utiliza evolución gramatical (Kunaver, 2020), para abordar el diseño de un oscilador y de filtros de segundo y tercer orden, con una función de adaptación multicriterio (Kunaver, 2020). Este trabajo es posterior al primer artículo que se publicó como consecuencia de esta tesis (Castejón & Carmona, 2018).

2.4.2. Codificación de desarrollo

En el caso de codificación de desarrollo, el cromosoma contiene un programa que define una sucesión de transformaciones que, partiendo de un circuito inicial o embrión, permiten producir un circuito final.

El enfoque basado en codificación de desarrollo fue introducido por John Koza y sus colaboradores en 1997 (Koza et al., 1997b; Koza et al., 1997a), utilizando PG como paradigma evolutivo. El conjunto de circuitos abordados mediante este enfoque fue creciendo a medida que nuevos trabajos eran publicados (Koza et al., 1998; Koza et al., 1999b; Koza et al., 2000a; Koza et al., 2000b; Koza et al., 2003; Mydlowec & Koza, 2000; Streeter et al., 2002), hasta finalmente abarcar un conjunto muy amplio de circuitos, con más de veinte circuitos de diferentes ámbitos como filtros, amplificadores, circuitos computacionales, controladores robóticos, sensor de temperatura y referencia de voltaje (Koza et al., 1999a).

El enfoque basado en codificación de desarrollo también ha sido utilizado en otros trabajos, como (Lohn & Colombano, 1999), donde se define un lenguaje de instrucciones de construcción de circuitos con el que se componen programas que permiten diseñar circuitos determinados. En este caso los cromosomas son cadenas lineales de dichas instrucciones, representadas por valores numéricos (*bytecode*), el AE utilizado es un AG, y los circuitos abordados corresponden a filtros y amplificadores. Trabajos posteriores en la literatura han utilizado el enfoque de Lohn (Sripramong & Toumazou, 2002).

El simulador utilizado en los trabajos que utilizan el enfoque de desarrollo es generalmente SPICE con la excepción de (Hu et al., 2005), donde se utiliza un análisis simbólico basado en MATLAB.

Adicionalmente, cabe mencionar también un trabajo que maneja el enfoque de expresiones de desarrollo de Koza, pero mediante el uso de GE (Castejón & Carmona, 2013). El circuito sintetizado fue un amplificador de una etapa donde se utilizó un embrión con un transistor ya dispuesto inicialmente en la configuración de emisor común.

Existe evidencia de que la codificación de desarrollo permite una descripción más compacta de un circuito (Mattiussi & Floreano, 2007; Sapargaliyev, 2011) y se han obtenido evidencias de un mejor comportamiento en relación con el problema de la escalabilidad (Gordon & Bentley, 2005).

2.4.3. Codificación indirecta

La característica principal de la codificación indirecta es el uso de una codificación implícita de las conexiones entre los dispositivos. En esta aproximación, la interconexión entre dispositivos se codifica de forma indirecta mediante el concepto de fortaleza de las conexiones, que finalmente dará lugar a la introducción de resistencias que conectan componentes. Como casos particulares, nos encontramos los valores extremos de las resistencias como, por ejemplo, el valor cero, que se correspondería con una conexión directa entre dichos componentes, o bien, el valor infinito, que se correspondería con un circuito abierto o una falta de conexión entre dichos componentes.

Esta forma de codificación de un circuito presenta como ventaja el que permite reducir el número de elementos que se codifican en el cromosoma, ya que las resistencias no necesitan codificarse de forma explícita (Floreano & Mattiussi, 2008). Los trabajos principales que utilizan este enfoque son (Mattiussi, 2005; Mattiussi & Floreano, 2007) y se describen con más detalles en la sección 2.4.6.

2.4.4. Codificación basada en grafo

La codificación basada en grafo se basa en la consideración de un circuito electrónico como un grafo cerrado, donde los arcos del grafo corresponden a los componentes del circuito y los nodos del grafo corresponden a los nodos del circuito. En este enfoque se suele codificar una matriz de conexiones entre componentes y, además, suele utilizar alguna variante de evolución diferencial (ED) (Rojec et al., 2019), o bien, otros algoritmos derivados de los AE, como un sistema inmune artificial (AIS, por sus siglas en inglés) (Gan et al., 2010) o un algoritmo de estimación de la distribución (AED) (Slezák & Petržela, 2014). El simulador utilizado en estos trabajos es SPICE o alguna variante del mismo y los circuitos abordados fueron filtros, tanto pasivos como activos, un circuito de raíz cúbica y otro de referencia de voltaje.

2.4.5. Codificación de parámetros de dimensionamiento

Este enfoque es un caso particular en el que sólo se realiza la codificación de los parámetros de dimensionamiento de un circuito para el que ya se ha determina-

do previamente su topología. En este caso, sólo es necesario codificar un vector de valores de los parámetros de los componentes. El AE más utilizado es ED, con la excepción de AG (Zebulum et al., 1998b) y NSGA-II (Povoa et al., 2016). Los circuitos abordados con este enfoque fueron amplificadores de transconductancia, un transportador de corriente y bloques de un receptor de radiofrecuencia. Los simuladores empleados en los trabajos mencionados fueron Smash y variantes de SPICE, como WINSPICE (Sabat et al., 2009), con la excepción de (El Dor et al., 2016), donde se utiliza análisis simbólico.

2.4.6. Trabajos de especial interés

En esta sección se describen tres trabajos de especial interés pues sus resultados en la síntesis de circuitos son considerados circuitos de referencia o *benchmark* que nos permitirán realizar comparativas con los resultados obtenidos por los algoritmos propuestos en esta tesis. En primer lugar, se describirá la aproximación de Koza basada en PG (Koza et al., 1999a). En segundo lugar, se describirán la aproximación basada en AGE (Mattiussi & Floreano, 2007). Finalmente, se describirá la aproximación basada en una EE (Sapargaliyev & Kalganova, 2012).

Aproximación de Koza basada en programación genética

La aproximación del grupo de Koza (Koza et al., 1997b; Koza et al., 1999a) tiene un especial interés por haber logrado la síntesis de un conjunto muy amplio de circuitos. El impacto del trabajo de Koza y sus colaboradores en este campo ha hecho que sus resultados se tomen como circuitos de referencia en trabajos posteriores (Mattiussi & Floreano, 2007; Sapargaliyev & Kalganova, 2012).

Como se ha indicado en la sección 2.4.2, la aproximación de Koza introduce el uso de PG para la síntesis de circuitos, junto con la obtención de resultados mejores que los de otros trabajos anteriores. De hecho consiguieron algunos resultados comparables a los realizados por diseñadores humanos, redescubrieron estructuras conocidas, como el filtro en escalera LC o el par de transistores Darlington, diseñaron circuitos de dificultad reconocida (Koza et al., 1999a), y replicaron la funcionalidad de circuitos ya patentados (Koza et al., 2003; Koza et al., 2008).

En el enfoque de Koza *et al.* se introduce la codificación de desarrollo como forma de representar circuitos electrónicos, que son grafos cerrados, en árboles de parsea-

do (Koza et al., 1999a). Un árbol de parseado contiene una expresión de desarrollo formada por una sucesión de funciones de transformación que se aplica sobre un circuito básico de partida, llamado embrión, en el que existe un conjunto de conexiones iniciales elegidas por el usuario. Tanto las conexiones iniciales como los componentes o nuevas conexiones que se van añadiendo son modificables y, por tanto, pueden seguir transformándose según la función de transformación que se va aplicando en cada momento. El circuito final así obtenido se inserta en la parte fija (véase la figura 1.1) que, a diferencia del embrión, no se modifica por las funciones de transformación. En la figura 2.1 (a) se puede ver un ejemplo de embrión que consta de dos conexiones modificables.

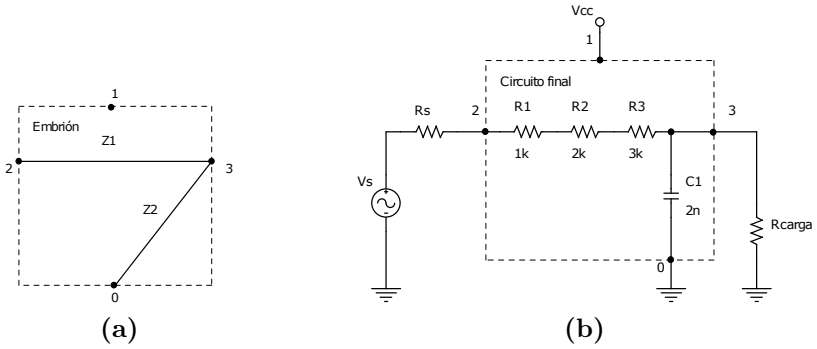


Figura 2.1. – Ejemplo de embrión con dos conexiones modificables Z_1 y Z_2 (a) y circuito final obtenido (b) en el enfoque de desarrollo de Koza.

Las funciones de transformación utilizadas en este enfoque se dividen en varios tipos:

1. Funciones de creación de componentes: permiten la creación nuevos componentes en el circuito. Por ejemplo, las funciones R y C crean una resistencia y un condensador, respectivamente, a partir de un hilo modificable. Existen también funciones de creación de componentes de tres terminales, como transistores, que son muy numerosas debido al elevado número de formas posibles de conexión de estos componentes.

2. Funciones de modificación de la topología: modifican la topología mediante conexiones en serie o paralelo, como realizan, por ejemplo, las funciones SERIES y PARALLEL0, respectivamente. También se dispone de funciones más complejas que crean, por ejemplo, conexiones a tierra, a alimentación o a otro nodo del circuito, tal como hacen las funciones THREE_GROUND, THREE_VCC o THREE_VIA, respectivamente.

3. Funciones de control del desarrollo: permiten terminar el desarrollo de un hilo modificable como, por ejemplo, la función END, o bien, eliminan un hilo modificable como, por ejemplo, la función CUT.
4. Funciones aritméticas para el cálculo de valores de los componentes: realizan cálculos aritméticos dirigidos a determinar el valor de un componente. Adicionalmente, se utiliza una función de escalado para obtener los valores finales de los componentes.
5. Funciones automáticamente definidas: permiten la reutilización de un subcircuito creado durante el proceso evolutivo como, por ejemplo, ADF0 o ADF1.

El repertorio completo consta de 40 funciones de creación de componentes y de 26 funciones de modificación de la topología y de control del desarrollo. Dicho repertorio permite la creación de cualquier tipo posible de topología de circuito (Koza et al., 1999a). Es importante indicar que las funciones de transformación pueden tener distinto número de parámetros o aridad. Este hecho hace necesario que los operadores de variación estén diseñados para asegurar que no alteran el número de parámetros de las funciones, con el objetivo de preservar la propiedad de clausura (véase la sección 3.1.1).

A continuación, se indica un ejemplo de expresión de desarrollo de este tipo de aproximación, donde se han simplificado las expresiones de cálculo de los valores de los componentes para mayor sencillez. La figura 2.1 (b) muestra el circuito final correspondiente a la ejecución de esta expresión:

LIST (SERIES(R (1.0e3)(END))(R (2.0e3)(END))(R (3.e3)(END)))(C (2.0e-9)(END))

La aproximación de Koza utiliza el simulador SPICE para evaluar el grado de adaptación de cada circuito y se basa en una implementación paralela del algoritmo para su distribución en un clúster de 64 procesadores, donde utiliza poblaciones de 10 000 individuos en cada procesador.

Aproximación de Mattiussi & Floreano basada en codificación genética analógica (AGE)

La codificación AGE (Mattiussi & Floreano, 2007) está inspirada en las redes de regulación genética (GRN, por sus siglas en inglés) biológicas, en las que los genes disponen de zonas de regulación que reciben mensajes procedentes de otras zonas

para regular su expresión: activación o desactivación. Un cromosoma en AGE codifica: (i) un número variable de dispositivos, que en electrónica analógica corresponden a transistores y condensadores; y (ii) el valor de la denominada fortaleza de las conexiones entre los mismos. Este valor de fortaleza entre conexiones de dispositivos se puede considerar como una conductancia por lo que su transformación en una resistencia que conecta ambos dispositivos es inmediata. Dado que sólo los transistores y condensadores se encuentran codificados de forma directa y, las resistencias, de forma indirecta, el modelo de codificación corresponde al denominado como codificación indirecta (véase la sección 2.4.3).

El cromosoma en AGE se compone de una cadena de caracteres alfabéticos de longitud variable. Dentro de la misma, se codifican los dispositivos que comienzan con un *token* predefinido que codifica el tipo de componente. Por ejemplo: "PBJT", "NBJT" y "C" codifican transistores PNP, transistores NPN y condensadores, respectivamente. A continuación, se codifican las denominadas cadenas de conexión, delimitadas por un *token* "TERM" y, finalmente, la cadena de parámetro delimitada por un *token* "PARM". La decodificación de un cromosoma conlleva un primer paso de extracción de dispositivos. Este paso consiste en leer la cadena de izquierda a derecha buscando los *tokens* anteriormente indicados y extrayendo la información completa de un dispositivo que comprende: un tipo de dispositivo, las cadenas de conexión para dos o tres terminales según el tipo de componente y, finalmente, una cadena de parámetro que codifica el valor del componente en su caso.

Una vez extraídos los componentes, las cadenas de conexión se convierten a conexiones finales dentro del circuito mediante un proceso que compara cada cadena de conexión contra todas las demás cadenas de conexión. El parecido entre ambas cadenas determinará la fortaleza de cada conexión. Este grado de parecido se realiza con una función de distancia, denominada alineamiento global, que se basa en buscar una secuencia de operaciones de sustitución, inserción y borrado de caracteres, que permita transformar una cadena de conexión en la otra (Mattiussi, 2005). El método de alineamiento global permite construir el denominado mapa de interacción de dispositivos. El mencionado mapa es una función que permite determinar un valor de conductancia a partir de dos cadenas de conexión correspondientes a los terminales de dos dispositivos. Esta conductancia corresponderá a una resistencia que interconectará ambos terminales de los dispositivos considerados. En caso de que el parecido entre cadenas de conexión sea muy bajo, existe un valor umbral por debajo del cual no existirá una resistencia de conexión y ambos dispositivos no

estarán conectados.

Finalmente, la conexión con los terminales de la parte fija se realiza mediante la definición de *tokens* de puerto entrada/salida, como por ejemplo “IOPTA”, “IOPTB”, etc. Es necesario que existan tantos tokens de conexión como terminales tenga disponibles la parte fija. Para estos puertos de conexión, también se evolucionarán sus cadenas de conexión correspondientes que permitirán la conexión con los terminales de los dispositivos que forman el circuito evolucionado.

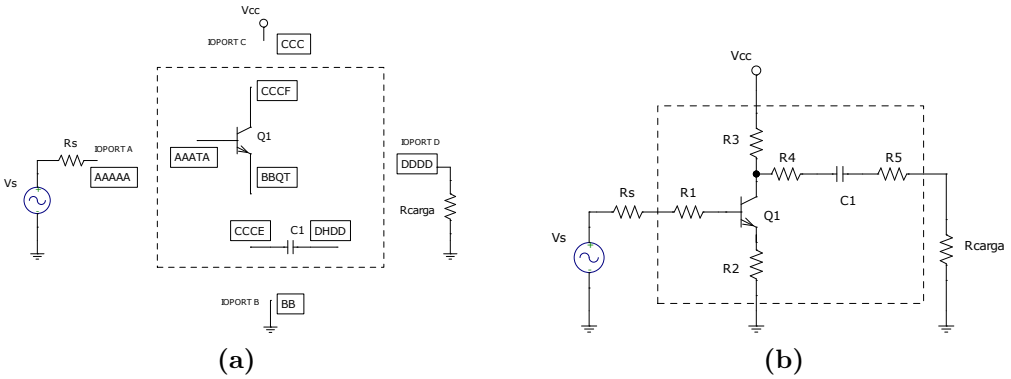


Figura 2.2. – Ejemplo de decodificación de un cromosoma codificado en AGE, mostrando la extracción de dispositivos (a) y el circuito final obtenido (b).

A continuación, se indica un ejemplo de un cromosoma perteneciente a este tipo de aproximación:

> NBJTCCFTERMAAATATERMBBQTTERMHHPARMZZZZZZZZ
> CCCCETERMDHDDTERMGGGPARMZZZZIOPTAAAAAATERM
> IOPTBBBTERMIOPTCCCCTERMIOPTDDDDDTERM

La figura 2.2 (a) muestra la extracción de dispositivos de dicho cromosoma y la figura 2.2 (b) muestra el circuito final obtenido.

Los circuitos sintetizados con esta aproximación corresponden a circuitos *benchmark* propuestos por Koza, comprendiendo un circuito de referencia de voltaje, un circuito sensor de temperatura y un generador de función gaussiana. La evaluación de circuitos se hace utilizando el simulador SPICE.

Aproximación de Sapargaliyev & Kaganova basada en estrategias evolutivas

En esta aproximación se recalca el uso de evolución abierta (*open-ended*), con el que se busca encontrar soluciones no convencionales a los circuitos objetivo (Sapar-

galiyev, 2011; Sapargaliyev & Kalganova, 2012). Frente a las críticas de falta de confiabilidad de los circuitos sintetizados mediante un AE (McConaghy & Gielen, 2006b), se ofrece un enfoque no convencional para el que se indica que se puede obtener robustez sin la confiabilidad buscada por diseñadores humanos (Sapargaliyev, 2011).

La aproximación de Sapargaliyev & Kalganova se basa en el uso de una EE para la síntesis de circuitos. En este enfoque, se omite el uso del operador de cruce, basando toda la evolución en el uso del operador de mutación. Adicionalmente, se utiliza una codificación directa, por lo que el cromosoma se compone de genes donde cada uno codifica un único componente. Cada componente, a su vez, comprende un tipo de componente, dos o tres terminales y un parámetro, para un total de cuatro elementos por componente o gen. Dado que se utiliza el mecanismo de codificación directa, la transformación de un cromosoma en una *netlist* es directa.

El operador de mutación utilizado en este enfoque es un operador *ad hoc* basado en el enfoque de operadores de variación que conservan la viabilidad del nuevo circuito generado. El operador de mutación puede actuar de diferentes formas:

1. Mutación de estructura del circuito: modifican el circuito existente sin incrementar o disminuir su tamaño. En caso de que se produzcan terminales no conectados, dichos terminales se conectan a otro nodo del circuito elegido de forma aleatoria.
    a) Cambio de tipo de componente. Esta transformación cambia un componente por otro tipo de componente. Se tiene en cuenta el número de terminales del componente inicial y del nuevo componente.
    b) Reglas de mutación de conexión de nodos. Se elige un terminal de un componente aleatoriamente y se conecta a cualquier otro nodo.
    c) Mutación de parámetros: modifica los valores de los componentes.
2. Mutación por adición de nuevo elemento: elige un tipo de componente aleatoriamente y lo conecta al circuito. Se eligen de forma aleatoria, tanto los nodos de conexión, como el valor del parámetro, en su caso.
3. Mutación por borrado de componente: elige aleatoriamente un componente existente y lo borra del cromosoma. En caso de que aparezcan terminales no conectados, se resuelven mediante conexión a otro nodo del circuito elegido de forma aleatoria.

4. Mutación por reutilización de estructuras: este tipo de mutación permite la reutilización de un subcircuito creado durante el proceso evolutivo.

Los circuitos sintetizados en esta aproximación corresponden a cuatro circuitos computacionales *benchmark* propuestos por Koza y que implementan las siguientes funciones matemáticas: potencia al cuadrado, raíz cuadrada, potencia al cubo y raíz cúbica. La evaluación de circuitos en el AE se realiza utilizando el simulador SPICE.

2.5. Trabajos previos en síntesis de circuitos electrónicos digitales

Aunque el ámbito de esta tesis son los circuitos electrónicos analógicos, por completitud, se ha considerado incluir también una breve descripción de los trabajos previos más relevantes en la síntesis de circuitos electrónicos digitales mediante electrónica evolutiva. En esta sección, se describen brevemente dichos trabajos y, a modo de resumen, se muestran también agrupados y resumidos en la tabla 2.2.

En el diseño de circuitos digitales, la representación más habitual corresponde a una codificación matricial de las puertas lógicas, siguiendo la estructura de una FPGA. Esta representación fue introducida en (Miller et al., 1997) y, posteriormente, se generalizó con el nombre de programación genética cartesiana (CGP, por sus siglas en inglés) en (Miller & Thomson, 2000) como alternativa a PG, donde se utilizan matrices en lugar de árboles de parseado como cromosomas. Esta representación matricial, bien como CGP (Kazarlis et al., 2015; Vasicek & Sekanina, 2016), o bien en forma similar (Coello Coello & Aguirre, 2002), se ha utilizado con éxito en la síntesis de circuitos digitales combinacionales. En el caso de CGP, las matrices de puertas lógicas se convierten posteriormente a cadenas lineales de enteros, por lo que el AE utilizado posteriormente puede ser un AG o una EE (Miller et al., 1997).

Alternativamente a la representación matricial, aparece la representación como suma de productos lógicos de las variables de entrada de un circuito combinacional. Dicho cromosoma permite codificar funciones booleanas que posteriormente se pueden convertir a un circuito (Zebulum et al., 2000).

Finalmente, una tercera aproximación a la síntesis de circuitos digitales sería mediante la generación directa de código en un lenguaje descriptivo del hardware como

Verilog. El código generado describiría un circuito que se podría implementar posteriormente en un ASIC. En este caso el AE utilizado está basado en EG (Karpuzcu, 2005).

Las aproximaciones anteriores se han utilizado en el ámbito de la síntesis de circuitos combinacionales, donde las entradas vienen indicadas en forma de una tabla de verdad y se pueden sintetizar utilizando puertas lógicas únicamente. Los circuitos secuenciales se describen mediante una tabla de estados y transiciones y, se abordan mediante una partición en dos circuitos: uno secuencial y otro combinacional puro. La parte secuencial gestiona los estados y sus transiciones y la parte combinacional traduce los estados a las salidas correspondientes. Si no se dispone de la tabla de estados, se puede partir de una descripción de nivel superior y obtener la máquina de estados mediante un AE como, por ejemplo, EG (Colmenar et al., 2013). Si se dispone de la tabla de estados, la síntesis de la parte secuencial conlleva la asignación de los estados a valores, cuya asignación óptima permite reducir el volumen, tanto de la parte combinacional como de la secuencial. Así, las dos partes: secuencial y combinacional se pueden resolver de forma conjunta, en dos etapas mediante un AG (Ali et al., 2004), o bien, de forma independiente, abordando la parte secuencial mediante un algoritmo multiobjetivo (Al Jassani et al., 2011) o, ya fuera del ámbito de los AE, mediante un algoritmo basado en un enjambre de partículas binario (BPSO, por sus siglas en inglés) (El-Maleh et al., 2013).

Atendiendo a la evaluación de los circuitos, los trabajos anteriormente mencionados utilizaban evolución extrínseca, usando simuladores de FPGA (Miller et al., 1997; Kalganova et al., 1998; Coello Coello & Aguirre, 2002; Yan et al., 2006), si bien, también es posible realizar la evolución de forma intrínseca (Lanchares et al., 2013; Lanchares et al., 2014). Además de los simuladores de FPGA, también se encuentran simuladores de lenguajes descriptivos del hardware como Verilog (Karpuzcu, 2005) o VHDL (Karpuzcu, 2005; Hounsell & Arslan, 2000).

También se han realizado propuestas para eliminar el uso de simuladores, utilizando equivalencia funcional o soluciones aproximadas para circuitos muy grandes, caso de los diagramas de decisión binarios (BDD, por sus siglas en inglés) (Vasicek & Sekanina, 2016; Vasicek & Sekanina, 2011). El caso de BDD se ha visto como una alternativa válida para abordar con éxito el problema de la escalabilidad.

Los circuitos combinacionales diseñados suelen ser multiplexores o circuitos de paridad, siendo los sumadores y multiplicadores de 1 o 2 bits los más habituales. Los

trabajos más modernos llegan hasta 16 bits (Mrazek & Vasicek, 2018) o abordan circuitos de más de 50 salidas (Vasicek & Sekanina, 2016). En la síntesis de circuitos secuenciales se han propuesto diseños de contadores y detectores de secuencias de hasta 4 bits (Ali et al., 2004) y, posteriormente, circuitos *benchmark* de hasta 19 salidas y 48 estados (Al Jassani et al., 2011; El-Maleh et al., 2013).

Tabla 2.1. – Trabajos previos en diseño automático de circuitos analógicos.

Referencia	Representación	Tipo de codificación	Se conserva la viabilidad	AE	Longitud del cromosoma	Multiobjetivo	Simulador	Circuitos sintetizados
(Grimbleby, 1995)	Lista de componentes y conexiones	Directa	Sí	AG	Fija	No	Análisis simbólico	Filtros
(Koza et al., 1997b)	Árboles de parseado	Desarrollo	Sí	PG	Variable	No	SPICE	Circuitos computacionales
(Thompson, 1997)	Cadena binaria	Directa	No	AG	Fija	No	Intrínseco	Discriminador de tono
(Zebulum et al., 1998a)	Lista de componentes y conexiones	Directa	No	SAGA	Fija y variable	No	Smash	Filtros
(Zebulum et al., 1998b)	Vector de parámetros	Dimensionamiento	No	AG	Fija	Sí	Smash	Amplificador de transconductancia de Miller
(Koza et al., 1999a)	Árboles de parseado	Desarrollo	Sí	PG	Variable	No	SPICE	Más de 20 circuitos

Tabla 2.1. – Trabajos previos en el diseño automático de circuitos analógicos (continuación).

Referencia	**Representación**	**Tipo de codificación**	**Se conserva la viabilidad**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Circuitos sintetizados**
(Lohn & Colombano, 1999)	cadena de *bytecode*	Desarrollo	Sí	AG	Variable	No	SPICE	Filtros y amplificadores
(Zebulum et al., 1999)	Lista de componentes y conexiones	Directa	No	AG	Fija	Sí	Smash, SPICE	Filtro activo
(Ando & Iba, 2000)	Lista de componentes y conexiones	Directa	No	*Messy* GA	Variable	No	No indicado	Filtros
(Grimbleby, 2000)	Lista de componentes y conexiones	Directa	Sí	*Hybrid* GA	Fija	No	Análisis simbólico	Filtros
(Mydlowec & Koza, 2000)	Arbol de parseado	Desarrollo	Sí	PG	Variable	No	SPICE	Circuitos computacionales
(Zebulum et al., 2000)	Lista de componentes y conexiones	Directa	No	SAGA	Variable	Sí	Smash	Filtros y amplificadores

Tabla 2.1. – Trabajos previos en el diseño automático de circuitos analógicos (continuación).

Referencia	**Representación**	**Tipo de codificación**	**Se conserva la viabilidad**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Circuitos sintetizados**
(Sripramong & Toumazou, 2002)	Arbol de parseado	Desarrollo	Sí	PG	Variable	Sí	SPICE	Amplificadores
(Streeter et al., 2002)	Arbol de parseado	Desarrollo	Sí	PG	Variable	No	SPICE	Circuitos computacionales
(Aggarwal, 2003)	Lista de componentes y conexiones	Directa	No	AG	Variable	No	Análisis simbólico	Oscilador sinusoidal
(Hu et al., 2005)	Arbol de parseado	Desarrollo	Sí	PG	Variable	No	Matlab	Filtros
(Trefzer, 2006)	Celdas de una FPTA	Directa	Sí	GA y *turtle* GA	Variable	Sí	Intrínseco	Amplificador
(Mattiussi & Floreano, 2007)	AGE	Directa	No aplicable	GRN	Variable	No	SPICE	Referencia de voltaje, sensor de temperatura, función gaussiana

Tabla 2.1. – Trabajos previos en el diseño automático de circuitos analógicos (continuación).

Referencia	**Representación**	**Tipo de codificación**	**Se conserva la viabilidad**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Circuitos sintetizados**
(McConaghy et al., 2007)	Lista de bloques, conexiones y valores	Directa	No aplicable	NSGA-II	No se indica	Sí	HSPICE	Amplificador operacional
(Tlelo-Cuautle & Duarte-Villaseñor, 2008)	Cada gen determina un tipo de bloque predefinido	Directa	No	AG	Fija	No	T-SPICE	Seguidor de voltaje
(Wang et al., 2008)	Dos niveles: bloques y componentes	Directa	No	TLGP	Variable	No	SPICE	Filtros y amplificadores
(Sabat et al., 2009)	Vector de parámetros	Dimensionamiento	No	DE	Fija	No	WINSPICE	Amplificador de transconductancia de Miller
(Gan et al., 2010)	Grafo	Grafo	Sí	AIS	Variable	No	SPICE	Filtros pasivos
(Kim et al., 2010)	No se indica	Directa	Sí	EE	No se indica	No	WINSPICE	Filtro

Tabla 2.1. – Trabajos previos en el diseño automático de circuitos analógicos (continuación).

Referencia	Representación	Tipo de codificación	Se conserva la viabilidad	AE	Longitud del cromosoma	Multiobjetivo	Simulador	Circuitos sintetizados
(Yuan & He, 2010)	cadena de *bytecode*	Desarrollo	Sí	VDE	Variable	No	SPICE	Amplificador con realimentación
(Kim & Cho, 2012)	Lista de componentes y conexiones	Directa	Sí	EE	Variable	No	WINSPICE	Filtros
(Sapargaliyev & Kalganova, 2012)	Lista de componentes y conexiones	Directa	Sí	EE	Variable	No	PSPICE	Circuitos computacionales
(Castejón & Carmona, 2013)	Cadena binaria	Desarrollo	Sí	GE	Variable	No	NGSpice	Amplificadores
(Slezák & Petržela, 2014)	Grafo e hipergrafo	Grafo	No	GhEDA	Fija	No	NGSpice	Función raíz cúbica
(El Dor et al., 2016)	Vector de parámetros	Dimensionamiento	No	MODE	Fija	Sí	Análisis simbólico	Transportador de corriente
(Povoa et al., 2016)	Vector de parámetros	Dimensionamiento	No	NSGA-II	Fija	Sí	HSPICE/RF	Bloques receptor RF

Tabla 2.1. – Trabajos previos en el diseño automático de circuitos analógicos (continuación).

Referencia	**Representación**	**Tipo de codificación**	**Se conserva la viabilidad**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Circuitos sintetizados**
(Rojec et al., 2019)	Matriz de conexiones y vector de parámetros	Grafo	No	PSADE y NSGA-II	Fija	Sí	SPICE	Filtro activo y referencia de voltaje
(Kunaver, 2020)	EG	Directa	No	AG	Variable	Sí	SPICE	Oscilador, filtros de segundo y tercer orden

Tabla 2.2. – Trabajos previos en diseño automático de circuitos digitales.

Referencia	**Representación**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Tipos de circuitos**	**Circuitos sintetizados**
(Miller et al., 1997)	Matriz de puertas lógicas	AG	Fija	No	Simulador FPGA	Combinacionales	Sumador y multiplicador de 3 bits
(Kalganova et al., 1998)	Matriz de puertas lógicas	AG	Fija	No	Simulador FPGA	Combinacionales	Sumador de 1 bit
(Hounsell & Arslan, 2000)	Celdas de puertas lógicas	AG	Fija	No	Simulador VHDL	Combinacionales	Circuitos aritméticos
(Kalganova, 2000)	Matriz de puertas lógicas	EE	Fija	No	Simulador FPGA	Combinacionales	Función lógica de 10 salidas
(Zebulum et al., 2000)	Funciones booleanas	SAGA	Variable	Sí	No indicado	Combinacionales	Multiplexor
(Coello Coello & Aguirre, 2002)	Matriz de puertas lógicas	MGA	Fija	Sí	Simulador FPGA	Combinacionales	Circuito de paridad y funciones lógicas

Tabla 2.2. – Trabajos previos en diseño automático de circuitos digitales (continuación).

Referencia	**Representación**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Tipos de circuitos**	**Circuitos sintetizados**
(Ali et al., 2004)	Matriz de puertas lógicas y cromosoma asignación de estados	AG	Fija	Sí	Simulador FPGA	Secuenciales	Contador, detector de secuencia de 4 bits
(Karpuzcu, 2005)	Código Verilog	GE	Variable	No	Icarus Verilog	Combinacionales	Sumador de 1 bit
(Yan et al., 2006)	Matriz de puertas lógicas	GEP	Fija	No	Simulador FPGA	Combinacionales	Sumador de 1 bit
(Al Jassani et al., 2011)	Cadena de enteros	MOGA propio	Fija	Sí	No indicado	Secuenciales	Circuitos *benchmark* con hasta 48 estados
(El-Maleh et al., 2013)	Cadena de enteros	BPSO	Fija	No	No indicado	Secuenciales	Circuitos *benchmark* con hasta 48 estados

Tabla 2.2. – Trabajos previos en diseño automático de circuitos digitales (continuación).

Referencia	**Representación**	**AE**	**Longitud del cromosoma**	**Multiobjetivo**	**Simulador**	**Tipos de circuitos**	**Circuitos sintetizados**
(Lanchares et al., 2013)	Matriz de puertas lógicas	EE	Fija	No	Intrínseco	Combinacionales	Sumador de 2 bits
(Kazarlis et al., 2015)	CGP	AG	Fija	No	Simulador descrito en (Kazarlis et al., 2014)	Combinacionales	Sumador, Comparador y Multiplicador de 2 bits
(Vasicek & Sekanina, 2016)	CGP	CGP	Fija	Sí	BDD	Combinacionales	Circuitos combinacionales con hasta más de 50 salidas
(Mrazek & Vasicek, 2018)	CGP	CGP	Fija	Sí	BDD	Combinacionales	Sumadores hasta 16 bits

3. Métodos propuestos

En este capítulo se describen las dos aproximaciones propuestas en esta tesis para abordar el problema del diseño automático de circuitos electrónicos analógicos. En primer lugar, a modo de introducción, la sección 3.1 describe y resume las principales ideas sobre EG. A continuación, en la sección 3.2, se describe el primer método propuesto, basado en la adaptación de EG al diseño de circuitos electrónicos analógicos. Seguidamente, en la sección 3.3, se presenta la segunda propuesta, una nueva variante de EG, desarrolla en esta tesis y que hemos denominado evolución multigramatical (EMG). A continuación, la sección 3.4 se centrará en el diseño de un nuevo método basado en EMG para el diseño de circuitos electrónicos analógicos. Finalmente, la sección 3.5 describe cómo abordar el problema de cumplir las especificaciones de diseño y, simultáneamente, reducir el número de componentes de los circuitos que evolucionan, con objeto de obtener circuitos más sencillos, mediante un enfoque multiobjetivo simple.

3.1. Introducción a la evolución gramatical

En esta sección se realiza una introducción al algoritmo EG. En primer lugar, se presenta un breve resumen del algoritmo PG del que EG es una variante. En segundo lugar, se describen los principales aspectos del algoritmo EG. En tercer lugar, finalmente, se muestra un ejemplo de decodificación de un cromosoma en el marco del algoritmo EG.

3.1.1. Programación genética

El paradigma evolutivo PG fue introducido por John Koza en 1992 (Koza, 1992) y ha mostrado ser un algoritmo muy potente en problemas de optimización. Los trabajos de Koza utilizando PG cubrieron inicialmente un abanico muy amplio de campos,

tales como regresión simbólica o diseño de circuitos, entre otros (Koza et al., 1999a) y, desde entonces, PG ha alcanzado resultados muy importantes en otros campos, tales como minería de datos (Cano et al., 2017), optimización de bases de datos (Estébanez et al., 2014), desarrollo de modelos empíricos (Dabhi & Chaudhary, 2015), planificación (Aler et al., 2002), diseño de redes neuronales convolucionales (Suganuma et al., 2017), aplicaciones financieras (Aguilar-Rivera et al., 2015), o predicción de resistencia de materiales (González-Taboada et al., 2016), entre otros.

Sin embargo, PG requiere el uso de una representación particular basada en árboles de parseado. Básicamente, las especificaciones de cómo representar los individuos en PG se traducen en definir la sintaxis de dichos árboles. Para ello, se recurre normalmente a un conjunto de funciones y a un conjunto de terminales cuyos elementos se combinan para formar el árbol. Sin embargo, los símbolos del primer conjunto sólo pueden aparecer en los nodos internos del árbol y, los del segundo, en los nodos hoja. Además, para acabar de especificar completamente la sintaxis, también es necesario definir un conjunto de reglas que permitan comprobar si la expresión resultante de cada árbol es válida. Otra característica de PG es la de requerir el uso de operadores de variación específicos, que deben diseñarse de forma que se garantice la consistencia de los programas resultantes mediante la preservación de la denominada propiedad de clausura *(closure property,* en inglés*)* (Koza, 1992), es decir, cada función en el conjunto de funciones disponibles debe ser capaz de aceptar como argumento cualquier salida de una función y/o algún terminal del conjunto de terminales disponibles. En otro caso, habría que comprobar sistemáticamente la validez de los árboles producidos por los operadores de variación.

Para garantizar la preservación de la propiedad de clausura, han aparecido diferentes extensiones en PG. Estas extensiones han recibido diferentes nombres genéricos, tales como programación genética basada en gramáticas (GBGP, por sus siglas en inglés) (Lourenço et al., 2017) o programación genética guiada por gramáticas *(*GGGP, por sus siglas en inglés) (Mckay et al., 2010). En particular, la EG pertenece a esta nueva categoría.

3.1.2. Evolución gramatical

El algoritmo EG es una variante de PG, dentro de las GGGP, desarrollado por Conor Ryan, John J. Collins y Michael O'Neill, (Ryan et al., 1998; O'Neill & Ryan, 2001), capaz de generar código en cualquier lenguaje de programación. A diferencia de PG,

que utiliza árboles para representar los individuos, EG se basa en el uso de cromosomas formados por cadenas binarias de longitud variable. Además, la decodificación del cromosoma se hace mediante una gramática libre de contexto, normalmente especificada mediante notación Backus-Naur (Backus-Naur form o BNF), del lenguaje objetivo. Este proceso de decodificación garantiza la propiedad de clausura (O'Neill & Ryan, 2001) y, adicionalmente, el uso de cadenas lineales permite la utilización de operadores de variación estándar, tales como los que se usan, por ejemplo, en AG. Debido a estas particularidades, en el algoritmo EG existe una separación entre la forma de representación (genotipo) y el motor de búsqueda utilizado para encontrar la solución y, por tanto, este algoritmo puede utilizar todo tipo de motores de búsqueda que permitan manejar cromosomas en forma de cadenas lineales.

La unidad básica de información en EG se llama *codón*, que es un nombre inspirado en la biología y se considera que un codón está formado normalmente por un *byte*, por lo que se dispondrá de 256 valores para cada codón. La decodificación de una cadena determinada se realiza leyéndola de izquierda a derecha y utilizando un codón para la elección de una de las reglas de producción del símbolo no terminal que, en cada momento, se está expandiendo. Para este proceso, se asume que las opciones de cada símbolo no terminal están numeradas consecutivamente, empezando desde cero. La elección de la opción se realiza mediante el operador módulo, tal y como se muestra en la expresión (3.1), donde se determina la regla seleccionada a partir del valor del codón actual y del número de reglas posibles, NR, asociadas al símbolo no terminal que se está expandiendo. El uso del operador módulo garantiza que se obtendrá un valor adecuado, en el rango $\{0, 1, ..., (NR - 1)\}$, para cualquier valor original del codón.

$$regla = codón\ MOD\ NR \tag{3.1}$$

El proceso de decodificación, tal como se ha descrito, se denomina decodificación en profundidad (*depth-first*, en inglés), sin embargo, también se han utilizado otras alternativas dentro de EG (Fagan et al., 2010), tales como decodificación en anchura, aleatoria o πGE (O'Neill et al., 2004), entre otras.

Finalmente, EG introduce también el concepto de *wrapping*, basado en el fenómeno de superposición de genes observado en biología (O'Neill & Ryan, 2001). El mecanismo de *wrapping* consiste en comenzar de nuevo la lectura de la cadena binaria,

una vez se ha recorrido ésta de forma completa, habiendo agotado todos los codones de la misma sin que el proceso de decodificación haya finalizado.

3.1.3. Ejemplo de decodificación en EG

En esta sección se describe un ejemplo sencillo de decodificación en EG. En primer lugar, se muestra la gramática utilizada en este ejemplo (tabla 3.1), que permite resolver un problema de regresión simbólica. Esta gramática es muy similar a la mostrada en (O'Neill & Ryan, 2001).

Una gramática libre de contexto se define mediante una tupla de cuatro componentes, $\{S, T, N, R\}$ donde S es el símbolo de comienzo, T es el conjunto de símbolos terminales, N es el conjunto de símbolos no terminales o variables, y finalmente, R es el conjunto de las reglas de producción de cada símbolo no terminal. A su vez, cada símbolo no terminal es aquel que puede expandirse mediante las reglas de producción, en una cadena formada por símbolos terminales y/o no terminales. Un símbolo terminal es un literal que ya no puede expandirse más. La aplicación de las reglas de producción de forma reiterada a la cadena de símbolos, empezando por el símbolo de comienzo, permitirá obtener una cadena formada únicamente por símbolos terminales.

Es importante comentar que, aunque en la literatura las gramáticas utilizadas en EG se suelen expresar en notación BNF, en esta tesis se utiliza el formato BNF extendido o EBNF (*Extended* BNF, en inglés), acorde al estándar ISO (ISO/IEC-14977) (ISO/IEC-14977, 1996). El uso de una notación formal estándar sistematiza y facilita la codificación de algoritmos, por lo que resulta más adecuado para la implementación de los métodos propuestos.

A continuación, se describe un ejemplo del proceso de decodificación de la siguiente cadena de codones: (6, 8, 24, 27, 64, 8, 27, 3). Esta cadena se decodifica finalmente en la expresión SIN(X + 1.0). El proceso de decodificación es el siguiente:

1. La decodificación comienza con el símbolo de comienzo, S, que corresponde a EXPRESSION.
2. La regla de producción para el símbolo no terminal EXPRESSION tiene cuatro opciones y el primer codón vale 6. Aplicando la expresión 3.1, que da lugar a $6\,MOD\,4$, obtenemos la regla 2, produciendo la expansión: FUNCTION(EXPRESSION).

Tabla 3.1. – Gramática de regresión simbólica simple.

S =	EXPRESSION	
T =	{"+", "-", "*", "/", "SIN", "COS", "EXP", "LOG", "X", "1.0"}	
N =	{EXPRESSION, OPERATOR, FUNCTION, VARIABLE}	
R = *formado por las siguientes reglas de producción*		
EXPRESSION =	EXPRESSION, OPERATOR, EXPRESSION |	(0)
	"(", EXPRESSION, OPERATOR, EXPRESSION, ")" |	(1)
	FUNCTION, "(", EXPRESSION, ")" |	(2)
	VARIABLE	(3)
OPERATOR =	"+" |	(0)
	"-" |	(1)
	"* " |	(2)
	"/"	(3)
FUNCTION =	"SIN" |	(0)
	"COS" |	(1)
	"EXP" |	(2)
	"LOG";	(3)
VARIABLE =	"X" |	(0)
	"1.0";	(1)

3. La expansión de la regla del siguiente símbolo no terminal FUNCTION se hace con el siguiente codón, con valor 8, dando lugar a la regla 0, que corresponde a SIN, por lo que la nueva expresión expandida es ahora: SIN(EXPRESSION).

4. El siguiente codón del cromosoma, con valor 24, determina el uso de la regla 0 para el siguiente símbolo no terminal EXPRESSION, dado que el símbolo SIN es terminal, lo que da lugar a: EXPRESSION, OPERATOR, EXPRESSION. Con ello la nueva expresión expandida es ahora: SIN(EXPRESSION, OPERATOR, EXPRESSION).

5. El siguiente codón, con valor 27, selecciona la regla 3, que aplicada al siguiente símbolo no terminal EXPRESSION corresponde a VARIABLE, por lo que la nueva expresión es ahora: SIN(VARIABLE, OPERATOR, EXPRESSION).

6. El siguiente codón, con valor 64, determina la regla 0, que aplicada al siguiente símbolo no terminal VARIABLE corresponde a X, por lo que la nueva expresión es ahora: SIN(X OPERATOR, EXPRESSION).

7. El siguiente codón, con valor 8, selecciona la regla 0, que aplicada al siguiente símbolo no terminal OPERATOR da lugar a un signo más, por lo que la expresión expandida es ahora: SIN(X + EXPRESSION).

8. El siguiente codón, con valor 27, selecciona la regla 3, que aplicada al siguiente símbolo no terminal EXPRESSION corresponde a VARIABLE, por lo que la expresión expandida es ahora: SIN(X + VARIABLE).
9. El siguiente codón, con valor 3, selecciona la regla 1, que aplicada sobre el siguiente símbolo no terminal VARIABLE corresponde a la constante 1.0, dando lugar a la expresión final, formada ya sólo por símbolos terminales: SIN(X + 1.0).

3.2. Diseño de circuitos electrónicos analógicos basado en EG

En esta sección se describe uno de los dos métodos propuestos en esta tesis para abordar el problema del diseño automático de circuitos electrónicos analógicos, en particular, el método basado en EG. En la sección 3.2.1 se describe la gramática genérica desarrollada para abordar este problema. En la sección 3.2.2 se introduce el parámetro denominado *máximo número de nodos*, dada su importancia en el enfoque propuesto. En la sección 3.2.3 se definen los conceptos de cromosoma inexpresable e inviable y se indica el tratamiento de los mismos. En la sección 3.2.4 se indican los pasos que componen el posproceso de la *netlist* obtenida mediante EG y que son necesarios antes de poder evaluarla en SPICE. En la sección 3.2.5 se describe la forma de evaluar cada individuo (circuito) de la población. En la sección 3.2.6 se detalla un ejemplo de decodificación utilizando la gramática desarrollada para el diseño de circuitos electrónicos analógicos. Finalmente, en la sección 3.2.7 se describe el motor de búsqueda utilizado en esta aproximación.

3.2.1. Gramática para el diseño de circuitos electrónicos analógicos

En esta sección se describe la gramática desarrollada para decodificar un cromosoma dando lugar a una *netlist* que define un circuito electrónico analógico. En particular, la tabla 3.2 muestra la gramática utilizada en este enfoque, que comprende todos los tipos de componentes que se utilizarán en los casos de estudio. Como convenio, esta gramática considera que el nodo 0 corresponde a tierra, de acuerdo con el criterio del simulador de circuitos utilizado para evaluar cada individuo.

El símbolo de comienzo de la gramática es el símbolo no terminal LIST, que se corresponde directamente con el símbolo no terminal COMPONENTS. La expansión del símbolo no terminal COMPONENTS puede dar lugar a uno de los componentes permitidos (resistencia, condensador, transistor BJT o transistor MOSFET). Para permitir que la *netlist* esté formada por más de un componente, el símbolo no terminal COMPONENTS también tiene reglas recursivas que permitirán su expansión como uno de los componentes permitidos y la continuación de su expansión con el mismo símbolo no terminal COMPONENTS. Las opciones recursivas permiten que la *netlist* continúe su expansión con nuevos componentes, mientras que las opciones no recursivas únicamente añaden el componente determinado.

La gramática desarrollada para el diseño de circuitos electrónicos analógicos está especialmente diseñada para que cada componente consuma un bloque de tamaño fijo, es decir, un bloque formado por un número determinado de codones. Por este motivo, a una gramática definida con esta propiedad la denominaremos *gramática de bloques*. La idea de definir bloques en el cromosoma facilitará la definición de operadores de cruce (véase la sección 3.2.7) que permitan el intercambio de material a nivel de bloques, que contendrán componentes completos, minimizando así el potencial efecto destructivo asociado a operadores de cruce estándar, como, por ejemplo, el operador de cruce de un punto. Para conseguir que se cumpla esta propiedad, se utiliza el símbolo no terminal DUMMY. Este símbolo puede expandirse como dos posibles símbolos terminales: “null1” y “null2”. Es importante comentar que todo símbolo no terminal que sólo tenga una opción no consume ningún codón del cromosoma (O’Neill & Ryan, 2001). Por lo tanto, la asignación de dos opciones al símbolo no terminal DUMMY fuerza la lectura de un codón para su expansión. Para ajustar el número de codones requerido por cada componente, al tamaño de bloque establecido, basta con añadir el número de símbolos no terminales DUMMY necesario. Con ello, es posible ajustar la expansión de cada tipo de componente de forma que todos los componentes consuman siempre un número fijo de codones. En concreto, la gramática mostrada en la tabla 3.2 utiliza bloques de ocho codones. Finalmente, es importante comentar que los símbolos “null1” y “null2” no tienen ningún significado en la *netlist* y son eliminados posteriormente, mediante la realización de un posprocesado de la *netlist* obtenida (véase la sección 3.2.4).

Cada componente dispone de dos, tres o cuatro pines, o conexiones, que se representan en la *netlist* indicando el nodo del circuito al que están conectados. El número de nodo se obtiene mediante la expansión del símbolo no terminal NODE. El rango

del número de nodos comienza con el valor cero que, como se ha comentado previamente, corresponde a tierra. El valor más alto del rango vendrá determinado por el usuario a través del parámetro denominado *máximo número de componentes* (MNN) que se explica en la sección 3.2.2.

Los no terminales RESISTOR y CAPACITOR, que dan lugar a resistencias y condensadores, respectivamente, tienen un valor expresado en notación científica, $m \cdot 10^e$, donde $0.1 \leq m < 10$ y el orden de magnitud del exponente, e, depende del tipo de componente. Los transistores pueden ser de tipo BJT o MOSFET y, a su vez, los transistores BJT podrán ser PNP o NPN y, los MOSFET, de canal N (NMOS) o de canal P (PMOS). La elección del tipo de transistor viene determinada por los símbolos no terminales BJTTYPE y MOSTYPE. En el caso de los transistores MOSFET, existe una cuarta conexión a sustrato. En este caso, se asume que la conexión al sustrato de los PMOS y NMOS se realiza a los nodos de voltaje máximo y mínimo del circuito, respectivamente. Estos pines se codifican, de forma fija, en los símbolos no terminales MODELPMOS y MODELNMOS. La longitud de canal MOSFET queda fijada a $10\mu m$, mientras que la anchura del canal podrá evolucionar con valores pertenecientes al intervalo $[10, 199]\,\mu m$, de la misma manera que se hace en (Koza et al., 1999a).

La gramática mostrada en la tabla 3.2 es una gramática genérica, por lo que debe ser ajustada a cada problema de diseño específico al que se vaya a aplicar. Por ejemplo, en caso de que no se vaya a utilizar un tipo de componente para el diseño de un circuito, dicho componente debería eliminarse. Por el contrario, también es posible añadir nuevos tipos de componentes, tales como, por ejemplo, bobinas, diodos, etc., si las especificaciones de diseño del circuito así lo requieren. Se mostrará un ejemplo de decodificación utilizando la gramática aquí descrita en la sección 3.2.6.

3.2.2. Máximo número de nodos (MNN)

Como se ha indicado previamente, el símbolo no terminal NODE se expande según su regla de producción como un número de nodo del circuito en el rango que va desde cero hasta un número máximo. Dicho valor máximo viene determinado por la suma de dos valores. El primer valor corresponde al máximo número de nodo asociado a la parte fija (véase la sección 1.2). Es necesario recordar que la parte fija es la parte del circuito que no se modifica durante el proceso evolutivo y consta de las entradas, salidas, tierra, alimentación, etc. La parte fija se define a partir de las

Tabla 3.2. – Gramática genérica para la generación de *netlists* en el enfoque basado en EG (véase la sección 3.2.1, para una descripción detallada).

S =	LIST
T =	{ "R", "C", "Q", "M", "QNPN", "QPNP", "null1", "null2", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "e", ".", end-of-line }
N =	{ LIST, LINE, RESISTOR, CAPACITOR, BJT, MOSFET, BJTTYPE, MOSTYPE, MODELNMOS, MODELPMOS, DUMMY, NODE, RESISTORVAL, CAPACITORVAL, CHANELWIDTH, DIGIT, NONZERODIGIT, EXPONENT, EOL }
R =	*Está formado por las siguientes reglas de producción:*
LIST =	COMPONENTS;
COMPONENTS =	RESISTOR | RESISTOR, COMPONENTS | CAPACITOR | CAPACITOR, COMPONENTS | BJT | BJT, COMPONENTS | MOSFET | MOSFET, COMPONENTS;
RESISTOR=	"R", NODE, NODE, DUMMY, RESISTORVAL, DUMMY, EOL;
CAPACITOR =	"C", NODE, NODE, DUMMY, CAPACITORVAL, DUMMY, EOL;
BJT =	"Q", NODE, NODE, NODE, BJTTYPE, DUMMY, DUMMY, DUMMY, EOL;
MOSFET =	"M", NODE, NODE, NODE, MOSTYPE, CHANELWIDTH, EOL;
BJTTYPE =	"QNPN" | "QPNP";
MOSTYPE =	MODELNMOS | MODELPMOS;
MODELNMOS =	"0", "NMOS1 L=10u W="
MODELPMOS =	"1", "PMOS1 L=10u W="
DUMMY =	"null1" | "null2";
NODE =	"0" | "1" | "2" | "3" | "4" | "5" | ... | "Máximo nodo de la parte fija+MNN";
RESISTORVAL =	NONZERODIGIT, ".", DIGIT, "e", DIGIT;
CAPACITORVAL =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
CHANELWIDTH =	NONZERODIGIT, DIGIT, "u" | "1", DIGIT, DIGIT, "u";
DIGIT =	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
NONZERODIGIT =	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
EXPONENT =	"-12" | "-11" | "-10" | "-9" | "-8" | "-7" | "-6" | "-5" | "-4" | "-3";
EOL=	*carácter de fin de línea*

especificaciones de diseño y consideraremos que sus nodos de circuito se encuentran numerados consecutivamente, comenzando desde el nodo con número 0, correspondiente a tierra. El segundo valor, elegido por el usuario, define el máximo número de nodos nuevos de circuito que no pertenecen a la parte fija. Este valor es el que se denomina máximo número de nodos (MNN). Cuanto mayor sea MNN, el circuito candidato podrá ser más complejo. El parámetro MNN tiene un fuerte impacto en el rendimiento del algoritmo pues un valor demasiado bajo del mismo limitará el número de posibles topologías a considerar, mientras que un valor demasiado elevado dará lugar a circuitos más complejos o con un número mayor de componentes con terminales no conectados (Zebulum et al., 1999).

En definitiva, el rango del símbolo no terminal NODE, aparece como enumeración de opciones de número de nodos, comenzando por el cero y llegando hasta la suma del máximo nodo de la parte fija más el valor de MNN. Por ejemplo, si se considera la parte fija mostrada en la figura 1.1, que contiene los nodos de 0 a 3, y se elige un valor de MNN de 10, la enumeración de opciones dentro de la regla de producción para la expansión del no terminal NODE deberá contemplar el rango de enteros desde 0 a 13.

Finalmente, hay que indicar que, aunque los números de nodo que se obtienen de la expansión del símbolo no terminal NODE pueden ser utilizados para conectar componentes, puede ocurrir que haya nodos a los que no sea deseable conectar componentes del circuito. Por ejemplo, si las especificaciones del circuito objetivo incluyen una fuente de voltaje de entrada con una resistencia serie interna (véase la figura 1.1), ningún componente del circuito candidato debería poder conectarse directamente a dicha fuente, pues invalidaría el uso de la resistencia interna, no respetando las especificaciones. La forma propuesta para resolver este problema es asignar a dicho nodo un número que quede fuera del rango de opciones incluidas en la regla de producción para la expansión del no terminal NODE.

3.2.3. Cromosomas inexpresables y cromosomas inviables

Como se ha visto en la sección 3.1, EG incorpora un mecanismo de *wrapping* que permite repetir la lectura del cromosoma si se llega al final de la lectura del mismo sin haber conseguido obtener una expresión final. Sin embargo, este mecanismo tiene asociado un valor límite, que indica el máximo número de veces que se puede aplicar.

A pesar del *wrapping*, puede haber cromosomas que no se puedan decodificar, por no haber podido expandir todos los símbolos no terminales (O'Neill & Ryan, 2001). Estos cromosomas, denominados *inexpresables*, no dan lugar a una *netlist* válida y, por lo tanto, no pueden ser simulados.

La existencia de cromosomas inexpresables resulta indeseable ya que no representan soluciones válidas. Por este motivo, es necesario reducir su número cuanto sea posible. Para ello, se aplica una fuerte penalización de su valor de adaptación, reduciendo la probabilidad de ser seleccionados en la fase de selección de padres. Como se ha visto en la sección 1.2, el problema de diseño de circuitos es un problema de minimización, es decir, valores de la función de adaptación más bajos corresponderán a mejores cromosomas. En la implementación del algoritmo, el valor de penalización se ha fijado arbitrariamente al máximo valor de adaptación permitido (en este caso como 5×10^7), por lo que cualquier otro valor de adaptación será inferior a dicho máximo. Dado que, como se verá en la sección 3.2.7, la selección de padres se realizará mediante torneo, un cromosoma inexpresable solamente podrá ser seleccionado, si todos los cromosomas participantes en el torneo son también inexpresables.

Además de los cromosomas inexpresables, también puede ocurrir otra situación indeseable a nivel fenotípico: el proceso de decodificación produce una expresión final sintácticamente válida, pero ésta no corresponde a un circuito válido. Este caso se puede deber a varias circunstancias como, por ejemplo, cuando el circuito obtenido no tiene conexión con todos y cada uno de los nodos de la parte fija. Estos circuitos se denominan *inviables* y sus cromosomas recibirán también el mismo nombre. La ausencia del tipo de conexiones indicadas responden a restricciones que no se pueden implementar dentro de la gramática, y se abordan en una etapa de posprocesado (véase la sección 3.2.4). Adicionalmente, puede que los problemas del circuito obtenido se hagan evidentes a la hora de simular el mismo, dando lugar a un error del simulador. En dicho caso, también se etiqueta el cromosoma como inviable.

La aparición de cromosomas inexpresables se considera peor que la de cromosomas inviables, ya que los primeros no dan lugar a una *netlist*. De esta manera, la penalización aplicada a los cromosomas inviables es menor que la aplicada a los cromosomas inexpresables. En particular, se ha fijado una penalización por inviabilidad con valor mitad de la penalización por inexpresabilidad (2.5×10^7).

Existe un tercer caso de circuitos problemáticos, si bien es menos grave que los dos casos anteriores. Este tercer caso hace referencia a la existencia de componentes

con alguno de sus terminales no conectados a ningún otro componente del circuito, que denominaremos informalmente como componentes con terminales «al aire». En estos casos, es preferible adoptar alguna medida para corregir esta situación, antes que etiquetar el circuito como inviable. En la literatura existen varias medidas para abordar este problema (véase la sección 2.3.2). En nuestra aproximación se ha optado por insertar una resistencia de $1G\Omega$ entre el terminal no conectado y masa, de igual manera que se hace en (Mattiussi & Floreano, 2007). La motivación de esta decisión se basa en su sencillez de implementación, al ser independiente del número de terminales del componente afectado.

Finalmente, se considera que todos los cromosomas que no sean inexpresables o inviables, serán cromosomas viables, incluyendo entre estos últimos a aquellos cromosomas que den lugar a componentes con terminales «al aire», pero modificados según la estrategia indicada.

3.2.4. Posprocesado de la *netlist*

La *netlist* que se obtiene al decodificar un cromosoma no puede simularse directamente, pues todavía le falta incluir los componentes de la parte fija, así como comprobar su viabilidad y retirar los símbolos "null1" y "null2". Por tanto, es necesario aplicar una etapa de posprocesado para terminar de preparar la *netlist* antes de proceder a su simulación. Los pasos de posprocesado que se realizan sobre la *netlist* decodificada son los siguientes:

1. Inclusión de los componentes de la parte fija.
2. Comprobación de la viabilidad del circuito. Se hace una comprobación sencilla, por la que se chequea que todos los nodos de la parte fija tienen conexión con el circuito candidato. En caso de que no sea así, la *netlist* se marca como inviable y no será simulada. También se aplica la penalización correspondiente al cromosoma asociado (véase la sección 3.2.3.)
3. Borrado de los símbolos "null1" y "null2" (véase la sección 3.2.1).
4. Conexión de resistencias a los componentes con terminales «al aire». Se utiliza una resistencia de $1G\Omega$ entre el terminal no conectado y masa (véase la sección 3.2.3).
5. Eliminación de componentes cortocircuitados. El algoritmo puede dar lugar a componentes que tengan sus terminales cortocircuitados y que, por lo tanto,

no cumplen ninguna función. Estos componentes se eliminan antes de simular el circuito candidato.

6. Numeración de componentes. El simulador requiere que cada componente esté identificado por una etiqueta única (p. ej. R1, R2, R3). En este paso se asigna un número único a la etiqueta que define el componente. Nótese que no es posible incorporar una etiqueta numérica en la gramática, ya que no es posible modelar estas etiquetas mediante una gramática libre de contexto.
7. Transformación de los valores de los componentes. Los valores de los componentes, que se encuentran en notación científica, se convierten a notación estándar y se les añaden los prefijos de unidades correspondientes.
8. Incorporación de modelos de los componentes utilizados. En concreto, se añaden los modelos de los transistores utilizados. Se utilizan los modelos BJT 2N3904 y 2N3906, NPN y PNP, respectivamente, en el caso de transistores bipolares. En el caso de los MOSFET, se utiliza un modelo genérico de SPICE para este tipo de transistores, en el que el valor de anchura del canal es aprendido por el evolutivo.

3.2.5. Evaluación de individuos

La evaluación de cada individuo (*netlist*) se realiza mediante el simulador de circuitos electrónicos NGSpice (véase la sección 1.6).

Posteriormente, se procesa la salida del simulador para obtener la señal de salida a evaluar en el circuito, esto es, el voltaje de salida o la corriente de salida. Dicha salida se compara con las especificaciones de diseño para obtener el valor de adaptación del circuito evaluado. En caso de error en la simulación, se marca el circuito evaluado como inviable y se penaliza correspondientemente el cromosoma asociado, tal como se indicó en la sección 3.2.3.

3.2.6. Ejemplo de decodificación

A continuación, se muestra un ejemplo de decodificación de un cromosoma, asumiendo que el problema a resolver consiste en obtener un amplificador basado en transistores BJT, considerando un conjunto de especificaciones y usando como parte fija del circuito la mostrada en la figura 1.1.

Para dimensionar el número de nodos, en la mencionada figura se observa que, la parte fija usa los nodos 0, 1, 2 y 3. También se observa que la fuente de entrada, Vs, dispone de una resistencia de entrada en serie, Rs, cuyo nodo de conexión entre ambas se etiquetará con el valor 50. Nótese que no está permitido que los circuitos candidatos puedan conectarse directamente a dicho nodo, pues puentearían la resistencia de entrada, Rs. Por ello, se asigna un número de nodo que no pueda generarse en la expansión del no terminal NODE, tal como se indicó en la sección 3.2.2. Adicionalmente, el valor de MNN se fija en 6. Con ello, el rango de nodos asociado al símbolo NODE es de 0 a 9, es decir:

NODE = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

El ejemplo de cromosoma que se va a decodificar es el siguiente: (53, 34, 22, 60, 4, 122, 32, 71, 9, 251, 74, 7, 82, 140, 93, 232, 66, 94, 33, 102, 28, 40, 63, 67, 255, 3, 45, 32). El proceso de decodificación utilizará la gramática mostrada en la tabla 3.2, modificada para incluir la regla de producción del símbolo no terminal NODE anteriormente indicada. El proceso sigue los siguientes pasos:

1. El proceso de decodificación comienza con el símbolo de comienzo, S, que se expande con el no terminal LIST.
2. El símbolo no terminal LIST se expande como COMPONENTS. Nótese que como la regla de producción sólo tiene una opción, no es necesario leer ningún codón del cromosoma. La expresión obtenida hasta el momento queda como: COMPONENTS.
3. La expansión del símbolo no terminal COMPONENTS tiene varias opciones, por lo que es necesario leer un codón del cromosoma, que en este caso es el primer codón, que vale 53. Aplicando la ecuación 3.1, $53\,MOD\,8$, da lugar a la elección de la regla 5, por lo que la expresión queda como: BJT, COMPONENTS.
4. Recorriendo la nueva expresión obtenida de izquierda a derecha, el primer símbolo no terminal a expandir es BJT. La regla de producción para expandir dicho símbolo tiene sólo una opción, por lo que no es necesario leer un codón. La nueva expresión expandida es: Q, NODE, NODE, NODE, BJTTYPE, DUMMY, DUMMY, DUMMY, EOL, COMPONENTS.
5. Q es un símbolo terminal, por lo que no es necesario expandirlo. A continuación, se expande el primer no terminal, NODE, y dado que su regla de

producción tiene varias opciones, se hace necesario leer un codón. La aplicación de la ecuación 3.1 para el siguiente codón leído, con valor 34, da lugar a la elección de la regla 4 ($34\,MOD\,10$), que corresponde al nodo 4, por lo que la nueva expresión expandida queda como sigue: Q 4, NODE, NODE, BJTTYPE, DUMMY, DUMMY, DUMMY, EOL, COMPONENTS.

6. La expansión de los otros dos símbolos no terminales NODE siguientes, con los siguientes valores de codones (22 y 60), nos devuelve los nodos 2 ($22\,MOD\,10$) y 0 ($60\,MOD\,10$), respectivamente, dando lugar a la siguiente expresión expandida: Q 4 2 0, BJTTYPE, DUMMY, DUMMY, DUMMY, EOL, COMPONENTS.

7. La regla de producción del siguiente símbolo no terminal, BJTTYPE, usa el siguiente codón de valor 4. Entonces, se selecciona la regla 0 ($4\,MOD\,2$), que está asociada con el símbolo terminal QNPN, y que corresponde al tipo de transistor BJT. La nueva expresión expandida queda como sigue: Q 4 2 0 QNPN, DUMMY, DUMMY, DUMMY, EOL, COMPONENTS.

8. Los siguientes tres símbolos no terminales DUMMY se utilizan como codones de relleno para completar un bloque de ocho codones de tamaño. La expansión de dichos no terminales con los codones de valores 122, 32 y 71 dan lugar a los símbolos terminales null1, null1 and null2, respectivamente, quedando la nueva expresión expandida como sigue: Q 4 2 0 QNPN null1 null1 null2 EOL, COMPONENTS.

9. La regla de producción para la expansión del símbolo no terminal EOL tiene sólo una opción y corresponde a un carácter de final de línea. En este momento finaliza la decodificación del primer componente de la *netlist,* siendo la nueva expresión expandida como sigue: Q 4 2 0 QNPN null1 null1 null2. Sin embargo, para no complicar la notación, la información asociada a dicho componente no se mostrará en las siguientes expansiones, dando por supuesto que está incluido en la primera línea de la *netlist*.

10. A continuación, el siguiente símbolo no terminal, COMPONENTS, se expande con el codón de valor 9, seleccionando la regla 1, que da lugar a la nueva expresión expandida: RESISTOR, COMPONENTS.

11. La regla de producción del siguiente símbolo no terminal, RESISTOR, consta sólo de una opción, por lo que no es necesario consumir un codón. La expansión

de dicho símbolo da lugar a la siguiente nueva expresión expandida: R, NODE, NODE, DUMMY, RESISTORVAL, DUMMY, EOL, COMPONENTS.

12. La decodificación continúa con los símbolos no terminales NODE que, con los codones con valores 251 y 74, dan lugar a los nodos 1 y 4, respectivamente. La nueva expresión expandida es: R 1 4 DUMMY, RESISTORVAL, DUMMY, EOL, COMPONENTS.

13. El proceso de decodificación continúa hasta que la expresión expandida resultante consta sólo de símbolos terminales y, por lo tanto, no quedan más símbolos no terminales que expandir. En este ejemplo, el proceso termina con la lectura del codón en la posición 24, de valor 67. Es decir, los cuatro últimos codones del cromosoma no se leen y, por lo tanto, no se tienen en cuenta en la decodificación.

La *netlist* resultante es la siguiente:

```
Q 4 2 0 QNPN null1 null1 null2
R 1 4 null2 1.0e3 null1
C 4 3 null1 1.0e-9 null2
```

Sin embargo, dado que esta netlist no se puede simular directamente, a continuación, se aplica la etapa de posprocesado, que comprende los pasos indicados en la sección 3.2.4. En este caso particular, sólo se requieren los siguientes pasos: (a) Inclusión de la *netlist* de la parte fija (véase la figura 1.1), (b) borrado de los símbolos "null1" y "null2", (c) numeración de los componentes, y (d) uso de los sufijos adecuados a los valores de los componentes. Las acciones asociadas a los pasos de detección de circuitos inviables, borrado de componentes cortocircuitados o tratamiento de componentes con terminales no conectados, no son de aplicación en este caso. La *netlist* finalmente resultante es la siguiente:

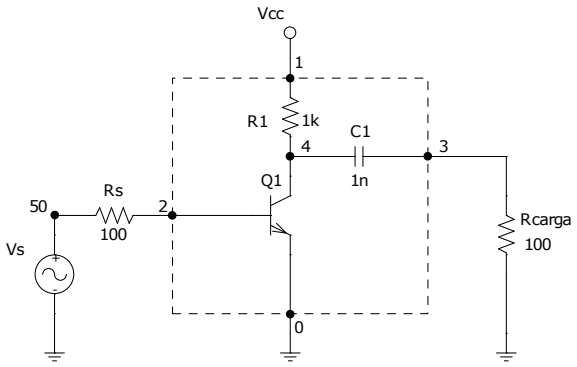


Figura 3.1. – Ejemplo de circuito candidato, obtenido mediante evolución y tras su inserción en la parte fija de circuito mostrada en la figura 1.1.

** Parte de la netlist correspondiente a la parte fija*
Vcc 1 0 dc 5
Vs 0 50 0.0 ac 0.001
Rs 50 2 100
Rload 3 0 100
** Netlist obtenida por el algoritmo evolutivo*
Q1 4 2 0 QNPN
R1 1 4 1.0k
C1 4 3 1.0n

El circuito asociado a la *netlist* final se muestra en la figura 3.1.

3.2.7. Motor de búsqueda

Como se ha indicado en la sección 3.1, una de las características de EG es la separación entre la forma de representación/decodificación y el motor de búsqueda utilizado para encontrar la solución. Esta separación hace que EG sea un algoritmo evolutivo modular, ya que permite utilizar cualquier algoritmo evolutivo, que maneje cadenas lineales, como motor de búsqueda. Habitualmente, se utiliza un AG como motor de búsqueda (Ryan et al., 1998; O'Neill & Ryan, 2001), pero también es posible el uso de otras opciones, como pueden ser aquellas basadas en enjambres de partículas (O'Neill & Brabazon, 2004; O'Neill & Brabazon, 2006b) o en evolución diferencial (O'Neill & Brabazon, 2006a). En esta tesis, se utiliza el modelo habitual basado en AG.

En esta sección, se describen las características del motor de búsqueda basado en AG. En particular, se utilizan operadores estándar, con excepción del operador de cruce, para el que se propone un operador que hace uso del concepto de bloque asociado a la gramática utilizada. La estructura del resto de la sección es la siguiente: en primer lugar se describe el operador de selección de padres. A continuación, se describen los operadores de recombinación. Posteriormente, se describe el operador de mutación. Finalmente, se describe el operador de selección de supervivientes.

Selección de padres

En cada generación, es necesario seleccionar un conjunto de μ padres que servirán para crear una población de hijos como consecuencia del proceso de recombinación. Esta selección se hace teniendo en cuenta los valores de adaptación de los individuos para que exista presión selectiva y el algoritmo tienda progresivamente a producir mejores soluciones.

Existen métodos de selección de padres con probabilidad proporcional a los valores de adaptación de los individuos como, por ejemplo, el método de ruleta. Sin embargo, estos métodos presentan problemas de convergencia prematura, ante casos de individuos con valores de adaptación excepcionales, y también, problemas de falta de presión selectiva, cuando todos los individuos presentan valores de adaptación similares (Eiben & Smith, 2003). Dichos métodos, también requieren una transformación de los valores de adaptación en el caso de que el problema sea de minimización (Eiben & Smith, 2003), como es nuestro caso.

Alternativamente, existen métodos de selección no proporcionales a los valores de adaptación. Uno de ellos es el método de selección por torneo, que consiste en la toma de una muestra aleatoria de un número k de individuos de la población, y la selección como padre del individuo con mejor valor de adaptación que se encuentre en dicha muestra. Este proceso se realiza sucesivamente hasta obtener μ padres. El número de individuos que componen cada torneo es un parámetro de configuración. El método de selección por torneo es uno de los más utilizados en la literatura (Eiben & Smith, 2003).

En esta tesis se utilizará el método de selección por torneo, debido a que no presenta los problemas asociados a los métodos proporcionales a los valores de adaptación de los individuos, ni tampoco requiere transformación de los valores de adaptación.

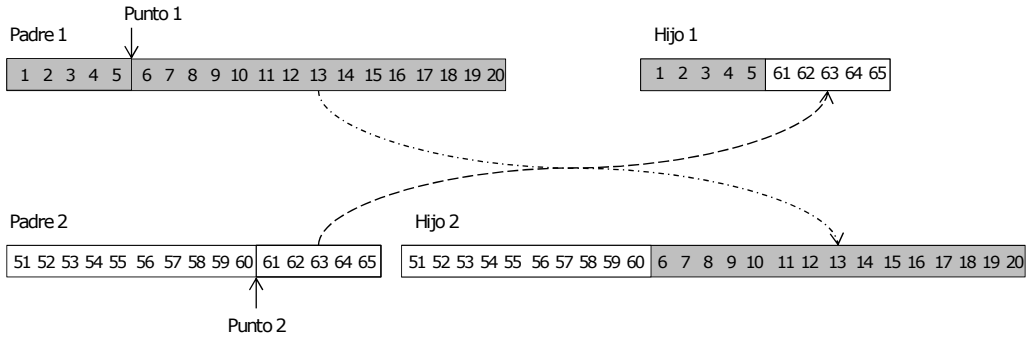


Figura 3.2. – Operador estándar de cruce de un punto.

Operador de recombinación

El operador de recombinación combina la información contenida en dos cromosomas padre para obtener dos nuevos cromosomas hijos, con información de ambos padres. Este operador es propio de los algoritmos evolutivos y los distingue frente a otros algoritmos de búsqueda o de optimización. La aplicación del operador viene parametrizada por el valor de tasa o probabilidad de recombinación, que suele variar entre 0.5 y 1.0 (Eiben & Smith, 2003), y que refleja la probabilidad de que los dos padres seleccionados se recombinen.

Aunque en EG se propone el uso del operador de cruce estándar de un punto (O'Neill et al., 2003), este operador se ha asociado a efectos destructivos sobre los cromosomas (Nordin et al., 1995), dado que el intercambio de información no tiene en cuenta la información genotípica que se intercambia. Por este motivo, en esta tesis se propone un nuevo operador denominado *operador de cruce one-block,* con el objetivo de evitar dichos problemas. Las prestaciones de dicho operador también se compararán con las del operador estándar de un punto (véase la sección 5.4.1).

El operador de cruce de un punto se caracteriza por su sencillez. En la figura 3.2, se muestra su funcionamiento, que consta de los siguientes pasos:

1. Se obtiene un número aleatorio de codón para cada cadena padre, limitado por el tamaño de cada una.
2. Se intercambian la parte final de cada cadena, a partir del punto obtenido en el paso anterior.

Por otro lado, el nuevo operador de cruce propuesto en esta tesis, operador de cruce one-block, es una variante del operador de cruce descrito en (Nicolau & Dempsey, 2006). En particular, el punto de cruce es múltiplo del tamaño de bloque que se

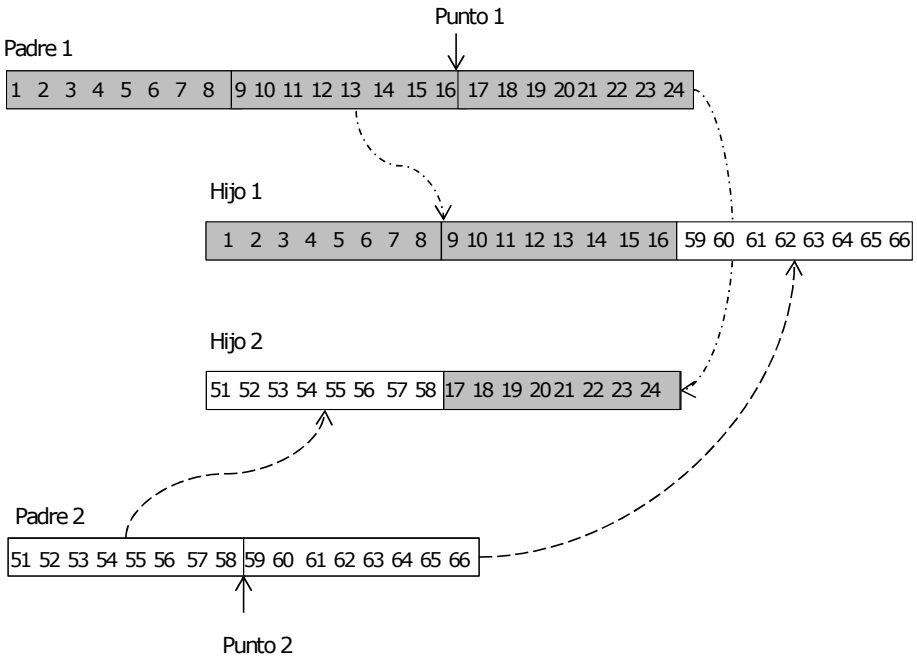


Figura 3.3. – Operador de cruce *one-block* usado en la aproximación basada en EG. Nótese que los puntos de cruce siempre deben ser múltiplos del tamaño de bloque que, en este caso, es de 8 codones.

define en la gramática (véase la sección 3.2.1). De esta manera, la información de los cromosomas se intercambia siempre en la frontera entre componentes, forzando que el intercambio use componentes completos (véase la figura 3.3). Nuestra hipótesis es que el uso de este operador permitirá evitar el efecto destructivo asociado al operador de cruce estándar de un punto. Adicionalmente, el nuevo operador definido dispone de un control de tamaño máximo del cromosoma que permite mitigar el efecto engorde (véase la sección 2.4.1). En caso de que se vaya a generar un cromosoma hijo mayor que el tamaño máximo, automáticamente se recorta a dicho tamaño.

Operador de mutación

Los operadores de mutación trabajan sobre un cromosoma, generalmente obtenido como resultado del operador de recombinación, y producen una variación de forma aleatoria sobre el mismo. Se parametrizan mediante una tasa o probabilidad de mutación cuya interpretación es propia del tipo de operador utilizado.

El operador de mutación utilizado en esta tesis es un operador de mutación de cambio de bit o *bitwise.* Este operador recorre la cadena de codones bit a bit de izquierda a derecha y para cada uno de ellos obtiene un número aleatorio. En el caso de que dicho

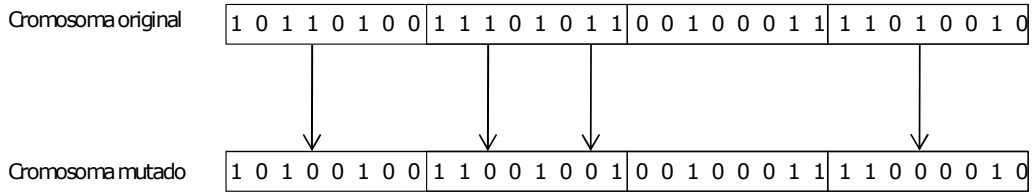


Figura 3.4. – Operador de mutación de cambio de bit o *bitwise* utilizado en la aproximación basada en EG.

número aleatorio sea menor que la tasa o probabilidad de mutación anteriormente indicada, el algoritmo realizará el cambio (*flip*) del valor del bit considerado (véase la figura 3.4).

Selección de supervivientes

Después de la selección de los μ padres, se generan λ hijos mediante aplicación de los operadores de variación anteriormente indicados. El modelo de selección de supervivientes, también llamado de reemplazo, define cómo se construye la siguiente generación a partir de una población de (μ, λ) individuos.

En esta tesis se ha utilizado el denominado modelo generacional, es decir, para cada generación, se reemplazan todos los individuos de la población actual por la descendencia obtenida ($\mu = \lambda$). Para evitar la pérdida de los mejores individuos, generalmente se suele utilizar elitismo, que permite preservar los n mejores individuos de una generación copiándolos directamente a la siguiente. El número n de individuos copiados es un parámetro del elitismo.

En los artículos seminales de EG (O'Neill & Ryan, 1999; O'Neill & Ryan, 2001; O'Neill et al., 2003; O'Neill & Ryan, 2004), se proponía el uso de una estrategia *steady-state*, en la que se reemplazaba un número de individuos menor que el total de la población, es decir $\lambda < \mu$. Dicho valor, λ, que se suele llamar también *gap,* tomaba el valor $\lambda = 1$ en el modelo *steady-state* originalmente propuesto.

Sin embargo, en los trabajos previos realizados en esta tesis, se han obtenido mejores resultados utilizando la estrategia generacional conjuntamente con elitismo que con *steady-state*, por lo que se ha optado por el uso del modelo generacional.

3.3. Una nueva variante de EG: Evolución multigramatical

En esta sección se presenta un nuevo enfoque, denominado evolución multigramatical (EMG), que es una nueva variante de EG surgida como consecuencia del desarrollo de esta tesis, en el intento de mejorar los resultados obtenidos por el método descrito en la sección 3.2.

El contenido de este epígrafe se estructura de la siguiente forma. En la sección 3.3.1, se presentarán dos propiedades deseables que debería tener una variante basada en EG: modularidad y homología. En la sección 3.3.2, se describirá el nuevo algoritmo EMG, focalizando en sus dos aspectos clave: la forma de codificar la información en el cromosoma y el proceso de decodificación. En la sección 3.3.3, se desarrollará un ejemplo de decodificación de un cromosoma en el ámbito de EMG. Debido a la existencia de cierta problemática a la hora de incluir la propiedad de homología en el marco de EMG, en la sección 3.3.4, se muestra una forma de abordar y resolver dicha problemática. Finalmente, en la sección 3.3.5, se presentará una extensión opcional en EMG, para permitir el aprendizaje de parámetros pertenecientes a las propias gramática utilizadas en el proceso de decodificación.

3.3.1. Homología y modularidad

EG es uno los algoritmos GGGP más utilizado en la actualidad (Mckay et al., 2010), si bien, hay cierta controversia sobre su rendimiento (Lourenço et al., 2017; Whigham et al., 2015; Ryan, 2017). En particular, la discusión se centra en el operador de cruce. Desde sus comienzos, EG se basó en el uso del operador de cruce estándar de un punto, muy utilizado en los AG (O'Neill & Ryan, 2001). Aunque éste es un operador muy sencillo y válido, su principal desventaja es su comportamiento potencialmente destructivo cuando se utiliza en EG, debido a que parte de la información recibida en los hijos puede interpretarse de forma diferente a cómo se hace en los padres. Alternativamente, existe la idea del operador de cruce homólogo, inspirado en la biología (Francone et al., 1999; Langdon, 2000). La propiedad de homología en la naturaleza implica que los fragmentos de cromosoma que se intercambian entre cromosomas siempre pertenecen a la misma posición dentro de los mismos y son de tamaños similares. Aunque hay evidencia en la literatura sobre intentos de construcción de operadores homólogos en EG, estos no han dado mejores resultados que los

obtenidos con el operador de cruce estándar de un punto (O'Neill et al., 2003). Adicionalmente, hay que tener en cuenta que un operador homólogo no resulta sencillo de implementar en EG, pues requiere almacenar el histórico de reglas de producción utilizadas durante la decodificación de cada cromosoma.

Por otro lado, la propiedad de modularidad y sus beneficios son bien conocidos en informática (Pressman, 1997). Dicha propiedad permite abordar un problema mediante la descomposición del mismo en subproblemas o módulos. De esta manera, dicho módulo puede enfocarse en un aspecto o funcionalidad del problema original, encapsulando la complejidad inicial dentro de cada módulo. La propiedad de modularidad también aparece de forma recurrente en la biología pues facilita la propiedad de evolucionabilidad, al limitar la interferencia entre la adaptación de funciones distintas (Wagner & Altenberg, 1996).

En el siguiente apartado se describe una nueva variante de EG, cuyo principal objetivo es incorporar las propiedades de homología y modularidad en su esquema de representación y decodificación, en un intento de mejorar las prestaciones del paradigma original.

3.3.2. Evolución multigramatical

Como en EG, un cromosoma en EMG es una cadena de codones de longitud variable. Sin embargo, a diferencia de lo que ocurre en EG, un cromosoma en EMG se divide en tantas particiones como subproblemas aparezcan durante el proceso de descomposición del problema de optimización que se esté abordando. Cada partición representa una solución a cada subproblema y se decodifica utilizando una gramática específica propia de dicho subproblema. La solución completa para el problema inicial emerge como agregación de los resultados de la decodificación de las diferentes particiones del cromosoma. EMG puede verse como un proceso descendente o *top-down*, en el que la primera gramática decodifica las características de alto nivel de abstracción del problema original, y la última gramática decodifica las características de bajo nivel. Con este nuevo enfoque, lo que se persigue es incorporar la propiedad de modularidad en EG.

No obstante, la idea de incluir la propiedad de modularidad en EG no es nueva. Por ejemplo, en (Swafford et al., 2011), aquellos módulos dentro del cromosoma que contienen información beneficiosa también evolucionan y, además, se incorporan en

la gramática utilizada. Sin embargo, la idea de modularidad que se incorpora en EMG es diferente, ya que surge del resultado de descomponer el problema original en diferentes subproblemas, por lo que la solución a cada subproblema se codifica en diferentes particiones dentro del cromosoma.

Por otro lado, el uso de más de una gramática en el proceso de decodificación también se ha explorado en la literatura relacionada con EG. Por ejemplo, el enfoque denominado evolución gramatical mediante evolución gramatical (EG^2) (O'Neill & Ryan, 2004) describe el uso de dos gramáticas diferentes: la gramática universal o gramática de gramáticas y la gramática de la solución. La primera permite construir la segunda, es decir, durante la ejecución del algoritmo la gramática de la solución evoluciona, permitiendo encontrar la mejor gramática para resolver el problema. De forma similar, otra variante de EG^2, denominada algoritmo genético basado en metagramática (mGGA, por sus siglas en inglés) (O'Neill et al., 2004), también hace uso de gramáticas que evolucionan durante la ejecución del algoritmo. Por el contrario, en EMG, las gramáticas utilizadas serán siempre fijas, es decir, no evolucionan y tampoco se busca aprender una mejor gramática.

El proceso de decodificación en EMG se basa en el proceso de decodificación de EG, pero con la modificación necesaria para utilizar un conjunto de gramáticas de forma secuencial. El proceso comienza utilizando el símbolo de comienzo de la primera gramática y opera igual que en EG: se parte del primer codón del cromosoma y se aplica la regla del módulo estándar descrita en la ecuación 3.1. La diferencia con EG aparece cuando se obtiene una expresión completa en el ámbito de la primera gramática, esto es, cuando la expresión decodificada está formada únicamente por símbolos terminales de la primera gramática. En este punto, el proceso de decodificación continúa de la siguiente forma: la expresión completa obtenida en el paso previo actúa como *símbolo de comienzo*[1] y se utiliza la segunda gramática para continuar el proceso de decodificación. Ahora sólo se expandirán aquellos símbolos que son no terminales respecto de la segunda gramática. El resultado de la decodificación utilizando la segunda gramática da lugar a una expresión que sólo contiene símbolos terminales respecto de dicha gramática. Este proceso se repetirá para cada una de las gramáticas incluidas en el conjunto de gramáticas. Al aplicar la última gramática, se obtendrá una expresión final que corresponderá al fenotipo asociado al cromosoma completo y será una solución candidata al problema original.

[1] Rigurosamente hablando se debería hablar aquí de «expresión de comienzo», dado que se parte de la expresión obtenida en el paso anterior.

Desde el punto de vista estructural, un cromosoma en EMG queda dividido implícitamente en diferentes particiones, determinadas por las gramáticas utilizadas para codificar la solución de cada subproblema. Para identificar cada partición, se utilizan unas marcas denominadas puntos de partición o P-points. EMG determina que se incluya una marca P-point posteriormente al último codón leído por cada gramática utilizada. Con ello, aparecerán tantas particiones como gramáticas se utilicen en el proceso de decodificación. Adicionalmente, si el proceso de decodificación termina antes de recorrer el cromosoma completo, los codones a continuación del último codón leído se ignoran. De esta manera, el conjunto de P-points refleja la disposición de las diferentes particiones, donde la longitud de cada partición no es fija y depende de los valores concretos de los codones del cromosoma, de la gramática utilizada y de la expresión utilizada como símbolo de comienzo. Finalmente, es importante notar que la división en particiones sólo aparece cuando la decodificación ha terminado, es decir, no se pueden saber las particiones de un cromosoma *a priori*.

La figura 3.5 muestra un diagrama de bloques del proceso de decodificación, adaptado para el caso particular de tres gramáticas. Sobre el cromosoma se muestran las tres particiones resultantes de la decodificación, correspondiendo cada partición a cada gramática utilizada. Adicionalmente, aparece una partición final etiquetada como *resto*, donde se encuentran los codones no utilizados en la decodificación. También se puede ver cómo la expresión obtenida al decodificar la partición i-ésima, se utiliza como símbolo de comienzo por la gramática $(i+1)$-ésima en el proceso de decodificación de la partición $(i+1)$-ésima del cromosoma. Toda esta operativa queda resumida, de manera más formal, en al algoritmo 3.1. Finalmente, se mostrará un ejemplo del proceso de decodificación en la sección 3.3.3.

EMG, como variante de EG, también es un AE que permite separar el proceso de decodificación del proceso de búsqueda. Así, también se puede utilizar como motor de búsqueda un AG o cualquier optimizador que represente la solución como una cadena lineal. De igual manera, se pueden utilizar en EMG los operadores de variación y selección que resultaban eficaces en EG. No obstante, si se quiere incorporar la propiedad de homología, es necesario definir un operador de cruce que haga un uso eficiente de las particiones incluidas en cada cromosoma padre. Este operador podría basarse en el cruce de particiones homólogas de cada padre, utilizando los P-point como punto de cruce (véase la figura 3.6). Sin embargo, aquí hay un problema: aquellas particiones heredadas por cada hijo, que originalmente se encontraban a la derecha de los puntos de cruce en sus respectivos padres, sólo

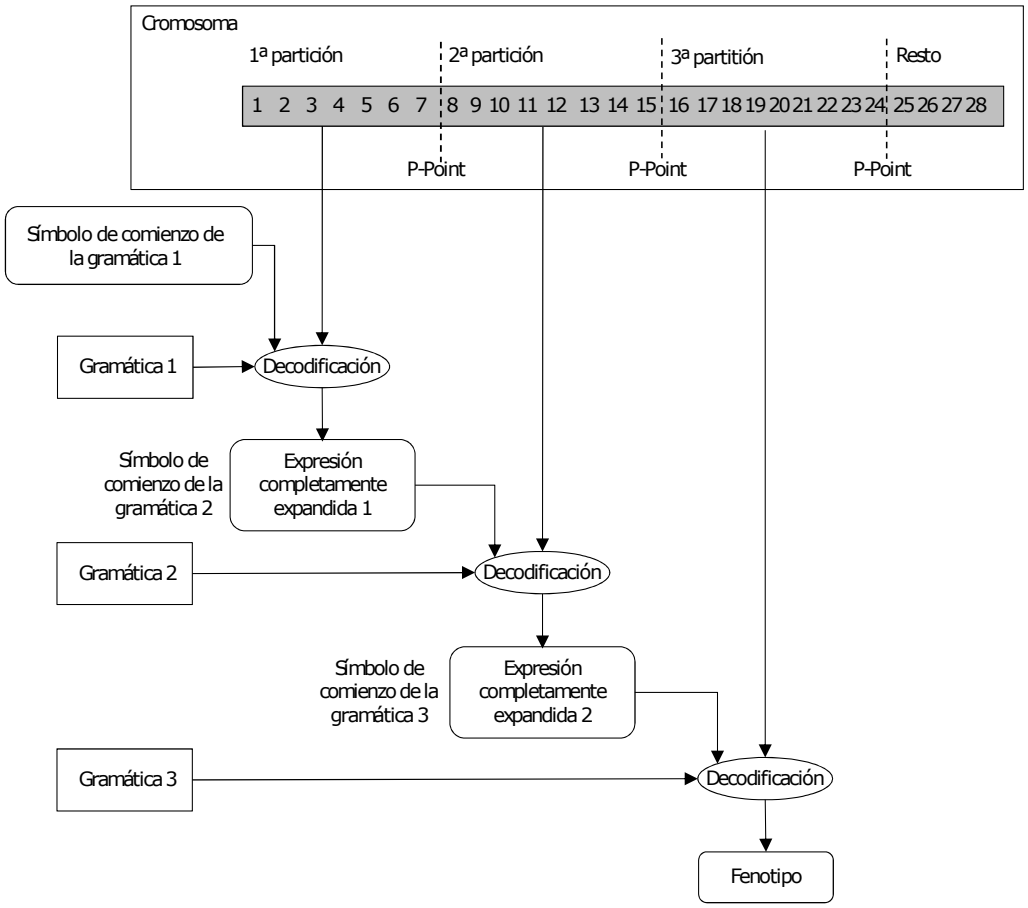


Figura 3.5. – Ejemplo de decodificación en EMG particularizado para el caso de tres gramáticas. También se muestran las particiones obtenidas en el proceso de decodificación del cromosoma.

Algoritmo 3.1 Pseudocódigo del proceso de decodificación en EMG.

Entradas: Cromosoma a decodificar (cromosoma); símbolo de comienzo de la gramática I (*símbolo_de_comienzo*); gramáticas a utilizar (*gramática*[i], con $i = 1, ..., n$)

Salida: expresión decodificada (*expresión*)

```
lista_de_P-points ← {∅};
puntero_de_lectura ← 0;
expresión ← símbolo_de_comienzo;
FOR i ← 1 TO n
   WHILE hay_símbolos_no_terminales(expresión) DO
      no_terminal ← siguiente_símbolo_no_terminal(expresión, gramática[i]);
      IF regla_de_producción(no_terminal, gramática[i]) tiene_varias_reglas
THEN
         puntero_de_lectura ← puntero_de_lectura+1;
         codón ← leer_cromosoma(puntero_de_lectura, cromosoma);
         regla ← mod(codón, número_de_opciones(no_terminal, gramática[i]));
      ELSE
         regla ← 0;
      End-IF
      expresión ← expandir_regla_de_producción(expresión, regla);
   End-WHILE
   lista_de_P-points ← lista_de_P-points ∪ puntero_de_lectura;
End-FOR
```

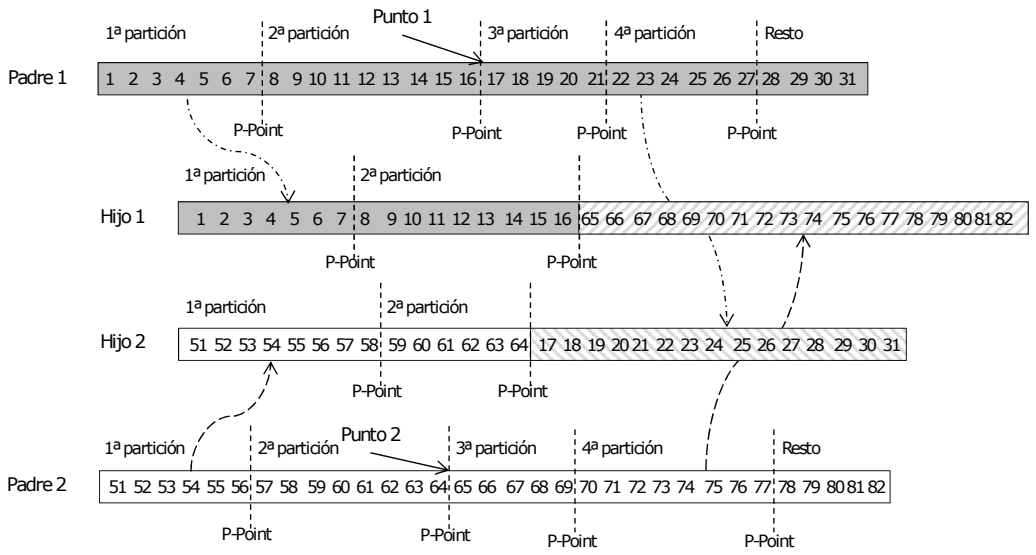


Figura 3.6. – Ejemplo de operador de cruce pseudo-homólogo en EMG basado en el operador de un punto. Nótese que los puntos de cruce elegidos deben apuntar a P-point homólogos. Sin embargo, las particiones que se encuentran en el lado derecho de los P-points de los padres no se conservarán necesariamente en los respectivos hijos.

son particiones estimadas. Las particiones definitivas sólo aparecerán después de realizar la decodificación de cada hijo. En otras palabras, el operador así definido no garantizaría la conservación de todas las particiones de los padres y, por ello, estrictamente hablando, sólo podría ser etiquetado como operador de cruce pseudo-homólogo. Sin embargo, en la sección 3.3.4, se describirá cómo, en determinados problemas de optimización, resulta más fácil introducir la idea de operador de cruce homólogo.

La introducción del algoritmo evolutivo EMG requiere una mayor expresividad de las gramáticas que las utilizadas hasta ahora. en EG. Para ello se recurrirá a lo que se denomina *extensiones gramaticales*. En concreto, aquí se utilizará una de las extensiones gramaticales propuestas en (Nicolau & Dempsey, 2006) y, además, se proponen otras nuevas que son necesarias para abordar el formalismo usado en EMG. Con la incorporación de estas extensiones, las gramáticas utilizadas dejan de cumplir el estándar ISO/IEC, así como también dejan de ser gramáticas puramente BNF. Sin embargo, se ha hecho necesario el uso de estas extensiones para mejorar la expresividad de las gramáticas y proveerlas de mayor funcionalidad.

Como se verá a continuación, las extensiones gramaticales vienen delimitadas por el uso de los caracteres “<” y “>”. Cada extensión se identifica por una palabra

clave pudiendo llevar además algunos argumentos. Las extensiones propuestas son las siguientes:

1. <GEPpointMarker> Esta extensión permite introducir una marca de punto de partición o P-point en el cromosoma, que será utilizada posteriormente por el operador de cruce. Esta marca se introduce a continuación del último codón leído.

2. <GECodonValue: *valor mínimo*, *valor máximo*> Esta extensión permite abreviar una lista de opciones formada por enteros sucesivos (comprendidos entre *valor mínimo* y *valor máximo*) en la expansión de un símbolo no terminal, a partir del valor del codón leído.

3. <GEResult> esta extensión indica que el algoritmo debe continuar la decodificación utilizando la expresión completa decodificada con la gramática anterior. Por tanto, permite formalizar el hecho de que el símbolo de comienzo de la gramática $(i+1)$-ésima contenga la expresión completa decodificada por la gramática (i)-ésima.

4. *<GEDecTimeVar: %Var>* esta extensión permite definir una variable durante el proceso de decodificación del cromosoma que puede ser utilizada en expresiones o en reglas de producción. Por tanto, esta extensión permite el ajuste de los parámetros de gramática sin necesidad de modificar la propia gramática. Estas variables se denominan con el prefijo "%" (p. ej. %VAR).

Dado que, con el uso de estas extensiones, las gramáticas dejan de ser gramáticas EBNF o BNF puras, es necesario usar un formalismo ligeramente diferente al usado por las gramáticas usadas en EG (véase la sección 3.1.3). Ahora, las gramáticas utilizadas en EMG vienen definidas por una tupla $< S, T, N, R, E >$, donde S, T, N y R tienen el mismo significado que en las gramáticas BNF, pero donde se introduce un nuevo conjunto, E, que corresponde al conjunto de extensiones gramaticales utilizadas.

Finalmente, es preciso señalar que no todas las extensiones definidas anteriormente pueden ser necesarias para todos los problemas abordados mediante EMG. Sin embargo, se han visto necesarias para completar el formalismo EMG utilizado para los problemas abordados en esta tesis. Igualmente, se podrían incluir nuevas extensiones si el problema así lo requiriera.

3.3.3. Ejemplo de decodificación con EMG

Sea un problema de regresión simbólica basada en funciones *kernel*, donde la función a aproximar, $f(x)$, se expresa como una suma ponderada de funciones *kernel*:

$$f(x) = w_1 K_1 + \ldots + w_n K_n \tag{3.2}$$

siendo K_i una instancia de una función *kernel* perteneciente a un conjunto de funciones *kernel* definido por el usuario y $w_i \in \mathbb{R}$ su peso asociado. Aunque el conjunto de funciones *kernel*, $\left\{K^{(1)}, \ldots, K^{(M)}\right\}$, está determinado por el usuario, el número de instancias de cada tipo de función *kernel* obtenidas (cero, una o más de una), así como el número total de instancias, sólo depende de la información contenida en cada cromosoma y, por tanto, ambos números no están predefinidos por el usuario. El tipo de funciones *kernel* utilizadas en este ejemplo son las siguientes:

$$\begin{cases} K^{(1)}(\gamma, c) = exp(-\gamma \left\| c - x \right\|^2) & \gamma \in R, c \in R \\ K^{(2)}(\alpha, \beta, d) = (\alpha x + \beta)^d & \alpha \in R, \beta \geq 0, d \in \mathbb{Z}^+ \\ K^{(3)}(\alpha, \beta) = tanh(\alpha x + \beta) & \alpha \in R, \beta \in R \end{cases} \tag{3.3}$$

Para abordar este problema mediante EMG, se utiliza un conjunto de dos gramáticas. La gramática I (véase la tabla 3.3) determina la composición de funciones *kernel* que aproximan la función objetiva. La gramática II (véase la tabla 3.4) define el peso y los valores de los parámetros de cada *kernel* utilizado. Como puede observarse en la tabla 3.4, la gramática II introduce el uso de la extensión <GECodonValue>. Esta extensión nos permite evitar el uso de la enumeración explícita de todos los elementos pertenecientes a un intervalo de enteros para las reglas de producción de los no terminales DEGREE, DIGIT, NONZERODIGIT y EXPONENT.

El cromosoma ejemplo que se va a decodificar es: (5, 7, 13, 8, 31, 2, 3, 4, 2, 2, 1, 1, 6, 7, 8, 9, 15, 34, 7, 23, 92, 115, 3, 22, 31, 48, 14, 78, 92, 21, 62, 23, 24, 32, 92, 1, 31, 56, 27, 88, 19, 31, 15, 23, 7, 84, 32, 13, 32, 34, 73, 48, 51, 32). El procedimiento de decodificación usado por la primera gramática sigue los mismos pasos que los usados por EG (véase la sección 3.2.6). En particular, para obtener una expresión formada sólo por símbolos terminales de acuerdo a la gramática I se requieren los primeros cuatro codones. El proceso de decodificación inserta un P-point después del cuarto

Tabla 3.3. – Gramática I usada en EMG para el problema de regresión simbólica.

S =	START
T =	{ "KNL1", "KNL2", "KNL3", " + ", "END" }
N =	{ START, KERNELS }
E =	{ <GEPpointMarker> }
R =	Comprende las siguientes reglas de producción*:*
START =	KERNELS, "END", <GEPpointMarker>;
KERNELS =	"KNL1" | "KNL1", " + ", KERNELS | "KNL2" | "KNL2", " + ", KERNELS | "KNL3", " + " | "KNL3", KERNELS;

Tabla 3.4. – Gramática II usada en EMG para el problema de regresión simbólica.

S =	START
T =	{ "K1", "K2", "K3", "+", "-", " * ", ".", "e", ", ", "(", ")" }
N =	{ KNL1, KNL2, KNL3, WEIGHT, FLOATPAR, POSFLOATPAR, INTPAR, SIGN, DIGIT, NONZERODIGIT, EXPONENT, END }
E =	{ <GEResult>, <GECodonValue>, <GEPpointMarker> }
R =	Comprende las siguientes reglas de producción*:*
START =	<GEResult>;
KNL1 =	WEIGHT, " * ", "K1(", FLOATPAR, ", ", FLOATPAR, "),";
KNL2 =	WEIGHT, " * ", "K2(", FLOATPAR, ", ", POSFLOATPAR, ", ", DEGREE, ")";
KNL3 =	WEIGHT, " * ", "K3(", FLOATPAR, ", ", FLOATPAR, "),";
END=	"", <GEPpointMarker>;
FLOATPAR=	SIGN, NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
POSFLOATPAR =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
DEGREE =	<GECodonValue: 0, 9>;
WEIGHT =	"(", SIGN, NONZERODIGIT, ".", DIGIT, "e", EXPONENT, ")";
SIGN=	"+" | "-";
DIGIT =	<GECodonValue: 0, 9>;
NONZERODIGIT =	<GECodonValue: 1, 9>;
EXPONENT =	<GECodonValue: -9, 9>;

codón del cromosoma. La expresión obtenida es la siguiente:

KNL3 + KNL1 + KNL1 + KNL2

Sin embargo, dicha expresión todavía no es una expresión completa desde el punto de vista fenotípico, pues todavía hay símbolos no terminales con respecto a la gramática II. Por este motivo, es necesario utilizar esta última para finalizar el proceso de decodificación. La expresión finalmente obtenida es la siguiente:

(-3.3e-5) * K3(+3.1e-8, +8.8e0) +

(-8.7e-5) * K1(+8.3e-6, -4.4e-7) +

(+4.2e-5) * K1(+6.2e-8, -3.7e3) +

(-5.5e-5) * K2(-4.2e4, 6.4e7, 8)

Esta expresión corresponde a un fenotipo completo. El último codón leído se encuentra en la posición 52 (valor 48). Con ello, los codones situados en las posiciones 53 y 54 (con valores 51 y 32 respectivamente) no llegan a ser leídos y, por tanto, se ignoran. Nótese que el proceso de decodificación inserta un P-point en el último codón leído.

3.3.4. Introduciendo la propiedad de homología en EMG

En esta sección se presenta una forma de introducir la propiedad de homología cuando EMG se aplica a una clase particular de problemas de optimización. En concreto, hablamos de aquellos problemas que admiten una descomposición jerárquica. Desde un punto de vista general, este es el caso de la mayoría de los problemas de diseño y, en particular, es también el caso del problema que nos ocupa en esta tesis, el del diseño de circuitos electrónicos.

El objetivo de la mayoría de los problemas de diseño es el aprendizaje de un modelo que cumple un conjunto de especificaciones y puede describirse mediante una jerarquía de n niveles de abstracción (véase un ejemplo en la figura 3.7). El nivel 1 contiene la descomposición del modelo inicial en diferentes componentes que sólo pertenecen a dicho nivel de abstracción; el nivel 2 contendrá la descomposición en subcomponentes de cada componente perteneciente al nivel 1, y donde cada subcomponente pertenece sólo al nivel 2; y así sucesivamente, hasta alcanzar el nivel

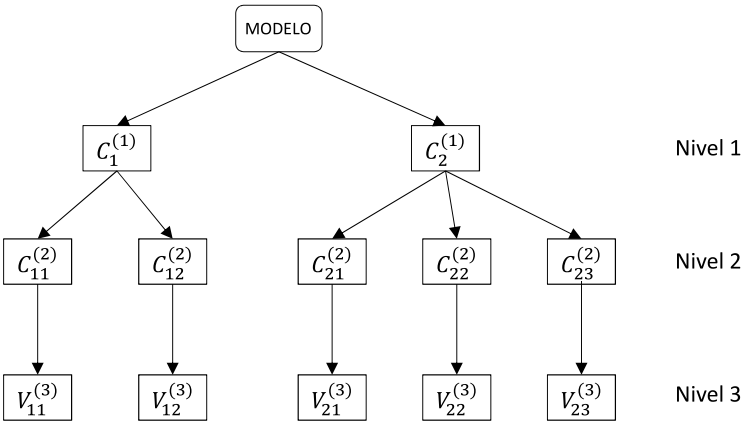


Figura 3.7. – Ejemplo de modelo de diseño descrito por una jerarquía de tres niveles de abstracción.

$n-1$ de abstracción. Finalmente, el nivel n almacenará los valores de los parámetros que caracterizan a cada componente perteneciente al nivel $n-1$. De hecho, es fácil encontrar problemas de diseño en la literatura, que puedan descomponerse en una jerarquía de niveles de abstracción. Por ejemplo, en el problema de diseño de circuitos, que nos ocupa en esta tesis, existe el nivel de topología (conteniendo tipos de componentes y sus conexiones), que puede estar compuesto por uno o más niveles (conteniendo, por ejemplo, subcircuitos y sus conexiones, componentes de cada subcircuito y sus conexiones, etc.), y el nivel de dimensionamiento (conteniendo los valores de cada componente electrónico). Otro ejemplo posible de problema de diseño podría ser el diseño de redes neuronales artificiales, donde también existen varios niveles para describir la arquitectura de la red (nivel de capa y nivel de neuronas) y un nivel más que define los pesos de las conexiones de cada neurona. Nótese que EMG, tal como se ha descrito hasta ahora, puede utilizarse también para resolver un problema de diseño. Es necesario definir una gramática para cada nivel de abstracción y, de esta manera, cada partición del cromosoma tendrá un conjunto de componentes pertenecientes a cada nivel de abstracción. Sin embargo, para introducir la propiedad de homología en este tipo de problemas, también es necesario imponer una restricción adicional en el tipo de gramáticas utilizado. En concreto, es necesario que las gramáticas sean sólo recursivas por la derecha, es decir, que las reglas de producción que definan los componentes a utilizar tengan la forma

siguiente:

$$C = C_1, C|...|C_n, C|C_1|...|C_n \tag{3.4}$$

donde $C_1, ..., C_n$ son símbolos no terminales que representan diferentes componentes que pueden utilizarse en un nivel de abstracción determinado. Por el contrario, la recursión general (3.5) y la recursión por la izquierda (3.6) no están soportadas:

$$C = C, C_1, C|...|C, C_n, C|C_1|...|C_n \tag{3.5}$$

$$C = C, C_1|...|C, C_n|C_1|...|C_n \tag{3.6}$$

La motivación de la restricción de recursión por la derecha es forzar que la gramática sea una gramática de bloques, como ya se hizo con las gramáticas propuestas para la aproximación propuesta en la sección 3.2.1. En dicha sección se mostraba cómo se podía codificar un componente completo en un único bloque de codones consecutivos en el cromosoma.

Con esta restricción en EMG, cada partición queda dividida en un conjunto de bloques, donde cada bloque representa el componente del nivel de abstracción representado en dicha partición. Adicionalmente, aparece una relación entre cada componente y su descomposición en el nivel inmediatamente inferior, es decir, una relación uno a uno entre cada bloque de la partición i-ésima y un conjunto de bloques consecutivos de la partición $(i + 1)$-ésima.

Con esta nueva estrategia emergen dos tipos de homologías: una homología intercromosómica, definida por la lista de P-points, y una homología intracromosómica, definida por la relación entre bloques de particiones consecutivas. La figura 3.8 muestra un ejemplo de cromosoma dividido en tres particiones, así cómo las relaciones intercromosómicas que aparecen.

Los dos tipos de homología nos permitirán definir un operador de cruce que aproveche la existencia de ambas, y que denominaremos operador de cruce homólogo

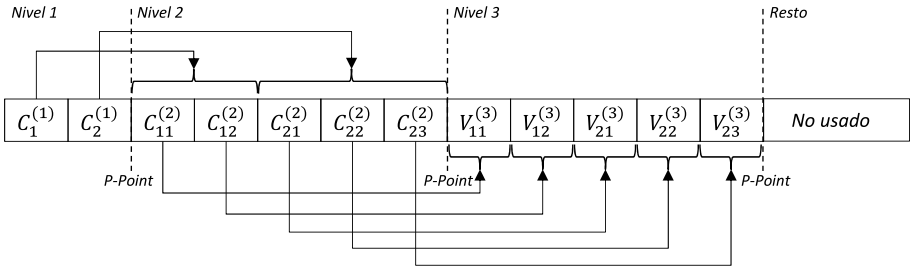


Figura 3.8. – Cromosoma que codifica una instancia del modelo de diseño presentado en la figura 3.7 mediante una aproximación basada en EMG. Cada partición corresponde a un nivel de abstracción y está delimitada por un P-point, lo que permite la definición de una homología intercromosómica. Adicionalmente, la relación uno a uno entre cada componente (bloque) del nivel i-ésimo (partición) y su descomposición correspondiente (subconjunto de bloques consecutivos) en el nivel $(i+1)$-ésimo (partición) también permite definir una homología intracromosómica. Las líneas verticales discontinuas delimitan cada partición.

multipartición basado en gramáticas de bloques (BG-MHX, por sus siglas en inglés). Para ilustrar un ejemplo de este nuevo operador de cruce, volvemos al problema de regresión simbólica, presentado en la sección 3.3.3. Obsérvese que dicho problema ya se había definido como un problema de diseño, donde se utilizan dos niveles de abstracción. El primer nivel contiene las funciones *kernel* a utilizar en la expresión candidata y el segundo nivel incluye los valores de los parámetros de cada *kernel*. En relación con las gramáticas utilizadas (véanse las tablas 3.3 y 3.4), ambas cumplen la restricción de ser recursivas por la derecha y, además, generan bloques de tamaño fijo, si bien, el valor de este tamaño fijo es diferente en cada gramática: la primera gramática utiliza bloques de tamaño uno y la segunda utiliza bloques de tamaño doce. De esta manera, cada cromosoma está formado por dos particiones y, a su vez, cada partición está formada por bloques diferentes: cada bloque de la primera partición codifica una función *kernel* y cada bloque de la segunda partición codifica el peso y los valores de los parámetros necesarios para definir dicha función *kernel*.

La figura 3.9 muestra un ejemplo del funcionamiento del operador de cruce homólogo, actuando sobre los dos cromosomas padre para producir dos cromosomas hijos. En primer lugar, selecciona aleatoriamente los puntos de cruce dentro de la primera partición de cada cromosoma padre. Para ello, elige una posición aleatoria que coincide con la frontera de los bloques. A continuación, el punto de cruce intrahomólogo queda automáticamente determinado en la segunda partición de cada padre. Por lo tanto, los tipos de funciones *kernel* elegidos para su copia de la primera partición de un cromosoma padre determinan los bloques correspondientes de la segunda

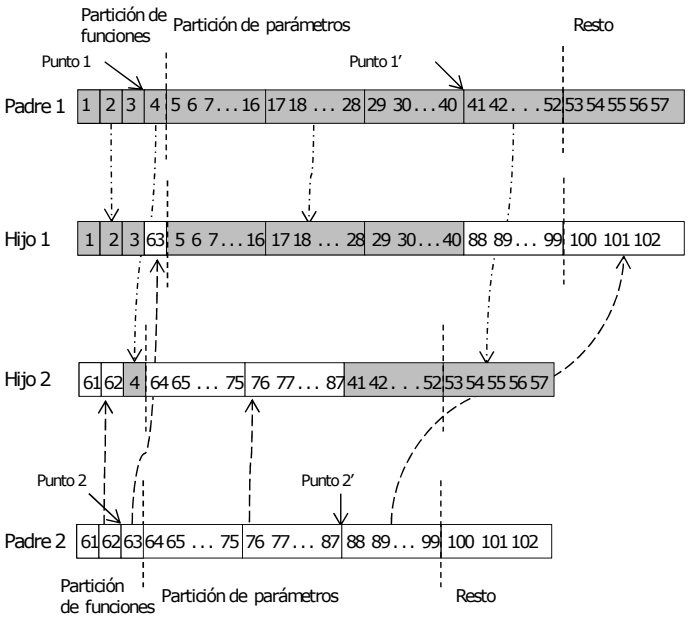


Figura 3.9. – Ejemplo de uso del operador de cruce homólogo (BG-MHX) en EMG. La elección aleatoria de los puntos de cruce, 1 y 2, asociados a bloques de la primera partición, determina automáticamente la elección de los puntos 1' y 2', respectivamente (homología intracromosómica), asociados a bloques de la segunda partición. El intercambio de bloques en los hijos (homología intercromosómica) se hace teniendo en cuenta, convenientemente, los puntos 1-1' y 2-2'. Las líneas verticales discontinuas delimitan cada partición.

partición que también serán copiados. De esta manera, se garantiza que la información de cada función *kernel* (tipo de función *kernel* y valores de los parámetros y peso), que se copia al cromosoma hijo de cada padre es completa. Se evita así, el comportamiento destructivo asociado al operador de cruce clásico de un punto, que podría intercambiar información parcial de dos funciones *kernel* diferentes o mezclar la información de los valores de parámetros de forma que no correspondan a la función *kernel* asociada. Este tipo de operadores homólogos deberían incluir también una restricción de tamaño máximo de longitud cromosoma con objeto de mitigar el efecto engorde (véase la sección 3.2.7). De esta forma, en el caso de que el cromosoma hijo superara el tamaño máximo, la cadena se recortaría al tamaño máximo. Adicionalmente, si un cromosoma hijo no puede decodificarse (i.e. es un cromosoma inexpresable) o tiene lugar *wrapping* en la decodificación del cromosoma, el operador de cruce homólogo debería incluir algún mecanismo para solucionar este inconveniente, por ejemplo, generando los dos hijos como clonación directa de los padres.

3.3.5. Aprendizaje de parámetros de gramática

En la sección 3.2.2, se presentó el parámetro de usuario MNN que determina el máximo número de nodos que puede utilizar el algoritmo para diseñar un circuito en un problema dado, además del número de nodos ya incluidos en la parte fija del circuito. El ajuste de dicho parámetro se hacía modificando la propia gramática, concretamente modificando la enumeración explícita de opciones en la expansión del símbolo no terminal NODE.

Dado el impacto que puede tener el parámetro MNN en la eficacia del algoritmo para encontrar el circuito óptimo (véase la sección 5.4.1), resulta de interés encontrar un método automatizado para el ajuste del mismo. De forma más genérica, en esta sección se propone un método para realizar el aprendizaje de parámetros pertenecientes a una gramática en EMG. En adelante, estos parámetros se denotarán como *parámetros de gramática*.

Para soportar el aprendizaje de parámetros de gramática con EMG, es necesario poder almacenar dichos parámetros en el cromosoma y poderlos decodificar posteriormente mediante la gramática utilizada. También es necesario que dichos parámetros tengan algún efecto sobre la gramática, por lo que ésta necesita poder ser parametrizada. El formalismo de las gramáticas BNF o EBNF no permiten esta opción. Para permitir esta parametrización, se utiliza una de las extensiones de la gramática definidas en la sección 3.3.2, concretamente, las *variables de tiempo de decodificación*.

Para poder almacenar este tipo de parámetros en el cromosoma se introduce una nueva partición, que llamaremos *partición de aprendizaje de parámetros de gramática*. Esta partición se encuentra al comienzo del cromosoma y se lee al comienzo del proceso de decodificación. Los valores decodificados se asignan al parámetro de la gramática correspondiente y el proceso de decodificación del resto del cromosoma se realiza de la misma manera que se ha visto en EMG, pero integrando en dicho proceso el valor asignado al parámetro.

Para ilustrar el uso de aprendizaje de parámetros de gramática en EMG, se mostrará un ejemplo a partir del problema de regresión simbólica basada en funciones *kernel* mostrado en la sección 3.3.3. En particular, las tablas 3.5 y 3.6 muestran, respectivamente, las gramáticas modificadas a partir de las mostradas en las tablas 3.3 y 3.4. Dicha modificación permite incluir el aprendizaje de parámetros de gramática. En

Tabla 3.5. – Gramática I para el problema de regresión simbólica abordado mediante EMG con aprendizaje de parámetros de gramática.

S =	START
T =	{ "KNL1", "KNL2", "KNL3", " + " }
N =	{ START, KERNELS, EXP }
E =	{ <GEPpointMarker>, <GECodonValue>, *<GEDecTimeVar: %*DEG>, %DEG }
R =	Comprende las siguientes reglas de producción*:*
<GEDecTimeVar: %DEG> =	<GECodonValue: 0, 12>;
START =	EXP, <GEPpointMarker>;
EXP =	KERNELS, <GEPpointMarker>;
KERNELS =	"KNL1" | "KNL1", " + ", KERNELS | "KNL2" | "KNL2", " + ", KERNELS | "KNL3", "+ " | "KNL3", KERNELS;

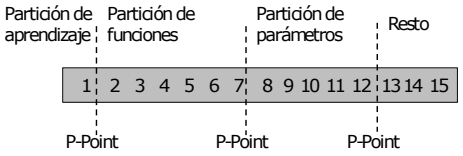


Figura 3.10. – Ejemplo de partición de un cromosoma dentro del problema de regresión simbólica abordado mediante EMG, incluyendo la partición en la que se codifican los parámetros de gramática.

concreto, el parámetro a aprender será el *máximo grado de las funciones kernel polinómicas* que pueden utilizarse en el problema de regresión. Para permitir este ajuste, se utilizará la variable de gramática %DEG que se lee a partir del cromosoma como expansión de la regla de producción "<GEDecTimeVar: %DEG>=<GECodonValue: 0, 20>" contenida en la gramática I. Dicha regla indica la lectura del primer codón del cromosoma, su conversión a un valor entre 0 y 20, de acuerdo al resultado de aplicar el operador módulo, y la carga del mismo en la variable de tiempo de decodificación. Esta variable es accesible durante el proceso de decodificación, por lo que puede utilizarse en la gramática II para la expansión del símbolo no terminal DEGREE, lo que permite parametrizar una regla de producción de la propia gramática. De esta manera, la gramática se ha podido modificar a través de la variable %DEG. La figura 3.10 muestra un ejemplo de cromosoma que incluye una partición de aprendizaje de parámetros de gramática. Nótese que el último codón leído en esta partición también incluye un P-point en la lista de P-points generada en la decodificación. Se mostrará un ejemplo más de aprendizaje de parámetros de gramática aplicado al diseño de circuitos analógicos en la sección 3.4.3.

Tabla 3.6. – Gramática II para el problema de regresión simbólica abordado mediante EMG con aprendizaje de parámetros de gramática.

S =	START
T =	{ "K1", "K2", "K3", " * ", ".", "e", ", ", "(", ")"}
N =	{ KNL1, KNL2, KNL3, WEIGHT, FLOATPAR, INTPAR, DIGIT, NONZERODIGIT, EXPONENT }
E =	{ <GEResult>, <GECodonValue>, <GEPpointMarker> }
R =	Comprende las siguientes reglas de producción*:*
START =	<GEResult>, <GEPpointMarker>;
KNL1 =	WEIGHT, " * ", "K1(", FLOATPAR, ", ", FLOATPAR, "),";
KNL2 =	WEIGHT, " * ", "K2(", FLOATPAR, ", ", POSFLOATPAR, ", ", DEGREE, ")";
KNL3 =	WEIGHT, " * ", "K3(", FLOATPAR, ", ", FLOATPAR, "),";
FLOATPAR =	SIGN, NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
POSFLOATPAR =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
DEGREE =	<GECodonValue: 0, %DEG>;
WEIGHT =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
SIGN=	"+" | "-";
DIGIT =	<GECodonValue: 0, 9>;
NONZERODIGIT =	<GECodonValue: 1, 9>;
EXPONENT =	<GECodonValue: -9, 9>;

3.4. Diseño de circuitos electrónicos basado en EMG

En este apartado se describe el diseño de circuitos electrónicos analógicos mediante un enfoque basado en EMG. Así, la sección 3.4.1 aborda las consideraciones a tener en cuenta por las gramáticas utilizadas en dicho enfoque. La sección 3.4.2 describe el operador de cruce homólogo desarrollado. La sección 3.4.3 muestra cómo abordar el aprendizaje del parámetro MNN. Finalmente, la sección 3.4.4 describe los requisitos del motor de búsqueda utilizado.

3.4.1. Gramáticas en EMG para el diseño de circuitos analógicos

Para la aplicación de los conceptos de EMG al diseño de circuitos analógicos, se han utilizado dos gramáticas. La primera gramática se utiliza para la selección de la topología y la segunda para el dimensionamiento. Ambas gramáticas se basan en una división adecuada de la gramática única usada en EG (véase la tabla 3.2), y su objetivo final es también la generación directa de una *netlist*.

Como su nombre indica, el diseño de circuitos es un problema de diseño que puede describirse como una jerarquía de niveles de abstracción, donde el nivel 1 contiene los componentes electrónicos a utilizar y el nivel 2 contiene los valores que definen cada componente. Adicionalmente, se definirán las gramáticas de forma que sean recursivas por la derecha, para poder utilizar la propiedad de homología definida en la sección 3.3.4. Adicionalmente, las gramáticas utilizadas serán gramáticas de bloques (véanse las secciones 3.2.1 y 3.3.4), es decir, se definirán para utilizar bloques de codones de tamaño fijo en la decodificación tanto de los componentes como de los valores de los mismos.

Gramática I: topología

El objetivo de la gramática de topología (véase la tabla 3.7) es determinar la topología del circuito como *netlist*. El símbolo de comienzo de esta gramática es el símbolo no terminal LIST, que se expande directamente como el símbolo no terminal COMPONENTS. Dicho símbolo no terminal, a su vez, se puede expandir como cualquiera de los componentes permitidos, seguido, bien por un nuevo símbolo COMPONENTS (lo que permitiría la continuación de la *netlist* mediante expansión de dicho símbolo), o bien por un símbolo no terminal END (que indicaría el final de la *netlist* e introduciría un P-point utilizando la extensión de la gramática <GEPpointMarker>).

De igual manera que ocurría con la gramática única usada en EG (véase la tabla 3.2), la gramática mostrada en la tabla 3.7, es una gramática genérica para el diseño de circuitos analógicos, por lo que debe ser ajustada ligeramente a cada problema de diseño específico al que se vaya a aplicar. Concretamente, los requisitos de diseño del circuito objetivo determinarán el tipo de componentes permitidos en los circuitos candidatos. Por ello, puede ser necesario eliminar algunos tipos de componentes mostrados en la gramática, o por el contrario, puede ser necesario añadir nuevos tipos de componentes.

Como se indicó en la sección 3.2.2, el rango del número de nodos viene determinado por la suma de dos valores: el primero viene determinado por el número de nodos de la parte fija y, el segundo, por el valor asignado al parámetro MNN. Esto se diferencia de la gramática usada en la aproximación basada en EG (véase la tabla 3.2) donde los valores posibles para la expansión del no terminal NODE se enumeraban explícitamente como opciones dentro de la regla de producción. Por el contrario,

Tabla 3.7. – Gramática I para la selección de topología del circuito en la aproximación basada en EMG (véase la sección 3.4.1 para una descripción más detallada).

S =	LIST
T =	{ "R", "C", "Q", "M", "null1", "null2", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "e", ".", "RESISTORVAL", "CAPACITORVAL", "BJTTYPE", "MOSTYPE", "CHANNELWIDTH", "ENDFIRST", end-of-line character}
N =	{ LIST, COMPONENTS, RESISTOR, CAPACITOR, BJT, MOSFET, DUMMY, NODE, END, EOL }
E =	{ <GEPpointMarker>, <GECodonValue> }
R =	*Comprende las siguientes reglas de producción:*
LIST =	COMPONENTS;
COMPONENTS =	RESISTOR, END | RESISTOR, COMPONENTS | CAPACITOR, END | CAPACITOR, COMPONENTS | BJT, END | BJT, COMPONENTS | MOSFET, END | MOSFET, COMPONENTS;
RESISTOR=	"R", NODE, NODE, DUMMY, "RESISTORVAL", EOL;
CAPACITOR =	"C", NODE, NODE, DUMMY, "CAPACITORVAL", EOL;
BJT =	"Q", NODE, NODE, NODE, "BJTTYPE", EOL;
MOSFET =	"M", NODE, NODE, NODE, "MOSTYPE", "CHANELWIDTH", EOL;
NODE =	<GECodonValue: 0, MNN + test_fixture_nodes-1 >;
DUMMY =	"null1" | "null2";
END =	"* ENDFIRST", EOL, <GEPpointMarker>;
EOL=	*carácter de fin de línea*

en la gramática de topología se utiliza la extensión <GECodonValue>, indicando el primer y último valores del rango consecutivo de valores que pueden tomar los nodos. De esta manera, se utiliza una regla de producción más compacta y sencilla.

La gramática de topología se define para utilizar bloques de tamaño fijo e igual a cuatro codones (gramática de bloques). En el caso de componentes que requieran menos de cuatro codones, se utiliza el símbolo no terminal DUMMY, ya definido en la aproximación basada en EG (véase la sección 3.2.1), que permite consumir los codones restantes para completar una secuencia de cuatro codones.

A continuación, se muestra un ejemplo de *netlist* obtenida como resultado de aplicar el proceso de decodificación utilizando esta gramática:

Q 1 2 4 BJTTYPE

R 4 0 null2 RESISTORVAL

C 4 3 null1 CAPACITORVAL

* ENDFIRST

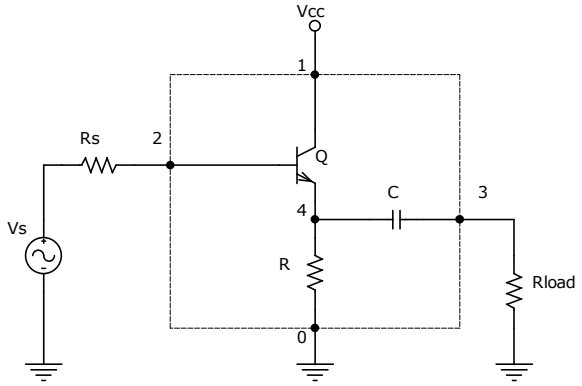


Figura 3.11. – Ejemplo de la topología de un circuito candidato (dentro del cuadrado). La parte fija corresponde al conjunto de componentes que se encuentran fuera del cuadrado.

Nótese que todos los símbolos obtenidos son símbolos terminales respecto de la gramática de topología. A diferencia de la *netlist* obtenida en la aproximación basada en EG (véase la sección 3.2.6), esta *netlist* sólo contiene los tipos de componentes y sus conexiones, es decir, sólo define la topología del circuito. La expansión de los valores de los componentes se realizará utilizando la gramática de dimensionamiento. La figura 3.11 muestra el circuito asociado a la *netlist* obtenida y su conexión a la parte fija.

Gramática II: dimensionamiento

El objetivo de la gramática de dimensionamiento (véase la tabla 3.8) es determinar el valor de los componentes obtenidos con la gramática de topología.

Para permitir que el proceso de decodificación continúe con la gramática de dimensionamiento, habrá que expandir todos aquellos símbolos que son no terminales desde el punto de vista de dicha gramática. Por ejemplo, las cadenas RESISTORVAL o CAPACITORVAL, que eran símbolos terminales para la gramática de topología, sin embargo, son símbolos no terminales en la de dimensionamiento. Como se indicó en el algoritmo 3.1, cuando hay un cambio de gramática, la expresión decodificada por la gramática i-ésima, se considera el símbolo de comienzo de la gramática $(i+1)$-ésima. En particular, en la gramática de dimensionamiento, el símbolo de comienzo es el símbolo no terminal LIST, que se expande con la extensión de la gramática <GEResult>. Esta extensión indica al algoritmo que LIST debe expandirse como la expresión final obtenida por la gramática de topología, lo que permitirá continuar

Tabla 3.8. – Gramática II para el dimensionamiento del circuito en la aproximación basada en EMG (véase la sección 3.4.1 para una descripción más detallada.

S =	LIST
T =	{ "R", "C", "Q", "M", "QNPN", "QPNP", "NMOS1 L=10u W=", "PMOS1 L=10u W=", "null1", "null2", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "e", ".", "END", end-of-line character}
N =	{ LIST, BJTTYPE, TYPE, MOSTYPE, MODELNMOS, MODELPMOS, DUMMY, NODE, RESISTORVAL, CAPACITORVAL, CHANELWIDTH, DIGIT, NONZERODIGIT, EXPONENT, ENDFIRST, EOL }
E =	{ <GEResult>, <GEPpointMarker>, <GECodonValue> }
R =	*Comprende las siguientes reglas de producción:*
LIST =	<GEResult>;
RESISTORVAL =	NONZERODIGIT, ".", DIGIT, "e", DIGIT;
CAPACITORVAL =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
BJTTYPE =	TYPE, DUMMY, DUMMY, DUMMY;
TYPE =	"QNPN" | "QPNP";
MOSTYPE =	MODELNMOS | MODELPMOS;
MODELNMOS =	"0", "NMOS1 L=10u W="
MODELPMOS =	"1", "PMOS1 L=10u W="
CHANELWIDTH =	NONZERODIGIT, DIGIT, "u" | "1", DIGIT, DIGIT, "u";
DIGIT =	<GECodonValue: 0, 9>;
NONZERODIGIT =	<GECodonValue: 1, 9>;
EXPONENT =	<GECodonValue: -12, -3>;
DUMMY =	"null1" | "null2";
ENDFIRST =	"END", <GEPpointMarker>;
EOL=	*carácter de fin de línea*

con la decodificación.

El símbolo no terminal ENDFIRST no da lugar a una parte funcional de la *netlist*. Se ha utilizado para permitir la introducción de una marca P-point, utilizando la extensión de la gramática <GEPpointMarker>, lo que nos permite marcar el punto del cromosoma en el que termina de utilizarse la gramática de topología.

Una vez más, el uso de la extensión de la gramática <GECodonValue> en las reglas de expansión de los valores de los componentes, nos permite evitar una enumeración explícita de opciones en los símbolos no terminales DIGIT, NONZERODIGIT y EXPONENT. Los valores posibles de los componentes se expanden como valores en notación científica, $m \cdot 10^e$, donde $0,1 \leq m < 10$ y el orden de magnitud del exponente, e, depende del tipo de componente, de igual manera que se hizo en la aproximación basada en EG (véase la sección 3.2.1).

Nótese que la gramática de dimensionamiento también está definida para utilizar bloques de tamaño fijo e igual a cuatro codones, mediante el uso del símbolo no terminal DUMMY cuando sea necesario. Cada bloque corresponde al valor de un componente definido en la partición de topología.

A continuación, se muestra un ejemplo de *netlist* final resultante de continuar el proceso de decodificación utilizando la gramática de dimensionamiento, a partir de la expresión decodificada por la gramática de topología:

Q 1 2 4 QNPN null1 null1 null2

R 4 0 null2 1.0e3 null1

C 4 3 null1 1.0$e-9$ null2

* END

Igual que en el caso de la aproximación basada en EG, aquí también es necesario utilizar una etapa de posprocesado (véase la sección 3.2.4), para obtener la *netlist* final que se pueda simular utilizando NGSpice. Siguiendo con el ejemplo, a continuación, se muestra la *netlist* posprocesada:

** Parte de la netlist correspondiente a la parte fija*

Vcc 1 0 dc 5

Vs 0 50 0.0 ac 0.001

Rs 50 2 1k

Rload 3 0 1k

** Netlist obtenida por el algoritmo*

Q 1 2 4 QNPN

R 4 0 1k

C 4 3 1n

* END

3.4.2. Cruce homólogo para el diseño de circuitos analógicos

Esta sección describe la particularización del operador de cruce homólogo, BG-MHX, ya definido en la sección 3.3.4, para su uso en el problema de diseño de circuitos. Así, el uso de las dos gramáticas definidas en la sección anterior da lugar a que el cromosoma quede divido en dos particiones: partición de topología y partición de dimensionamiento. El proceso de decodificación puede terminar sin haber consumido todos los codones que forman el cromosoma, por lo que puede aparecer una tercera partición que contenga los codones restantes, que son simplemente ignorados. Por tanto, cada componente de circuito se codifica en dos bloques intrahomólogos: un bloque dentro de la partición de topología, que almacena el tipo de componente y sus conexiones, y otro bloque dentro de la partición de dimensionamiento, que almacena el valor de sus parámetros. De esta manera, si la primera partición codifica un número N de componentes, la segunda partición deberá constar también de N bloques. Nótese que los bloques relacionados aparecen en el mismo orden en ambas particiones. La figura 3.12 muestra un ejemplo de la estructura del cromosoma así descrita.

El operador de cruce homólogo adaptado al diseño de circuitos permite cruzar información completa de componentes de los dos padres. La figura 3.13 muestra un ejemplo de cómo actúa el operador de cruce en el problema de diseño de circuitos sobre dos padres para producir dos hijos. El operador selecciona aleatoriamente un

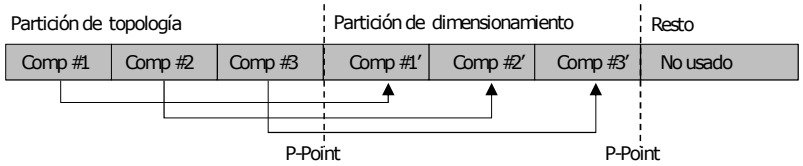


Figura 3.12. – Ejemplo de la partición del cromosoma para el problema de diseño de circuitos abordado mediante EMG, donde se observa que aparece una partición para cada gramática formada por diferentes bloques en cada partición. Nótese que cada bloque dentro de la partición de topología tiene un bloque homólogo en la partición de dimensionamiento.

punto de cruce en la partición de topología sobre cada cromosoma padre. El bloque intrahomólogo se determina de forma automática en la partición de dimensionamiento de cada padre. Por ello, cuando se seleccionan bloques para su copia desde la partición de topología de uno de los padres, también se deberán copiar los bloques intrahomólogos correspondientes de la partición de dimensionamiento. De esta manera, se copiará la información completa de los componentes (incluyendo tipo, nodos de conexión y valores) al hijo correspondiente.

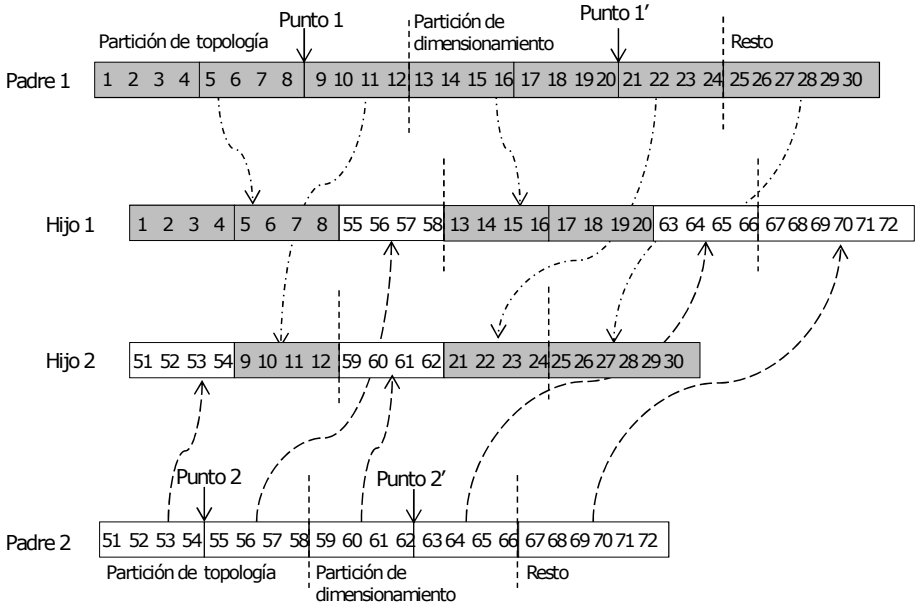


Figura 3.13. – Ejemplo de uso del operador de cruce homólogo (BG-MHX) para el problema de diseño de circuitos abordado mediante EMG. Nótese cómo la elección de los puntos 1 y 2 determinan la elección de los puntos $1'$ y $2'$ (bloques intrahomólogos), respectivamente.

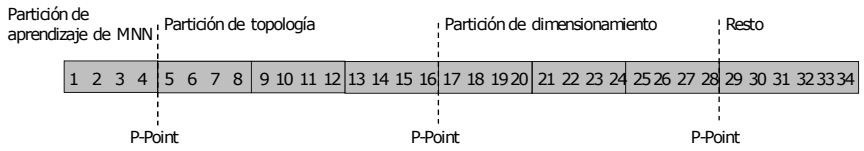


Figura 3.14. – Ejemplo de partición del cromosoma incluyendo la nueva partición de aprendizaje de parámetros de gramática, donde se codifica el parámetro MNN, para el problema de diseño de circuitos electrónicos abordado mediante EMG.

3.4.3. Aprendizaje del parámetro MNN

Como se ha visto en la sección 3.3.5, EMG puede expandirse para introducir el aprendizaje de parámetros de gramática. En particular, se aplicará esta posibilidad para el estudio de cómo aprender el parámetro de la gramática MNN. La gramática de topología mostrada en la tabla 3.9 contiene las modificaciones necesarias de la gramática descrita en la tabla 3.7 para incorporar el aprendizaje del parámetro MNN. En particular, el valor para este parámetro, se lee del cromosoma, como expansión de la regla de producción <GEDecTimeVar: %MNN>=<GECodonValue: 0,30>", que lee el primer codón del cromosoma y carga el valor obtenido en la variable de tiempo de decodificación %MNN. Esta variable se utiliza en la gramática de topología en la expansión del símbolo no terminal NODE. La variable %MNN se utiliza en una expresión <GECodonValue>. De esta manera, la gramática se parametriza mediante el uso de la variable %MNN, que contiene el valor aprendido para el parámetro MNN.

Con el objetivo de que la partición de parámetros de gramática tenga también un tamaño de bloque de cuatro codones, se utilizan tres símbolos no terminales DUMMY. De esta manera, el tamaño de bloque de cuatro codones se conserva en todo el cromosoma. El aprendizaje de parámetros de gramática se introduce al comienzo del cromosoma, como puede verse en la figura 3.14. Sin embargo, desde la perspectiva del operador de cruce homólogo, la partición de aprendizaje de parámetros de gramática se considera siempre unida a la partición correspondiente de la primera gramática, esto es, aplicará un mecanismo que evite que el punto de cruce pueda caer en la frontera entre ambas particiones. Finalmente, es importante notar que la gramática de dimensionamiento, no necesita ser modificada en este caso, por lo que es la misma que la mostrada en la tabla 3.8.

Tabla 3.9. – Gramática I para topología de circuito considerando el aprendizaje del parámetro de la gramática MNN en la aproximación basada en EMG (véase la sección 3.4.3 para una descripción detallada).

S =	LIST
T =	{ "R", "C", "Q", "M", "QNPN", "QPNP", "null1", "null2", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "e", ".", "RESISTORVAL", "CAPACITORVAL", "BJTTYPE", "MOSTYPE", "CHANNELWIDTH", "ENDFIRST", end-of-line character}
N =	{ LIST, LINE, RESISTOR, CAPACITOR, BJT, MOSFET, DUMMY, NODE, END, EOL }
E =	{ <GEPpointMarker>, <GECodonValue>, *<GEDecTimeVar: %*MNN>, %MNN }
R =	*Comprende las siguientes reglas de producción:*
LIST =	HEADER, COMPONENTS;
<GEDecTimeVar: %MNN> =	<GECodonValue: 0, 30>;
HEADER=	"* MNN:", %MNN, DUMMY, DUMMY, DUMMY, EOL, <GEPpointMarker>;
COMPONENTS =	RESISTOR, END | RESISTOR, COMPONENTS | CAPACITOR, END | CAPACITOR, COMPONENTS | BJT, END | BJT, COMPONENTS | MOSFET, END | MOSFET, COMPONENTS;
RESISTOR=	"R", NODE, NODE, DUMMY, "RESISTORVAL", EOL;
CAPACITOR =	"C", NODE, NODE, DUMMY, "CAPACITORVAL", EOL;
BJT =	"Q", NODE, NODE, NODE, "BJTTYPE", EOL;
MOSFET =	"M", NODE, NODE, NODE, "MOSTYPE", "CHANELWIDTH", EOL;
NODE =	<GECodonValue: 0, %MNN + test_fixture_nodes-1 >;
DUMMY =	"null1" | "null2";
END =	"* ENDFIRST", EOL, <GEPpointMarker>;
EOL=	*carácter de fin de línea*

3.4.4. Motor de búsqueda en MGE para el diseño de circuitos

En el diseño de circuitos electrónicos abordado mediante EMG se ha utilizado un motor de búsqueda basado en AG, donde se ha usado el operador de cruce descrito en la sección 3.4.2. Los demás operadores y parámetros del motor de búsqueda son los mismos que los utilizados en la aproximación basada en EG (véase la sección 3.2.7).

3.5. Reducción del número de componentes

Como se ha indicado en la sección 2.3, en el campo de la electrónica evolutiva se buscan soluciones que no sólo cumplan los objetivos de diseño, sino que también sean eficientes como, por ejemplo, añadiendo como objetivo adicional del proceso evolutivo que el circuito buscado contenga el menor número posible de componentes. Hasta ahora, los problemas abordados se han considerado con un único objetivo correspondiente a cumplir un juego de especificaciones. En esta sección se aborda el paso a un enfoque multiobjetivo para incorporar también el número de componentes como objetivo a minimizar.

En lugar de abordar el paso a un algoritmo basado en la dominancia de Pareto, como podría ser NSGA-II (Deb et al., 2000), se ha optado por un enfoque más simple, basado en clasificar los objetivos en dos tipos: «objetivos a satisfacer» frente a «objetivos a optimizar». De esta manera, los requisitos de las especificaciones se consideran objetivos a satisfacer, ya que deben cumplirse unos umbrales mínimos para considerar que el circuito cumple las especificaciones. Por otro lado, el objetivo de reducir el número de componentes se considera un objetivo a optimizar, ya que se busca un circuito con un número mínimo de componentes, pero que debe cumplir las especificaciones. También es posible ver este método como un mecanismo de manejo de restricciones, donde las especificaciones son restricciones a cumplir, mientras se optimiza el número de componentes.

Adicionalmente, se utilizará un enfoque basado en la aproximación presentada en (Floreano & Mattiussi, 2008). Para ello, se ha considerado una ordenación de los objetivos, siendo el objetivo de cumplimiento de las especificaciones el de mayor prioridad. La minimización del número de componentes será un objetivo de menor prioridad y se abordará únicamente cuando la solución candidata ya cumpla las

especificaciones. Para conseguir esto, se utilizará una sucesión de funciones de adaptación, que en este caso serán dos: $f_1(\boldsymbol{x})$ y $f_2(\boldsymbol{x})$, donde $\boldsymbol{x}$ representa una solución candidata en el espacio de búsqueda, $f_1(\boldsymbol{x})$representa el grado de adaptación respecto al cumplimiento de las especificaciones de diseño y, $f_2(\boldsymbol{x})$, el grado de adaptación respecto a la simplicidad del circuito. El valor de adaptación de dicha solución $\boldsymbol{x}$ vendrá indicado por la función $f_1(\boldsymbol{x})$ mientras no se alcance el valor umbral mínimo. Si la solución candidata $\boldsymbol{x}$ cumpla dicho valor umbral, su valor de adaptación vendrá determinado por la segunda función $f_2(\boldsymbol{x})$. Es importante notar que, los valores de adaptación provistos por la función $f_2(\boldsymbol{x})$ deben ser siempre menores que los de la función $f_1(\boldsymbol{x})$ para el buen funcionamiento del algoritmo. De esta manera, el problema de diseño de los circuitos se enfocará en primer lugar en conseguir circuitos que cumplan las especificaciones y, una vez se consiga esto, el algoritmo se centrará en la reducción del número de componentes, pero siempre manteniendo el cumplimiento de dichas especificaciones.

4. Casos de estudio

En este capítulo se describen los casos de estudio que se abordarán usando las aproximaciones propuestas en el capítulo 3. En particular, se describen las especificaciones de los circuitos a sintetizar. Estos circuitos fueron definidos por Koza (Koza et al., 1999a) y han sido utilizados en trabajos sucesivos, por lo que se han establecido como circuitos de referencia o *benchmark*. Los circuitos que se abordarán pueden agruparse en dos conjuntos diferentes. El primer conjunto consta de tres circuitos: un circuito sensor de temperatura, un circuito de referencia de voltaje y un circuito de generación de función gaussiana. El segundo conjunto consta de cuatro circuitos que implementan varias funciones matemáticas: potencia al cuadrado, potencia al cubo, raíz cuadrada y raíz cúbica. Nos referiremos al primer conjunto como circuitos no computacionales y, al segundo, como circuitos computacionales. Esta división nos permitirá hacer comparativas con trabajos previos y relacionados que se centraron en el diseño de alguno de estos conjuntos.

La sección 4.1 describe las especificaciones del conjunto de circuitos no computacionales. La sección 4.2 describe las especificaciones del conjunto de circuitos computacionales.

4.1. Especificaciones de los circuitos no computacionales

En esta sección se describen las especificaciones del conjunto de circuitos no computacionales, que consta de tres circuitos: un circuito sensor de temperatura, un circuito de referencia de voltaje y un circuito de generación de función gaussiana.

4.1.1. Circuito sensor de temperatura

Este circuito toma su propia temperatura como entrada y entrega como salida un voltaje proporcional al valor de la temperatura. El rango de temperatura de entrada es de $0°C \leq T \leq 100°C$ y el rango de voltaje de salida debe ser de $0V \leq V_o \leq 10V$. Los circuitos candidatos se simulan mediante un barrido de temperatura en el rango anteriormente mencionado, a intervalos de $5°C$, lo que corresponde a 21 puntos de ajuste. En cada punto de ajuste, se compara la salida deseada V_{o_i} con la salida obtenida $\widetilde{V}_{o_i}$. Si la diferencia absoluta de dichos valores es menor que un umbral de $0.1V$, se dice que se alcanza un *hit*. Alcanzar un *hit* significa que la salida obtenida es suficientemente cercana a la salida deseada en el punto de ajuste. Por ejemplo, un circuito que consiga 15 *hits* sobre el total de 21 puntos de ajuste, su ratio de *hits* se dice que es 15/21. Un circuito se considera exitoso cuando consigue un *hit* en todos los puntos de ajuste.

La función de adaptación se calcula mediante la expresión (4.1), donde los factores de peso, w_i, se calculan mediante la expresión (4.2), dependiendo de si se ha conseguido o no un *hit* en el punto de ajuste considerado.

$$fitness = \sum_i w_i |V_{o_i} - \widetilde{V}_{o_i}| \tag{4.1}$$

$$w_i = \begin{cases} 1.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| \leq 0.1V \\ 10.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| > 0.1V \end{cases} \tag{4.2}$$

Los tipos de componentes utilizados en este circuito son transistores BJT (2N3904 y 2N3906), resistencias y condensadores. La parte fija consta de cuatro nodos accesibles: dos fuentes de alimentación ($+15V$ y $-15V$), una resistencia de carga ($1K\Omega$) y masa.

4.1.2. Circuito de referencia de voltaje

El objetivo de este circuito es proporcionar una salida de voltaje fijo de $2V$ cuando recibe una entrada en el rango de voltaje $4V \leq V_i \leq 6V$ y, también, considerando

un rango de temperatura $0°C \leq T \leq 100°C$.

Los circuitos candidatos se simulan con un barrido de voltaje dentro del rango considerado, a intervalos de $0.1V$, y con un barrido de temperaturas a intervalos de $25°C$. El resultado de ambos barridos dar lugar a 105 (21×5) puntos de ajuste. Cuando el valor absoluto de la diferencia entre el voltaje deseado, V_{o_i}, y el voltaje obtenido, $\widetilde{V}_{o_i}$, es menor o igual que un umbral de $0.02V$, se alcanza un *hit* en dicho punto. La función de adaptación se calcula mediante la expresión (4.3), donde los factores de peso, w_i, se obtienen mediante la expresión (4.4), dependiendo de si se ha conseguido un *hit* o no en el punto de ajuste considerado. Un circuito se considera exitoso cuando consigue un *hit* en todos los puntos de ajuste.

$$fitness = \sum_i w_i |V_{o_i} - \widetilde{V}_{o_i}| \tag{4.3}$$

$$w_i = \begin{cases} 1.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| \leq 0.02V \\ 10.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| > 0.02V \end{cases} \tag{4.4}$$

Los tipos de componentes utilizados en este circuito son transistores BJT (2N3904 y 2N3906) y resistencias. La parte fija consta de tres nodos accesibles: una fuente de alimentación (con una resistencia serie de $1K\Omega$), una resistencia de salida ($10K\Omega$), y masa.

4.1.3. Circuito generador de función gaussiana

El objetivo de este circuito es generar una corriente de salida que sea una función gaussiana del voltaje de entrada. La corriente de salida se mide en un generador de voltaje conectado a la salida. Este problema se atribuye a Adrian Stoica del Jet Propulsion Laboratory de Pasadena, California (Koza et al., 1999a). El voltaje de entrada varía en el intervalo $2V \leq V_i \leq 3V$. La función gaussiana de salida queda definida mediante una media que se alcanza cuando la entrada vale $2.5V$ y con una desviación estándar de $0.1V$. El pico de la corriente de salida es de $80nA$. Los circuitos candidatos se simulan con un barrido de voltaje DC en el rango anteriormente mencionado a intervalos de $0.01V$, lo que corresponde a un total de

101 puntos de ajuste. Cuando el valor absoluto de la diferencia entre la corriente de salida esperada, I_{o_i}, y la corriente de salida obtenida , $\widetilde{I}_{o_i}$, es menor que un umbral de $5nA$, se alcanza un *hit* en dicho punto. La función de adaptación se calcula mediante la expresión (4.5), donde los factores de peso, w_i, se obtienen mediante la expresión (4.6), dependiendo de si se ha conseguido un hit o no en el punto de ajuste considerado. Un circuito se considera exitoso cuando consigue un hit en todos los puntos de ajuste.

$$fitness = \sum_i w_i |I_{o_i} - \widetilde{I}_{o_i}| \tag{4.5}$$

$$w_i = \begin{cases} 10^6 & si\ |I_{o_i} - \widetilde{I}_{o_i}| \leq 5nA \\ 10^7 & si\ |I_{o_i} - \widetilde{I}_{o_i}| > 5nA \end{cases} \tag{4.6}$$

Los tipos de componente utilizados en este circuito son transistores MOSFET de canal N y canal P, y resistencias. La parte fija consta de cuatro nodos accesibles: una fuente de alimentación ($+5V$), un generador de señal ($2-3V$) con una resistencia interna en serie de 1Ω, un generador de voltaje conectado a la salida y masa.

4.2. Especificaciones de los circuitos computacionales

El conjunto de circuitos computacionales genera diferentes funciones matemáticas que dependen del voltaje de entrada. Aunque los cuatro circuitos utilizados cumplen las mismas especificaciones de circuito, el mismo tipo de componentes y la misma función de adaptación que las propuestas por Koza (Koza et al., 1999a), aquí propondremos tres variantes del análisis de los mismos.

4.2.1. Especificaciones generales de los circuitos computacionales

El conjunto de circuitos computacionales consta de cuatro circuitos que calculan una función matemática del voltaje de entrada. Las funciones utilizadas son: potencia al cuadrado, potencia al cubo, raíz cuadrada y raíz cúbica. El voltaje de entrada varía en el intervalo $-250mV$ to $250mV$, con excepción del circuito de raíz cuadrada, en el que varía en el intervalo $0V$ y $500mV$. Los circuitos candidatos se simulan con un barrido de voltaje en los rangos mencionados, a intervalos de $25mV$ dando lugar a un total de 21 puntos de ajuste para estos circuitos. Cuando el valor absoluto de la diferencia entre el voltaje de salida esperado, V_{o_i}, y el voltaje de salida obtenido, $\widetilde{V}_{o_i}$, es menor o igual que un umbral porcentaje del valor de voltaje esperado (fijado en $1\,\%V_{o_i}$), se alcanza un *hit* en dicho punto. La función de adaptación se calcula mediante la expresión (4.7), donde los factores de peso w_i, se obtienen mediante la expresión (4.8), dependiendo de si se ha conseguido un hit o no en el punto de ajuste considerado. Un circuito se considera exitoso cuando consigue un hit en todos los puntos de ajuste.

$$fitness = \sum_i w_i |V_{o_i} - \widetilde{V}_{o_i}| \tag{4.7}$$

$$w_i = \begin{cases} 1.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| \leq 1\,\%V_{o_i} \\ 10.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| > 1\,\%V_{o_i} \end{cases} \tag{4.8}$$

Los tipos de componentes utilizados en el diseño de este circuito son transistores BJT (2N3904 y 2N3906), y resistencias. La parte fija consta de cinco nodos accesibles: dos fuentes de alimentación ($+15V$ y $-15V$), un generador de señal de entrada (con una resistencia serie de $1K\Omega$), una resistencia de carga ($1K\Omega$), y masa.

4.2.2. Formas de evaluar los circuitos computacionales

En el caso de los circuitos computacionales, han surgido diversas consideraciones que han dado lugar a la introducción de variantes en la forma en la que se han

evaluado. En este apartado se indican las tres variantes de formas de evaluación de dichos circuitos.

Primera variante

En esta primera variante del análisis de estos circuitos, la simulación de los circuitos candidatos se realiza como en el caso de los no computacionales, es decir, mediante un análisis basado en un barrido DC del simulador, acorde a la definición original (Koza et al., 1999a).

Segunda variante

El análisis mediante barrido DC, como su nombre indica, es un análisis en continua para un voltaje de entrada fija. Estas condiciones no se corresponden con un funcionamiento en condiciones reales de un circuito computacional, en el que se espera introducir una señal de entrada variable para calcular su correspondiente valor de salida. Sobre este tipo de análisis se ha dicho que no conduce necesariamente a circuitos robustos que ofrezcan una función matemática correcta en el tiempo (Mydlowec & Koza, 2000), siendo preferible un análisis transitorio del circuito candidato para obtener circuitos más robustos que los obtenidos con el análisis DC, aunque sea necesario un tiempo mayor de computación (Sapargaliyev & Kalganova, 2012).

Por tanto, para conseguir un funcionamiento más robusto, se realiza la simulación de cada circuito mediante una rampa de entrada (Sapargaliyev & Kalganova, 2012; Mydlowec & Koza, 2000), con un tiempo total de subida de $200ms$. El voltaje de salida se mide en intervalos de $10ms$, lo que corresponde a incrementos de $25mV$, dando lugar a un total de 21 puntos de ajuste. Esta simulación se realiza mediante un análisis transitorio (*transient,* en inglés) del simulador. Esta nueva simulación mantiene las especificaciones iniciales de cada problema, pero introduce una exigencia adicional de comportamiento en frecuencia.

Tercera variante

Las condiciones de éxito de los circuitos computacionales anteriormente definidas resultan ser muy estrictas, por lo que puede ser muy difícil conseguir muchos circuitos que las cumplan. Con el objetivo de poder comparar estadísticamente el rendimiento

de distintos algoritmos sobre este conjunto de circuitos, en algunos experimentos se ha optado por suavizar estas condiciones, obteniendo así tasas de éxito más elevadas.

En esta variante, cuando el valor absoluto de la diferencia entre el voltaje de salida esperado, V_{o_i}, y el voltaje de salida obtenido, $\widetilde{V}_{o_i}$, es menor o igual que un 5 % del valor máximo del voltaje de salida esperado, $max(V_{o_i})$, se considera que alcanza un *hit* en el punto de ajuste considerado.

Esta variante parte de las especificaciones de la segunda variante, por lo que se utiliza también un análisis transitorio del simulador.

La función de adaptación sigue siendo la indicada en la expresión (4.7), pero con los factores de peso obtenidos mediante la expresión (4.9), cuyo valor depende de si se ha conseguido un hit o no en el punto de ajuste considerado.

$$w_i = \begin{cases} 1.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| \leq 5\,\%\ max(V_{o_i}) \\ 10.0 & si\ |V_{o_i} - \widetilde{V}_{o_i}| > 5\,\%\ max(V_{o_i}) \end{cases} \tag{4.9}$$

4.3. Parsimonia simple en MGE

En esta sección se presenta un enfoque multiobjetivo, denominado *parsimonia simple*, que fue definido en la sección 3.5, con el propósito de reducir el número de componentes del circuito a diseñar. Para ello, recordemos que se utilizan dos funciones de adaptación diferentes y que, para obtener el grado de adaptación de un individuo, se utiliza una u otra, dependiendo de si, se alcanzan o no, las especificaciones de diseño.

Las dos funciones se definen de la siguiente forma: $f_1(\widetilde{V}_{o_i})$ y $f_2(ncomp)$, donde la función f_1 será igual a la definida anteriormente para los circuitos *benchmark* propuestos (véanse las expresiones (4.1), (4.3), (4.5) y (4.7)), y tiene como objetivo que los circuitos cumplan la funcionalidad establecida en las especificaciones de los mismos. Por otra parte, la función f_2 tiene como objetivo reducir el número de componentes, por lo que se propone una función inversamente proporcional al número de componentes del circuito, *ncomp*. Mientras un circuito no cumpla las especificaciones de diseño, será evaluado mediante la función f_1. En otro caso, es decir, cuando el circuito sea considerado como exitoso, entonces pasa a ser evaluado por la función f_2, con el objeto de reducir ahora el número de componentes.

La expresión (4.10) formaliza matemáticamente el proceso de seleccionar en cada caso la función de adaptación adecuada, donde los factores de peso, w_i, se siguen calculando igualmente a como se realizaba en el circuito inicial sin parsimonia, k es un factor de escala y *maxcomp* es el máximo número de componentes que se pueden representar en un cromosoma teniendo en cuenta el máximo *wrapping* permitido.

$$fitness = \begin{cases} f_1(\widetilde{V}_{o_i}) = \sum_i w_i |V_{o_i} - \widetilde{V}_{o_i}| & si\ n^o\ de\ hits\ no\ es\ máximo \\ f_2(ncomp) = k \cdot min(ncomp, maxcomp) & en\ otro\ caso \end{cases} \tag{4.10}$$

Es importante que los valores obtenidos por la función de adaptación cuando se alcanza el máximo número de hits sean siempre menores que los obtenidos cuando se minimiza el número de componentes. Si los valores se solaparan, no se conseguiría el efecto buscado. Para ello, se ajusta el valor del factor k para que f_2 en un circuito que esté formado por un numero de componentes igual a *maxcomp*, tome el valor mínimo de f_1. Se puede comprobar que dicho valor mínimo ocurre para el caso en que todas las diferencias absolutas incluidas en la función f_1 valgan cero, menos un caso en el que la diferencia sea justo el valor umbral (realmente debe ser justo un poco mayor, para que no se cumpla la condición del *hit* en dicho punto). De esta manera, para ajustar el valor de k se utiliza la expresión 4.11, donde $max(w_i) = 10,0$ en todos los circuitos diseñados y *umbralhit* es el valor umbral para alcanzar un hit en la especificación del circuito objetivo.

$$k = \frac{max(w_i) \times umbralhit}{maxcomp} \tag{4.11}$$

De esta manera, los valores de adaptación mayores que k corresponderán a circuitos no exitosos y, valores menores, a circuitos exitosos, pero para los que se persigue reducir su número de componentes.

5. Resultados y discusión

Este capítulo se centra en los resultados de los experimentos realizados con los dos algoritmos propuestos y descritos en el capítulo 3, incluyendo un análisis de los mismos. De aquí en adelante, se denominará ACID-GE (*Analog Circuit Design based on Grammatical Evolution*) a la implementación del enfoque basado en EG y ACID-MGE (*Analog Circuit Design based on Multi-Grammatical Evolution*) a la implementación del enfoque basado en EMG.

La sección 5.1 describe los experimentos realizados. La sección 5.2 indica los parámetros de configuración de los dos algoritmos implementados. La sección 5.3 define las medidas de prestaciones que se utilizarán para evaluar los resultados de los experimentos. En la sección 5.4, se muestran y discuten los resultados utilizando ACID-GE, realizando una comparativa con trabajos previos. En la sección 5.5, se muestran y discuten los resultados utilizando ACID-MGE y diferentes variantes sobre el mismo, realizando una comparativa con los resultados obtenidos con ACID-GE. Finalmente, en la sección 5.6, se muestran los mejores circuitos obtenidos con ambas aproximaciones.

5.1. Descripción de los experimentos realizados

Los experimentos realizados sobre los circuitos *benchmark,* descritos en el capítulo 4, se agruparon en cuatro grupos y se describen a continuación.

En el primer juego de experimentos, se sintetizaron los circuitos *benchmark* mediante el algoritmo ACID-GE, utilizando gramáticas (véase el apéndice D.1) adaptadas a partir de la gramática general mostrada en la tabla 3.2 (véase la sección 3.2.1). En particular, en los experimentos realizados sobre circuitos no computacionales, se realizó un análisis estadístico del rendimiento del algoritmo ACID-GE, probando configuraciones alternativas del número de generaciones y del valor del parámetro

MNN. Una vez probado que el algoritmo implementado produce tasas de éxito razonables en el caso de los circuitos no computacionales, se hizo uso de las configuraciones exitosas obtenidas y se aplicó el algoritmo al caso de los circuitos computaciones, pero ahora focalizando más en el propósito de obtener circuitos exitosos que en analizar el comportamiento estadístico de dicho algoritmo. Adicionalmente, se realizó una comparativa de estos resultados con resultados obtenidos en trabajos previos en la literatura.

El segundo juego de experimentos se realizó sobre ACID-MGE, con gramáticas (véase el apéndice D.2) derivadas de las mostradas en las tablas 3.7 y 3.8 (véase la sección 3.4.1). Se realizó un análisis estadístico para la síntesis de todos los circuitos de estudio y una comparativa con el algoritmo ACID-GE.

En el tercer juego de experimentos se estudió la característica opcional de EMG relacionada con el aprendizaje de parámetros de gramática, focalizando sobre el aprendizaje del parámetro MNN. Finalmente, se realizó una comparación de los resultados aquí obtenidos con ACID-MGE sin aprendizaje sin parámetros.

Finalmente, el cuarto juego de experimentos analizó la incorporación del mecanismo de parsimonia simple sobre ACID-MGE. Los resultados obtenidos fueron comparados con los conseguidos por a ACID-MGE.

Debido al carácter estocástico de los algoritmos evolutivos, cada experimento implicó la realización de 50 ejecuciones del algoritmo objeto de estudio. La elección de este número de ejecuciones supone un compromiso entre tiempo de ejecución y consideraciones de potencia estadística (véase el apéndice E).

5.2. Configuración de parámetros

En esta sección se describen los parámetros de configuración usados con los algoritmos ACID-GE y ACID-MGE en la realización de los experimentos descritos en la sección 5.1. En particular, en la sección 5.2.1, se describen los parámetros de configuración del algoritmo ACID-GE y, en la sección 5.2.2, los parámetros de configuración del algoritmo ACID-MGE.

5.2.1. Parámetros de configuración en ACID-GE

Las gramáticas utilizadas en los experimentos con ACID-GE se han derivado de la gramática general mostrada en la tabla 3.2. La adaptación consistió en eliminar de dicha gramática los componentes no utilizados en el circuito objetivo y en modificar convenientemente la regla de producción del símbolo no terminal COMPONENTS. Adicionalmente, también se modificó la regla de producción del símbolo no terminal NODE para poder generar el rango apropiado de números de nodos, teniendo en cuenta el MNN del experimento a realizar. Las gramáticas utilizadas con ACID-GE en la síntesis de los diferentes circuitos *benchmark*, se incluyen en el apéndice D.1.

La tabla 5.1 muestra los parámetros de configuración utilizados con ACID-GE para la síntesis de los circuitos computacionales y no computacionales. Los valores de estos parámetros se han ajustado experimentalmente, mediante el análisis de resultados obtenidos de las ejecuciones preliminares sobre el conjunto de circuitos no computacionales.

En particular, en relación con el parámetro de máximo *wrapping*, hay evidencia en la literatura de que la falta de uso de *wrapping* conlleva una degradación del rendimiento (O'Neill & Ryan, 2003). Sin embargo, no hay una regla orientativa para un ajuste óptimo de este parámetro. Este parámetro se sintonizó con diferentes valores, sin obtener diferencias significativas de rendimiento, por lo que finalmente, se dejó un valor de 4, con objeto de impedir la aparición de un alto número de individuos inexpresables en las primeras generaciones.

5.2.2. Parámetros de configuración en ACID-MGE

Las gramáticas de topología utilizadas en los experimentos con ACID-MGE se han derivado de las gramáticas generales mostradas en las tablas 3.7 y 3.9, dependiendo del no uso o sí uso de aprendizaje del parámetro de gramática MNN, respectivamente. Las gramáticas de dimensionamiento utilizan siempre la mostrada en la tabla 3.8, dado que ésta no requiere ser adaptada para cada circuito analizado. La adaptación de la gramática de topología consistió en eliminar los componentes no utilizados en el circuito objetivo y, por tanto, en modificar convenientemente la regla de producción del símbolo no terminal COMPONENTS. La modificación de la regla de producción del símbolo no terminal NODE, para poder sintonizar el rango apropiado de números de nodos sólo fue necesaria cuando se utiliza ACID-MGE sin aprendizaje (véase

Tabla 5.1. – Algoritmo ACID-GE: parámetros de configuración.

Circuito objetivo	Sensor de temperatura (ST), referencia de voltaje (RV), función gaussiana (FG) o circuitos computacionales (CC)
Gramática	Gramática genérica (véase la tabla 3.2) adaptada al circuito objetivo (véase la sección 5.2.1)
Motor de búsqueda	AG con cromosomas de tamaño variable
Función de adaptación	Dependiente del circuito objetivo (véanse las secciones 4.1 y 4.2)
Condición de éxito	Cuando el circuito cumple todos los *hits* (véanse las secciones 4.1 y 4.2)
Tamaño de la población (μ)	1 000
Representación	Cadenas de codones de longitud variable
Generaciones	3 000
Inicialización	Cadenas de codones aleatorias de longitud en el rango $150 - 250$
Longitud máxima del cromosoma	294 (ST), 294 (RV), 336 (FG) y 294 (CC) codones
Operador de cruce	Operador *one-block* (véase la sección 3.2.7)
Tasa de cruce	0.5
Tamaño de bloque (componente)	7 (TS), 7 (VR), 8 (GF) y 7 (CC) codones
Operador de mutación	Operador de mutación *Bitwise* (véase la sección 3.2.7)
Tasa de mutación	0.001
Selección de padres	Selección por torneo (tamaño=3)
Selección de supervivientes	Reemplazo generacional ($\lambda = \mu$)
Elitismo	2
Wrapping	4
Condición de terminación	Máximo número de generaciones
Generador de números aleatorios	Mersenne *twister*
Ejecuciones por experimento	50

la tabla 3.7), dado que cuando se usa aprendizaje el número de nodos es adaptado automáticamente. Las gramáticas utilizadas con ACID-MGE para los diferentes circuitos *benchmark*, considerando los casos de ausencia o presencia del mecanismo de aprendizaje de parámetros de gramática, se muestran en los apéndices D.2 y D.3, respectivamente.

La tabla 5.2 muestra los parámetros de configuración utilizados con ACID-MGE para la síntesis de los circuitos computacionales y no computacionales. Tal y como puede verse, comparando esta tabla con la 5.1, para poder hacer una comparación justa entre ACID_MGE y ACID-GE, los valores de los parámetros se mantuvieron iguales, con la excepción del uso del operador de cruce *one-block*, que fue usado en ACID-GE, y el operador de cruce homólogo BG-MHX, que fue usado en ACID-MGE. Respecto al parámetro longitud máxima de cromosoma, hay que tener en cuenta que, aunque los valores utilizados en cada tipo de algoritmo sean diferentes, en realidad son equivalentes. Esto es así porque, en ambos algoritmos, se ha dimensionado la longitud máxima del cromosoma para permitir la codificación de un máximo de 42 componentes sin utilizar *wrapping*. Atendiendo ahora, al tamaño de bloque, nos encontramos con que ACID-GE utiliza bloques de 7 codones, con excepción del circuito de función gaussiana que los usa de 8, así como que ACID-MGE utiliza bloques de 4 codones en todos los casos, si bien son necesarios dos bloques para codificar un componente completo. Con ello, la longitud máxima para ACID-GE será de 294 codones, con excepción del circuito de función gaussiana, que será de 336 codones y, para ACID-MGE, será de 336 codones, en todos los casos.

5.3. Medidas de prestaciones de un algoritmo evolutivo

En esta sección se definen varios índices de medidas que permitan evaluar las prestaciones de un AE y realizar una comparativa con otros algoritmos. Los índices de medidas que más se utilizarán son la tasa de éxito (SR, por sus siglas en inglés) y la mejor adaptación media (MBF, por sus siglas en inglés). Dado un experimento compuesto de un número de ejecuciones del algoritmo, el valor de SR se define como la relación entre el número de ejecuciones exitosas sobre el total de ejecuciones. Se conseguirá un éxito, en nuestro caso, cuando un circuito alcance todos los *hits* (véanse las secciones 4.1 y 4.2). Por otro lado, el valor de MBF corresponde al valor medio

Tabla 5.2. – Algoritmo ACID-MGE: parámetros de configuración.

Circuito objetivo	Sensor de temperatura, referencia de voltaje, función gaussiana o circuitos computacionales
Gramáticas	Conjunto de dos gramáticas mostradas en las tablas 3.7, 3.8 o 3.9 adaptadas a cada circuito objetivo. (véase la sección 5.2.1)
Motor de búsqueda	AG con cromosomas de tamaño variable
Función de adaptación	Dependiente del circuito objetivo (véanse las secciones 4.1 y 4.2)
Condición de éxito	Cuando el circuito cumple todos los *hits* (véanse las secciones 4.1 y 4.2)
Tamaño de la población (μ)	1 000
Representación	Cadena de codones de longitud variable
Número máximo de generaciones	3 000
Initialización	Cadenas de codones aleatorias de longitud en el rango $150 - 250$
Longitud máxima del cromosoma	336 codones
Operador de cruce	Operador BG-MHX (véase la sección 3.4.2)
Probabilidad de cruce	0.5
Tamaño de bloque	4 codones
Operador de mutación	Operador de mutación *Bitwise* (véase la sección 3.2.7)
Probabilidad de mutación	0.001
Selección de padres	Selección por torneo (tamaño=3)
Reemplazo	Reemplazo generacional ($\lambda = \mu$)
Elitismo	2
Wrapping	4
Condición de terminación	Número máximo de generaciones
Generador de números aleatorios	Mersenne *twister*
Número de ejecuciones por experimento	50

de la adaptación de los mejores circuitos obtenidos en la última generación de cada ejecución del algoritmo, tras agotar el máximo número de generaciones establecido o haber alcanzado la condición de terminación. Una tercera medida, muy apropiada en un problema de diseño, como es el caso que nos ocupa, es la mejor adaptación obtenida, que se corresponde con el mínimo de las mejores adaptaciones (minBF, por sus siglas en inglés). Este valor corresponde al circuito que mejor cumple los criterios definidos, es decir, es el mejor circuito obtenido en un experimento.

Adicionalmente, se utilizarán dos índices relacionados con la velocidad de convergencia del algoritmo: número medio de generaciones para obtener un éxito y número medio de evaluaciones para obtener un éxito (AES, por sus siglas en inglés). Ambos índices tienen en cuenta únicamente las ejecuciones exitosas en un experimento. En general, valores más bajos en dichos índices corresponden a menor coste computacional y a mayor velocidad de convergencia del algoritmo.

5.4. Resultados de experimentos con ACID-GE

En esta sección se muestran los resultados de los experimentos utilizando la aproximación ACID-GE, y también su comparación con los resultados de trabajos previos en la literatura.

Para el conjunto de circuitos no computacionales, se realiza un análisis estadístico del rendimiento del algoritmo evolutivo, que incluye un estudio del impacto de dos de sus parámetros sobre el rendimiento: el número máximo de generaciones y el máximo número de nodos (MNN). En el caso de los circuitos computacionales, se utilizan los resultados del estudio estadístico previo para sintonizar adecuadamente el algoritmo. Para ambos tipos de circuitos se mostrará también la salida medida en el mejor circuito obtenido y se comparará con la esperada.

Adicionalmente, el esquema de los mejores circuitos obtenidos se muestran en la sección 5.6.1 y sus *netlists* asociadas se muestran en el apéndice C.1.

5.4.1. Circuitos no computacionales

La tabla 5.3 muestra los resultados de los experimentos sobre los circuitos no computacionales para diferentes valores del parámetro MNN. Como se puede comprobar,

Tabla 5.3. – Algoritmo ACID-GE: resultados obtenidos en los experimentos sobre los circuitos no computacionales, dependiendo del parámetro MNN y con 3 000 generaciones. Se indica el número de éxitos, la tasa de éxitos SR, el valor mínimo de las mejores adaptaciones minBF y el valor medio de las mismas MBF.

Objetivo	MNN	N.º de éxitos	SR (%)	minBF	MBF±SD	Hits±SD / max
sensor	4	21	42.0	0.176	6.559 ± 11.196	19.2 ± 2.6 / 21
sensor	6	21	42.0	0.169	5.294 ± 6.526	19.5 ± 1.7 / 21
sensor	8	24	48.0	0.172	3.653 ± 4.256	19.9 ± 1.4 / 21
sensor	10	26	**52.0**	0.140	2.628 ± 2.942	20.2 ± 1.1 / 21
sensor	15	22	44.0	0.140	4.574 ± 5.326	19.7 ± 1.4 / 21
sensor	20	16	32.0	0.286	5.849 ± 7.007	19.3 ± 2.1 / 21
sensor	30	13	26.0	0.066	6.328 ± 8.598	19.3 ± 1.9 / 21
vref	4	1	2.0	0.773	33.756 ± 21.167	49.3 ± 27.4 / 105
vref	6	4	8.0	0.112	26.567 ± 16.228	56.5 ± 26.3 / 105
vref	8	5	10.0	0.183	18.145 ± 13.498	68.8 ± 23.3 / 105
vref	10	13	**26.0**	0.125	14.093 ± 15.096	78.0 ± 26.5 / 105
vref	15	6	12.0	0.231	20.429 ± 16.451	65.3 ± 26.1 / 105
vref	20	5	10.0	0.171	17.955 ± 17.072	71.5 ± 25.9 / 105
vref	30	6	12.0	0.309	18.255 ± 16.550	70.0 ± 27.5 / 105
gauss	4	6	12.0	0.078	10.632 ± 6.990	$69,6 \pm 15.2$ / 101
gauss	6	12	24.0	0.040	6.702 ± 6.782	78.1 ± 17.8 / 101
gauss	8	8	16.0	0.068	6.645 ± 6.372	77.4 ± 15.9 / 101
gauss	10	12	24.0	0.047	5.945 ± 6.276	78.9 ± 18.2 / 101
gauss	15	13	**26.0**	0.030	6.474 ± 6.682	79.8 ± 16.8 / 101
gauss	20	13	**26.0**	0.041	6.856 ± 7.230	80.3 ± 17.5 / 101
gauss	30	8	16.0	0.060	7.649 ± 6.674	74.9 ± 17.7 / 101

los valores de SR y MBF dependen fuertemente del circuito objetivo, mostrando que algunos circuitos son más difíciles de sintetizar que otros. Sin embargo, es importante indicar que en un enfoque de diseño, el objetivo principal es conseguir un algoritmo que sea capaz de conseguir una buena solución al menos una vez (Eiben & Smith, 2003). Esto vendrá indicado por el valor de la tasa de éxitos, SR, donde se puede comprobar que se obtienen circuitos exitosos en todos los experimentos realizados.

Efecto del parámetro MNN

La figura 5.1 muestra el efecto del parámetro MNN sobre el SR, para los tres circuitos estudiados. Como puede comprobarse, en cada caso aparece un valor en el que el SR

alcanza su valor máximo. Para ser más precisos, en el caso del circuito de función gaussiana aparece una meseta para el rango $MNN = 10 - 20$ y en el caso de los circuitos sensor de temperatura y de referencia de voltaje, se alcanza el valor de pico de SR para $MNN = 10$.

Dado que el valor óptimo de MNN no se puede predecir de antemano para el diseño de un circuito dado, se puede adoptar la siguiente estrategia para ajustar el valor óptimo de dicho parámetro: primero se empieza con un valor pequeño y se realizan varias ejecuciones del algoritmo, analizando el porcentaje de *hits* obtenidos. Por último, se incrementa progresivamente el valor de MNN hasta encontrar un valor para el que el número de *hits* del circuito obtenido sea máximo.

El número de componentes del circuito también se ve afectado por el valor de MNN. Esto se puede comprobar en la figura 5.2 que muestra el número medio de componentes de los circuitos exitosos obtenidos en cada experimento frente al valor de MNN. Nótese que las ejecuciones no exitosas no se tienen en cuenta en esta gráfica. En el cálculo del número medio de componentes, se han ignorado los componentes de la parte fija y no se ha realizado ninguna simplificación de componentes (por ejemplo, sustitución de un grupo de resistencias por su resistencia equivalente). Puede observarse que el número de componentes alcanza un valor mínimo para los valores de MNN 4 y 8 en los circuitos de referencia de voltaje y de función gaussiana, respectivamente. Por otro lado, en el caso del circuito sensor de temperatura, aparece una cierta meseta para valores de MNN entre 10 y 20.

Efecto del número máximo de generaciones

Para estudiar el impacto del número máximo de generaciones sobre el rendimiento, se incrementa su número de 3 000 a 10 000. Los resultados se muestran en la tabla 5.4. Como puede comprobarse, en todos lo casos, los valores de SR, MBF y minBF mejoran frente el caso con 3 000 generaciones. Aunque, esta mejora se hace a expensas de un incremento del coste computacional, se puede afirmar que el incremento del número de generaciones permite obtener mejores circuitos.

Tabla 5.4. – Algoritmo ACID-GE: resultados obtenidos en los experimentos sobre los circuitos no computacionales, dependiendo del número de generaciones y con un valor de $MNN = 6$. Se indica el número de éxitos, la tasa de éxitos SR, el valor mínimo de las mejores adaptaciones minBF y el valor medio de las mismas MBF.

Target	N.º gen.	N.º de éxitos	SR (%)	minBF	MBF±SD	Hits±SD / max
sensor	10 000	29	**58.0**	0.065	3.881 ± 5.936	19.9 ± 1.8 / 21
sensor	3 000	21	42.0	0.169	5.294 ± 6.526	19.5 ± 1.7 / 21
vref	10 000	7	**14.0**	0.188	17.353 ± 14.588	70.7 ± 25.0 / 105
vref	3 000	4	8.0	0.112	26.567 ± 16.228	56.5 ± 26.3 / 105
gauss	10 000	22	**44.0**	0.036	3.943 ± 5.167	85.0 ± 17.1 / 101
gauss	3 000	12	24.0	0.040	6.702 ± 6.782	78.1 ± 17.8 / 101

Salidas obtenidas por los mejores circuitos

Finalmente, se comparan las salidas esperadas y las medidas en los mejores circuitos obtenidos. Concretamente en las figuras 5.3a, 5.3b y 5.3c, se muestran el voltaje de salida del mejor circuito sensor de temperatura, el voltaje de salida del mejor circuito de referencia de voltaje, y la corriente de salida del mejor circuito de generación gaussiana, respectivamente. Como se puede observar, en los tres casos se consigue un buen ajuste. Los esquemas de los circuitos se muestran en las figuras 5.10, 5.11 y 5.12, respectivamente, y sus correspondientes *netlists*, en el apéndice C.1.

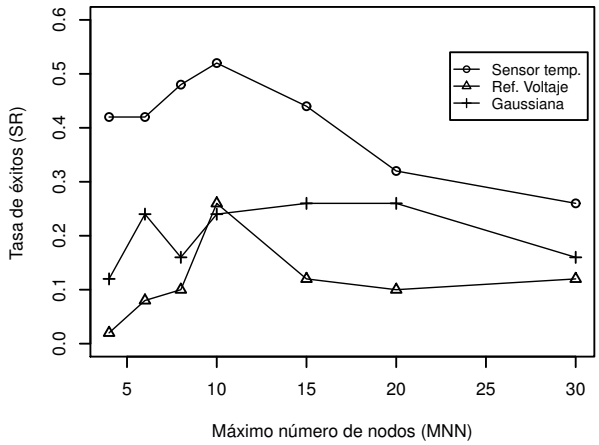
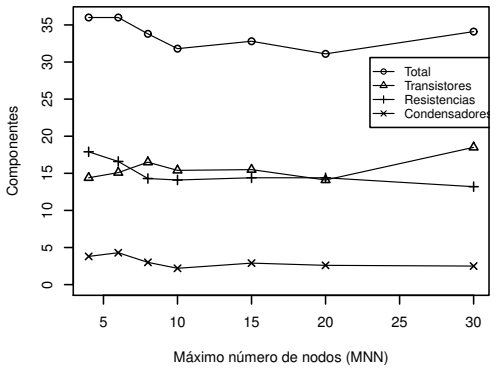
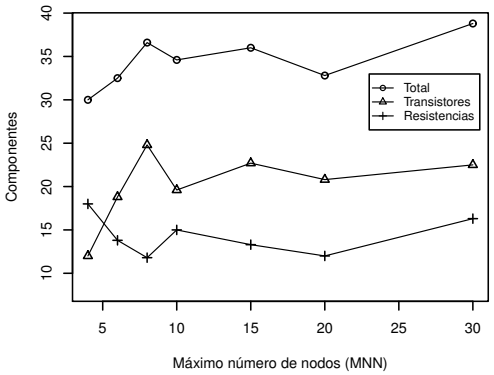


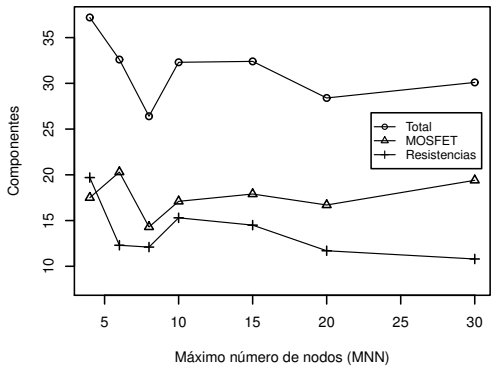
Figura 5.1. – Algoritmo ACID-GE: tasa de éxitos (SR) frente al máximo número de nodos (MNN) para el caso de circuitos no computacionales.



(a) Sensor de temperatura.

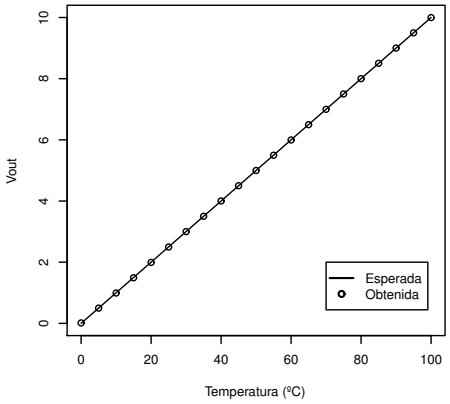


(b) Referencia de voltaje.

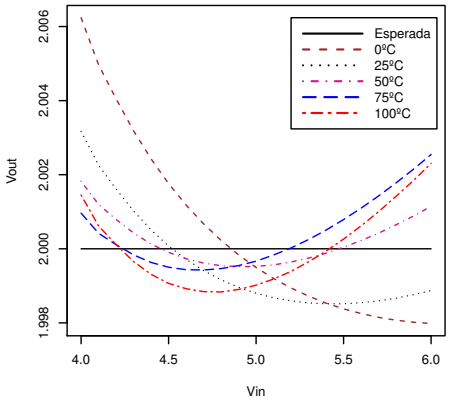


(c) Generador de función gaussiana.

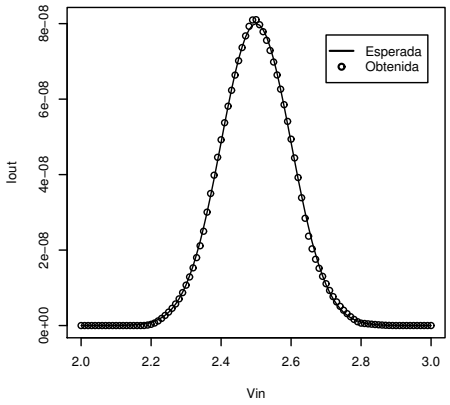
Figura 5.2. – Algoritmo ACID-GE: número medio de componentes frente al máximo número de nodos (MNN) para el caso de circuitos no computacionales.



(a) Sensor de temperatura.



(b) Referencia de voltaje. Nótese la escala reducida en el eje Y ($\triangle V_{out} = 2mV$).



(c) Generador de función gaussiana.

Figura 5.3. – Algoritmo ACID-GE: salida medida frente a salida esperada para los mejores circuitos no computacionales obtenidos.

Tabla 5.5. – Algoritmo ACID-GE: comparativa entre los operadores de cruce estándar de un punto y *one-block* en relación a la tasa de éxitos (SR) para los tres circuitos no computacionales. Se muestran los p-valores de los test de hipótesis de diferencia de proporciones, de una cola.

Objetivo	MNN	Operador	Nº de ejecuciones	SR (%)	p-valor
Sensor de temperatura	6	*One-point*	50	28.0	0.071
Sensor de temperatura	6	*One-block*	50	**42.0**	
Referencia de voltaje	10	*One-point*	50	14.0	0.067
Referencia de voltaje	10	*One-block*	50	**26.0**	
Generador de función gaussiana	15	*One-point*	50	22.0	0.320
Generador de función gaussiana	15	*One-block*	50	**26.0**	

Comparativa entre operadores de cruce

La introducción del operador de cruce *one-block*, definido en la sección 3.2.7, permite el intercambio de bloques completos entre los cromosomas padre, cada uno de los cuales corresponde a un componente de circuito completamente definido. Por otro lado, el operador estándar de un punto usado en EG, puede tener efectos destructivos, ya que podría cortar un cromosoma en cualquier punto e intercambiar información con diferente significado. La tabla 5.5 muestra una comparativa entre los resultados obtenidos al utilizar ambos tipos de operadores. Este estudio se ha realizado para valores de MNN que producen un valor de SR elevado (véase la figura 5.1). Como puede observarse, en los tres casos, el operador *one-block* obtiene un valor de SR más elevado que con el operador estándar de un punto. No obstante, para comprobar si estos resultados son estadísticamente significativos, se ha realizado un test de hipótesis de diferencia de proporciones de una cola. Los p-valores obtenidos (véase la tabla 5.5) indican que los resultados son estadísticamente significativos ($\alpha = 0.1$) para los circuitos sensor de temperatura y referencia de voltaje. Para el circuito generador de función gaussiana, aún no obteniendo una diferencia estadística significativa, el SR obtenido para el nuevo operador de cruce propuesto es también mayor. Por lo tanto, es posible concluir que, para los casos estudiados, el operador *one-block* mejora o iguala los resultados obtenidos con el operador de cruce estándar de un punto, añadiendo así evidencia experimental de nuestra hipótesis sobre la ventaja del operador de cruce *one-block* frente al operador estándar de un punto.

5.4.2. Circuitos computacionales

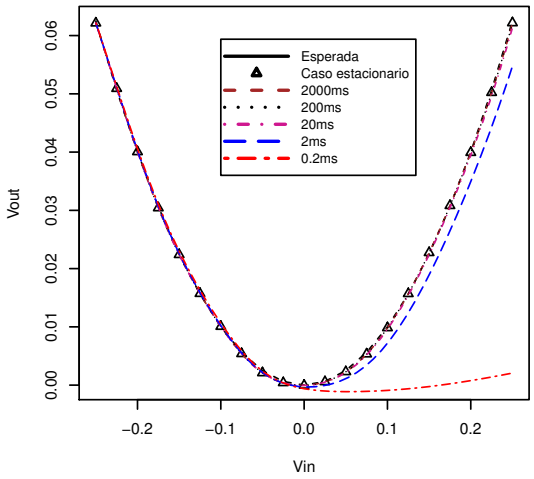
Los experimentos incluidos en esta sección sólo van dirigidos a comprobar si es posible obtener un circuito que cumpla con las especificaciones de diseño de cada circuito computacional. Esta sección sobre circuitos computacionales se centra únicamente en el diseño de los mismos. Por este motivo, no se realiza un análisis estadístico, ni se muestran los valores de SR y MBF en los resultados obtenidos.

En este caso, también se han obtenido circuitos exitosos en los cuatros casos, con la misma configuración utilizada en los circuitos no computacionales y un número máximo de generaciones igual a 3 000 generaciones. Los valores de MNN utilizados en cada circuito se han obtenido mediante la estrategia iterativa indicada en la sección 5.4.1, y son los siguientes: 10 para los circuitos de potencia al cuadrado y raíz cuadrada, 20 para potencia al cubo y 30 para raíz cúbica. La figura 5.4 muestra una comparativa entre las salidas esperada y obtenida para los mejores circuitos obtenidos en condiciones estacionarias.

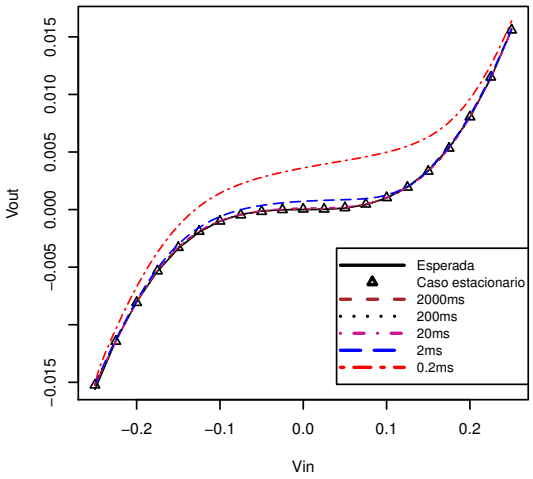
Análisis en el dominio del tiempo

El análisis en DC no necesariamente conduce a circuitos robustos cuando la frecuencia con la que varía la entrada puede repercutir en la salida del circuito (Mydlowec & Koza, 2000). Por ello, se ha realizado un análisis en el dominio del tiempo, de los circuitos computacionales obtenidos (condiciones no estacionarias). En particular, se ha utilizado una rampa de entrada con diferentes tiempos de subida Δ_{rt}. Las salidas de este análisis se muestran también en la figura 5.4, junto con las salidas obtenidas para el caso estacionario. Como puede observarse, todos los circuitos son capaces de ofrecer una salida adecuada cuando el tiempo de subida de la rampa de entrada es de $\Delta_{rt} = 200ms$ o mayor. Sin embargo, las salidas obtenidas empeoran notablemente cuando los tiempos de subida disminuyen por debajo del valor mencionado. Es importante indicar, que los circuitos se han obtenido con una función de adaptación que sólo tiene en cuenta el comportamiento de los circuitos mediante un análisis en continua para diferentes voltajes de entrada (barrido de voltaje).

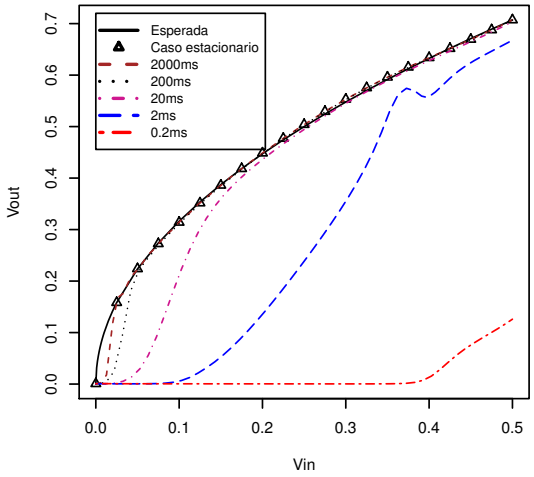
En la literatura se encuentran evidencias de que el uso de una función de adaptación basada únicamente en análisis en DC lleva a circuitos poco robustos (Mydlowec & Koza, 2000; Sapargaliyev & Kalganova, 2012). Por estos motivos, se ha realizado un nuevo experimento que permita analizar si es posible mejorar el comportamiento



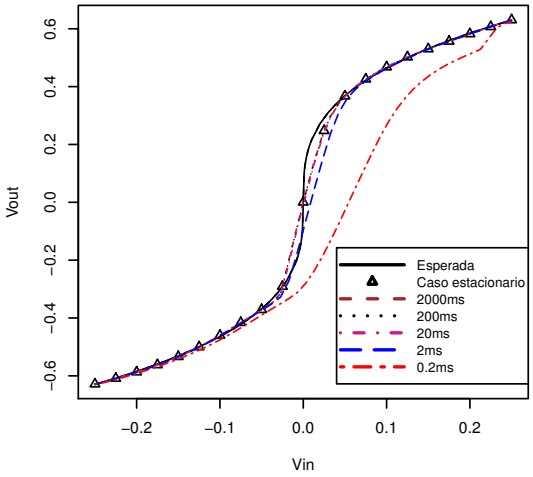
(a) Circuito de potencia al cuadrado.



(b) Circuito de potencia al cubo.



(c) Circuito de raíz cuadrada.



(d) Circuito de raíz cúbica.

Figura 5.4. – Algoritmo ACID-GE: salida medida frente a salida esperada para los mejores circuitos computacionales obtenidos. Estos resultados se han obtenido mediante un barrido DC en condiciones estacionarias, y mediante análisis transitorios en condiciones no estacionarias (utilizando rampas de entrada con diferentes tiempos de subida). Los circuitos fueron obtenidos con una función de adaptación que sólo tenía en cuenta los análisis estacionarios basados en un barrido de voltaje.

de estos circuitos en el dominio del tiempo. En concreto, para realizar este estudio, se ha elegido arbitrariamente el circuito de potencia al cuadrado y, además, se ha utilizado la denominada segunda variante de la función de adaptación (véase la sección 4.2.2), en la que se realiza un análisis transitorio y se utiliza una rampa con tiempo de subida de la rampa de $\Delta_{rt} = 0.2s$. La figura 5.5 muestra una comparativa entre las salidas esperada y medida para el mejor circuito obtenido en este experimento, utilizando rampas de entrada con diferentes tiempos de subida. Como puede observarse en dicha figura, el circuito obtenido en este experimento se comporta mucho mejor que el obtenido con análisis en condiciones estacionarias (véase la figura 5.4a). Adicionalmente, la salida medida es muy cercana a la salida esperada, incluso para tiempos de subida de la rampa muy bajos ($\Delta_{rt} = 0.2ms$). Los resultados de este estudio son consistentes con los obtenidos en (Sapargaliyev & Kalganova, 2012; Mydlowec & Koza, 2000), donde se indica que el uso de una función de adaptación basada en un análisis en el dominio del tiempo parece producir circuitos más robustos que los obtenidos con una función de adaptación basada en un análisis en condiciones estacionarias.

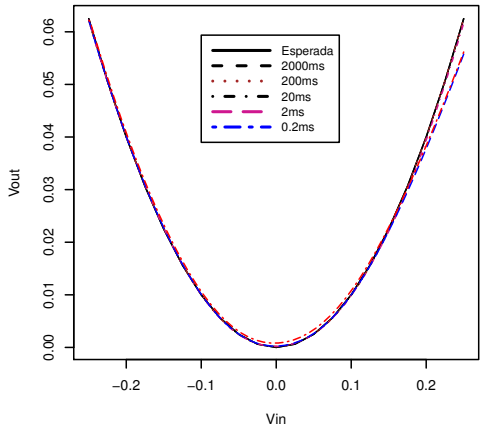


Figura 5.5. – Algoritmo ACID-GE: salida medida frente a salida esperada, para diferentes tiempos de subida de una rampa de entrada, en el mejor circuito de potencia al cuadrado obtenido utilizando una función de adaptación basada en un análisis en el dominio del tiempo.

La topología y dimensionamiento de los mejores circuitos computacionales obtenidos en cada caso, se muestran en las figuras 5.13, 5.14, 5.15 y 5.16. Nótese que el mejor circuito de potencia al cuadrado mostrado corresponde al obtenido mediante análisis en el dominio del tiempo.

5.4.3. Comparativa de ACID-GE con trabajos previos

En esta sección se realiza una comparativa entre los resultados obtenidos con ACID-GE y diferentes trabajos previos en la literatura, para los dos conjuntos de circuitos utilizados.

Circuitos no computacionales

Los trabajos previos en los que se sintetizaron los circuitos no computacionales, utilizaron PG (Koza et al., 1999a) y AGE (Mattiussi & Floreano, 2007). La tabla 5.6 muestra la comparativa de ACID-GE y dichos trabajos previos, teniendo en cuenta que los resultados de estos últimos se muestran tal como se presentan en (Mattiussi & Floreano, 2007). Es importante señalar que, para realizar una comparación justa, los valores de adaptación mostrados se han calculado con la misma función de adaptación. Las especificaciones de cada circuito son las mismas definidas por Koza (Koza et al., 1999a), con excepción del circuito de referencia de voltaje, donde se utiliza una resistencia de carga diferente ($10K\Omega$), tal y como se hizo en (Mattiussi & Floreano, 2007). En otro caso, con las especificaciones originales de Koza, es imposible obtener un circuito exitoso.

Como puede observarse en la tabla 5.6 y atendiendo al valor de adaptación, ACID-GE ofrece mejores resultados en todos los casos, utilizando un número de evaluaciones similar o menor. Adicionalmente, el número de componentes utilizado es más bajo para el caso del circuito de referencia de voltaje y, para los dos circuitos restantes, dicho número está comprendido entre los valores obtenidos en los dos trabajos con los que se realiza la comparación.

Circuitos computacionales

Los trabajos previos en los que se sintetizaron los circuitos computacionales utilizaron PG (Koza et al., 1999a; Mydlowec & Koza, 2000; Streeter et al., 2002) y EE (Sapargaliyev & Kalganova, 2012). En el caso del circuito de potencia al cubo, los resultados también se comparan con un diseño convencional (Cipriani & Takeshian, 2000). La tabla 5.7 muestra una comparativa entre los resultados de ACID-GE con dichos trabajos, donde los resultados de los trabajos previos se muestran tal como se presentan en (Sapargaliyev & Kalganova, 2012). Además, para hacer una comparación justa entre los distintos algoritmos, se utiliza ahora el error medio absoluto

Tabla 5.6. – Algoritmo ACID-GE: comparativa con trabajos previos para los mejores circuitos no computacionales obtenidos. Los resultados se presentan tal como aparecen en (Mattiussi & Floreano, 2007). Para realizar una comparación justa, los valores de adaptación mostrados se han calculado con la misma función de adaptación. Nótese que, en el caso de AGE (Mattiussi & Floreano, 2007), los resultados de número de componentes fueron promediados por los autores del mencionado trabajo para cinco circuitos, por lo que el valor mostrado no es un entero.

	PG (Koza et al., 1999a)	**AGE** (Mattiussi & Floreano, 2007)	**ACID-GE**
Sensor de temperatura			
Fitness	26.4	1.13	**0.065**
Nº de evaluaciones	1.6×10^7	6.5×10^6	$\mathbf{6.14 \times 10^6}$
Nº de componentes	54	**27.8**	33
Referencia de voltaje			
Fitness	6.6	2.64	**0.112**
Nº de evaluaciones	5.12×10^7	5.6×10^6	$\mathbf{1.86 \times 10^6}$
Nº de componentes	67	70.2	**32**
Generador de función gaussiana			
Fitness	0.094	0.3	**0.036**
Nº de evaluaciones	2.30×10^7	$\mathbf{4.30 \times 10^6}$	6.23×10^6
Nº de componentes	**14**	36	28

(MAE, por sus siglas en inglés), que se define mediante (5.1), donde O_i es la salida esperada, $\widetilde{O}_i$ es la salida medida en el circuito obtenido y n es el número de puntos de ajuste. Como puede observarse, ACID-GE ofrece el mínimo valor de MAE con el mínimo número de evaluaciones en los cuatro circuitos considerados. El número de componentes utilizados por ACID-GE también es competitivo en relación con los resultados obtenidos por el resto de los algoritmos usados en la comparación.

$$MAE = \frac{1}{n} \sum_{i=1}^{i=n} |O_i - \widetilde{O}_i| \tag{5.1}$$

5.5. Resultados de experimentos con ACID-MGE

En esta sección se muestran los resultados utilizando el algoritmo ACID-MGE y, además, se comparan directamente con los resultados obtenidos con ACID-GE para,

Tabla 5.7. – Algoritmo ACID-GE: comparación con trabajos previos para los mejores circuitos computacionales obtenidos, donde PG1 corresponde al trabajo de (Koza et al., 1999a), PG2, al de (Mydlowec & Koza, 2000), PG3, al de (Streeter et al., 2002), diseño manual, al de (Cipriani & Takeshian, 2000) y EE, al de (Sapargaliyev & Kalganova, 2012). Los resultados se muestran tal como aparecen en (Sapargaliyev & Kalganova, 2012). Para hacer una comparación justa entre los distintos algoritmos, se utiliza el valor de MAE definido en (5.1).

	PG1	**PG2**	**PG3**	**Diseño manual**	**EE**	**ACID-GE**
Raíz cuadrada						
MAE (mV)	183.57	20.00	-	-	9.23	**2.048**
N.º de componentes	64	39	-	-	**22**	26
N.º de evaluaciones	-	6.7×10^9	-	-	3.7×10^6	$\mathbf{1.83 \times 10^6}$
Potencia al cuadrado						
MAE (mV)	-	27.00	-	-	1.44	**0.109**
N.º de componentes	39	37	-	-	35	**29**
N.º de evaluaciones	-	1.1×10^9	-	-	2.7×10^6	$\mathbf{1.87 \times 10^6}$
Raíz cúbica						
MAE (mV)	80.00	-	-	-	11.90	**4.178**
N.º de componentes	50	-	-	-	39	**28**
N.º de evaluaciones	3.8×10^7	-	-	-	4.5×10^6	$\mathbf{1.85 \times 10^6}$
Potencia al cubo						
MAE (mV)	1.04	-	0.99	7.13	0.29	**0.0585**
N.º de componentes	56	-	47	**12**	44	36
N.º de evaluaciones	-	-	2.94×10^6	-	2.34×10^6	$\mathbf{1.87 \times 10^6}$

Tabla 5.8. – ACID-GE vs. ACID-MGE: comparación de resultados para los circuitos no computacionales con 3 000 generaciones.

Circuito	Algoritmo	MNN	SR	p-valor (SR)	minBF	MBF±SD	Hits±SD
Sensor de temperatura	ACID-MGE	6	**70.0 %**	0.0016	**0.033**	**2.13 ± 3.34**	**97.1 %±5.3**
	ACID-GE	6	42.0 %		0.169	5.29 ± 6.53	93.0 %±7.9
Función gaussiana	ACID-MGE	6	**58.0 %**	0.0001	**0.028**	**1.00 ± 1.75**	**94.1 %±9.8**
	ACID-GE	6	24.0 %		0.040	6.70 ± 6.78	77.3 %±17.6
Referencia de voltaje	ACID-MGE	6	8.0 %	0.5000	**0.053**	**19.35 ± 14.04**	**64.2 %±23.3**
	ACID-GE	6	8.0 %		0.112	26.57 ± 16.23	53.8 %±25.0

de esta forma, analizar si el enfoque basado en EMG aporta alguna ventaja.

5.5.1. Comparación de resultados de ACID-MGE y ACID-GE

Las tablas 5.8 y 5.9 muestran los resultados de ACID-MGE frente a ACID-GE para los circuitos no computacionales y computacionales, respectivamente. En particular, los resultados de los circuitos no computacionales con ACID-GE son los que ya se mostraron en la tabla 5.3. Sin embargo, en el caso de los circuitos computacionales, se han realizado nuevos experimentos con ACID-GE, pero usando la tercera variante de las especificaciones (véase sección 4.2.2), con el objetivo de poder obtener unos valores de SR más altos y, por lo tanto, que esto facilite un análisis más robusto desde un punto de vista estadístico. Dichas especificaciones también conllevan que las funciones de adaptación estén basadas en análisis en el dominio del tiempo para todos los circuitos computacionales. Finalmente, es importante indicar que para los experimentos con ACID-MGE se han utilizado los mismos valores del parámetro MNN que en el caso de ACID-GE para permitir una comparación más justa.

En relación con los resultados mostrados, se muestran los valores de SR, p-valores, minBF, MBF y el número medio de hits. Para el caso de los circuitos computacionales, también se incluye el valor del MAE, expresado en mV. En todos los casos, se han realizado tests de hipótesis para verificar si las mejoras de SR observadas son estadísticamente significativas o no.

Atendiendo al SR, se observa una mejora estadísticamente significativa ($\alpha < 0.05$) en cinco circuitos del total de siete. Sin embargo, en dos circuitos no es posible rechazar la hipótesis nula, si bien, aún en estos casos el valor obtenido de SR por

Tabla 5.9. – ACID-GE vs. ACID-MGE: comparación de resultados para los circuitos computacionales con 3 000 generaciones.

Circuito	Algoritmo	MNN	SR	p-valor (SR)	MAE ±SD	minBF	MBF ±SD	Hits ±SD
Potencia al cuadrado	ACID-MGE	10	**84.0 %**	0.0001	**0.53 ±0.24**	**0.002**	**0.06 ±0.14**	**97.7 % ±6.5**
	ACID-GE	10	52.0 %		0.93 ±0.41	0.009	0.73 ±1.17	81.0 % ±28.5
Raíz cuadrada	ACID-MGE	10	**66.0 %**	0.0117	**4.01 ±2.47**	**0.003**	**2.71 ±5.78**	**89.0 % ±24.2**
	ACID-GE	10	44.0 %		5.55 ±4.32	0.028	6.25 ±7.61	74.0 % ±32.3
Potencia al cubo	ACID-MGE	20	**84.0 %**	0.1574	**0.10 ±0.04**	**0.001**	**0.03 ±0.08**	**95.7 % ±13.7**
	ACID-GE	20	76.0 %		0.18 ±0.07	0.002	0.05 ±0.12	92.2 % ±19.3
Raíz cúbica	ACID-MGE	30	**22.0 %**	0.0006	**7.13 ±3.70**	**0.036**	**13.87 ±25.02**	**70.2 % ±29.4**
	ACID-GE	30	2.0 %		10.81 ±0.0	0.227	76.95 ±23.74	11.5 % ±20.2

el algoritmo ACID-MGE es igual o mejor que en el caso de ACID-GE. Estos dos circuitos son el circuito de referencia de voltaje y el circuito de potencia al cubo. Con ello, se puede concluir que, desde el punto de vista del SR, ACID-MGE mejora o iguala los resultados ofrecidos por ACID-GE en todos los experimentos realizados.

Considerando la medida minBF, que corresponde al mejor circuito obtenido, se observa también una mejora en seis de los siete circuitos. Sólo en el caso del circuito de generador de función gaussiana se obtiene un valor igual de minBF para los dos algoritmos.

En cuanto al MBF, se observa que ACID-MGE mejora los resultados obtenidos frente a ACID-GE en todos los experimentos realizados.

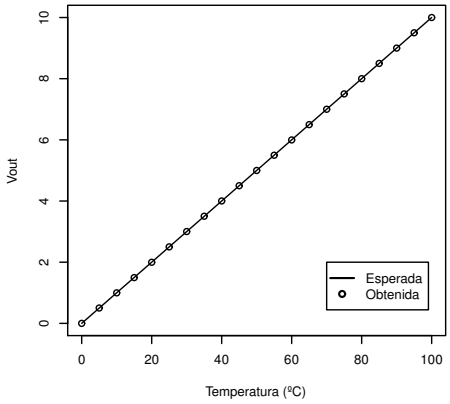
Finalmente, desde el punto de vista del MAE, medido sólo en el caso de los circuitos computacionales, también se observa que ACID-MGE mejora los resultados obtenidos frente a ACID-GE.

5.5.2. Salidas de los circuitos obtenidos con ACID-MGE

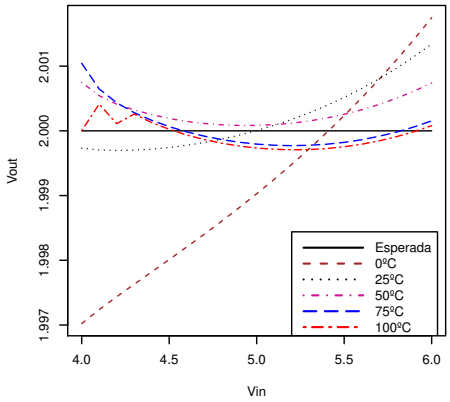
En esta sección se analizan las comparativas entre las salidas esperadas y las medidas en los mejores circuitos obtenidos utilizando el algoritmo ACID-MGE. En las figuras 5.6a, 5.6b y 5.6c, se muestra el voltaje de salida del mejor circuito sensor de temperatura, el voltaje de salida del mejor circuito de referencia de voltaje y la corriente de salida del mejor circuito de generación gaussiana, respectivamente. En las figuras 5.7a, 5.7b, 5.7c y 5.7d, se muestran los voltajes de salida de los circuitos de potencia al cuadrado, potencia al cubo, raíz cuadrada y raíz cúbica, respectivamente. Nótese que, en todos los casos, la aproximación obtenida es bastante buena. Las *netlists* correspondientes a estos circuitos se muestran en el apéndice C.2.

5.5.3. Comparativa de ACID-MGE con trabajos previos

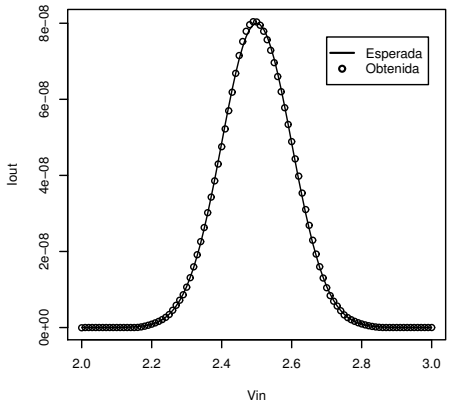
En esta sección se realiza una comparativa entre los resultados obtenidos con ACID-MGE y diferentes trabajos previos en la literatura, para los dos conjuntos de circuitos utilizados.



(a) Sensor de temperatura.

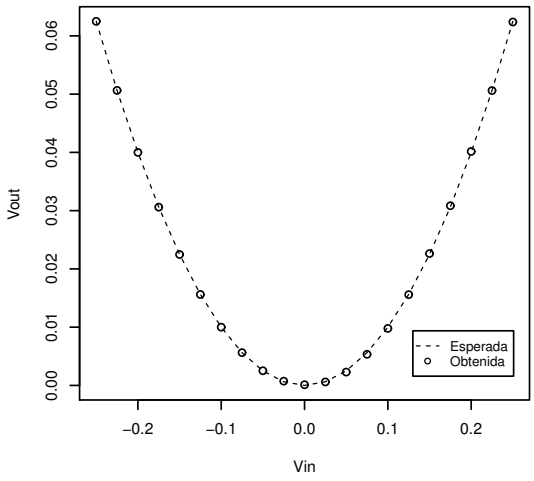


(b) Referencia de voltaje. Nótese la escala reducida en el eje Y ($\triangle V_{out} = 2mV$).

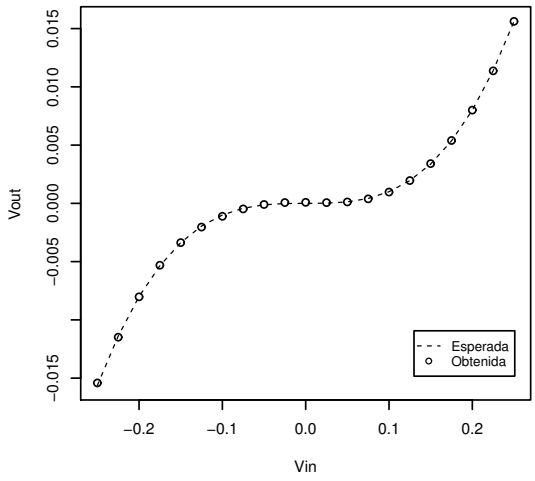


(c) Generador de función gaussiana.

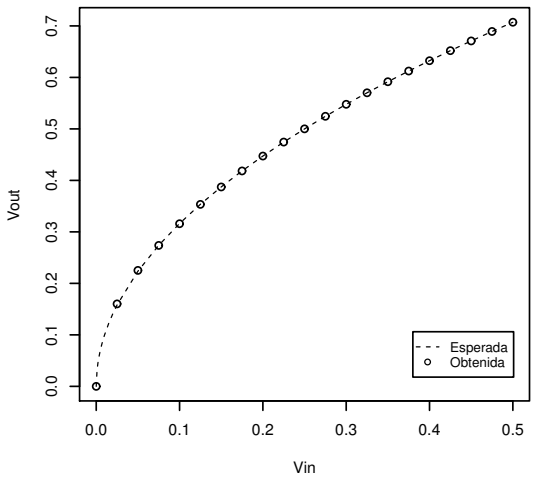
Figura 5.6. – Algoritmo ACID-MGE: salida medida frente a salida esperada para los mejores circuitos no computacionales obtenidos.



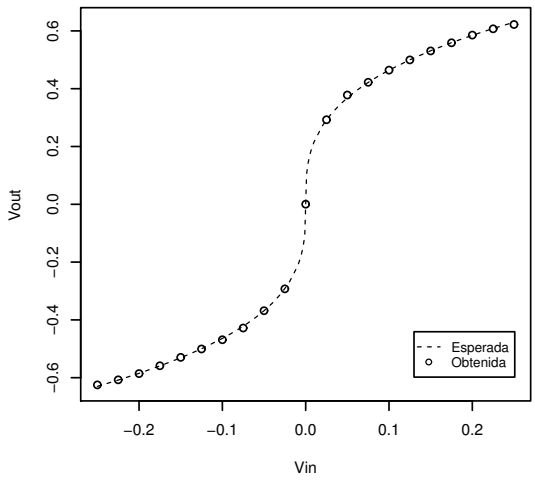
(a) Circuito de potencia al cuadrado.



(b) Circuito de potencia al cubo.



(c) Circuito de raíz cuadrada.



(d) Circuito de raíz cúbica.

Figura 5.7. – Algoritmo ACID-MGE: salida medida frente a salida esperada para los mejores circuitos computacionales obtenidos.

Tabla 5.10. – Algoritmo ACID-MGE: comparativa con trabajos previos para los mejores circuitos no computacionales obtenidos. Los resultados se presentan tal como aparecen en (Mattiussi & Floreano, 2007). Para realizar una comparación justa, los valores de adaptación mostrados se han calculado con la misma función de adaptación.

	PG (Koza et al., 1999a)	**AGE** (Mattiussi & Floreano, 2007)	**ACID-GE**	**ACID-MGE**
Sensor de temperatura				
Fitness	26.4	1.13	0.065	**0.033**
Referencia de voltaje				
Fitness	6.6	2.64	0.112	**0.053**
Generador de función gaussiana				
Fitness	0.094	0.3	0.036	**0.028**

Circuitos no computacionales

Los trabajos previos en los que se sintetizaron los circuitos no computacionales utilizaron PG (Koza et al., 1999a) y AGE (Mattiussi & Floreano, 2007). La tabla 5.10 muestra la comparativa de ACID-MGE y dichos trabajos previos, teniendo en cuenta que los resultados de estos últimos se muestran tal como se presentan en (Mattiussi & Floreano, 2007). Es importante señalar que, para realizar una comparación justa, los valores de adaptación mostrados se han calculado con la misma función de adaptación. Las especificaciones de cada circuito son las mismas definidas por Koza (Koza et al., 1999a), con excepción del circuito de referencia de voltaje, donde se utiliza una resistencia de carga diferente ($10K\Omega$), tal y como se hizo en (Mattiussi & Floreano, 2007). En otro caso, con las especificaciones originales de Koza, es imposible obtener un circuito exitoso. Como puede observarse en la tabla 5.10, ACID-MGE ofrece mejores resultados en todos los casos.

Circuitos computacionales

Los trabajos previos en los que se sintetizaron los circuitos computacionales utilizaron PG (Koza et al., 1999a; Mydlowec & Koza, 2000; Streeter et al., 2002) y EE (Sapargaliyev & Kalganova, 2012). En el caso del circuito de potencia al cubo, los resultados también se comparan con un diseño convencional (Cipriani & Takeshian, 2000). La tabla 5.11 muestra una comparativa entre los resultados de ACID-MGE con dichos trabajos, donde los resultados de los trabajos previos se muestran tal

Tabla 5.11. – Algoritmo ACID-MGE: comparación con trabajos previos para los mejores circuitos computacionales obtenidos, donde PG1 corresponde al trabajo de (Koza et al., 1999a), PG2, al de (Mydlowec & Koza, 2000), PG3, al de (Streeter et al., 2002), diseño manual, al de (Cipriani & Takeshian, 2000) y EE, al de (Sapargaliyev & Kalganova, 2012). Los resultados se muestran tal como aparecen en (Sapargaliyev & Kalganova, 2012). Para hacer una comparación justa entre los distintos algoritmos, se utiliza el valor de MAE definido en (5.1).

	PG1	**PG2**	**PG3**	**Diseño manual**	**EE**	**ACID-GE**	**ACID-MGE**
Raíz cuadrada							
MAE (mV)	183.57	20.00	-	-	9.23	2.048	**0.228**
Potencia al cuadrado							
MAE (mV)	-	27.00	-	-	1.44	0.109	**0.084**
Raíz cúbica							
MAE (mV)	80.00	-	-	-	11.90	4.178	**2.038**
Potencia al cubo							
MAE (mV)	1.04	-	0.99	7.13	0.29	0.0585	**0.0502**

como se presentan en (Sapargaliyev & Kalganova, 2012). Además, para hacer una comparación justa entre los distintos algoritmos, se utiliza ahora el valor de MAE, definido mediante (5.1). Como puede observarse, ACID-MGE ofrece el mínimo valor de MAE en los cuatro circuitos considerados.

5.5.4. Resultados de ACID-MGE incluyendo aprendizaje del parámetro MNN

La tabla 5.12 muestra una comparativa de los resultados de ACID-MGE con y sin aprendizaje del parámetro MNN, para los circuitos no computacionales. La tabla 5.13 muestra la misma comparativa para el caso de los circuitos computacionales. En el caso de aprendizaje, se muestra el valor del parámetro MNN aprendido por el algoritmo, mientras que, en el caso de no utilizarlo, se ha elegido un valor de MNN cercano al aprendido, con el objetivo de que la comparación sea más justa.

Se han realizado tests de hipótesis para comprobar si las diferencias de SR obtenidas, en los casos con y sin aprendizaje, son estadísticamente significativas. A la vista de los p-valores, se puede comprobar que la hipótesis nula no se puede rechazar en seis

Tabla 5.12. – Algoritmo ACID-MGE: resultados con y sin aprendizaje del parámetro MNN para el caso de circuitos no computacionales con 3 000 generaciones.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	SR	p-valor (SR)
Sensor de temperatura	ACID-MGE	Aprendizaje	10.5 ± 4.2	70.0 %	0.0780
	ACID-MGE	Prefijado	10	**82.0** %	
Función gaussiana	ACID-MGE	Aprendizaje	10.9 ± 5.9	62.0 %	0.1984
	ACID-MGE	Prefijado	10	**70.0** %	
Referencia de voltaje	ACID-MGE	Aprendizaje	12.8 ± 5.0	10.0 %	0.0486
	ACID-MGE	Prefijado	10	**22.0** %	

Tabla 5.13. – Algoritmo ACID-MGE: resultados con y sin aprendizaje del parámetro MNN para el caso de circuitos computacionales con 3 000 generaciones.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	SR	p-valor (SR)
Potencia al cuadrado	ACID-MGE	Aprendizaje	10.5 ± 5.3	84.0 %	0.5000
	ACID-MGE	Prefijado	10	84.0 %	
Raíz cuadrada	ACID-MGE	Aprendizaje	9.3 ± 2.9	60.0 %	0.2668
	ACID-MGE	Prefijado	10	**66.0** %	
Potencia al cubo	ACID-MGE	Aprendizaje	10.0 ± 4.7	84.0 %	0.5000
	ACID-MGE	Prefijado	10	84.0 %	
Raíz cúbica	ACID-MGE	Aprendizaje	15.8 ± 5.6	12.0 %	0.1362
	ACID-MGE	Prefijado	15	**20.0** %	

de los siete casos ($\alpha < 0.05$). En particular, sólo en el caso del circuito de referencia de voltaje hay una diferencia estadísticamente significativa a favor de la opción que usa aprendizaje. De estos resultados, se puede concluir lo siguiente:

1. Los valores aprendidos de MNN para los circuitos no computacionales son próximos a los valores de pico mostrados en la figura 5.1, $MNN = 10$ para los circuitos sensor de temperatura y referencia de voltaje, y se encuentra dentro de la meseta $MNN = 10 - 20$ para el circuito de función gaussiana. Por lo que se puede concluir que el uso de la aproximación basada en ACID-MGE con aprendizaje del parámetro MNN, en los casos probados, tiene la ventaja de realizar automáticamente el ajuste de dicho parámetro.

2. El aprendizaje del valor del parámetro MNN es más costoso para el algoritmo que utiliza esta opción, ya que tiene que hallar el valor óptimo de MNN y, además, encontrar el circuito óptimo. Sin embargo, a pesar de que se han mantenido las mismas condiciones en ambos algoritmos (incluyendo el máximo número de generaciones igual a 3 000), el SR obtenido con aprendizaje sólo empeora, de forma estadísticamente significativa, en uno de los siete circuitos analizados.

Por lo tanto, la evidencia obtenida nos permite afirmar que el aprendizaje del valor de MNN es útil, al evitar que sea el usuario el que tiene que sintonizar este parámetro por ensayo y error, y que este proceso no dificulta la búsqueda de la solución comparado con el caso en el que no se realiza el aprendizaje de dicho parámetro.

5.5.5. Resultados en relación con la velocidad de convergencia

Esta sección se centra en analizar la velocidad de convergencia de las aproximaciones propuestas en esta tesis, considerando los casos de ACID-MGE con y sin aprendizaje del parámetro MNN y ACID-GE. Se muestran dos índices de medida relacionados con la velocidad de convergencia de algoritmos evolutivos (véase la sección 5.3). El primero es el valor medio del número de generaciones necesarias para obtener un circuito exitoso (etiquetado como Nº gen.). El segundo es el número medio de evaluaciones para conseguir un éxito (AES). Valores más bajos de estos índices corresponderán a algoritmos más rápidos. Las tablas 5.14 y 5.15 muestran dichas medidas para los circuitos no computacionales y computacionales, respectivamente. En el caso de los circuitos computacionales de potencia al cubo y raíz cúbica,

Tabla 5.14. – ACID-GE vs. ACID-MGE: valor medio de generaciones para conseguir un éxito y número medio de evaluaciones para conseguir un éxito para circuitos no computacionales, 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	Nº gen.±SD	AES±SD
Sensor de temperatura	ACID-MGE	Aprendizaje	10.5 ± 4.2	904.2 ± 749.1	$7.42 \times 10^5 \pm 6.22 \times 10^5$
	ACID-MGE	Prefijado	10	$\mathbf{704.1 \pm 652.1}$	$\mathbf{5.75 \times 10^5 \pm 5.46 \times 10^5}$
	ACID-GE	Prefijado	10	$1,289.8 \pm 846.9$	$7.85 \times 10^5 \pm 5.37 \times 10^5$
Función gaussiana	ACID-MGE	Aprendizaje	10.9 ± 5.9	$\mathbf{835.4 \pm 580.8}$	$\mathbf{6.66 \times 10^5 \pm 4.81 \times 10^5}$
	ACID-MGE	Prefijado	10	931.4 ± 583.3	$7.46 \times 10^5 \pm 4.66 \times 10^5$
	ACID-GE	Prefijado	10	$1,458.3 \pm 857.0$	$8.65 \times 10^5 \pm 5.51 \times 10^5$
Referencia de voltaje	ACID-MGE	Aprendizaje	12.8 ± 5.0	$1,810.6 \pm 924.7$	$14.2 \times 10^5 \pm 7.36 \times 10^5$
	ACID-MGE	Prefijado	10	$\mathbf{1,153.5 \pm 646.1}$	$\mathbf{9.08 \times 10^5 \pm 5.16 \times 10^5}$
	ACID-GE	Prefijado	10	$1,574.2 \pm 665.5$	$9.63 \times 10^5 \pm 4.43 \times 10^5$

aparecen sendos experimentos más con ACID-MGE, que corresponden a valores de MNN de 20 y 30. Estos valores del parámetro MNN son los utilizados con ACID-GE y permiten realizar una comparativa con el mismo.

En relación con el valor medio del número de generaciones para obtener un circuito exitoso, ACID-MGE sin aprendizaje resulta ser más rápido que ACID-GE en todos los experimentos realizados. El algoritmo ACID-MGE con aprendizaje de MNN ofrece mejores resultados que ACID-GE en cinco de los siete circuitos. Por otro lado, ACID-MGE con aprendizaje de MNN resulta ser más lento que la versión de ACID-MGE sin aprendizaje, en cinco de los siete circuitos. Este último resultado es lógico, dado que el aprendizaje simultáneo del valor de MNN añade más dificultad a la búsqueda del circuito solución.

En relación con el índice AES, es necesario indicar que no hay una relación directa entre generaciones y número de evaluaciones, pues ambos algoritmos, ACID-GE y ACID-MGE, cuentan con un mecanismo de caché que sólo realiza la evaluación de un cromosoma la primera vez que aparece en una población. Si vuelve a aparecer un mismo cromosoma, por lo tanto, ya evaluado previamente, no será necesario volver a evaluarlo de nuevo, pudiendo tomar su valor de adaptación obtenido previamente. Este mecanismo permite reducir el número de evaluaciones y hacer más eficiente la implementación. Por este motivo, tampoco existe una relación exacta entre los

Tabla 5.15. – ACID-GE vs. ACID-MGE: valor medio de generaciones para conseguir un éxito y número medio de evaluaciones para conseguir un éxito para circuitos computacionales, 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	Nº gen.±SD	AES±SD
Potencia al cuadrado	ACID-MGE	Aprendizaje	10.5 ± 5.3	778.7±656.2	$6.06 \times 10^5 \pm 5.24 \times 10^5$
	ACID-MGE	Prefijado	10	$\mathbf{671.5 \pm 574.9}$	$\mathbf{4.94 \times 10^5 \pm 4.12 \times 10^5}$
	ACID-GE	Prefijado	10	$1,176.9 \pm 828.3$	$6.52 \times 10^5 \pm 4.73 \times 10^5$
Raíz cuadrada	ACID-MGE	Aprendizaje	9.3 ± 2.9	$\mathbf{835.8 \pm 569.2}$	$\mathbf{6.55 \times 10^5 \pm 4.55 \times 10^5}$
	ACID-MGE	Prefijado	10	$1,154.9 \pm 655.0$	$9.13 \times 10^5 \pm 5.25 \times 10^5$
	ACID-GE	Prefijado	10	$1,329.3 \pm 797.3$	$7.71 \times 10^5 \pm 4.89 \times 10^5$
Potencia al cubo	ACID-MGE	Aprendizaje	10.0 ± 4.7	637.8 ± 538.8	$4.92 \times 10^5 \pm 4.26 \times 10^5$
	ACID-MGE	Prefijado	10	630.9 ± 538.1	$4.90 \times 10^5 \pm 4.22 \times 10^5$
	ACID-MGE	Prefijado	20	$\mathbf{539.8 \pm 361.7}$	$\mathbf{4.22 \times 10^5 \pm 2.87 \times 10^5}$
	ACID-GE	Prefijado	20	$1,008.0 \pm 703.6$	$6.24 \times 10^5 \pm 4.41 \times 10^5$
Raíz cúbica	ACID-MGE	Aprendizaje	15.8 ± 5.6	$1,804.8 \pm 559.2$	$14.4 \times 10^5 \pm 4.39 \times 10^5$
	ACID-MGE	Prefijado	15	$1,604.7 \pm 511.6$	$12.8 \times 10^5 \pm 4.22 \times 10^5$
	ACID-MGE	Prefijado	30	$\mathbf{1,561.1 \pm 587.8}$	$12.6 \times 10^5 \pm 4.75 \times 10^5$
	ACID-GE	Prefijado	30	$1,885.0 \pm 0.0$	$\mathbf{11.9 \times 10^5 \pm 0.00 \times 10^5}$

índices AES y número medio de generaciones para obtener un éxito, que, de otra manera, correspondería al número de generaciones multiplicado por el tamaño de la población.

Por otro lado, los resultados atendiendo al valor del índice AES dan como mejores resultados a los mismos algoritmos que el número medio de generaciones para obtener un éxito, con la excepción del circuito de raíz cúbica. En este caso, el algoritmo que mejor índice AES ofrece es ACID-GE. Además de la explicación indicada en el párrafo anterior, es importante indicar que en el caso de ACID-GE sólo se pudo obtener un único éxito en todas las ejecuciones y, por lo tanto, dicho resultado podría ser poco representativo.

En relación con el índice AES, ACID-MGE sin aprendizaje ofrece un valor menor de evaluaciones en cinco de los siete circuitos cuando se compara con ACID-GE. El algoritmo ACID-MGE con aprendizaje de MNN ofrece mejores resultados que ACID-GE en cuatro de los siete circuitos. Finalmente, el uso de ACID-MGE con aprendizaje de MNN muestra un peor valor de AES que ACID-MGE sin aprendizaje en cinco de los siete circuitos, lo que vuelve a remarcar la idea de que el aprendizaje simultáneo del valor de MNN añade más dificultad a la búsqueda del circuito solución y, además, estando alineado con los resultado anteriormente vistos en relación con el número de generaciones.

5.5.6. Efecto engorde o *bloat*

Esta sección se centra sobre el denominado efecto engorde o *bloat* que se manifiesta por el crecimiento progresivo de los cromosomas a medida que aumenta el número de generaciones. Para medir el impacto del efecto engorde, se introducen nuevas estadísticas. Dado que todos los algoritmos utilizados en la síntesis de circuitos utilizan cromosomas de longitud variable, una de las medidas utilizadas será la longitud media del cromosoma. Adicionalmente, el proceso de decodificación puede terminar antes de haber leído el cromosoma completo, quedando una parte final no utilizada (véase la sección 3.2.6). Para medir este efecto se introduce el concepto de *longitud expresada media*, que corresponde al número de codones leídos del cromosoma para obtener una expresión expandida completamente. Finalmente, en relación con el efecto engorde, también vamos a medir el número medio de componentes de los circuitos exitosos.

Es necesario tener en cuenta que, para mitigar el impacto del efecto engorde en el rendimiento de los algoritmos, las implementaciones de los operadores de cruce utilizados incluyen un control de tamaño máximo del cromosoma (véanse las secciones 3.2.7 y 3.4.2). Sin embargo, puede ser útil analizar cuáles de los algoritmos implementados tienden a utilizan más codones para representar los circuitos solución.

Para facilitar la comparación entre longitudes de cromosoma, tanto la longitud media como la expresada se calculan respecto a la longitud máxima del cromosoma (véase la sección 5.2.2). En ambos algoritmos, se ha dimensionado la longitud máxima del cromosoma para permitir 42 componentes, sin utilizar *wrapping*, con la excepción en el caso de ACID-MGE con aprendizaje de MNN, donde se utiliza un bloque de 4 codones situado justo al principio del cromosoma que codifica el valor de dicho parámetro. En este caso no se ha aumentado el tamaño máximo de 336 codones, por lo que sólo cabrían 41 componentes sin *wrapping*.

Adicionalmente, se introducen dos nuevos índices que, tal y como indica su nombre, miden el *número de cromosomas expresables* y el *número de cromosomas viables*. Recordando que un cromosoma es inexpresable cuando no se puede decodificar, mientras que el cromosoma inviable es aquel cuyo fenotipo es un circuito inviable (véase la sección 3.2.3). Un circuito inviable no puede ser simulado correctamente, bien por no tener conexiones a la alimentación, masa, entrada o salida, o bien porque da lugar a un error del simulador.

Por un lado, las tablas 5.16 y 5.17 muestran la longitud media y la longitud expresada media (en ambos casos, calculadas respecto al valor de longitud máxima) de los cromosomas en la población de la última generación de los algoritmos ACID-MGE, con y sin aprendizaje de MNN, y ACID-GE, para los circuitos no computacionales y computacionales, respectivamente.

Por otro lado, las tablas 5.18 y 5.19 muestran el número medio de componentes de los circuitos exitosos obtenidos, el número medio de cromosomas expresables y el número medio de cromosomas viables en la población de la última generación de los algoritmos ACID-MGE, con y sin aprendizaje de MNN, y ACID-GE, para los circuitos no computacionales y computacionales, respectivamente. Es importante notar que el número medio de componentes se ha obtenido sobre el conjunto de circuitos exitosos en la última generación.

En relación con la longitud media del cromosoma y la longitud media expresada, se puede observar que la longitud media de los cromosomas está muy cerca de la

Tabla 5.16. – ACID-GE vs ACID-MGE: longitud media (LM) y longitud expresada media (LEM) de los cromosomas en la población de la última generación de los algoritmos para los circuitos no computacionales, 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	LM(%) ± SD	LEM(%) ±SD
Sensor de temperatura	ACID-MGE	Aprendizaje	10.5 ± 4.2	96.0±13.3	93.9±13.2
	ACID-MGE	Prefijado	10	95.7±14.1	93.5±14.7
	ACID-GE	Prefijado	10	**85.1±6.6**	**48.7±4.7**
Función gaussiana	ACID-MGE	Aprendizaje	10.9 ± 5.9	96.1±13.7	93.4±14.0
	ACID-MGE	Prefijado	10	93.6±16.4	90.4±18.0
	ACID-GE	Prefijado	10	**84.3±4.8**	**42.8±12.2**
Referencia de voltaje	ACID-MGE	Aprendizaje	12.8 ± 5.0	98.8±6.0	96.2±9.3
	ACID-MGE	Prefijado	10	98.4±7.6	96.0±9.4
	ACID-GE	Prefijado	10	**86.1±6.8**	**55.3±5.3**

Tabla 5.17. – ACID-GE vs ACID-MGE: longitud media (LM) y longitud expresada media (LEM) de los cromosomas en la población de la última generación de los algoritmos para los circuitos computacionales, 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	LM(%) ± SD	LEM(%) ±SD
Potencia al cuadrado	ACID-MGE	Aprendizaje	10.5 ± 5.3	97.8±11.1	95.7±10.9
	ACID-MGE	Prefijado	10	92.2±17.4	89.8±17.3
	ACID-GE	Prefijado	10	**86.8±1.5**	**52.0±5.8**
Raíz cuadrada	ACID-MGE	Aprendizaje	9.3 ± 2.9	92.0±17.0	89.3±17.8
	ACID-MGE	Prefijado	10	94.5±12.3	92.4±12.6
	ACID-GE	Prefijado	10	**82.5±7.2**	**56.9±8.9**
Potencia al cubo	ACID-MGE	Aprendizaje	10.0 ± 4.7	99.0±7.0	96.8±7.0
	ACID-MGE	Prefijado	10	99.7±2.2	97.5±2.3
	ACID-MGE	Prefijado	20	98.3±8.6	96.0±9.2
	ACID-GE	Prefijado	20	**86.0±6.8**	**58.0±6.9**
Raíz cúbica	ACID-MGE	Aprendizaje	15.8 ± 5.6	99.4±4.0	96.2±10.5
	ACID-MGE	Prefijado	15	98.8±7.7	96.2±7.9
	ACID-MGE	Prefijado	30	100.0±0.1	97.6±0.8
	ACID-GE	Prefijado	30	**83.3±2.3**	**73.3±8.1**

Tabla 5.18. – ACID-GE vs ACID-MGE: número medio de componentes (NMC) de los circuitos exitosos obtenidos, número medio de cromosomas expresables (NMCE) y número medio de cromosomas viables (NMCV) en la población de la última generación de los algoritmos para los circuitos no computacionales, 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	NMC±SD	NMCE±SD	NMCV±SD
Sensor de temperatura	ACID-MGE	Aprendizaje	10.5 ± 4.2	43.4±6.2	**1 000.0±0.0**	971.5±24.8
	ACID-MGE	Prefijado	10	45.0±7.3	**1 000.0±0.0**	**977.1±25.4**
	ACID-GE	Prefijado	10	**31.8±6.3**	941.7±15.9	894.2±39.6
Función gaussiana	ACID-MGE	Aprendizaje	10.9 ± 5.9	42.4±8.7	**1 000.0±0.0**	**966.8±24.2**
	ACID-MGE	Prefijado	10	43.9±6.5	**1 000.0±0.0**	**966.8±31.2**
	ACID-GE	Prefijado	10	**32.3±7.4**	945.6±21.9	891.4±28.9
Referencia de voltaje	ACID-MGE	Aprendizaje	12.8 ± 5.0	43.0±4.6	**1 000.0±0.0**	**975.5±22.6**
	ACID-MGE	Prefijado	10	45.6±6.2	**1 000.0±0.0**	971.8±22.9
	ACID-GE	Prefijado	10	**34.6±5.9**	922.7±21.9	866.0±44.6

longitud máxima en ACID-MGE, tanto si se utiliza aprendizaje como si no. En el caso de ACID-GE, la longitud media expresada es más baja que ACID-MGE en todos los casos. Adicionalmente, la longitud media expresada se encuentra más cerca de la longitud media del cromosoma en ACID-MGE, con y sin aprendizaje, que en el caso de ACID-GE.

Finalmente, el número de componentes en los circuitos exitosos es mayor para los casos de ACID-MGE, con y sin aprendizaje, que en el caso de ACID-GE. Este número resulta ser incluso mayor que el número de componentes que se pueden codificar en un cromosoma sin *wrapping*, lo que resulta indicativo de que se está utilizando *wrapping* en dichos casos.

Las conclusiones tras analizar todos estos resultados son las siguientes: (i) el efecto *bloat* parece ser ligeramente mayor en ACID-MGE (con y sin aprendizaje) que en el caso de ACID-GE; (ii) el número medio de componentes de circuito utilizados por ACID-MGE (con y sin aprendizaje) es ligeramente mayor que en ACID-GE, estando este resultado en consonancia con la conclusión anterior; y (iii) el número medio de cromosomas expresables y viables es mayor en ACID-MGE (con y sin aprendizaje) que en ACID-GE, indicando con ello que la variante MGE favorece la aparición de este tipo de cromosomas, facilitando así la búsqueda de la solución. Esto

Tabla 5.19. – AACID-GE vs ACID-MGE: número medio de componentes (NMC) de los circuitos exitosos obtenidos, número medio de cromosomas expresables (NMCE) y número medio de cromosomas viables (NMCV) en la población de la última generación de los algoritmos para los circuitos computacionales, 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN(±SD)	NMC±SD	NMCE±SD	NMCV±SD
Potencia al cuadrado	ACID-MGE	Aprendizaje	10.5 ± 5.3	43.0±7.6	**1 000.0±0.0**	**983.2±19.7**
	ACID-MGE	Prefijado	10	42.7±8.3	**1 000.0±0.0**	976.0±24.5
	ACID-GE	Prefijado	10	**32.9±7.2**	930.7±19.0	883.3±35.4
Raíz cuadrada	ACID-MGE	Aprendizaje	9.3 ± 2.9	41.5±8.2	**1 000.0±0.0**	971.6±23.5
	ACID-MGE	Prefijado	10	44.7±9.6	**1 000.0±0.0**	**976.5±16.3**
	ACID-GE	Prefijado	10	**30.1±9.1**	916.0±30.5	871.6±32.1
Potencia al cubo	ACID-MGE	Aprendizaje	10.0 ± 4.7	42.9±6.5	**1 000.0±0.0**	985.4±15.5
	ACID-MGE	Prefijado	10	43.8±6.6	**1 000.0±0.0**	**987.5±8.6**
	ACID-MGE	Prefijado	20	43.9±5.6	**1 000.0±0.0**	969.3±16.5
	ACID-GE	Prefijado	20	**34.2±8.7**	912.4±33.9	859.7±49.7
Raíz cúbica	ACID-MGE	Aprendizaje	15.8 ± 5.6	42.7±3.5	**1 000.0±0.0**	**965.1±32.0**
	ACID-MGE	Prefijado	15	44.6±5.6	**1 000.0±0.0**	959.7±32.0
	ACID-MGE	Prefijado	30	45.5±5.7	**1 000.0±0.0**	949.1±24.1
	ACID-GE	Prefijado	30	**41.0±0.0**	835.8±38.7	749.1±67.3

Tabla 5.20. – Algoritmo ACID-MGE: comparación del número medio de componentes (NMC) y del número de componentes del mejor circuito (NCMC) usando o no parsimonia (PAR), para los circuitos no computacionales con 3 000 generaciones.

Circuito	Algoritmo	MNN	SR	NMC	NCMC
Sensor de temperatura	ACID-MGE + PAR	10	60.0 %	38.7±9.7	**8**
	ACID-MGE	10	**82.0** %	45.0±7.3	50
Función gaussiana	ACID-MGE + PAR	10	66.0 %	39.2±11.1	**9**
	ACID-MGE	10	**70.0** %	43.9±6.5	44
Referencia de voltaje	ACID-MGE + PAR	10	20.0 %	37.1±11.0	**13**
	ACID-MGE	10	**22.0** %	45.6±6.2	42

último podría ayudar a explicar por qué la calidad de las soluciones obtenidas por ACID-MGE sean mejores que las obtenidas por ACID-GE, tal y como evidencian los resultados mostrados en las tablas 5.8 y 5.9.

5.5.7. Parsimonia simple

Según se ha visto en la sección 5.5.6, el algoritmo ACID-MGE sufre un efecto *bloat* algo más acusado que ACID-GE. Con vistas a reducir el impacto de dicho efecto, se realizaron experimentos introduciendo parsimonia simple en ACID-MGE, que conllevaron el uso de una función de adaptación modificada (véase la sección 4.3). Las tablas 5.20 y 5.21 muestran los resultados de ACID-MGE, comparando el uso y no uso de parsimonia para los circuitos computacionales y no computacionales, respectivamente. Se muestran los valores de SR, el valor medio de componentes y el número de componentes del mejor circuito obtenido.

Atendiendo al número medio de componentes, se observa que ACID-MGE con parsimonia logra reducir dicho número frente al algoritmo ACID-MGE sin parsimonia. Adicionalmente, atendiendo al número de componentes del mejor circuito obtenido, se observa que ACID-MGE con parsimonia logra reducir notablemente este número frente a ACID-MGE sin parsimonia en los siete circuitos. En el caso del circuito de raíz cúbica, sin embargo, ha sido necesario aumentar el número de generaciones a 6 000, además de utilizar el valor de $MNN = 15$, para conseguir una reducción notable del número de componentes. Es importante indicar que este circuito es el de mayor complejidad, tanto en número de componentes, como dando lugar a uno

Tabla 5.21. – Algoritmo ACID-MGE: comparación del número medio de componentes (NMC) y del número de componentes del mejor circuito (NCMC) usando o no parsimonia (PAR), para los circuitos computacionales usando 3 000 generaciones, con excepción del circuito de raíz cúbica que, con parsimonia, requirió 6 000 generaciones.

Circuito	Algoritmo	MNN	Nº Gen.	SR	NMC	NCMC
Potencia al cuadrado	ACID-MGE + PAR	10	3 000	**88.0** %	34.0±11.7	**7**
	ACID-MGE	10	3 000	84.0 %	42.7±8.3	42
Raíz cuadrada	ACID-MGE + PAR	10	3 000	**76.0** %	40.5±10.0	**11**
	ACID-MGE	10	3 000	66.0 %	44.7±9.6	44
Potencia al cubo	ACID-MGE + PAR	20	3 000	**88.0** %	40.7±8.7	**10**
	ACID-MGE	20	3 000	84.0 %	43.9±5.6	47
Raíz cúbica	ACID-MGE + PAR	15	6 000	**32.0** %	42.8±7.4	**22**
	ACID-MGE + PAR	30	3 000	18.0 %	44.8±2.2	50
	ACID-MGE	30	3 000	22.0 %	45.5±5.7	57

de los SR más bajos obtenidos, frente a los demás circuitos sintetizados (véanse las tablas 5.8 y 5.9).

Atendiendo al SR y, en el caso de raíz cúbica comparando con el experimento con 3 000 generaciones, se observa que ACID-MGE con parsimonia ofrece un mejor valor en tres de los siete circuitos, mientras que ACID-MGE sin parsimonia ofrece el mejor valor en los cuatro circuitos restantes. Sin embargo, estas diferencias de SR se consideran espurias, pues los algoritmos ACID-MGE con parsimonia y sin parsimonia comparten función de adaptación en la primera parte del algoritmo, hasta conseguir un circuito exitoso. La mejora posterior no afecta a la calificación de circuito exitoso ya obtenido, ni se ha observado ningún caso en el que se haya obtenido un circuito al final de las generaciones totales que resulte ser no exitoso cuando previamente se había conseguido un éxito en una generación anterior. No obstante, es cierto que la nueva restricción impuesta por el mecanismo de parsimonia, dificulta la obtención del circuito óptimo, dado que, en este caso, se persigue la obtención de un circuito que cumpla las especificaciones de diseño y, además, minimice el número de componentes utilizados.

5.5.8. Salidas de los circuitos obtenidos mediante ACID-MGE con parsimonia simple

En esta sección se analizan las salidas esperadas y las medidas en los mejores circuitos obtenidos mediante el algoritmo ACID-MGE con parsimonia simple. En las figuras 5.8a, 5.8b y 5.8c se muestra el voltaje de salida del mejor circuito sensor de temperatura, el voltaje de salida del mejor circuito de referencia de voltaje y la corriente de salida del mejor circuito de generación gaussiana, respectivamente. En las figuras 5.20, 5.22, 5.21 y 5.23, se muestran los voltajes de salida de los circuitos de potencia al cuadrado, potencia al cubo, raíz cuadrada y raíz cúbica, respectivamente.

Como se puede observar, el ajuste no es tan bueno como en el caso de ACID-MGE sin parsimonia (véanse las figuras 5.6 y 5.7). Dado que el mecanismo de parsimonia implica la optimización de un objetivo adicional respecto del caso sin parsimonia, se podría pensar en aumentar el número máximo de generaciones en caso del primero, para aumentar el tiempo disponible en la búsqueda de circuitos aún más competitivos que los ya obtenidos. Este estudio no se ha abordado en esta tesis y se plantea como trabajo futuro.

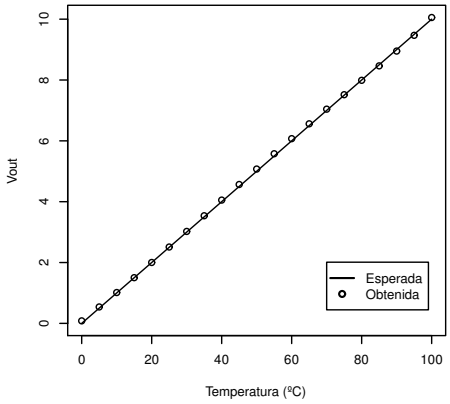
Los esquemas de estos circuitos se muestran en la sección 5.6 (véanse las figuras 5.17, 5.18, 5.19, 5.20, 5.21, 5.22 y 5.23) y sus *netlists* asociadas se muestran en el apéndice C.3.

5.6. Mejores circuitos obtenidos por ACID-GE y ACID-MGE

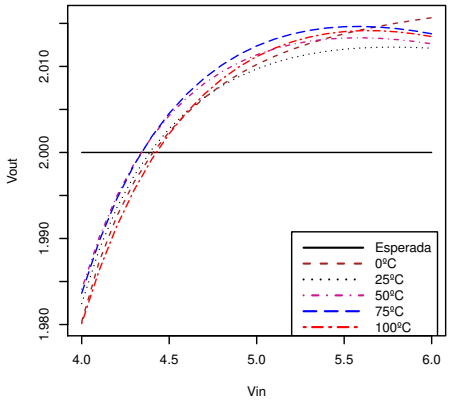
En esta sección se muestran los mejores circuitos obtenidos con los algoritmos propuestos en esta tesis. En primer lugar, se muestran los circuitos obtenidos con el algoritmo ACID-GE y, a continuación, se muestran los circuitos obtenidos con el algoritmo ACID-MGE con parsimonia simple.

5.6.1. Circuitos obtenidos con el algoritmo ACID-GE

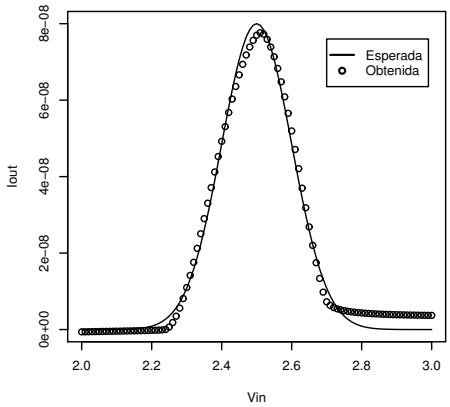
Para mostrar los esquemas de los circuitos obtenidos mediante ACID-GE, se han sustituido bloques de resistencias conectadas por su resistencia equivalente. Adicionalmente, la tabla 5.22 muestra los tipos de transistor MOSFET y la anchura del



(a) Sensor de temperatura.

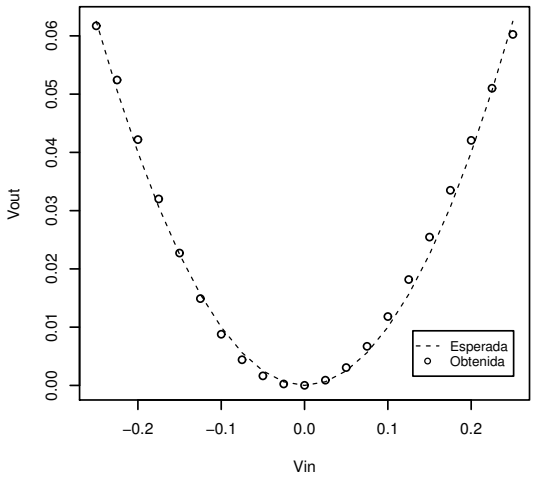


(b) Referencia de voltaje. Nótese la escala reducida en el eje Y ($\triangle V_{out} = 2mV$).

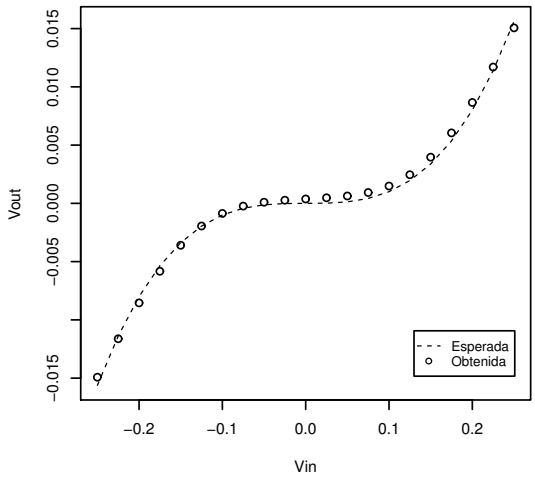


(c) Generador de función gaussiana.

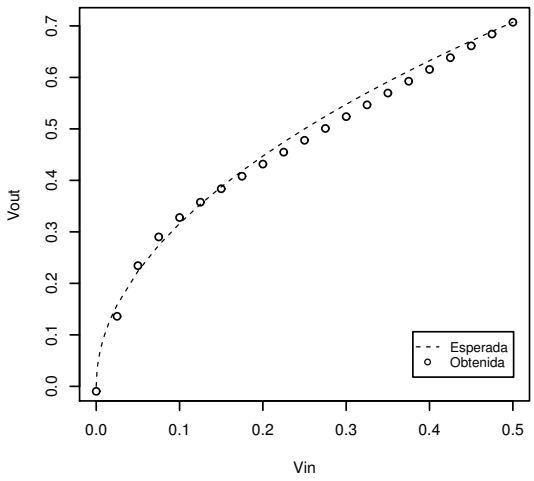
Figura 5.8. – Algoritmo ACID-MGE: salida medida frente a salida esperada para los mejores circuitos no computacionales obtenidos usando parsimonia simple.



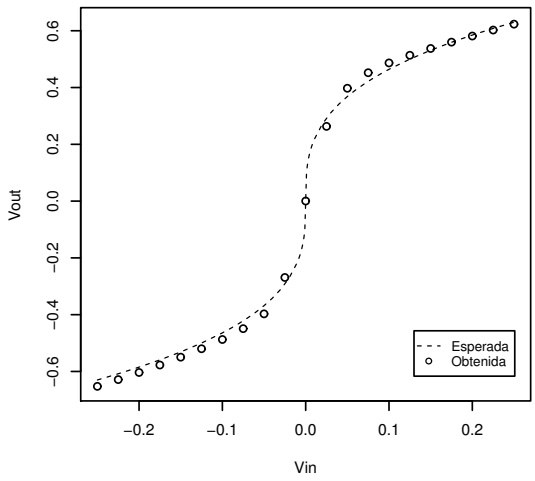
(a) Circuito de potencia al cuadrado.



(b) Circuito de potencia al cubo.



(c) Circuito de raíz cuadrada.



(d) Circuito de raíz cúbica.

Figura 5.9. – Algoritmo ACID-MGE: salida medida frente a salida esperada para los mejores circuitos computacionales obtenidos usando parsimonia simple.

canal obtenidos por el algoritmo para el circuito de función gaussiana mostrado en la figura 5.12.

En relación con los circuitos no computacionales, las figuras 5.10, 5.11 y 5.12 muestran el mejor circuito sensor de temperatura, el mejor circuito de referencia de voltaje y el mejor circuito generador de función gaussiana, respectivamente. En relación con los circuitos computaciones, las figuras 5.13, 5.14, 5.15 y 5.16 muestran el mejor circuito de función potencia al cuadrado, el mejor circuito de raíz cuadrada, el mejor circuito de función potencia al cubo y el mejor circuito de raíz cúbica, respectivamente.

Tabla 5.22. – Algoritmo ACID-GE: anchura del canal y tipo de los transistores MOSFET del mejor circuito de función gaussiana obtenido, mostrado en la figura 5.12.

Transistor	**Tipo**	**Anchura del canal** (μm)
M1	Canal N	175
M2	Canal N	199
M3	Canal N	74
M4	Canal N	199
M5	Canal P	142
M6	Canal N	11
M7	Canal P	10
M8	Canal N	91
M9	Canal N	199
M10	Canal N	199
M11	Canal N	199
M12	Canal N	99
M13	Canal N	35
M14	Canal N	179
M15	Canal P	46
M16	Canal N	197
M17	Canal P	124

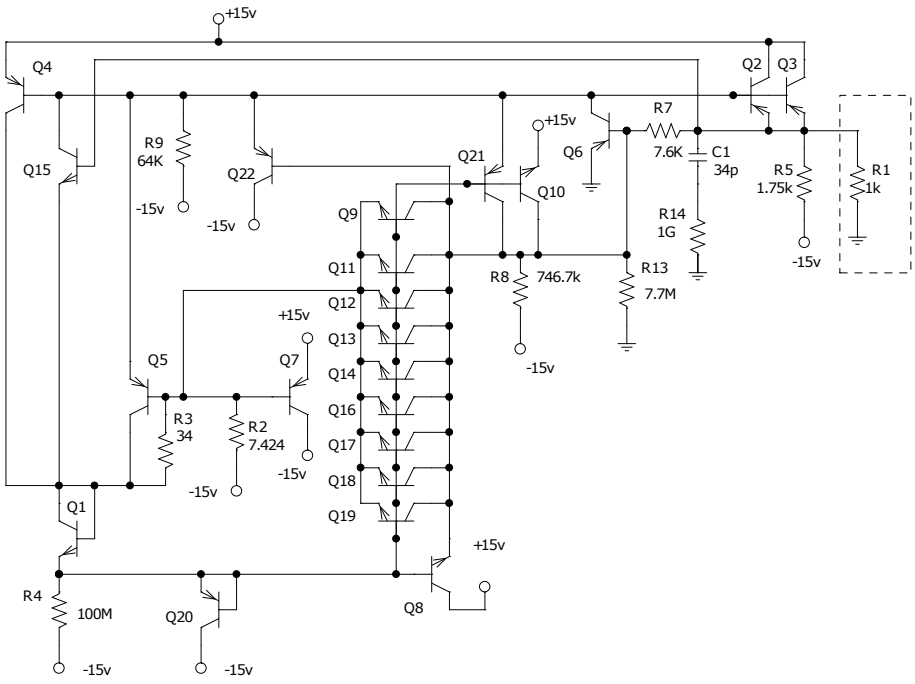


Figura 5.10. – Algoritmo ACID-GE: mejor circuito sensor de temperatura obtenido, con un valor de $MNN = 6$ y 10 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de temperatura.

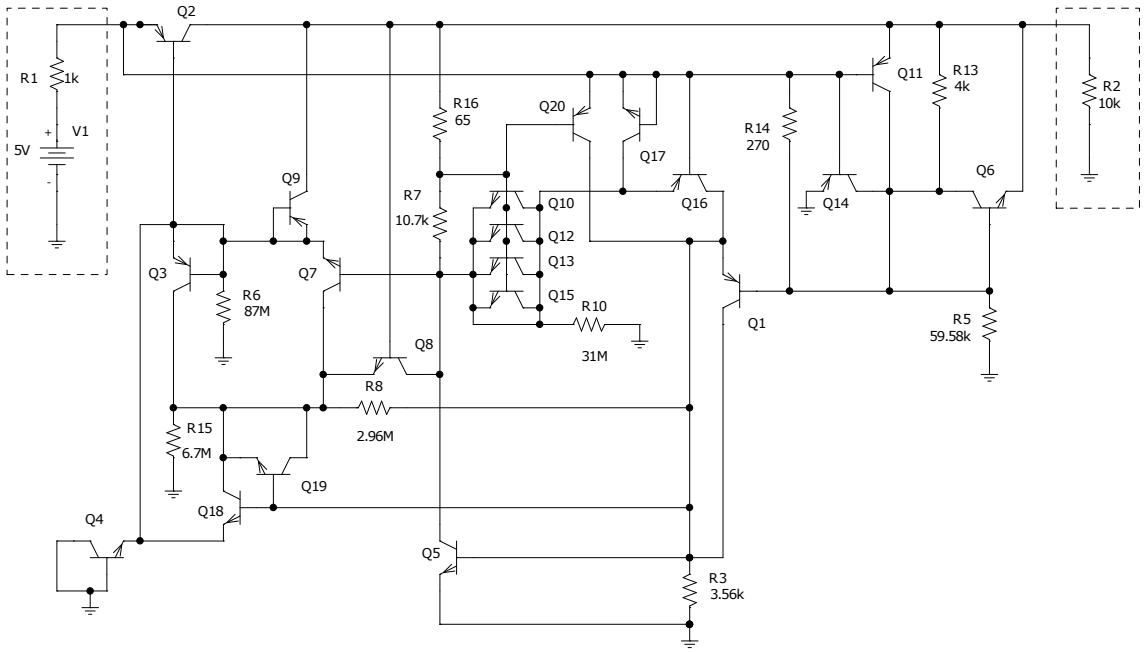


Figura 5.11. – Algoritmo ACID-GE: mejor circuito de referencia de voltaje obtenido, con un valor de $MNN = 6$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC y temperatura.

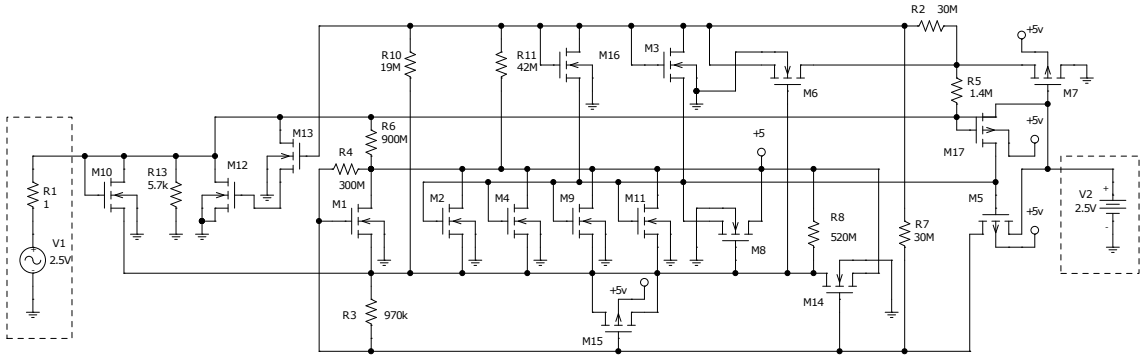


Figura 5.12. – Algoritmo ACID-GE: mejor circuito de función gaussiana obtenido, con un valor de $MNN = 6$ y 10 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC.

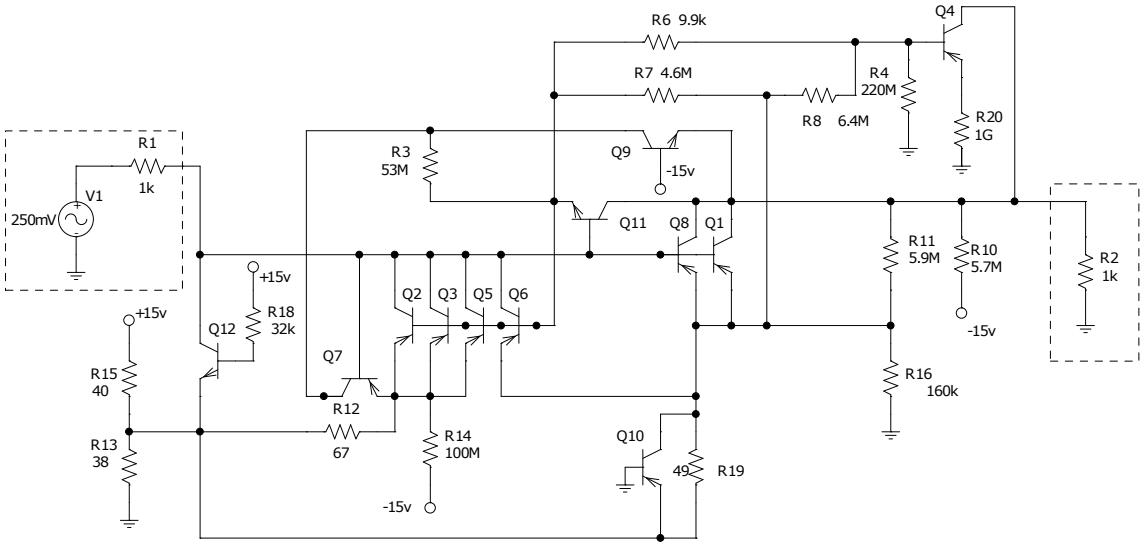


Figura 5.13. – Algoritmo ACID-GE: mejor circuito computacional para la función potencia al cuadrado obtenido, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis en el dominio del tiempo.

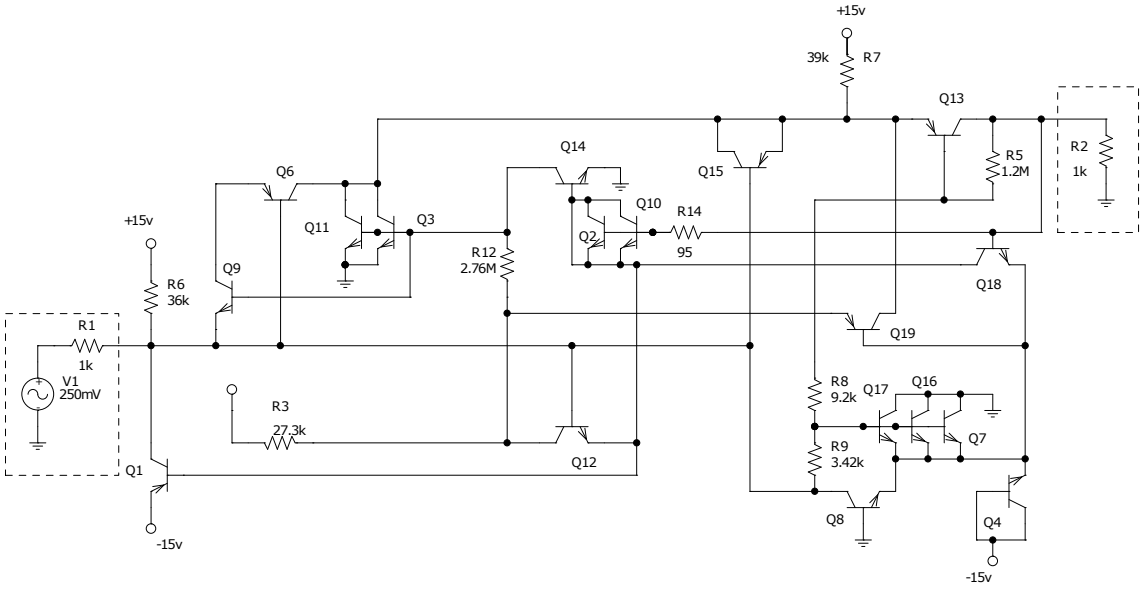


Figura 5.14. – Algoritmo ACID-GE: mejor circuito computacional para la función raíz cuadrada obtenido, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC.

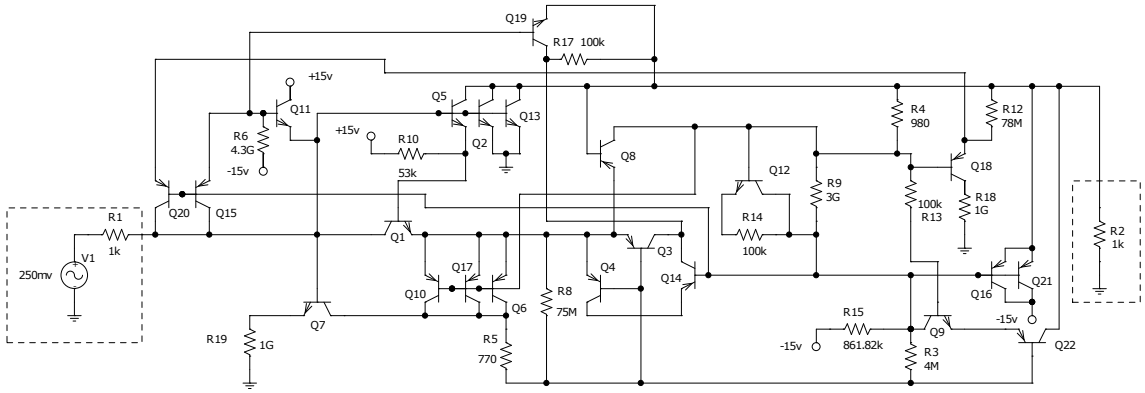


Figura 5.15. – Algoritmo ACID-GE: mejor circuito computacional para la función potencia al cubo obtenido, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC.

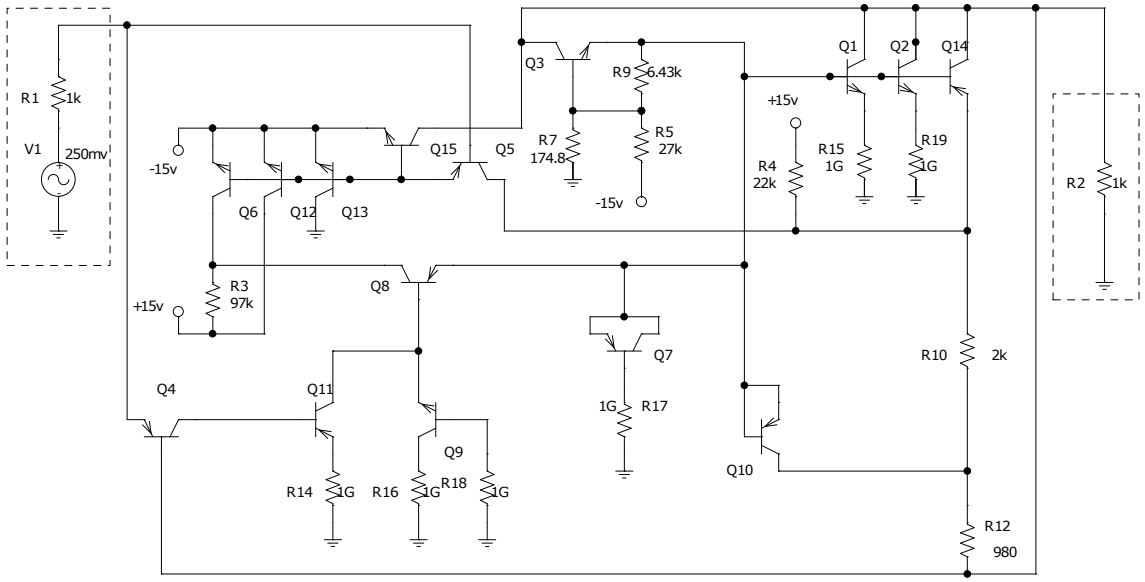


Figura 5.16. – Algoritmo ACID-GE: mejor circuito computacional para la función raíz cúbica obtenido, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC.

5.6.2. Circuitos obtenidos con el algoritmo ACID-MGE sin parsimonia simple

Es importante indicar que la tarea de representar el esquema un circuito a partir de su *netlist* no es directa, ni existen herramientas *open-source* para automatizar esta tarea. Por ello, en los casos en los que se ha abordado en esta tesis, se ha realizado manualmente. Adicionalmente, el número medio de componente en los circuitos obtenidos con ACID-MGE es mayor que con ACID-GE, tal y como se describió en la sección 5.5.6, aumentando así la dificultad que ya se experimentó al representar los esquemáticos de los circuitos obtenidos por ACID-GE.

Por estos motivos, no se muestran los esquemáticos de los mejores circuitos obtenidos con ACID-MGE sin parsimonia. No obstante, pensamos que esto no es una carencia grave, dado que la salida por ellos producida sí se mostró en las figuras 5.6 y 5.7 y, además, en el apéndice C.2 se muestran las *netlists* de los respectivos mejores circuitos con este algoritmo, tanto computacionales como no computacionales. Adicionalmente, el uso de la netlist permite reproducir fácilmente el comportamiento cada uno de los circuitos obtenidos mediante el simulador SPICE, o alguna de sus variantes.

5.6.3. Circuitos obtenidos con el algoritmo ACID-MGE con parsimonia simple

En esta sección se muestran los circuitos obtenidos con el algoritmo ACID-MGE con parsimonia simple. Los circuitos obtenidos se muestran tal cual se han obtenido del algoritmo, sin realizar ninguna simplificación, salvo donde se indique lo contrario. Adicionalmente, la tabla 5.23 muestra los tipos de transistor MOSFET y la anchura del canal obtenidos para el circuito de función gaussiana mostrado en la figura 5.19.

En relación con los circuitos no computacionales, la figura 5.17, 5.18 y 5.19 muestran el mejor circuito sensor de temperatura, el mejor circuito de referencia de voltaje y el mejor circuito generador de función gaussiana, respectivamente. En relación con los circuitos computacionales, la figura 5.20, 5.21, 5.22 y 5.23 muestran el mejor circuito de función potencia al cuadrado, el mejor circuito de raíz cuadrada, el mejor circuito de función potencia al cubo y el mejor circuito de raíz cúbica, respectivamente.

Tabla 5.23. – Algoritmo ACID-MGE: anchura del canal y tipo de los transistores MOSFET del mejor circuito de función gaussiana obtenido usando parsimonia simple, mostrado en la figura 5.19.

Transistor	**Tipo**	**Anchura del canal** (μm)
M1	Canal N	121
M2	Canal P	31
M3	Canal N	108
M4	Canal N	56

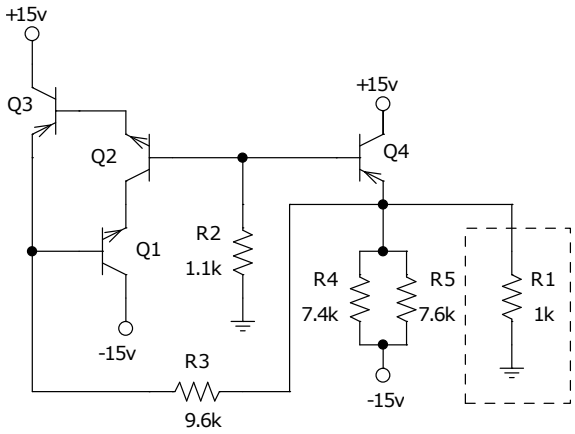


Figura 5.17. – Algoritmo ACID-MGE: Mejor circuito sensor de temperatura obtenido usando parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de temperatura.

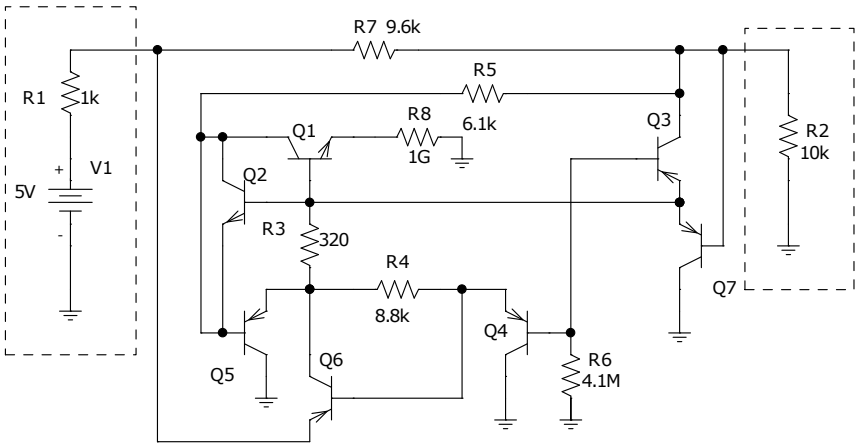


Figura 5.18. – Algoritmo ACID-MGE: Mejor circuito de referencia de voltaje obtenido usando parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC y temperatura.

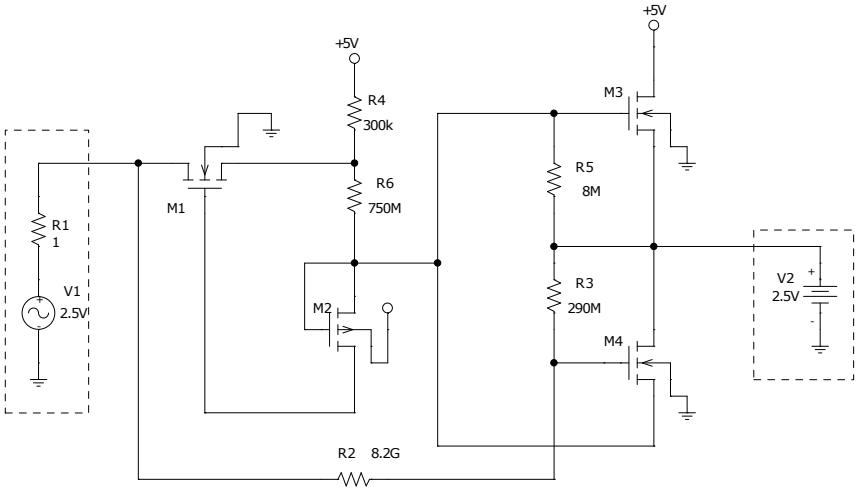


Figura 5.19. – Algoritmo ACID-MGE: Mejor circuito de función gaussiana obtenido usando parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis por barrido de voltaje DC.

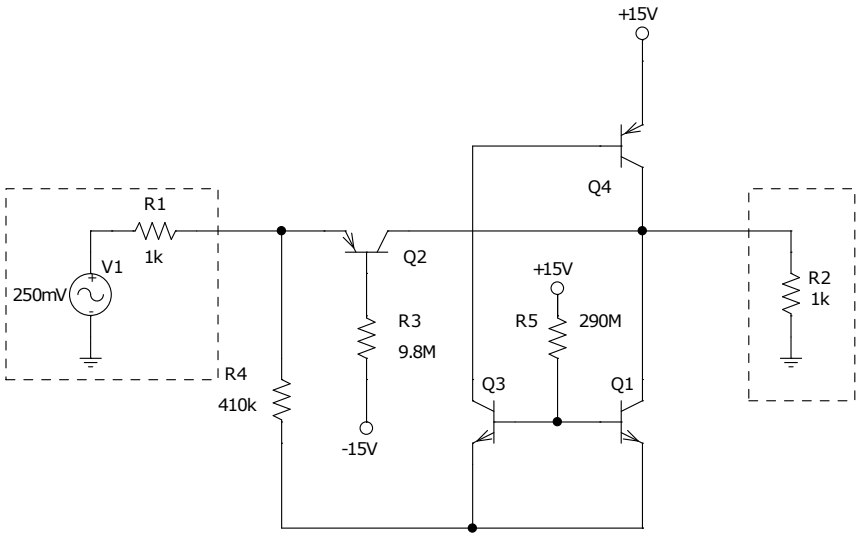


Figura 5.20. – Algoritmo ACID-MGE: Mejor circuito computacional para la función potencia al cuadrado obtenido usando parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis en el dominio del tiempo.

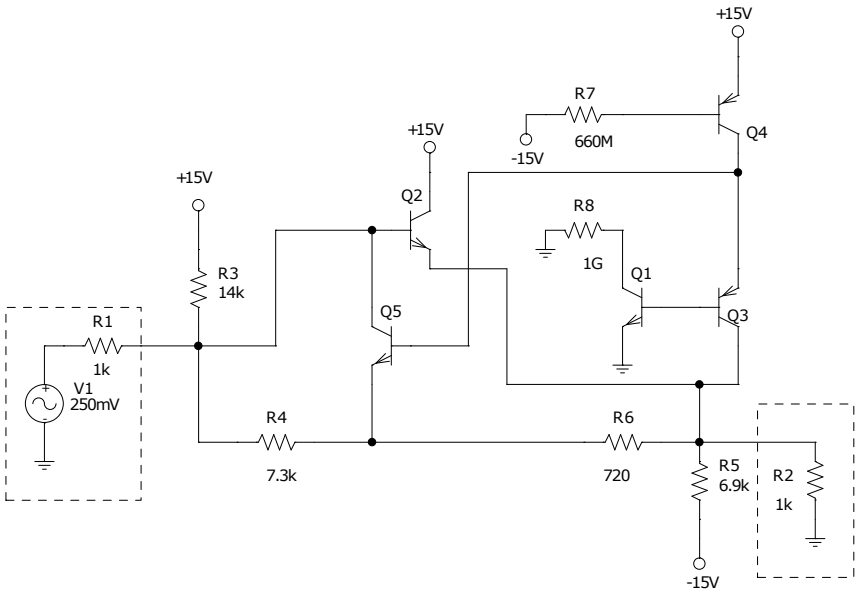


Figura 5.21. – Algoritmo ACID-MGE: Mejor circuito computacional para la función raíz cuadrada obtenido usando parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis en el dominio del tiempo.

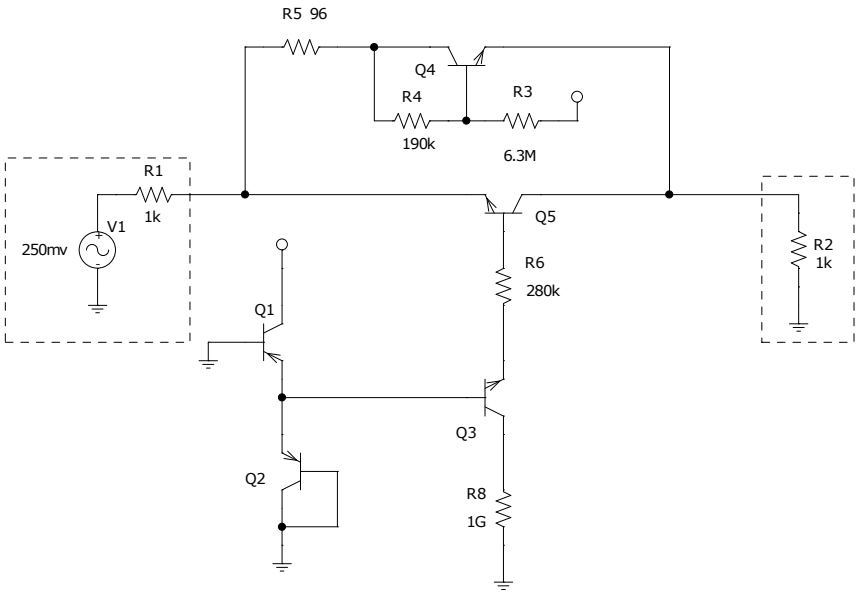


Figura 5.22. – Algoritmo ACID-MGE: Mejor circuito computacional para la función potencia al cubo obtenido usando parsimonia simple, con un valor de $MNN = 20$ y 3 000 generaciones. En este esquemático se ha eliminado la resistencia R7, de $1G\Omega$, utilizada para conectar la alimentación negativa de $-15V$ a masa, que no tiene función alguna en el circuito. Este circuito se obtuvo mediante una función de adaptación basada en un análisis en el dominio del tiempo.

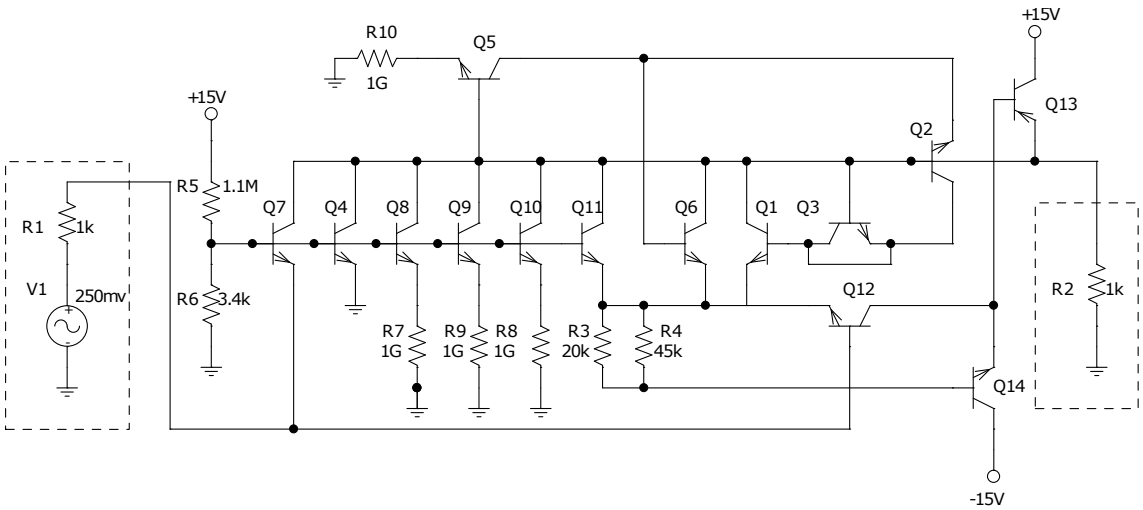


Figura 5.23. – Algoritmo ACID-MGE: Mejor circuito computacional para la función raíz cúbica obtenido usando parsimonia simple, con un valor de $MNN = 15$ y 6 000 generaciones. Este circuito se obtuvo mediante una función de adaptación basada en un análisis en el dominio del tiempo.

6. Conclusiones y trabajo futuro

En este capítulo se exponen las conclusiones y trabajo futuro de esta tesis. En primer lugar, se revisa el cumplimiento de los objetivos planteados en la sección 1.3, junto con las conclusiones derivadas de dicha revisión. A continuación, se analiza el cumplimiento de la hipótesis principal planteada en esta tesis. Finalmente, se plantean posibles trabajos futuros que podrían continuar alguna de las líneas de investigación abiertas en esta tesis.

6.1. Conclusiones

En esta sección se analizará el grado de consecución de cada uno de los objetivos planteados inicialmente (véase la sección 1.3), junto con cada una de las conclusiones parciales resultante de dicho análisis. Finalmente, dichas conclusiones permitirán analizar la validez de la hipótesis de investigación planteada en esta tesis.

Se han elegido siete circuitos *benchmark*, agrupados en dos conjuntos que se han denominado: circuitos computacionales y no computacionales, dando cumplimiento al subobjetivo 1. La elección de estos circuitos *benchmark* es representativa del estado del arte y, por tanto, se puede concluir que la comparación de resultados de la aproximación propuesta con otros trabajos relevantes en la literatura que han abordado dichos circuitos es representativa del estado del arte.

Se ha desarrollado una gramática genérica, fácilmente adaptable a cada circuito abordado, para el diseño de circuitos electrónicos analógicos en el marco de EG. Esta gramática hace uso del concepto de gramática de bloques, lo que permite el uso de operadores de cruce más eficaces, tal y como el que se ha propuesto, el denominado operador de cruce *one-block*. Adicionalmente, se ha aportado evidencia de que este operador se comporta igual o mejor que el operador de cruce estándar de un punto en todos los casos de estudio analizados. Estas dos contribuciones de la tesis permiten dar cumplimiento a los subobjetivos 2 y 3.

Se ha implementado un algoritmo basado en EG, denominado ACID-GE, capaz de diseñar con éxito los siete circuitos *benchmark,* dando cumplimiento al subobjetivo 4.

Se ha evaluado el algoritmo ACID-GE, sobre los siete circuitos *benchmark*, realizando un estudio estadístico del impacto de diferentes parámetros, como el número máximo de generaciones y el máximo número de nodos (MNN) sobre la tasa de éxitos. Esta contribución ha permitido cumplir el subobjetivo 5.

Se ha realizado una comparación de los resultados obtenidos por el algoritmo ACID-GE frente a los resultados obtenidos por otros algoritmos previamente descritos en la literatura, dando cumplimiento al subobjetivo 6. El resultado de la comparativa permite afirmar que los resultados obtenidos con ACID-GE son competitivos.

La búsqueda de alternativas para estudiar la mejora de los resultados obtenidos con ACID-GE ha dado lugar a la propuesta de una nueva variante de EG, que se ha denominado *evolución multigramatical* (EMG). Esta contribución da cumplimiento así al subobjetivo 7.

Se ha desarrollado un conjunto de gramáticas, dentro del marco de EMG, para el diseño de circuitos analógicos. Esta actuación da lugar al cumplimiento del subobjetivo 8.

Las características de EMG obligan a definir un nuevo operador de cruce si, como se hizo en el enfoque basado en EG, se quiere evitar el potencial efecto destructivo de un operador de cruce estándar de un punto. Siguiendo esta dirección, este nuevo operador propuesto se ha denominado operador de cruce BG-MHX. Esta actuación da lugar al cumplimiento del subobjetivo 9.

Se ha implementado un algoritmo basado en EMG, denominado ACID-MGE, capaz de diseñar con éxito los siete circuitos *benchmark,* dando cumplimiento al subobjetivo 10.

Se ha evaluado el algoritmo ACID-MGE sobre los circuitos *benchmark* y los resultados obtenidos se han comparado con ACID-GE. Esta actuación ha dado lugar al cumplimiento del subobjetivo 11. En la comparativa de los resultados obtenidos por ACID-MGE y ACID-GE, desde el punto de vista del SR, se observa una mejora estadísticamente significativa ($\alpha < 0.05$) en cinco circuitos del total de siete. Sin embargo, en los dos circuitos restantes no es posible rechazar la hipótesis nula, si bien, aun en estos dos casos, el valor obtenido de SR por ACID-MGE es igual o mejor que el conseguido por ACID-GE.

Se ha introducido en ACID-MGE un mecanismo para el aprendizaje de parámetros de gramática, para lo que también se han desarrollado nuevas extensiones para la definición de una gramática. Dicho mecanismo se ha aplicado al parámetro MNN. Esto ha supuesto también una adaptación en la definición de las gramáticas utilizadas inicialmente en ACID-MGE. Estas actuaciones han dado lugar al cumplimiento del subobjetivo 12

Se han realizado pruebas de evaluación del algoritmo ACID-MGE con aprendizaje del parámetro MNN sobre los circuitos *benchmark,* incluyendo un estudio estadístico del mismo. Esta actuación ha dado lugar al cumplimiento del subobjetivo 13. En la comparativa de los resultados obtenidos utilizando ACID-MGE con y sin aprendizaje del parámetro MNN, se ha observado que se obtienen buenos resultados usando la opción del aprendizaje del parámetro MNN, sin una pérdida estadísticamente significativa de SR en seis de los siete circuitos *benchmark*. La valoración de este resultado debería ser aún más positiva, teniendo en cuenta que el aprendizaje del parámetro MNN tiene un coste adicional al del cumplimiento de las especificaciones de diseño y, sin embargo, se mantuvo idéntico el número máximo de generaciones al comparar ambas aproximaciones.

Se ha comprobado que el algoritmo ACID-MGE sufre el efecto engorde de una forma algo más acusada que el algoritmo ACID-GE, dando lugar, el primero, a circuitos con un mayor número de componentes. Para mitigar este efecto, se ha incorporado un mecanismo de manejo de restricciones, denominado parsimonia simple, en ACID-MGE, buscando obtener circuitos más simples y que, además, cumplan las especificaciones de diseño. Esto ha permitido cumplir con el subobjetivo 14.

Se han realizado pruebas del algoritmo ACID-MGE con parsimonia simple y se han obtenido circuitos competitivos, es decir, que garantizan las especificaciones de diseño y, además, con un número menor de componentes que el obtenido con ACID-MGE sin parsimonia simple. Esto ha garantizado el cumplimiento del subobjetivo 15.

Finalmente, respecto a la hipótesis principal de esta tesis: **asumir que es posible diseñar e implementar algoritmos basados en EG para abordar el problema del diseño automático de circuitos analógicos de forma eficaz y eficiente**, y atendiendo a las distintas conclusiones derivadas del cumplimiento de los diferentes objetivos, se puede afirmar que **se aporta suficiente evidencia en esta tesis para validar dicha hipótesis**.

6.2. Trabajo futuro

Aunque los circuitos abordados son circuitos complejos en cuanto a la necesidad de conocimiento experto para su diseño por diseñadores humanos, resultan ser pequeños con relación al diseño de circuitos prácticos. Con vistas a permitir el uso de estos algoritmos de forma industrial, se plantea la necesidad de abordar circuitos de mayor tamaño, atendiendo al, ya mencionado, problema de la escalabilidad. Otro aspecto que sería necesario abordar como trabajo futuro es el de la robustez de los circuitos obtenidos, para que puedan funcionar dentro de un rango más amplio de condiciones, como un mayor rango de temperatura o, por otra parte, atendiendo a la dispersión de valores de los componentes.

El mecanismo de parsimonia simple incorporado en ACID-MGE se ha mostrado eficaz sobre los circuitos abordados, permitiendo la reducción del número de componentes de los circuitos obtenidos. Sin embargo, dicho mecanismo detiene la mejora de los resultados una vez se cumplen las especificaciones de diseño, para enfocarse en la reducción del número de componentes. Así, dicho mecanismo da lugar a circuitos que cumplen las especificaciones al mínimo. Para mejorar esta situación, se podría plantear un mecanismo basado en la dominancia de Pareto, lo que permitiría compaginar la mejora de ambos objetivos: mejora del cumplimiento de especificaciones y, además, reducción del número de componentes. Este trabajo también se podría abordar mediante un aumento del número de generaciones sobre la implementación realizada, para conseguir circuitos aún más competitivos que los ya obtenidos.

Finalmente, el nuevo algoritmo EMG, propuesto en esta tesis, se ha mostrado eficaz y competitivo en los problemas abordados. Sin embargo, se plantea la necesidad de más investigación sobre el mismo, para validar su competitividad frente a EG en un abanico mayor de problemas.

APÉNDICES

A. Publicaciones asociadas a esta tesis

Los resultados de esta tesis han dado lugar a las siguientes publicaciones:

- Castejón, F. & Carmona, E. J. (2013). Automatic design of electronic amplifiers using grammatical evolution. En Alonso-Betanzos, A. *et al.*, editores, Actas de Multiconferencia CAEPIA-13, páginas 703–712. Índices de calidad: Microsoft Academic Field Rating: 7. Área: Computer Science, Artificial Intelligence.
- Castejón, F. & Carmona, E. J. (2018). Automatic design of analog electronic circuits using grammatical evolution. Applied Soft Computing, 62:1003 – 1018. Índices de calidad de la revista: factor de impacto (JCR 2018): 4.873. Ranking: 20/134 (Q1). Área: Computer Science, Artificial Intelligence.
- Castejón, F. & Carmona, E. J. (2020). Multi-grammatical evolution: A new variant of grammatical evolution to solve hierarchically decomposable design problems. IEEE Access. [En revisión]. Índices de calidad de la revista: factor de impacto (JCR 2018): 4.098. Ranking 23/155 (Q1). Área: Computer Science, Information Sytems.
- Castejón, F. & Carmona, E. J. (2020). Efficient analog circuit design based on multi-grammatical evolution. Electronics. [En preparación]. Índices de calidad de la revista: factor de impacto (JCR 2018): 1.764. Ranking 154/266 (Q3). Área: Engineering, Electrical & Electronic.

B. Resultados del problema de regresión simbólica

En este apéndice se muestran los resultados del problema de regresión simbólica planteado en la sección 3.3.3. Este problema se planteó como una especie de «problema de juguete» para implementar y valorar inicialmente las ideas del paradigma EMG, antes de aplicarlo al problema del diseño automático de circuitos electrónicos analógicos.

B.1. Definición de la función objetivo

La función objetivo del problema de regresión simbólica se muestra en la ecuación (B.1). Las funciones *kernel* utilizadas en este problema de regresión fueron las definidas en la ecuación (3.3). El dominio de definición en el que realiza el ajuste es $[0, 0.2]$. La función de evaluación utilizada corresponde al error cuadrático medio evaluado en 41 puntos de ajuste equiespaciados sobre el dominio indicado. El criterio de éxito viene dado cuando el valor de adaptación es menor que el valor umbral 0.01, diciendo entonces que se obtiene una expresión exitosa.

$$\begin{aligned} f(x) = 2\,\exp(-10^3(0.11-x)^2) + \tanh(30x-4.5) + \\ \tanh(10^2x-2.5) + 10^3(0.3x-0.035)^2 \end{aligned} \tag{B.1}$$

B.2. Parámetros del algoritmo

En esta sección se describen los parámetros de configuración de los algoritmos EG y EMG utilizados en la solución del problema de regresión simbólica. Concretamente, las tablas B.1 y B.2 muestran los parámetros de los algoritmos EG y EMG,

respectivamente. Por otro lado, la tabla B.3 muestra la gramática utilizada en el experimento con el algoritmo EG, mientras que las gramáticas I y II utilizadas con EMG se muestran en las tablas 3.3 y 3.4, respectivamente.

Tabla B.1. – Algoritmo EG: parámetros de configuración para el problema de regresión simbólica.

Función objetivo	ver sección B.1
Rango de ajuste	$[0, 0.2]$
Gramática	Ver tablas B.3
Motor de búsqueda	AG con cromosomas de tamaño variable
Función de adaptación	Error cuadrático medio evaluado en 41 puntos de ajuste
Condición de éxito	$MSE < 0.01$
Tamaño de la población (μ)	1 000
Representación	Cadenas de codones de longitud variable
Generaciones	3 000
Inicialización	Cadenas de codones aleatorias de longitud en el rango 50 y 100
Longitud máxima del cromosoma	130
Operador de cruce	Operador de un punto estándar (ver sección 3.2.7)
Tasa de cruce	0.4
Operador de mutación	Operador de mutación *Bitwise* (véase la sección 3.2.7)
Tasa de mutación	0.002
Selección de padres	Selección por torneo (tamaño=3)
Selección de supervivientes	Reemplazo generacional ($\lambda = \mu$)
Elitismo	2
Wrapping	4
Condición de terminación	$MSE < 0.01$
Generador de números aleatorios	Mersenne *twister*
Ejecuciones por experimento	400

B.3. Resultados

En esta sección se muestran los resultados del problema de regresión simbólica para la función objetivo mostrada en la sección B.1. La tabla B.4 muestra los resultados de los experimentos con EMG y EG.

Como se puede observar, el valor de SR obtenido por EMG es mayor que el obtenido mediante EG. La diferencia es estadísticamente significativa, como se puede ver en

Tabla B.2. – Algoritmo EMG: parámetros de configuración para el problema de regresión simbólica.

Función objetivo	Ver sección B.1
Rango de ajuste	$[0, 0.2]$
Gramáticas	Ver tablas 3.3 y 3.4
Motor de búsqueda	AG con cromosomas de tamaño variable
Función de adaptación	Error cuadrático medio evaluado en 41 puntos de ajuste
Tamaño de la población (μ)	1 000
Representación	Cadena de codones de longitud variable
Número máximo de generaciones ($NG_{máx}$)	3 000
Inicialización	Cadenas de codones aleatorias de longitud en el rango 50 y 100
Longitud máxima del cromosoma	130 codones
Cruce	Operador de cruce BG-MHX (véase la sección 3.4.2)
Probabilidad de cruce	0.4
Tamaño de bloque	Dos particiones con tamaños de bloque 1y 12 codones respectivamente
Mutación	Operador de mutación *Bitwise* (véase la sección 3.2.7)
Probabilidad de mutación	0.002
Selección de padres	Selección por torneo (tamaño=3)
Reemplazo	Reemplazo generacional ($\lambda = \mu$)
Elitismo	2
Wrapping	4
Condición de terminación	$MSE < 0.01$
Generador de números aleatorios	Mersenne *twister*
Número de ejecuciones por experimento	400

el p-valor, obtenido mediante un test de hipótesis de una cola con $\alpha < 0.05$, donde dicho p-valor queda muy por debajo del umbral establecido.

Atendiendo al valor de MBF, sin embargo, el mejor valor lo ofrece EG, siendo algo más bajo que el ofrecido por EMG. Finalmente, atendiendo a la velocidad de convergencia, el algoritmo EMG es más rápido que el basado en EG, dado que el valor medio de generaciones es menor en el primero que en el segundo.

La tabla B.5 muestra las mejores funciones obtenidas con los algoritmos EG y EMG. Finalmente, la figura B.1 muestra las gráficas de la función objetivo y las mejores funciones obtenidas, con ambos algoritmos.

Tabla B.3. – Algoritmo EG: gramática para el problema de regresión simbólica.

S =	START
T =	{ "K1", "K2", "K3", " + ", "+", "-", " * ", ".", "e", ", ", "(", ")" }
N =	{ KNL1, KNL2, KNL3, WEIGHT, FLOATPAR, POSFLOATPAR, INTPAR, SIGN, DIGIT, NONZERODIGIT, EXPONENT, END }
R =	comprende las siguientes reglas de producción
START =	KERNELS;
KERNELS =	KNL1 | KNL1, " + ", KERNELS | KNL2 | KNL2, " + ", KERNELS | KNL3, " + " | KNL3, KERNELS;
KNL1 =	WEIGHT, " * ", "K1(", FLOATPAR, ", ", FLOATPAR, "),";
KNL2 =	WEIGHT, " * ", "K2(", FLOATPAR, ", ", POSFLOATPAR, ", ", DEGREE, ")";
KNL3 =	WEIGHT, " * ", "K3(", FLOATPAR, ", ", FLOATPAR, "),";
FLOATPAR=	SIGN, NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
POSFLOATPAR =	NONZERODIGIT, ".", DIGIT, "e", EXPONENT;
DEGREE =	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
WEIGHT =	"(", SIGN, NONZERODIGIT, ".", DIGIT, "e", EXPONENT, ")";
SIGN=	"+" | "-";
DIGIT =	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
NONZERODIGIT =	"1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
EXPONENT =	"-9" | "-8" | "-7" | "-6" | "-5" | "-4" | "-3" | "-2" | "-1" | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";

Tabla B.4. – EG vs. EMG: comparación de resultados para el problema de regresión simbólica con 3 000 generaciones.

Algoritmo	SR(%)	p-valor (SR)	minBF	MBF±SD	Nº gen.±SD
EMG	**21.0**	0.0004	**0.0075**	0.0325±0.0367	**2 629.9±796.4**
EG	12.3		0.0054	**0.0298±0.0193**	2 824.3 ± 521.7

Tabla B.5. – EG vs. EMG: comparación de las mejores funciones obtenidas en el problema de regresión simbólica.

Algoritmo	Función
EMG	(-4.6e-01) * K3(-9.8e+01, +2.2e+00) + (+3.7e-01) * K3(-4.8e+01, +6.2e+00) + (-4.9e-01) * K3(-4.4e+01, +7.9e+00) + (+2.8e-03) * K3(-1.7e-08, +6.6e+06) + (+4.2e-01) * K3(-4.6e+01, +6.0e+00) + (-6.4e-01) * K3(-8.8e+01, +7.9e+00) + (-6.4e+04) * K2(-1.7e-04, +3.9e-06, +1.0e+00)
EG	(+1.6e+02) * K3(+6.2e-02, +8.7e-06) + (+3.7e-01) * K1(+2.1e+03, +1.1e-01) + (+8.4e-04) * K2(+9.9e+00, +8.3e-02, +9.0e+00) + (+7.6e-01) * K1(+2.2e+03, +1.1e-01) + (+6.5e-07) * K1(+5.1e-05, -7.7e-05) + (-9.4e-01) * K1(+3.3e+03, +9.9e-03)

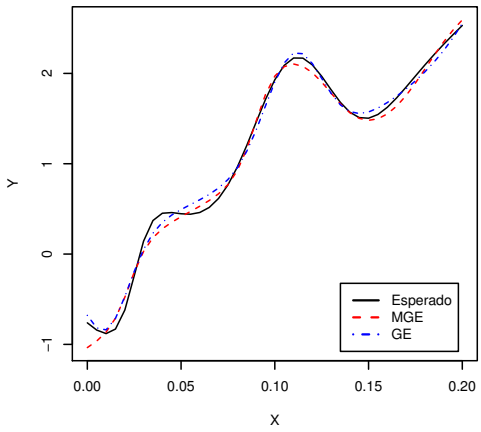


Figura B.1. – EG vs. EMG: función objetivo y mejores funciones obtenidas en el problema de regresión simbólica.

C. *Netlists* de los mejores circuitos

En este apéndice se recogen las *netlists* correspondientes a los mejores circuitos obtenidos con los diferentes algoritmos presentados en esta tesis. Las *netlists* se presentan tal cual se han obtenido en el algoritmo correspondiente, tras aplicar la fase de posprocesado.

C.1. *Netlists* de los circuitos obtenidos con ACID-GE

A continuación, se muestran las *netlists* correspondientes a los mejores circuitos obtenidos con el algoritmo ACID-GE. Es necesario recordar que, en los esquemas de los circuitos que corresponden a estas *netlists*, mostrados en la sección 5.6.1, se sustituyeron diferentes agrupaciones de resistencias interconectadas por sus resistencias equivalentes. Dicha simplificación no aparece recogida en estas *netlists,* que se muestran tal cual se han obtenido del algoritmo.

Circuito sensor de temperatura

Mejor circuito sensor de temperatura obtenido con ACID-GE, con un valor de $MNN = 6$ y 10 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.10.

```
Sensor de temperatura
V1 1 0 dc 15.0
V2 2 0 dc -15.0
R1 3 0 1K
R2 2 7 89.0
Q1 6 6 4 Q2N3904
```

```
Q2 1 5 3 Q2N3906
Q3 1 5 3 Q2N3906
R3 7 6 34.0
R4 4 2 100.0Meg
Q4 6 5 1 Q2N3906
Q5 6 7 5 Q2N3906
Q6 5 9 0 Q2N3906
R5 3 2 1.8K
Q7 2 7 1 Q2N3906
R6 2 7 8.1
R7 9 3 7.6K
Q8 1 4 9 Q2N3904
Q9 9 4 7 Q2N3904
Q10 9 4 1 Q2N3904
Q11 9 4 7 Q2N3904
Q12 9 4 7 Q2N3904
*C 9 9 3.4e-9
Q13 9 4 7 Q2N3904
Q14 9 4 7 Q2N3904
Q15 5 3 6 Q2N3904
Q16 9 4 7 Q2N3904
Q17 9 4 7 Q2N3904
Q18 9 4 7 Q2N3904
R8 9 2 2.0Meg
R9 5 2 64.0K
Q19 9 4 7 Q2N3904
C1 3 8 34.0p
R10 9 2 2.2Meg
R11 9 2 2.6Meg
R12 3 2 64.0K
Q20 2 4 4 Q2N3906
Q21 9 4 5 Q2N3906
R13 9 0 7.7Meg
Q22 2 9 5 Q2N3906
R14 8 0 1G
```

```
* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc temp 0 100 5
.print dc, V(3)
.end
```

Circuito de referencia de voltaje

Mejor circuito de referencia de voltaje obtenido con ACID-GE, con un valor de $MNN = 6$ y 3 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.11.

```
Circuito de Referencia de Voltaje
V1 1 0 dc 5.0
R1 1 2 1K
R2 3 0 10K
R3 6 0 52.0K
R4 6 0 85.0K
Q1 6 4 6 Q2N3906
R5 4 0 110.0K
Q2 3 8 2 Q2N3906
Q3 9 8 8 Q2N3906
Q4 0 0 8 Q2N3904
R6 0 8 87.0Meg
Q5 7 6 0 Q2N3904
Q6 4 4 3 Q2N3904
Q7 9 7 8 Q2N3904
Q8 7 3 9 Q2N3904
R7 5 7 23.0K
R8 6 9 10.0Meg
R9 5 7 20.0K
Q9 3 8 8 Q2N3906
R10 0 7 31.0Meg
Q10 7 5 7 Q2N3904
Q11 4 2 3 Q2N3906
Q12 7 5 7 Q2N3904
Q13 7 5 7 Q2N3904
Q14 4 2 0 Q2N3906
Q15 7 5 7 Q2N3904
Q16 6 2 7 Q2N3906
R11 0 6 4.0K
Q17 7 2 2 Q2N3904
R12 6 9 4.2Meg
R13 4 3 4.0K
R14 2 4 270.0
Q18 9 6 8 Q2N3904
Q19 9 6 9 Q2N3904
```

```
R15 9 0 6.7Meg
R16 3 5 65.0
R17 4 0 130.0K
Q20 6 5 2 Q2N3906
* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.temp 27
.dc V1 4 6 0.1 temp 0 100 25
.print dc, V(3)
.end
```

Circuito generador de función gaussiana

Mejor circuito de referencia de voltaje obtenido con ACID-GE, con un valor de $MNN = 6$ y 10 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.12.

```
Generador de funcion gaussiana
V1 1 0 dc 5.0
R1 1 2 1
V2 3 0 dc 2.5
V3 4 0 dc 5.0
M1 4 8 10 0 NMOS1 L=10u W=175u
M2 4 6 10 0 NMOS1 L=10u W=199u
M3 9 9 6 0 NMOS1 L=10u W=74u
M4 4 6 10 0 NMOS1 L=10u W=199u
R2 7 9 30.0Meg
R3 10 8 970.0K
R4 8 4 300.0Meg
M5 8 6 3 4 PMOS1 L=10u W=142u
M6 7 10 9 0 NMOS1 L=10u W=11u
M7 0 3 7 4 PMOS1 L=10u W=10u
M8 4 10 6 0 NMOS1 L=10u W=91u
R5 2 7 1.4Meg
R6 4 2 900.0Meg
M9 4 6 10 0 NMOS1 L=10u W=199u
M10 2 2 10 0 NMOS1 L=10u W=199u
M11 4 6 10 0 NMOS1 L=10u W=199u
M12 2 5 0 0 NMOS1 L=10u W=99u
M13 2 9 5 0 NMOS1 L=10u W=35u
M14 10 8 4 0 NMOS1 L=10u W=179u
R7 8 9 34.0Meg
M15 10 8 10 4 PMOS1 L=10u W=46u
R8 10 4 520.0Meg
R9 9 4 160.0Meg
R10 10 9 20.0Meg
M16 9 9 6 0 NMOS1 L=10u W=197u
```

```
R11 4 9 57.0Meg
R12 0 2 12.0K
R13 0 2 11.0K
R14 10 9 450.0Meg
R15 8 9 260.0Meg
M17 3 2 6 4 PMOS1 L=10u W=124u

* Model
.model NMOS1 NMOS
.model PMOS1 PMOS

.op
.temp 27
.dc V1 2 3 0.01
.print dc, V2#branch
.end
```

Circuito de potencia al cuadrado

Mejor circuito de potencia al cuadrado obtenido con ACID-GE, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.13. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 1 0 dc 1.0
V2 4 0 dc 15.0
V3 5 0 dc -15.0
R1 1 2 1K
R2 3 0 1K
Q1 3 2 11 Q2N3906
R3 14 6 53.0Meg
Q2 2 6 13 Q2N3906
R4 0 10 220.0Meg
R5 6 10 8.3Meg
Q3 2 6 13 Q2N3906
```

```
R6 6 10 9.9K
Q4 3 10 15 Q2N3906
Q5 2 6 13 Q2N3906
Q6 2 6 11 Q2N3906
R7 6 11 4.6Meg
Q7 14 2 13 Q2N3906
Q8 3 2 11 Q2N3906
R8 10 11 6.4Meg
R9 7 0 520.0
R10 12 3 5.7Meg
R11 11 3 5.9Meg
Q9 14 5 3 Q2N3904
R12 9 13 67.0
R13 0 9 38.0
R14 13 5 100.0Meg
R15 9 4 40.0
R16 7 11 160.0K
R17 12 5 30.0K
R18 8 4 32.0K
Q10 11 0 9 Q2N3906
Q11 3 2 6 Q2N3904
R19 9 11 49.0
Q12 2 8 9 Q2N3904
R20 15 0 1G
* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
```

```
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

Circuito de raíz cuadrada

Mejor circuito de raíz cuadrada obtenido con ACID-GE, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.14.

```
Circuitos computacionales
V1 1 0 dc 1.0
V2 4 0 dc 15.0
V3 5 0 dc -15.0
R1 1 2 1K
R2 3 0 1K
R3 7 4 78.0K
Q1 2 12 5 Q2N3906
R4 7 4 42.0K
R5 3 15 1.2Meg
R6 2 4 36.0K
```

```
R7 11 4 39.0K
R8 15 14 9.2K
R9 2 14 11.0K
R10 2 14 11.0K
R11 2 14 50.0K
Q2 12 8 12 Q2N3904
R12 13 7 10.0Meg
R13 13 7 16.0Meg
Q3 11 13 0 Q2N3904
Q4 5 5 10 Q2N3904
R14 3 8 95.0
Q5 4 0 4 Q2N3904
Q6 11 2 9 Q2N3906
R15 15 6 3.5
Q7 0 14 10 Q2N3904
Q8 2 0 10 Q2N3904
Q9 9 13 2 Q2N3904
R16 2 14 11.0K
Q10 12 8 12 Q2N3904
R17 13 7 10.0Meg
R18 13 7 10.0Meg
Q11 11 13 0 Q2N3904
Q12 7 2 12 Q2N3904
Q13 3 15 11 Q2N3906
Q14 13 12 0 Q2N3904
Q15 11 2 11 Q2N3906
R19 15 6 5.7Meg
Q16 0 14 10 Q2N3904
Q17 0 14 10 Q2N3904
Q18 12 3 10 Q2N3904
Q19 11 10 7 Q2N3906
* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
```

```
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 0mv 500mv 0.025
.print dc, V(3)
.end
```

Circuito de potencia al cubo

Mejor circuito de potencia al cubo obtenido con ACID-GE, con un valor de $MNN = 20$ y 3 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.15.

```
Circuitos computacionales
V1 1 0 dc 1.0
V2 4 0 dc 15.0
V3 5 0 dc -15.0
```

```
R1 1 2 1K
R2 3 0 1K
R3 14 7 4.0Meg
Q1 2 6 10 Q2N3904
Q2 3 2 0 Q2N3904
R4 21 3 980.0
*Q 9 9 9 Q2N3906
Q3 13 14 10 Q2N3906
R5 11 14 770.0
R6 15 5 4.3G
Q4 16 14 10 Q2N3906
R7 24 5 95.0
R8 10 14 75.0Meg
Q5 3 2 6 Q2N3904
Q6 11 21 10 Q2N3906
R9 21 7 3.0G
R10 4 6 53.0K
Q7 11 2 23 Q2N3904
Q8 21 3 10 Q2N3906
R11 20 7 150.0K
Q9 7 18 19 Q2N3904
Q10 11 21 10 Q2N3906
R12 3 22 78.0Meg
R13 21 18 100.0K
Q11 4 15 2 Q2N3904
Q12 7 21 25 Q2N3904
R14 7 25 100.0K
Q13 3 2 0 Q2N3904
R15 20 5 780.0K
Q14 13 7 16 Q2N3906
Q15 2 7 15 Q2N3906
Q16 5 7 3 Q2N3906
R16 20 7 180.0K
R17 3 13 100.0K
Q17 11 21 10 Q2N3906
```

```
Q18 9 21 22 Q2N3906
Q19 13 15 3 Q2N3904
Q20 2 7 22 Q2N3906
Q21 5 7 3 Q2N3906
Q22 3 14 19 Q2N3906
R18 9 0 1G
R19 23 0 1G
R20 24 0 1G
* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
```

```
.print dc, V(3)
.end
```

Circuito de raíz cúbica

Mejor circuito de raíz cúbica obtenido con ACID-GE, con un valor de $MNN = 30$ y 3 000 generaciones. La *netlist* se muestra sin simplificar y corresponde a la figura 5.16.

```
Circuitos computacionales
V1 1 0 dc 1.0
V2 4 0 dc 15.0
V3 5 0 dc -15.0
R1 1 2 1K
R2 3 0 1K
R3 24 4 97.0K
Q1 3 7 14 Q2N3904
Q2 3 7 31 Q2N3904
Q3 3 26 7 Q2N3904
Q4 10 3 2 Q2N3906
R4 4 11 22.0K
R5 26 5 27.0K
Q5 11 2 25 Q2N3906
R6 11 23 500.0
Q6 24 25 5 Q2N3904
Q7 7 20 7 Q2N3906
Q8 24 27 7 Q2N3906
R7 26 0 850.0
Q9 16 29 27 Q2N3904
R8 34 19 300.0
R9 26 7 10.0K
Q10 19 7 7 Q2N3906
Q11 27 10 33 Q2N3906
Q12 4 25 5 Q2N3904
Q13 0 25 5 Q2N3904
R10 34 23 1.2K
```

```
R11 26 0 220.0
R12 3 19 980.0
R13 26 7 18.0K
Q14 3 7 11 Q2N3906
Q15 3 25 5 Q2N3904
R14 33 0 1G
R15 14 0 1G
R16 16 0 1G
R17 20 0 1G
R18 29 0 1G
R19 31 0 1G
* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1
```

```
.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

C.2. *Netlists* de los circuitos obtenidos con ACID-MGE

En esta sección se muestran las netlists de los mejores circuitos obtenidos con el algoritmo ACID-MGE. Las *netlists* se muestran sin simplificar tal cual se han obtenido del algoritmo.

Circuito sensor de temperatura

Mejor circuito sensor de temperatura obtenido con ACID-MGE, con un valor de $MNN = 6$ y 3 000 generaciones.

```
Sensor de temperatura
V1 1 0 dc 15.0
V2 2 0 dc -15.0
R1 3 0 1K
Q1 2 3 6 Q2N3906
C1 1 3 59.0u
*C 1 1 6.4e-3
R2 4 5 560.0Meg
C2 6 8 4.8u
C3 6 5 580.0p
R3 1 9 11.0Meg
Q2 2 6 8 Q2N3904
Q3 1 7 5 Q2N3904
R4 5 1 730.0Meg
R5 8 1 120.0Meg
R6 3 8 760.0K
```

```
R7 0 4 5.5Meg
C4 8 2 4.6u
Q4 1 7 5 Q2N3904
R8 1 6 960.0K
C5 1 5 670.0n
Q5 1 7 3 Q2N3904
Q6 5 9 3 Q2N3904
R9 4 9 950.0K
Q7 9 8 1 Q2N3906
R10 2 3 8.4K
R11 3 8 960.0K
C6 2 9 93.0p
Q8 1 7 5 Q2N3904
Q9 6 4 3 Q2N3906
Q10 5 4 9 Q2N3906
Q11 7 0 0 Q2N3904
R12 1 5 310.0K
R13 5 0 3.0Meg
R14 5 1 1.2Meg
R15 2 5 8.2K
R16 3 8 980.0K
R17 3 5 66.0
Q12 1 7 3 Q2N3904
Q13 7 4 3 Q2N3904
C7 1 3 140.0p
Q14 3 4 3 Q2N3906
Q15 7 9 6 Q2N3906
C8 2 6 77.0p
Q16 1 7 5 Q2N3904
C9 8 7 6.2n
Q17 1 8 7 Q2N3906
* Fin

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
```

```
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc temp 0 100 5
.print dc, V(3)
.end
```

Circuito de referencia de voltaje

Mejor circuito de referencia de voltaje obtenido con ACID-MGE, con un valor de $MNN = 6$ y 3 000 generaciones.

```
Referencia de Voltaje
V1 99 0 dc 5.0
R1 99 1 1K
```

```
R2 2 0 10K
Q1 5 4 7 Q2N3904
R3 3 2 57.0Meg
R4 6 0 230.0K
Q2 2 5 1 Q2N3906
Q3 3 1 5 Q2N3904
R5 0 3 10.0K
Q4 0 0 8 Q2N3906
Q5 0 6 1 Q2N3906
Q6 2 3 6 Q2N3904
Q7 2 3 3 Q2N3904
Q8 2 3 1 Q2N3906
Q9 7 6 2 Q2N3904
Q10 5 1 7 Q2N3906
Q11 2 3 1 Q2N3906
R6 2 5 55.0K
Q12 3 7 1 Q2N3906
R7 3 5 300.0
Q13 2 3 3 Q2N3904
Q14 2 3 1 Q2N3906
R8 7 1 36.0K
R9 3 5 800.0
R10 3 6 25.0Meg
Q15 6 3 1 Q2N3906
Q16 6 3 1 Q2N3906
Q17 2 5 1 Q2N3906
Q18 2 3 7 Q2N3904
Q19 2 5 1 Q2N3906
Q20 7 6 3 Q2N3904
Q21 2 3 1 Q2N3904
R11 4 3 4.6Meg
Q22 3 2 1 Q2N3906
Q23 2 3 1 Q2N3906
R12 8 6 140.0K
Q24 8 8 4 Q2N3906
```

```
Q25 1 6 5 Q2N3904
Q26 2 3 1 Q2N3906
R13 4 2 840.0
R14 8 5 21.0Meg
*R 1 1 9.3e5
R15 5 2 100.0
Q27 2 3 1 Q2N3906
R16 0 6 410.0K
Q28 0 5 6 Q2N3906
* Fin


* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
```

```
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.temp 27
.dc V1 4 6 0.1 temp 0 100 25
.print dc, V(3)
.end
```

Circuito generador de función gaussiana

Mejor circuito generador de función gaussiana obtenido con ACID-MGE, con un valor de $MNN = 6$ y 3 000 generaciones.

```
Generador de funcion gaussiana
V1 99 0 dc 5.0
R1 99 1 1
V2 3 0 dc 2.5
V3 2 0 dc 5.0
M1 7 3 1 2 PMOS1 L=10u W=25u
M2 4 3 9 0 NMOS1 L=10u W=169u
M3 8 9 7 0 NMOS1 L=10u W=38u
R2 0 8 7.0K
R3 7 2 17.0Meg
R4 4 2 270.0Meg
M4 6 7 2 0 NMOS1 L=10u W=199u
M5 4 6 2 0 NMOS1 L=10u W=77u
R5 5 9 470.0K
M6 9 0 2 0 NMOS1 L=10u W=16u
*R 9 9 2.3e1
*R 3 3 5.2e1
M7 3 2 4 2 PMOS1 L=10u W=50u
M8 2 5 7 0 NMOS1 L=10u W=89u
M9 9 5 4 2 PMOS1 L=10u W=16u
R6 7 4 380.0Meg
R7 4 2 4.2G
R8 2 1 360.0
```

```
M10 2 1 6 0 NMOS1 L=10u W=189u
M11 5 2 3 2 PMOS1 L=10u W=76u
R9 2 8 6.5G
R10 8 5 66.0Meg
M12 2 7 6 0 NMOS1 L=10u W=189u
M13 2 5 7 0 NMOS1 L=10u W=185u
M14 8 6 5 2 PMOS1 L=10u W=172u
M15 8 2 6 0 NMOS1 L=10u W=122u
M16 5 9 8 0 NMOS1 L=10u W=160u
R11 2 4 4.0G
M17 9 5 3 2 PMOS1 L=10u W=89u
*R 2 2 4.2e7
R12 8 2 4.3
M18 1 3 6 0 NMOS1 L=10u W=139u
R13 0 6 100.0Meg
R14 6 4 1.9Meg
R15 8 2 370.0
R16 8 7 7.9Meg
R17 7 5 220.0K
M19 3 5 4 2 PMOS1 L=10u W=53u
R18 8 1 3.0
R19 8 0 5.3
M20 6 1 1 0 NMOS1 L=10u W=68u
*R 9 9 8.7e3
*R 0 0 6.4e3
* Fin

* Model
.model NMOS1 NMOS
.model PMOS1 PMOS

.op
.temp 27
.dc V1 2 3 0.01
.print dc, V2#branch
```

```
.end
```

Circuito de potencia al cuadrado

Mejor circuito de potencia al cuadrado obtenido con ACID-MGE, con un valor de $MNN = 10$ y 3 000 generaciones. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
R3 12 8 430.0
R4 3 9 2.0
R5 14 0 4.6Meg
Q1 9 4 10 Q2N3904
R6 12 13 140.0Meg
Q2 7 4 14 Q2N3904
Q3 9 14 6 Q2N3904
Q4 7 2 1 Q2N3906
Q5 4 10 5 Q2N3906
Q6 1 5 5 Q2N3904
R7 7 3 7.9K
Q7 13 1 3 Q2N3904
Q8 2 11 10 Q2N3906
Q9 12 5 8 Q2N3904
Q10 5 5 0 Q2N3906
Q11 1 5 3 Q2N3904
R8 1 5 1.4Meg
Q12 7 1 12 Q2N3906
Q13 11 0 3 Q2N3904
Q14 13 5 7 Q2N3906
R9 13 2 3.7Meg
```

```
R10 14 7 180.0K
Q15 5 8 5 Q2N3906
Q16 12 5 12 Q2N3904
R11 0 1 750.0Meg
R12 0 3 640.0Meg
R13 12 10 1.7G
Q17 1 5 6 Q2N3904
Q18 12 5 12 Q2N3904
R14 7 5 3.8Meg
Q19 12 5 8 Q2N3904
R15 4 7 1.5Meg
Q20 10 0 0 Q2N3906
R16 13 3 22.0Meg
R17 7 13 940.0Meg
Q21 6 8 0 Q2N3904
R18 0 11 5.0
Q22 9 6 10 Q2N3906
R19 13 2 5.8Meg
R20 5 2 5.7Meg
R21 7 4 13.0Meg
*R 5 5 5.2e8
Q23 10 13 14 Q2N3906
* Fin

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
```

```
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

Circuito de raíz cuadrada

Mejor circuito de raíz cuadrada obtenido con ACID-MGE, con un valor de $MNN = 10$ y 3 000 generaciones. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
*R 10 10 7.9e2
R3 0 14 66.0Meg
R4 5 9 36.0
R5 8 14 85.0Meg
```

```
R6 0 7 500.0K
R7 7 0 840.0Meg
R8 2 6 7.2Meg
Q1 7 11 13 Q2N3906
R9 14 11 87.0
Q2 8 1 8 Q2N3906
R10 7 12 770.0Meg
R11 13 4 14.0K
R12 9 10 75.0
R13 6 0 30.0K
R14 14 13 1.9K
R15 7 14 670.0K
R16 11 10 7.0G
R17 7 1 72.0
R18 4 3 220.0K
R19 7 8 3.8
Q3 6 7 12 Q2N3904
R20 5 1 6.9K
R21 2 12 370.0Meg
Q4 8 12 11 Q2N3906
R22 7 4 4.0G
Q5 4 8 14 Q2N3906
R23 6 0 21.0K
R24 7 4 1.0Meg
R25 3 9 700.0
Q6 10 3 0 Q2N3906
R26 14 11 14.0
*R 2 2 5.4e6
Q7 0 11 12 Q2N3906
R27 5 9 480.0Meg
R28 7 13 12.0Meg
R29 6 2 980.0Meg
Q8 13 10 10 Q2N3904
R30 1 5 5.5K
R31 8 4 750.0K
```

```
R32 10 1 45.0K
R33 4 1 75.0K
Q9 7 2 0 Q2N3906
R34 2 11 550.0
R35 12 0 220.0K
Q10 1 14 12 Q2N3906
R36 2 10 1.2Meg
* Fin

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
```

```
.dc V1 0mv 500mv 0.025
.print dc, V(3)
.end
```

Circuito de potencia al cubo

Mejor circuito de potencia al cubo obtenido con ACID-MGE, con un valor de $MNN = 20$ y 3 000 generaciones. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
R3 17 12 2.1K
R4 21 13 430.0
Q1 3 11 17 Q2N3906
R5 15 16 980.0
R6 17 3 49.0K
Q2 1 16 4 Q2N3906
Q3 21 0 11 Q2N3904
Q4 3 21 16 Q2N3904
R7 6 20 560.0Meg
R8 1 3 140.0K
R9 4 16 1.0Meg
Q5 21 13 17 Q2N3906
R10 1 16 290.0K
R11 4 20 380.0K
R12 2 3 36.0Meg
R13 2 3 10.0Meg
R14 11 4 28.0Meg
Q6 21 0 15 Q2N3904
R15 1 3 290.0K
```

```
*R 22 22 1.1e8
Q7 14 14 10 Q2N3906
Q8 3 17 18 Q2N3904
R16 19 13 1.3G
Q9 3 8 2 Q2N3906
Q10 21 5 3 Q2N3906
R17 15 2 680.0Meg
Q11 18 1 3 Q2N3904
R18 0 10 23.0K
R19 12 14 780.0K
Q12 0 1 5 Q2N3904
R20 21 18 98.0
R21 4 11 54.0Meg
R22 4 12 2.1Meg
Q13 22 9 9 Q2N3906
Q14 20 15 6 Q2N3904
Q15 18 21 12 Q2N3906
Q16 3 14 19 Q2N3904
Q17 5 1 6 Q2N3904
Q18 18 13 18 Q2N3906
R23 17 1 720.0
R24 4 11 260.0Meg
Q19 12 2 20 Q2N3906
Q20 17 21 6 Q2N3906
R25 5 22 520.0Meg
Q21 17 23 7 Q2N3906
* Fin
R26 7 0 1G
R27 8 0 1G
R28 23 0 1G

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
```

```
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

Circuito de raíz cúbica

Mejor circuito de raíz cúbica obtenido con ACID-MGE, con un valor de $MNN = 30$ y 3 000 generaciones. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
```

```
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
Q1 24 31 6 Q2N3906
R3 25 14 6.2G
R4 9 25 280.0Meg
R5 29 32 70.0Meg
R6 5 23 320.0K
R7 2 18 6.0G
Q2 6 20 18 Q2N3904
Q3 4 3 14 Q2N3906
Q4 16 0 5 Q2N3906
Q5 15 22 16 Q2N3906
R8 13 1 9.9
R9 28 26 990.0K
Q6 29 8 3 Q2N3904
R10 13 23 23.0Meg
Q7 6 19 1 Q2N3904
Q8 15 7 26 Q2N3904
R11 0 14 200.0
Q9 23 6 28 Q2N3906
R12 9 24 30.0
R13 12 19 4.7K
R14 8 25 690.0
Q10 13 27 34 Q2N3904
R15 2 9 7.5K
R16 2 15 300.0K
R17 2 15 1.0Meg
Q11 32 14 4 Q2N3906
Q12 32 13 27 Q2N3906
Q13 28 32 13 Q2N3904
Q14 6 17 6 Q2N3904
R18 19 20 1.9Meg
R19 4 3 20.0K
R20 26 28 690.0K
```

```
Q15 19 8 16 Q2N3906
R21 7 2 150.0Meg
Q16 17 31 18 Q2N3906
R22 18 1 1.6
Q17 12 27 25 Q2N3906
Q18 30 13 21 Q2N3906
Q19 9 8 33 Q2N3906
Q20 16 0 33 Q2N3906
Q21 15 26 7 Q2N3906
Q22 27 13 21 Q2N3906
Q23 27 12 29 Q2N3906
R23 25 3 4.1G
Q24 10 17 28 Q2N3906
Q25 14 6 24 Q2N3906
Q26 0 30 22 Q2N3906
Q27 23 25 17 Q2N3906
Q28 10 19 33 Q2N3906
R24 17 11 880.0K
R25 20 28 88.0Meg
R26 27 8 17.0K
Q29 29 32 12 Q2N3906
Q30 9 22 28 Q2N3906
R27 5 12 82.0K
* Fin
R28 11 0 1G
R29 34 0 1G

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
```

```
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

C.3. *Netlists* de los circuitos obtenidos con ACID-MGE con parsimonia simple

En esta sección se muestran las *netlists* de los mejores circuitos obtenidos con el algoritmo ACID-MGE con parsimonia simple. Las *netlists* se muestran sin simplificar tal cual se han obtenido del algoritmo. Los esquemas de circuitos correspondientes a estas *netlists*, fueron mostrados en la sección 5.6.3.

Circuito sensor de temperatura

Mejor circuito sensor de temperatura obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* corresponde al esquema

de circuito mostrado en la figura 5.17.

```
Sensor de temperatura
V1 1 0 dc 15.0
V2 2 0 dc -15.0
R1 3 0 1K
R2 8 0 1.1K
Q1 2 10 4 Q2N3904
Q2 4 8 5 Q2N3904
Q3 1 5 10 Q2N3906
R3 3 10 9.6K
R4 3 2 7.4K
R5 3 2 7.6K
Q4 1 8 3 Q2N3906
* Fin

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
```

```
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc temp 0 100 5
.print dc, V(3)
.end
```

Circuito de referencia de voltaje

Mejor circuito de referencia de voltaje obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* corresponde al esquema de circuito mostrado en la figura 5.18.

```
Circuito de Referencia de Voltaje
V1 99 0 dc 5.0
R1 99 1 1K
R2 2 0 10K
R3 12 4 320.0
Q1 3 12 5 Q2N3904
Q2 3 12 3 Q2N3904
R4 4 10 8.8K
Q3 2 9 12 Q2N3906
Q4 0 9 10 Q2N3906
R5 3 2 6.1K
Q5 0 3 4 Q2N3906
Q6 4 10 1 Q2N3906
Q7 0 2 12 Q2N3906
R6 0 9 4.1Meg
R7 1 2 9.6K
* Fin
R8 5 0 1G

* Model Generated by MODPEX *
```

```
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.temp 27
.dc V1 4 6 0.1 temp 0 100 25
.print dc, V(3)
.end
```

Circuito generador de función gaussiana

Mejor circuito de referencia de voltaje obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* corresponde al esquema de circuito mostrado en la figura5.19

```
Generador de funcion gaussiana
V1 99 0 dc 5.0
R1 99 1 1
V2 3 0 dc 2.5
V3 2 0 dc 5.0
M1 1 5 4 0 NMOS1 L=10u W=121u
M2 10 10 5 2 PMOS1 L=10u W=31u
R2 7 1 8.2G
R3 7 3 290.0Meg
R4 4 2 300.0K
R5 10 3 8.0Meg
M3 2 10 3 0 NMOS1 L=10u W=108u
R6 4 10 750.0Meg
M4 3 7 10 0 NMOS1 L=10u W=56u
* Fin

* Model
.model NMOS1 NMOS
.model PMOS1 PMOS

.op
.temp 27
.dc V1 2 3 0.01
.print dc, V2#branch
.end
```

Circuito de potencia al cuadrado

Mejor circuito de potencia al cuadrado obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* corresponde al esquema de circuito mostrado en la figura 5.20. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
```

```
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
Q1 3 7 13 Q2N3904
Q2 3 5 1 Q2N3906
R3 4 5 9.8Meg
R4 1 13 410.0K
Q3 9 7 13 Q2N3904
Q4 3 9 2 Q2N3906
R5 2 7 290.0Meg
* Fin

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
```

```
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

Circuito de raíz cuadrada

Mejor circuito de raíz cuadrada obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 10$ y 3 000 generaciones. La *netlist* corresponde al esquema de circuito mostrado en la figura 5.21. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
R3 1 2 14.0K
R4 12 1 7.3K
R5 4 3 6.9K
R6 12 3 720.0
R7 4 10 660.0Meg
Q1 11 8 0 Q2N3904
Q2 2 1 3 Q2N3904
Q3 3 8 5 Q2N3906
Q4 5 10 2 Q2N3906
Q5 1 5 12 Q2N3904
* Fin
R8 11 0 1G

* Model Generated by MODPEX *
```

```
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 0mv 500mv 0.025
.print dc, V(3)
.end
```

Circuito de potencia al cubo

Mejor circuito de potencia al cubo obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 20$ y 3 000 generaciones. La *netlist* corresponde al esquema de circuito mostrado en la figura 5.22, donde se ha eliminado la resistencia R7, de $1G\Omega$, utilizada para conectar la alimentación negativa de $-15V$ a masa, que no tiene

función alguna en el circuito. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
R3 2 16 6.3Meg
Q1 2 0 24 Q2N3906
Q2 0 0 24 Q2N3906
Q3 9 24 18 Q2N3904
R4 19 16 190.0K
Q4 19 16 3 Q2N3904
Q5 3 5 1 Q2N3904
R5 1 19 96.0
R6 18 5 280.0K
* Fin
R7 4 0 1G
R8 9 0 1G

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1
```

```
.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

Circuito de raíz cúbica

Mejor circuito de raíz cúbica obtenido con ACID-MGE y parsimonia simple, con un valor de $MNN = 15$ y 6 000 generaciones. La *netlist* corresponde al esquema de circuito mostrado en la figura 5.23. El análisis mostrado en la *netlist* no es el utilizado en el bucle del algoritmo, pues en su lugar se utilizó un análisis *transient* en el dominio del tiempo.

```
Circuitos computacionales
V1 99 0 dc 1.0
V2 2 0 dc 15.0
V3 4 0 dc -15.0
R1 99 1 1K
R2 3 0 1K
R3 18 7 20.0K
R4 18 7 45.0K
R5 2 8 1.1Meg
Q1 3 12 18 Q2N3904
Q2 12 3 16 Q2N3904
```

```
Q3 12 3 12 Q2N3904
Q4 3 8 0 Q2N3904
Q5 16 3 15 Q2N3904
Q6 3 16 18 Q2N3904
Q7 3 8 1 Q2N3904
Q8 3 8 5 Q2N3904
Q9 3 8 14 Q2N3904
Q10 3 8 13 Q2N3904
Q11 3 8 18 Q2N3904
R6 8 0 3.4K
Q12 11 1 18 Q2N3904
Q13 2 11 3 Q2N3906
Q14 4 7 11 Q2N3904
* Fin
R7 5 0 1G
R8 13 0 1G
R9 14 0 1G
R10 15 0 1G

* Model Generated by MODPEX *
.MODEL Q2N3904 npn
+IS=6.9716e-14 BF=545.416 NF=1.09328 VAF=10
+IKF=0.0228393 ISE=5.71808e-12 NE=1.88204 BR=4.70256
+NR=1.3912 VAR=2.31769 IKR=0.074093 ISC=5.71808e-12
+NC=1.36259 RB=1.733 IRB=1.12054 RBM=0.876202
+RE=0.356192 RC=1.78096 XTB=0.1 XTI=1
+EG=1.05 CJE=4.47982e-12 VJE=0.4 MJE=0.240345
+TF=4e-10 XTF=1.5 VTF=1 ITF=1
+CJC=3.76637e-12 VJC=0.4 MJC=0.241382 XCJC=0.8
+FC=0.533333 CJS=0 VJS=0.75 MJS=0.5
+TR=3.77901e-05 PTF=0 KF=0 AF=1

.MODEL Q2N3906 pnp
+IS=1.14615e-14 BF=535.453 NF=1.06473 VAF=10
+IKF=0.0234918 ISE=1.33613e-13 NE=1.62939 BR=4.66099
```

```
+NR=1.19618 VAR=2.77165 IKR=0.0740931 ISC=1.33613e-13
+NC=1.22182 RB=0.1 IRB=1.05964 RBM=0.1
+RE=0.0001 RC=1.39183 XTB=0.1 XTI=1
+EG=1.05 CJE=6.03788e-12 VJE=0.4 MJE=0.272764
+TF=4.8381e-10 XTF=1.5 VTF=1 ITF=1
+CJC=6.18444e-12 VJC=0.4 MJC=0.234098 XCJC=0.8
+FC=0.5415 CJS=0 VJS=0.75 MJS=0.5
+TR=2.42096e-06 PTF=0 KF=0 AF=1

.op
.dc V1 -250mv 250mv 0.025
.print dc, V(3)
.end
```

D. Gramáticas

Este apéndice incluye las gramáticas desarrolladas y utilizadas en los diferentes algoritmos implementados en esta tesis. Estas gramáticas pueden no coincidir completamente con las gramáticas genéricas definidas en el capítulo 3, pues contienen detalles de implementación cuya descripción no se ha considerado relevante en el ámbito de la tesis.

Con la excepción de los casos con algoritmo ACID-MGE con aprendizaje del parámetro MNN, las gramáticas mostradas consideran un valor de parámetro MNN fijo. Para modificar este valor, es necesario modificar el símbolo no terminal NODES, para que pueda expandirse por un número mayor de nodos. En cada gramática, se indica el número de nodos permitidos.

D.1. Gramáticas para EG

En esta sección se muestran las gramáticas utilizadas en EG. Las tablas D.1, D.2, D.3 y D.4 muestran las gramáticas utilizadas para los circuitos sensor de temperatura, referencia de voltaje, función gaussiana y para todos los circuitos computacionales, respectivamente.

D.2. Gramáticas para EMG

En esta sección se muestran las gramáticas utilizadas en EMG. En este caso, cada circuito se sintetiza utilizando dos gramáticas (gramática de topología y gramática de dimensionamiento). Las tablas D.5, D.6, D.7 y D.8 muestran las primeras gramáticas utilizadas para los circuitos sensor de temperatura, referencia de voltaje, función gaussiana y para todos los circuitos computacionales, respectivamente. Fi-

nalmente, la tabla D.9 muestra la gramática de dimensionamiento común a todos los circuitos.

D.3. Gramáticas para EMG con aprendizaje de MNN

En esta sección se muestran las gramáticas utilizadas en EMG, con aprendizaje del parámetro MNN. Nuevamente, cada circuito se sintetiza utilizando dos gramáticas. Las tablas D.10, D.11, D.12 y D.13 muestran las primeras gramáticas utilizadas para los circuitos sensor de temperatura, referencia de voltaje, función gaussiana y para todos los circuitos computacionales, respectivamente. Todos los circuitos utilizan la misma gramática de dimensionamiento que coincide con la ya mostrada en la tabla D.9.

```
(*
 * netlistSensorBloquesBJT4.ebnf
 *
 * Gramática para el circuito sensor de temperatura
 *
 * Bloques de tamaño 7 bytes
 *
 * MNN = 6
 *)

LIST = LINEA;

LINEA = RESISTENCIA | RESISTENCIA, LINEA | CONDENSADOR | CONDENSADOR, LINEA | BJT |
      BJT, LINEA;

RESISTENCIA =  "R", SEP, nodo, SEP, nodo, SEP, dummy, SEP, ValorResistencia, EOL;

CONDENSADOR =  "C", SEP, nodo, SEP, nodo, SEP, dummy, SEP, ValorCondensador, EOL;

BJT = "Q", SEP, nodo, SEP, nodo, SEP, nodo, SEP, TipoBJT, SEP, dummy, SEP, dummy,
    EOL;

(* 2N3904 es NPN y 2N3906 es PNP *)
TipoBJT = "Q2N3904" | "Q2N3906";

dummy = 'nulo1' | 'nulo2';

(* 6 nodos nuevos, 4 nodos de parte fija *)
nodo = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

ValorResistencia = digitonocero, DECIMAL, digito, EXP, digito;

ValorCondensador = digitonocero, DECIMAL, digito, EXP, expCondensador;

digito = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

digitonocero = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

expCondensador = "-12" | "-11" | "-10" | "-9" | "-8" | "-7" | "-6" | "-5" | "-4" |
    "-3";

SEP = ' ';

EXP = 'e';

DECIMAL = '.';

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.1. – Algoritmo ACID-GE: Gramática utilizada para el circuito sensor de temperatura

```
(*
 * netlistVrefBloquesBJT4.ebnf
 *
 * Gramática para el circuito de referencia de voltaje
 *
 * Bloques de tamaño 7 bytes, permite 6 nodos nuevos, no permite el nodo 1
 *
 * MNN = 6
 *)

LIST = LINEA;

LINEA = RESISTENCIA | RESISTENCIA, LINEA | BJT | BJT, LINEA;

RESISTENCIA =  "R", SEP, nodo, SEP, nodo, SEP, dummy, SEP, ValorResistencia, EOL;

BJT = "Q", SEP, nodo, SEP, nodo, SEP, nodo, SEP, TipoBJT, SEP, dummy, SEP, dummy,
    EOL;

(* 2N3904 es NPN y 2N3906 es PNP *)
TipoBJT = "Q2N3904" | "Q2N3906";

dummy = 'nulo1' | 'nulo2';

(* 6 nodos nuevos, parte fija 4, nodo 1 no se puede generar para evitar puentear
    resistencia de fuente *)
nodo = '0' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' ;

ValorResistencia = digitonocero, DECIMAL, digito, EXP, digito;

digito = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

digitonocero = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

SEP = ' ';

EXP = 'e';

DECIMAL = '.';

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.2. – Algoritmo ACID-GE: gramática utilizada para el circuito de referencia de voltaje

```
(*
 * netlistBloquesMOSFET4.ebnf
 *
 * Gramática para cirtuito de función gaussiana
 *
 * Bloques de 8 codones, permite 6 nodos nuevos, no permite el nodo 1
 *
 * MNN = 6
 *)

LIST = LINEA;

LINEA = RESISTENCIA | RESISTENCIA, LINEA | MOSFET | MOSFET, LINEA;

RESISTENCIA = "R", SEP, nodo, SEP, nodo, SEP, dummy, SEP, ValorResistencia, SEP,
    dummy, EOL;

MOSFET = "M", SEP, nodo, SEP, nodo, SEP, nodo, SEP, TIPOMOS, AnchoCanalMos, EOL;

TIPOMOS = MODELNMOS | MODELPMOS;

MODELNMOS = '0', SEP, "NMOS1 L=10u W=";

MODELPMOS = '4', SEP, "PMOS1 L=10u W=";

dummy = 'nulo1' | 'nulo2';

(* 6 nodos nuevos, 5 parte fija, nodo 1 no se puede generar para evitar puentear
    resistencia de fuente *)
nodo = '0' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10';

ValorResistencia = digitonocero, DECIMAL, digito, EXP, digito;

AnchoCanalMos = digitonocero, digito, 'u' | '1', digito, digito, 'u';

digito = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

digitonocero = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

SEP = ' ';

EXP = 'e';

DECIMAL = '.';
EOL = ?EOL?;
CR = ?Carriage return?;
```

Tabla D.3. – Algoritmo ACID-GE: gramática utilizada para el circuito de función gaussiana

```
(*
 * netlistComputaBloquesBJT4_n10.ebnf
 *
 * Gramática para circuitos computacionales
 *
 * Bloques de tamaño 7 codones, permite 10 nodos nuevos
 *
 * MNN = 10
 *)

LIST = LINEA;

LINEA = RESISTENCIA | RESISTENCIA, LINEA | BJT | BJT, LINEA;

RESISTENCIA =  "R", SEP, nodo, SEP, nodo, SEP, dummy, SEP, ValorResistencia, EOL;

BJT = "Q", SEP, nodo, SEP, nodo, SEP, nodo, SEP, TipoBJT, SEP, dummy, SEP, dummy,
    EOL;

(* 2N3904 es NPN y 2N3906 es PNP *)
TipoBJT = "Q2N3904" | "Q2N3906";

dummy = 'nulo1' | 'nulo2';

(* 10 nodos nuevos, 6 nodos de parte fija *)
nodo = '0' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' |
    '13' | '14' | '15';

ValorResistencia = digitonocero, DECIMAL, digito, EXP, digito;

digito = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

digitonocero = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';

SEP = ' ';

EXP = 'e';

DECIMAL = '.';

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.4. – Algoritmo ACID-GE: gramática utilizada para los circuitos computacionales

```
(*
 * MultiNetlistSensorTopology1.ebnf
 *
 * Multigramática para el circuito sensor de temperatura
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 4 nodos
 *
 * MNN = 10
 *
 * Último nodo es MNN + 3 (nodos 0 a 3 parte fija)
 * Ej: MNN=10, último nodo 13
 *)

LIST = COMPONENTS;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | CAPACITOR, FINAL | CAPACITOR,
     COMPONENTS | BJT, FINAL | BJT, COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

CAPACITOR =  "C", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "CAPACITORVAL", EOL;

BJT = "Q", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "BJTTYPE", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, 13 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.5. – Algoritmo ACID-MGE: gramática de topología para el circuito sensor de temperatura.

```
(*
 * MultiNetlistVrefTopology1.ebnf
 *
 * Multigramática para el circuito de referencia de voltaje
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 4 nodos, nodo de fuente con etiqueta fuera del rango de NODE
 *
 * MNN = 10
 *
 * Último nodo es MNN + 2 (nodos 0 a 2 parte fija y el nodo fuera de rango)
 * Ej: MNN=10, último nodo 12
 *)

LIST = COMPONENTS;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | BJT, FINAL | BJT, COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

BJT = "Q", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "BJTTYPE", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, 12 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.6. – Algoritmo ACID-MGE: gramática de topología para el circuito de referencia de voltaje.

```
(*
 * MultiNetlistMOSTopology1.ebnf
 *
 * Multigramática para el circuito de función gaussiana
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 5 nodos, nodo de fuente con etiqueta fuera del rango de NODE
 *
 * MNN = 10
 *
 * Último nodo es MNN + 3 (nodos 0 a 3 parte fija y el nodo fuera de rango)
 * Ej: MNN=10, último nodo 13
 *)

LIST = COMPONENTS;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | MOSFET, FINAL | MOSFET,
    COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

MOSFET = "M", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "MOSTYPE", "CHANNELWIDTH", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, 13 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.7. – Algoritmo ACID-MGE: gramática de topología para el circuito de función gaussiana.

```
(*
 * MultiNetlistComputaTopology1.ebnf
 *
 * Multigramática para los circuitos computacionales
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 6 nodos, nodo de fuente con etiqueta fuera del rango de NODE
 *
 * MNN = 10
 *
 * Último nodo es MNN + 4 (nodos 0 a 4 y el nodo fuera de rango)
 * Ej: MNN=10, último nodo 14
 *)

LIST = COMPONENTS;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | BJT, FINAL | BJT, COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

BJT = "Q", SEP, NODE, SEP, NODE, SEP, NODE, SEP, NODE, SEP, "BJTTYPE", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, 14 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.8. – Algoritmo ACID-MGE: gramática de topología para todos los circuitos computacionales.

```
(*
 * MultiNetlistComunTop2.ebnf
 *
 * 2a multigramática común para todas las gramáticas
 *)

LIST = <GEResult>;

RESISTORVAL = NONZERODIGIT, DECIMAL, DIGIT, EXP, DIGIT, SEP, DUMMY;

CAPACITORVAL = NONZERODIGIT, DECIMAL, DIGIT, EXP, EXPONENT, SEP, DUMMY;

(* 2N3904 es NPN y 2N3906 es PNP *)
BJTTYPE = TYPE, SEP, DUMMY, SEP, DUMMY, SEP, DUMMY;

TYPE = "Q2N3904" | "Q2N3906";

MOSTYPE = MODELNMOS | MODELPMOS;

MODELNMOS = '0', SEP, "NMOS1 L=10u W=";

MODELPMOS = '4', SEP, "PMOS1 L=10u W=";

CHANNELWIDTH = NONZERODIGIT, DIGIT, 'u' | '1', DIGIT, DIGIT, 'u';

DUMMY = "nulo1" | "nulo2";

DIGIT = <GECodonValue: 0, 9>;

NONZERODIGIT = <GECodonValue: 1, 9>;

EXPONENT = <GECodonValue: -12, -3>;

EXP = "e";

DECIMAL = ".";

SEP = " ";

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.9. – Algoritmo ACID-MGE: gramática de dimensionamiento común para todos los circuitos abordados.

```
(*
 * MultiNetlistSensorTopMNN1.ebnf
 *
 * Multigramática para el circuito sensor de temperatura
 *
 * Permite aprender el MNN
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 4 nodos
 *
 * Último nodo es MNN + 3 (nodos 0 a 3 parte fija)
 * Ej: MNN=10, último nodo 13
 *)

LIST = HEADER, COMPONENTS;

%MNN = <GECodonValue: 6, 30>;

HEADER = "* MNN:", %MNN, SEP, DUMMY, SEP, DUMMY, SEP, DUMMY, EOL, <GEXOMarker>;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | CAPACITOR, FINAL | CAPACITOR,
      COMPONENTS | BJT, FINAL | BJT, COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

CAPACITOR =  "C", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "CAPACITORVAL", EOL;

BJT = "Q", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "BJTTYPE", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, %MNN + 3 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.10. – Algoritmo ACID-MGE: gramática de topología, con aprendizaje de MNN, para el circuito sensor de temperatura.

```
(*
 * MultiNetlistVrefTopMNN1.ebnf
 *
 * Multigramática para el circuito de referencia de voltaje
 *
 * Permite aprender el MNN
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 4 nodos, nodo de fuente con etiqueta fuera del rango de NODE
 *
 * Último nodo es MNN + 2 (nodos 0 a 2 parte fija y el nodo fuera de rango)
 * Ej: MNN=10, último nodo 12
 *)

LIST = HEADER, COMPONENTS;

%MNN = <GECodonValue: 6, 30>;

HEADER = "* MNN:", %MNN, SEP, DUMMY, SEP, DUMMY, SEP, DUMMY, EOL, <GEXOMarker>;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | BJT, FINAL | BJT, COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

BJT = "Q", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "BJTTYPE", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, %MNN + 2 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.11. – Algoritmo ACID-MGE: gramática de topología, con aprendizaje de MNN, para el circuito de referencia de voltaje.

```
(*
 * MultiNetlistMOSTopMNN1.ebnf
 *
 * Multigramática para el circuito de función gaussiana
 *
 * Permite aprender el MNN
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 5 nodos, nodo de fuente con etiqueta fuera del rango de NODE
 *
 * Último nodo es MNN + 3 (nodos 0 a 3 parte fija y el nodo fuera de rango)
 * Ej: MNN=10, último nodo 13
 *)

LIST = HEADER, COMPONENTS;

%MNN = <GECodonValue: 6, 30>;

HEADER = "* MNN:", %MNN, SEP, DUMMY, SEP, DUMMY, SEP, DUMMY, EOL, <GEXOMarker>;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | MOSFET, FINAL | MOSFET,
    COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

MOSFET = "M", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "MOSTYPE", "CHANNELWIDTH", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, %MNN + 3 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.12. – Algoritmo ACID-MGE: gramática de topología, con aprendizaje de MNN, para el circuito de función gaussiana.

```
(*
 * MultiNetlistComputaTopMNN1.ebnf
 *
 * Multigramática para los circuitos computacionales
 *
 * Permite aprender el MNN
 *
 * Utiliza bloques de 4 codones
 *
 * Parte fija de 6 nodos, nodo de fuente con etiqueta fuera del rango de NODE
 *
 * Último nodo es MNN + 4 (nodos 0 a 4 y el nodo fuera de rango)
 * Ej: MNN=10, último nodo 14
 *)

LIST = HEADER, COMPONENTS;

%MNN = <GECodonValue: 6, 30>;

HEADER = "* MNN:", %MNN, SEP, DUMMY, SEP, DUMMY, SEP, DUMMY, EOL, <GEXOMarker>;

COMPONENTS = RESISTOR, FINAL | RESISTOR, COMPONENTS | BJT, FINAL | BJT, COMPONENTS;

RESISTOR =  "R", SEP, NODE, SEP, NODE, SEP, DUMMY, SEP, "RESISTORVAL", EOL;

BJT = "Q", SEP, NODE, SEP, NODE, SEP, NODE, SEP, "BJTTYPE", EOL;

DUMMY = "nulo1" | "nulo2";

NODE = <GECodonValue: 0, %MNN + 4 >;

SEP = " ";

FINAL = "* Fin", EOL, <GEXOMarker>;

EOL = ?EOL?;

CR = ?Carriage return?;
```

Tabla D.13. – Algoritmo ACID-MGE: gramática de topología, con aprendizaje de MNN, para todos los circuitos computacionales.

E. Análisis de potencia estadística

En este apéndice se muestra un análisis de la potencia estadística de los tests de hipótesis utilizados para comparar los experimentos realizados en esta tesis. Para poder realizar dicho análisis, resulta conveniente hacer uso del concepto de *tamaño del efecto* (Cohen, 1992), ya que permite formalizar las diferencias que puede detectar un test determinado. Sin embargo, dado que en la literatura relacionada con la inteligencia artificial y la informática no es habitual el uso de dicho concepto, a lo largo de esta tesis se ha utilizado el concepto más habitual como es el del p-valor.

Los tests de hipótesis realizados buscan comparar los valores obtenidos de tasa de éxitos (SR, por sus siglas en inglés) en dos experimentos dados, con el objetivo de verificar si la diferencia es estadísticamente significativa. El tipo de test de hipótesis que permite hacer esto es el denominado *test de comparación de proporciones*, donde SR es la proporción de éxitos, obtenida como $SR = n/N$, donde n es el número de éxitos sobre un tamaño muestral N. El tamaño muestral, N, se corresponde con el número de ejecuciones realizadas en un experimento y que consideraremos igual en los dos experimentos a comparar.

Consideraremos que cada ejecución del algoritmo se puede modelar como una prueba de Bernoulli, donde la probabilidad de éxito es p y la de fracaso es su complementaria $q = 1 - p$. De esta manera, un experimento se puede modelar como una prueba de Bernoulli repetida N veces, por lo que la probabilidad de la variable aleatoria número de éxitos, n, se distribuye según una distribución binomial con parámetros N y p. Teniendo en cuenta que el valor $SR = n/N$ es un estimador de p.

En un test de hipótesis se trabaja con una hipótesis nula que asume que no hay diferencia entre los dos experimentos comparados. La hipótesis alternativa asume lo contrario, es decir, que sí existe una diferencia entre ambos experimentos. Se define el valor de significación estadística α como la probabilidad de cometer un error de tipo I, o falso positivo, que se corresponde con la detección de una diferencia cuando realmente no existe. También se define el error de tipo II, o falso negativo,

con probabilidad β, que se corresponde con la no detección de una diferencia cuando realmente existe. Normalmente, se suele trabajar con el valor de potencia estadística, cuyo valor es $1 - \beta$, que representa la probabilidad de rechazar la hipótesis nula cuando realmente existe una diferencia.

El test de proporciones utilizado en esta tesis es un test z de dos proporciones de una cola, por lo que supondremos siempre que p_2 es mayor que p_1, con un valor de significación α, que podrá tomar los valores habituales, tales como 0.05 o 0.1 y, finalmente, una potencia estadística $1 - \beta$, cuyo valor más habitual es 0.8.

El tamaño del efecto en un test de comparación de proporciones se mide mediante el índice h de Cohen (Cohen, 1992), que se obtiene mediante la ecuación E.1, donde los valores ϕ_1 y ϕ_2 se obtienen a partir de los valores de $p1$ y $p2$, mediante la transformación de arco de seno, según la ecuación E.2.

$$h = \phi_2 - \phi_1 \tag{E.1}$$

$$\phi = 2\, arcsin(\sqrt{p}) \tag{E.2}$$

El índice h de Cohen puede tomar valores entre 0 y 1. Dicho parámetro se suele considerar cualitativamente como grande, pequeño y mediano, según la escala cualitativa mostrada en la tabla E.1.

Tabla E.1. – Escala cualitativa del tamaño del efecto en un test de hipótesis.

Tamaño del efecto	h de Cohen
Pequeño	0.2
Mediano	0.5
Grande	0.8

La librería «pwr» para uso con lenguaje R permite calcular el valor de la potencia estadística a partir los valores de nivel de significación, α, y del tamaño muestral, N, para un test de proporciones de una cola. Mediante esta librería se han obtenido las tablas E.2 y E.3 que muestran el valor de la potencia estadística, $1 - \beta$, para los valores de significación estadística $\alpha = 0.05$ y $\alpha = 0.1$, respectivamente. Ambas tablas muestran la potencia estadística para diferentes valores de h de Cohen y

tamaño muestral, N.

La librería «pwr» también permite obtener el tamaño muestral, N, a partir del nivel de significación estadística y la potencia estadística, $1-\beta$, para un test de proporciones de una cola. La tabla E.4 muestra el tamaño muestral, N, para una potencia estadística $1-\beta = 0.8$, para los dos valores indicados del nivel de significación, α, y varios valores de h de Cohen.

Tabla E.2. – Tabla de potencia estadística, $1-\beta$, para un test de proporciones de una cola, con $\alpha = 0.05$, para los valores de h de Cohen y N.

		N							
		12	25	50	75	100	150	200	400
h	0.1	0.080771	0.098300	0.126135	0.150923	0.174187	0.218041	0.259511	0.408797
	0.2	0.124054	0.174187	0.259511	0.337203	0.408797	0.534743	0.638760	0.881709
	0.3	0.181409	0.279545	0.442413	0.576232	0.683129	0.829761	0.912315	0.995309
	0.4	0.253007	0.408797	0.638760	0.789485	0.881709	0.965563	0.990742	0.999970
	0.5	0.337203	0.548912	0.803765	0.921760	0.970667	0.996377	0.999603	1.000000
	0.6	0.430477	0.683129	0.912315	0.978790	0.995309	0.999808	0.999993	1.000000
	0.7	0.527819	0.796736	0.968212	0.995876	0.999525	0.999995	1.000000	1.000000
	0.8	0.623520	0.881709	0.990742	0.999431	0.999970	1.000000	1.000000	1.000000
	0.9	0.712154	0.937869	0.997849	0.999945	0.999999	1.000000	1.000000	1.000000

Tabla E.3. – Tabla de potencias estadísticas, $1-\beta$, para un test de proporciones de una cola, con $\alpha = 0.1$, para los valores de h de Cohen y N.

		N							
		12	25	50	75	100	150	200	400
h	0.1	0.149961	0.176704	0.217239	0.251691	0.282833	0.338878	0.389144	0.552770
	0.2	0.214281	0.282833	0.389144	0.477350	0.552770	0.673825	0.763760	0.939053
	0.3	0.292291	0.412589	0.586460	0.710746	0.799481	0.906001	0.957143	0.998467
	0.4	0.381419	0.552770	0.763760	0.878584	0.939053	0.985466	0.996721	0.999994
	0.5	0.477350	0.686593	0.888473	0.962487	0.987901	0.998850	0.999900	1.000000
	0.6	0.574617	0.799481	0.957143	0.991637	0.998467	0.999955	0.999999	1.000000
	0.7	0.667526	0.883628	0.986738	0.998672	0.999878	0.999999	1.000000	1.000000
	0.8	0.751127	0.939053	0.996721	0.999851	0.999994	1.000000	1.000000	1.000000
	0.9	0.821994	0.971312	0.999356	0.999988	1.000000	1.000000	1.000000	1.000000

A la vista de las tablas E.2, E.3 y E.4, se puede comprobar que dado un nivel de significación, α, un mayor tamaño muestral permitirá detectar un tamaño del efecto menor, lo que resulta adecuado para poder distinguir entre experimentos cuyos SR sean próximos. Sin embargo, el tamaño muestral se encuentra limitado por el tiempo total de ejecución de los experimentos, que viene determinado por los

Tabla E.4. – Tabla de tamaños muestrales para un test de proporciones de una cola, con una potencia estadística $1 - \beta = 0.8$, para los valores de h de Cohen y α.

		α	
		0.05	0.1
h	0.1	1237	902
	0.2	310	226
	0.3	138	101
	0.4	78	57
	0.5	50	37
	0.6	35	26
	0.7	26	19
	0.8	20	15
	0.9	16	12

tiempos de simulación de los circuitos candidatos, por lo que será necesario buscar un compromiso entre el tamaño muestral y el tamaño del efecto que se podrá distinguir entre experimentos.

En esta tesis se ha utilizado un tamaño muestral, N, de 50, que para un nivel de significación $\alpha = 0.05$ y una potencia estadística $1 - \beta = 0.8$, permite distinguir un tamaño del efecto con un índice h de cohen mínimo de 0.5, lo que corresponde a un tamaño del efecto medio.

Con el mismo tamaño muestral $N = 50$, considerando un nivel de significación $\alpha = 0.1$ y una potencia estadística $1 - \beta = 0.8$, se puede distinguir un tamaño del efecto con h de Cohen mínimo de 0.42, algo más pequeño que el considerado tamaño del efecto medio.

Finalmente, en los experimentos realizados en el problema de regresión simbólica, mostrados en el apéndice B, al no estar limitados por el tiempo de simulación de circuitos, se ha podido utilizar un tamaño muestral más elevado, con $N = 400$. Dicho tamaño muestral, para un nivel de significación $\alpha = 0.05$ y una potencia estadística $1 - \beta = 0.8$, permite distinguir diferencias con un tamaño del efecto pequeño, concretamente hasta un valor mínimo de 0.18.

F. Resultados sobre *wrapping* y caché de evaluaciones

En este apéndice se muestran unas estadísticas adicionales sobre los resultados, en relación con los mecanismos de *wrapping* y del mecanismo de caché implementada en ACID-GE y ACID-MGE.

En la sección 3.1.2 se introdujo el concepto de *wrapping* y en este apéndice se mostrará el número medio de veces que se utiliza el mecanismo de *wrapping* en la decodificación de los cromosomas de la última generación de los algoritmos.

La evaluación de cada circuito es el proceso computacionalmente más costoso de los algoritmos propuestos. Por ello, resulta de interés tratar de reducir el número de evaluaciones sin impactar en la eficacia del mismo. En este sentido la validación previa de la viabilidad de los circuitos candidatos incluida en la etapa de posprocesado de las *netlists* (véase la sección 3.2.4), evita la simulación de circuitos etiquetados como inviables *a priori*.

Adicionalmente, la implementación del algoritmo propuesto incluye un mecanismo de caché de evaluaciones, que permite almacenar el resultado de adaptación de circuitos previamente evaluados, de forma que no sea necesaria su evaluación de nuevo, en caso de que el algoritmo mantenga el cromosoma en la población o, en caso de una nueva aparición del mismo circuito en una generación posterior.

Para medir el rendimiento de la caché, se ha introducido la medida de eficacia de caché, mostrada en la ecuación F.1, donde la eficacia se mide en porcentaje. Esta medida refleja el porcentaje de evaluaciones no realizadas, por encontrarse la evaluación de un individuo concreto ya almacenada en la caché, lo que se denomina un *hit* de caché.

$$eficacia = \frac{hits\,de\,caché}{número\,de\,evaluaciones\,totales} \tag{F.1}$$

Las tablas F.1 y F.2 muestran la eficiencia media de la caché durante la ejecución de los algoritmos propuestos y también, para la última generación de cada ejecución de los algoritmos, se muestra el número medio de *wrapping* utilizado en la decodificación de los cromosomas, el número de cromosomas viables y el número de cromosomas expresables en la población de la última generación de los algoritmos ACID-MGE, con y sin aprendizaje de MNN, y ACID-GE, para los circuitos no computacionales y computacionales, respectivamente.

Tabla F.1. – ACID-GE vs ACID-MGE: comparación de la eficiencia de la caché (EC) y *wrapping* usando 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	EC(%)±SD	*Wrapping*±SD
Sensor de temperatura	ACID-MGE	Aprendizaje	10.5 ± 4.2	17.0±2.5	0.88±0.25
	ACID-MGE	Prefijado	10	17.0±3.0	0.88±0.24
	ACID-GE	Prefijado	10	34.0±5.2	0.02±0.14
Función gaussiana	ACID-MGE	Aprendizaje	10.9 ± 5.9	18.3±2.8	0.88±0.23
	ACID-MGE	Prefijado	10	18.8±4.1	0.83±0.30
	ACID-GE	Prefijado	10	47.2±17.6	0.00±0.00
Referencia de voltaje	ACID-MGE	Aprendizaje	12.8 ± 5.0	22.6±2.4	0.93±0.16
	ACID-MGE	Prefijado	10	21.2±1.6	0.91±0.19
	ACID-GE	Prefijado	10	34.6±4.3	0.04±0.19

En relación con la eficiencia de la caché, es necesario indicar que un valor alto de la misma se corresponde con un menor número de evaluaciones de los cromosomas y, por lo tanto, sería positivo en eficiencia en tiempo. Sin embargo, otra posible lectura de un número más bajo de eficiencia de la caché podría corresponderse con que el algoritmo es capaz de proponer más candidatos diferentes y, por lo tanto, podría corresponderse con una mayor diversidad de la población evaluada. En este sentido, se observa que los valores de eficiencia de la caché de ACID-MGE son, en general, más bajos que en ACID-GE, lo que se puede relacionar con la mejoría del indicador SR que ofrece ACID-MGE, de forma que dicha mejora podría ser debida a una mayor diversidad de cromosomas candidatos, lo que se podría interpretar como que ACID-MGE resulta más eficiente explorando el espacio de búsqueda.

En relación con el valor medio de uso de *wrapping,* se observa que, en todos los casos, se encuentra por debajo del valor máximo fijado, 4, (véase sección 5.2.1).

Tabla F.2. – ACID-GE vs ACID-MGE: comparación de la eficiencia de la caché (EC) y *wrapping* usando 3 000 generaciones y dependiendo si se usa o no aprendizaje del parámetro MNN en ACID-MGE.

Circuito	Algoritmo	Ajuste de MNN	MNN±SD	EC(%)±SD	*Wrapping*±SD
Potencia al cuadrado	ACID-MGE	Aprendizaje	10.5 ± 5.3	22.6±4.4	0.94±0.13
	ACID-MGE	Prefijado	10	23.1±3.4	0.78±0.34
	ACID-GE	Prefijado	10	38.7±7.1	0.00±0.00
Raíz cuadrada	ACID-MGE	Aprendizaje	9.3 ± 2.9	21.7±3.9	0.77±0.36
	ACID-MGE	Prefijado	10	19.6±3.4	0.80±0.32
	ACID-GE	Prefijado	10	30.6±8.6	0.00±0.00
Potencia al cubo	ACID-MGE	Aprendizaje	10.0 ± 4.7	22.3±2.7	0.94±0.13
	ACID-MGE	Prefijado	10	21.8±1.5	0.94±0.09
	ACID-MGE	Prefijado	20	21.4±2.2	0.91±0.18
	ACID-GE	Prefijado	20	30.6±3.8	0.02±0.14
Raíz cúbica	ACID-MGE	Aprendizaje	15.8 ± 5.6	20.9±4.6	0.93±0.13
	ACID-MGE	Prefijado	15	19.6±1.6	0.93±0.14
	ACID-MGE	Prefijado	30	19.2±1.2	0.95±0.02
	ACID-GE	Prefijado	30	17.8±6.3	0.00±0.00

Adicionalmente, se observan valores más altos de uso de *wrapping* en ACID-MGE que en ACID-GE, es decir, los cromosomas de la última generación de los algoritmos utilizan más *wrapping* en el primer caso, estando entre un 0.77 y un 0.95 que, en el segundo, donde el uso es casi residual, siendo menor que 0.04. Estos valores no se pueden interpretar como porcentajes de uso de una lectura adicional del cromosoma, pues podría haber cromosomas que requieran más de una lectura. No obstante, sí ofrecen la indicación de un uso importante de este mecanismo en ACID-MGE.

G. Documentación del código

Está prevista la liberación de los algoritmos implementados en esta tesis bajo la licencia pública general (GPL, por sus siglas en inglés) de GNU en su versión 3. La liberación de ACID-GE se acometerá en primer lugar, seguida de la liberación de ACID-MGE, pendiente de la publicación del artículo que presenta dicho algoritmo (véase apéndice A). En este apéndice se incluye la documentación elaborada para la implementación de ACID-GE. Se presenta directamente en inglés porque será la lengua utilizada para compartir dicha documentación.

G.1. Software ACID-GE

G.1.1. Introduction

This document describes the *Analog Circuit Design using Grammatical Evolution* (ACID-GE) software. It covers some source code useful description, as well as distribution description, including instructions for its use.

G.1.2. Dependencies

ACID-GE software has been developed and tested in Ubuntu Linux, using the following tools:

- Maven 3.3.9
- Eclipse Oxygen
- Java: Oracle JDK 8u171
- NGSpice revision 27
- javaCC v5.0

G.1.3. Source code

The source code uses Maven as the tool for compiling and building the library. The source code is organized in a standard directory structure, which is described as follows:

```
acidge
|
|-- src
|   |-- main
|   |   |-- java <--- Java source code
|   |   |-- jjtree <--- sources for jjtree (Javacc)
|   |   |-- resources <--- scripts, grammar files and properties
|   |   `-- assembly <--- dist.xml for building distribution file
|   |
|   `-- test
|       |-- java <--- Junit test cases
|       `-- resources <--- grammar files and properties for testing
|
|-- target <--- Binaries are built here
|
|-- pom.xml <--- Maven project object model file
|-- NOTICE.txt <--- Notice.txt file
|-- README.txt <--- Readme.txt file
`-- LICENSE.txt <--- License information
```

G.1.4. Package description

This section covers a shallow description of the package structure of ACID-GE. The base package for all the source code is:

- es.uned.simda.acidge

es.uned.simda.acidge.ge

This package comprises the main classes for the algorithm, some of them are the following:

- Main: starting class
- GE: implements the evolutionary algorithm engine

- Poblacion: population class comprising Fenotipo instances
- Fenotipo: Fenotype
- Genotipo: genotype or chromosome class

es.uned.simda.acidge.ge.cache

This package implements a cache which stores the fitness of each genotype and avoids reevaluating those chromosomes whose genotype already appeared at an earlier moment in the evolutionary process. The key is a hash value of the generated netlist. This cache also implements a control for parallel execution.

es.uned.simda.acidge.ge.operadores

This package comprise all operators used by the algorithm, including mutation, crossover, initialization, parent selection and survivor selection operators.

es.uned.simda.acidge.ge.random

This class implements a random number generation interface for wrapping different implementing classes.

es.uned.simda.acidge.ge.random.mersenne

This package includes MersenneTwisterFast class which implements Mersenne twister random number generation. Refer to that file for the authors and license.

es.uned.simda.acidge.generador

This package implements a grammatical evolution decoder. It decodes a byte array from the chromosome and generates a final expression. The main decoder class is GeneradorGrammar.

es.uned.simda.acidge.generador.gramatica

This package implements an abstract grammar which is used to decode chromosomes by the grammatical evolution algorithm. The EBNF grammar parser reads a file and generates a Gramatica Class from this package.

es.uned.simda.acidge.generador.parser

This package implements an EBNF grammar parser, which is used for grammar file reading. The parser is based on Javacc and JJtree. It allows to write an abstract syntax tree (AST) from the grammar file read.

es.uned.simda.acidge.generador.arbol

This package implements a tree structure to be used while decoding a chromosome.

es.uned.simda.acidge.netlist

A raw decoded netlist cannot be fed to NGSpice as it is. A post-processing is needed to get the final netlist. This package implements this post-processing.

es.uned.simda.acidge.problema

This package implements an interface for evaluating chromosomes to solve a specific problem.

- Problema: an interface to wrap different kind of problems to be solved by the algorithm
- EvalRes: a superclass to contain a chromosome evaluation result

es.uned.simda.acidge.problema.dev

This package implements a circuit constructor class and a subclass a Problema for circuit developing.

es.uned.simda.acidge.problema.dev.eval

This package implements a superclass DevEval to evaluate different kind of circuits. KozaEvalRes is a subclass of EvalRes, and is used to contain a circuit evaluation result.

es.uned.simda.acidge.problema.dev.eval.cc

This package implements specific classes for evaluating computational circuits. A specific class is provided for squaring, square root, cubing and cube root circuits.

es.uned.simda.acidge.problema.dev.kozamat

This package implements specific classes for evaluating non-computational circuits. A specific class is provided for temperature sensing circuit, voltage reference circuit and gaussian function circuit.

es.uned.simda.acidge.problema.dev.netlist

This class implements a subclass of circuit constructor for generating netlists.

es.uned.simda.acidge.rmi

Classes for parallel execution, which is based on RMI.

es.uned.simda.acidge.spice

Classes for NGSpice calling and retrieving results. Main classes are:

- Spice: multithreading class for calling an NGSpice process and reading its output.
- ProcesaSalidaSpice: process NGSpice output and generates HashMap of Signal class
- Signal: it is a vector of values read from NGSpice output for processing in evaluation classes

es.uned.simda.acidge.stats

Utility class for statistics generation.

es.uned.simda.acidge.util

Logging classes.

G.1.5. Distribution

This section describes the directory structure for the distribution package. This directory allows algorithm run and testing.

A compressed tar file (.tgz) is built with the jar file, and all scripts, grammar files and properties files. The name of this file is: acidge-1.0.0-SNAPSHOT-distribution.tgz. Uncompressing of this file produces the following directory structure:

```
acidge-dist
|
|-- bin <--- scripts for executing the algorithm
|
|-- lib
|   `-- acidge.jar <--- library comprising the algorithm binaries
|
|-- grammar <--- grammar files in EBNF format
|   `-- various ebnf files
|
|-- properties
|   |-- sensor <--- properties for temperature sensing circuit
|   |   |-- GE.properties
|   |   `-- netlist.properties
|   |
|   |-- vref <--- properties for voltage reference circuit
|   |-- gaussian <--- properties for gaussian function circuit
|   |-- square <--- properties for squaring circuit
|   |-- sqroot <--- properties for square root circuit
|   |-- cube <--- properties for cubing circuit
```

```
    └── cuberoot <--- properties for cube root circuit
├── logs <--- log files
└── 20180504_... <--- temporal directories
```

G.1.6. Properties files

This section describes the properties files used by the algorithm. These files are GE.properties y netlist.properties. The algorithm looks for these files in the properties directory. Several example properties files are provided. The set of files to be used, need to be copied to the properties directory before running the algorithm.

GE.properties

This file comprises the properties for the algorithm. The properties are as follows:

- Paralelismo: execution in parallel. true or false.
- GeneratorClassName: fully qualified class name of Generator. It is already set to the appropriate class and should not be changed.
- ProblemClassName: fully qualified class name of problem to solve implementing class. It is already set to the appropriate class and should not be changed.
- CircuitConstructorClassName: fully qualified class name of circuit constructor class It is already set to the appropriate class and should not be changed.
- DevEvalClassName: fully qualified class name of circuit evaluation class. There is one class for each circuit tested. See the example properties files.
- IndividualsNumber: Number of chromosomes in the population. In the circuits tested it is set to 1000.
- MinGenesNumber: minimum chromosome length in random initialization.
- MaxGenesNumber: maximum chromosome length in random initialization.
- LimitMaxGenesNumber: maximum chromosome length. It is used in crossover operators.

- MaxEvaluationCount: alternative algorithm termination condition. Algorithm ends when this evaluation count is reached. It is not used in the circuits tested.
- GenerationsNumber: algorithm ends when this generation number is reached.
- TerminationConditionType: it is set to 5 in the circuits tested, which marks the first successful chromosome, and allows the algorithm to reach the maximum generations.
- PopulationInitializationClassName: fully qualified class name of population initialization operator class. It is already set to the appropriate class and should not be changed.
- ParentSelectionClassname: fully qualified class name of parent selection operator class. It is set to tournament selection for the circuits tested.
- TournamentSize: tournament size. It is set to 3 for the circuits tested.
- CrossoverClassName: fully qualified class name of crossover operator class. It is set to one-block operator for the circuits tested.
- CrossoverRate: crossover rate. It is set to 0.5 in the circuits tested.
- CrossoverPointsNumber: crossover points number. It should be set to one for the selected operator.
- CrossoverBlockSize: crossover block size. It has to agree with the block size of the grammar used.
- CrossoverRateGE2: not used
- CrossoverRateGE3: not used
- MutationClassName: fully qualified class name of mutation operator class. It is set to bitwise mutation for the circuits tested.
- MutationRate: mutation rate. It is set to 0.001 for the circuits tested
- MurationRateGE2: not used
- DuplicationClassName: fully qualified class name of duplication operator class. It is set to the appropriate class and should not be changed.
- DuplicationRate: duplication operator rate. It is set to zero for the circuits tested, since it is not really used.

- SurvivorSelectionClassName: fully qualified class name of survivor selection operator class. It is set to generational selection for the circuits tested.
- Elitism: number of the best chromosomes to save from one generation to the next. It is set to two for the circuits tested.
- GenerationalGap: number of chromosomes to replace on each generation, in steady-state survivor selection. It is not used for the circuits tested.
- MaxWrappingNumber: maximum wrapping allowed. It is set to 4 for the circuits tested.
- GrammarFileName: grammar filename. It should be in the grammar directory.
- FunAdaptacionClassName: fully qualified class name for fitness mapping class. It is not used for the circuits tested and should not be changed.
- UmbralError: not used
- GoalFitness: fitness goal objective. Not used in circuits generation which use its own criterium coded in evaluation class.
- HasGoal: wether problem has a goal. True or false. It is not used in circuits generation and should be set to false.
- RandomGeneratorClassName: fully qualified class name for random number generation class. It is set for Mersenne twister random generation for the circuits tested.
- MaxRecursionLevel: maximum recursion level in grammar decoding. It should not be changed.
- HashCache: Fitness cache using. It is set to true for the circuits tested.

netlist.properties

This file comprises the properties for netlist building. The properties are as follows:

- Simulaciones: number of simulations. It is set to one for the circuits tested.
- NodosProtegidos: a set of nodes separated by commas. If one of these nodes is left dangling or unconnected, the circuit will be considered unfeasible.
- EvitarComponentesColgando: if this property is set to one, a high valued resistor is connected from every dangling node to ground.

- ResistenciaElevada: this property is the value for the high valued resistor to be connected from dangling node s to ground. It is set to 1G for the circuits tested.
- netlist.header.0: this property contains the header lines to be appended at the top of the generated netlist. This property contains the fixed part of the circuit. It is a multiline property.
- netlist.modelo: this property contains the transistor models used. It is a multiline property.
- netlist.analisis.0: this property contains the NGSpice commands setting the analysis to be done in the simulation. It is a multiline property.

server.properties

This is an additional properties file used for parallel execution. This file contains the relation of servers in a cluster of nodes for parallel execution. It contains one line per node, indicating the name of the node and the number of server processes started on it, separated by a '=' character, as the following example:

```
node1=4
node2=3
```

G.1.7. Post-processing

This section describes the post-process done to a raw netlist generated by the decoding of the chromosome to prepare it for NGSpice simulation.

1. Inserting netlist.header.0 at the beginning of the netlist. This contains the fixed part of the circuit.
2. Appending netlist.modelo which contains transistor models used.
3. Appending netlist.analisis.0 which contains NGSPice commands
4. Deleting strings null1 and null2
5. Checking for protected nodes not to be dangling
6. Connecting 1GΩ resistors from dangling nodes to ground

7. Removing short circuited components
8. Converting component values from scientific notation to standard notation with suffixes.
9. Renumbering components

G.1.8. Scripts

This section describes several scripts which are used to run the algorithm. They are all in the bin directory.

- ejecutar.sh: basic script to launch an algorithm run.
- lanza.sh: script for launching several runs. It takes a number parameter which stands for the number of runs.
- nhlanza.sh: script for launching several runs as the former script. This script uses nohup command to allow disconnecting from the SSH session without interrupting the execution.
- filtra.sh: a script to extract useful information from the log files generated in a set of runs.
- server.sh: used to start a server used in parallel execution.
- nhserver.sh: used to start a number of servers with nohup, in parallel execution.

G.1.9. Parameters

Some parameters can be passed to the algorithm. They are the following:

- -server: it starts the program as a server for parallel execution
- -s number: used in conjunction with -server flag. It sets the server number
- -c value: it sets the crossover rate
- -m value: it sets the mutation rate

G.1.10. Log files

This section describes the log files generated by the algorithm during a run.

- salida_yyyymmdd_hhmiss_xxx.log: log file from an algorithm run.
- salserver_m_n: log file from server m. It is rotated so n can be 0 or 1.

While the algorithm is running, the following command can be used to monitor generations and best fitness so far:

- tail -f salida_yyyymmdd_hhmiss_xxx.log | grep BestFitness

The contents of the GE.properties file used are dumped to the log file when the algorithm start.

When the algorithm ends, the best fitness is reported in the log file with a line which start with the string "Mejor", followed by the fitness value, number of hits obtained. After that, the raw netlist generated is included, which comprise several lines. At the end of the netlist, some information of the chromosome is included, comprising the generation in which the chromosome appeared, the expressed length, and the complete byte string.

This raw netlist cannot be directly used as an input for the NGSpice simulator. The final netlist is reported in the log file with a line which start with the string "BestCircuit". The following lines comprises the netlist which can be directly used in the simulator.

Bibliografía

Aaserud, O. & Nielsen, I. R. (1995). Trends in current analog design - A panel debate. *Analog Integrated Circuits and Signal Processing*, 7(1), 5–9.

Aggarwal, V. (2003). Evolving sinusoidal oscillators using genetic algorithms. In *Proceedings of NASA/DoD Conference on Evolvable Hardware* (pp. 67–76).: IEEE.

Aguilar-Rivera, R., Valenzuela-Rendón, M., & Rodríguez-Ortiz, J. (2015). Genetic algorithms and darwinian approaches in financial applications: A survey. *Expert Systems with Applications*, 42(21), 7684–7697.

Al Jassani, B., Urquhart, N., & Almaini, A. (2011). State assignment for sequential circuits using multi-objective genetic algorithm. *IET computers & digital techniques*, 5(4), 296–305.

Aler, R., Borrajo, D., & Isasi, P. (2002). Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, 141(1-2), 29–56.

Ali, B., Almaini, A., & Kalganova, T. (2004). Evolutionary algorithms and theirs use in the design of sequential logic circuits. *Genetic Programming and Evolvable Machines*, 5(1), 11–29.

Ando, S. & Iba, H. (2000). Analog circuit design with a variable length chromosome. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 2 (pp. 994–1001).

Camenzind, H. (2005). *Designing analog chips*. Virtual Bookworm. Com Pub Incorporated.

Cano, A., Ventura, S., & Cios, K. J. (2017). Multi-objective genetic programming for feature extraction and data visualization. *Soft Computing*, 21(8), 2069–2089.

Castejón, F. & Carmona, E. (2013). Automatic design of electronic amplifiers using grammatical evolution. In A. Alonso-Betanzos & others (Eds.), *Actas de Multiconferencia CAEPIA-13* (pp. 703–712).

Castejón, F. & Carmona, E. J. (2018). Automatic design of analog electronic circuits using grammatical evolution. *Applied Soft Computing*, 62, 1003 – 1018.

Cipriani, S. & Takeshian, A. (2000). Compact cubic function generator. US Patent 6,160,427.

Coello Coello, C. A. & Aguirre, A. H. (2002). Design of combinational logic circuits through an evolutionary multiobjective optimization approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 16(1), 39–53.

Cohen, J. (1992). A power primer. *Psychological bulletin*, 112(1), 155–159.

Colmenar, J. M., Cuesta-Infante, A., Risco-Martín, J. L., & Hidalgo, J. I. (2013). An evolutionary methodology for automatic design of finite state machines. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO 2013 Companion (pp. 139–140). New York, NY, USA: Association for Computing Machinery.

Dabhi, V. K. & Chaudhary, S. (2015). Empirical modeling using genetic programming: A survey of issues and approaches. *Natural Computing*, 14(2), 303–330.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *International Conference on Parallel Problem Solving From Nature* (pp. 849–858).: Springer.

D&R (2018). Analog IC market forecast with strongest annual growth through 2022. Recuperado de `https://www.design-reuse.com/news/43374/analog-ic-market-forecast.html`. Último acceso 26/04/2020.

Eiben, A. E. & Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin: Springer.

El Dor, A., Fakhfakh, M., & Siarry, P. (2016). Multiobjective differential evolution algorithm using crowding distance for the optimal design of analog circuits. *Journal of Electrical Systems*, 12(3).

El-Maleh, A. H., Sheikh, A. T., & Sait, S. M. (2013). Binary particle swarm optimization (BPSO) based state assignment for area minimization of sequential circuits. *Applied soft computing*, 13(12), 4832–4840.

Estébanez, C., Saez, Y., Recio, G., & Isasi, P. (2014). Automatic design of noncryptographic hash functions using genetic programming. *Computational Intelligence*, 30(4), 798–831.

Fagan, D., O'Neill, M., Galván-López, E., Brabazon, A., & McGarraghy, S. (2010). An analysis of genotype-phenotype maps in grammatical evolution. In *European Conference on Genetic Programming* (pp. 62–73).: Springer.

Floreano, D. & Mattiussi, C. (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press.

Francone, F. D., Conrads, M., Banzhaf, W., & Nordin, P. (1999). Homologous crossover in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, volume 2 (pp. 1021–1026). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Gan, Z., Yang, Z., Shang, T., Yu, T., & Jiang, M. (2010). Automated synthesis of passive analog filters using graph representation. *Expert Systems with Applications*, 37(3), 1887–1898.

Gielen, G. G., Walscharts, H. C., & Sansen, W. M. (1990). Analog circuit design optimization based on symbolic simulation and simulated annealing. *Solid-State Circuits, IEEE Journal of*, 25(3), 707–713.

Gielen, G. G. E. & Rutenbar, R. A. (2002). Computer-aided design of analog and mixed-signal integrated circuits. In R. A. Rutenbar, G. G. E. Gielen, & B. A. Antao (Eds.), *Computer-Aided Design of Analog Integrated Circuits and Systems* (pp. 3–10). New York, NY, USA: John Wiley & Sons, Inc.

Gilbert, B. (2001). Analog design in the information age. In *Proceedings of the 2001 BIPOLAR/BiCMOS Circuits and Technology Meeting* (pp. 118–123).

González-Taboada, I., González-Fonteboa, B., Martínez-Abella, F., & Pérez-Ordóñez, J. L. (2016). Prediction of the mechanical properties of structural recycled concrete using multivariable regression and genetic programming. *Construction and Building Materials*, 106, 480–499.

Gordon, T. G. & Bentley, P. J. (2005). Development brings scalability to hardware evolution. In *Proceedings of NASA/DoD Conference on Evolvable Hardware* (pp. 272–279).: IEEE.

Grimbleby, J. (2000). Automatic analogue circuit synthesis using genetic algorithms. *IEEE Proceedings - Circuits, Devices and Systems*, 147(6), 319–323.

Grimbleby, J. B. (1995). Automatic analogue network synthesis using genetic algorithms. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications* (pp. 53–58).

Harjani, R., Rutenbar, R. A., & Carley, L. R. (1987). A prototype framework for knowledge-based analog circuit synthesis. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, DAC '87 (pp. 42–49). New York, NY, USA: ACM.

Hidalgo, J. I., Fernandez, F., Lanchares, J., Sanchez, J., Hermida, R., Tomassini, M., Baraglia, R., Perego, R., & Garnica, O. (2003). Multi-FPGA systems synthesis by means of evolutionary computation. In *Genetic and Evolutionary Computation Conference* (pp. 2109–2120).: Springer.

Higuchi, T., Iwata, M., Kajitani, I., Yamada, H., Manderick, B., Hirao, Y., Murakawa, M., Yoshizawa, S., & Furuya, T. (1996). Evolvable hardware with genetic learning. In *IEEE International Symposium on Circuits and Systems, ISCAS '96, Connecting the World.*, volume 4 (pp. 29–32).

Hounsell, B. I. & Arslan, T. (2000). A novel genetic algorithm for the automated design of performance driven digital circuits. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1 (pp. 601–608).: IEEE.

Hu, J., Zhong, X., & Goodman, E. D. (2005). Open-ended robust design of analog filters using genetic programming. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (pp. 1619–1626).: ACM.

ISO/IEC-14977 (1996). Information technology – syntactic metalanguage – extended BNF. doi: 10.3403/01300022.

Johnson, R. C. (2015). Analog EDA finally automated. Recuperado de `https://www.eenewseurope.com/news/analog-eda-finally-automated`. Último acceso 26/04/2020.

Kalganova, T. (2000). Bidirectional incremental evolution in extrinsic evolvable hardware. In *The Second NASA/DoD Workshop on Evolvable Hardware* (pp. 65–74).: IEEE.

Kalganova, T., Miller, J. F., & Fogarty, T. C. (1998). Some aspects of an evolvable hardware approach for multiple-valued combinational circuit design. In *Evolvable Systems: From Biology to Hardware* (pp. 78–89). Springer.

Karpuzcu, U. R. (2005). Automatic Verilog code generation through grammatical evolution. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation* (pp. 394–397).: ACM.

Kazarlis, S., Kalomiros, J., Balouktsis, A., & Kalaitzis, V. (2015). Evolving optimal digital circuits using cartesian genetic programming with solution repair methods. In *Proceedings of the 2015 International Conference on Systems, Control, Signal Processing and Informatics (SCSI 2015), Barcelona, Spain* (pp. 39–44).

Kazarlis, S., Kalomiros, J., Mastorocostas, P., Petridis, V., Balouktsis, A., Kalaitzis, V., & Valais, A. (2014). A method for simulating digital circuits for evolutionary optimization. In *Proceedings of the 10th Annual International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE 2014)*.

Kim, K.-J. & Cho, S.-B. (2012). Automated synthesis of multiple analog circuits using evolutionary computation for redundancy-based fault-tolerance. *Applied Soft Computing*, 12(4), 1309 – 1321.

Kim, K.-J., Wong, A., & Lipson, H. (2010). Automated synthesis of resilient and tamper-evident analog circuits without a single point of failure. *Genetic Programming and Evolvable Machines*, 11(1), 35–59.

Koza, J., Andre, D., Bennett III, F., & Keane, M. (1999a). *Genetic Programming III: Darwinian Invention & Problem Solving*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st edition.

Koza, J., Bennett III, F., Andre, D., & Keane, M. (1998). Evolutionary design of analog electrical circuits using genetic programming. In I. C. Parmee (Ed.), *Adaptive Computing in Design and Manufacture: The Integration of Evolutionary and Adaptive Computing Technologies with Product/System Design and Realisation* (pp. 177–192). Springer London.

Koza, J., Bennett III, F., Andre, D., & Keane, M. (1999b). The design of analogue circuits by means of genetic programming. In P. J. Bentley (Ed.), *Evolutionary Design by Computers* chapter 16, (pp. 365–385). John Wiley&Son.

Koza, J., Bennett III, F., Andre, D., & Keane, M. (2000a). Automatic design of analog electrical circuits using genetic programming. In H. Cartwright (Ed.), *Intelligent Data Analysis in Science* chapter 8, (pp. 172–200). Oxford: Oxford University Press.

Koza, J., Bennett III, F., Andre, D., & Keane, M. (2000b). Synthesis of topology and sizing of analog electrical circuits by means of genetic programming. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4), 459–482.

Koza, J. R. (1992). *Genetic Programming: On The Programming Of Computers By Means Of Natural Selection*, volume 1. Cambridge, MA, USA: MIT press.

Koza, J. R., Bennett, III, F. H., Andre, D., & Keane, M. A. (1997a). Evolution using genetic programming of a low-distortion, 96 decibel operational amplifier. In *Proceedings of the 1997 ACM Symposium on Applied Computing* (pp. 207–216). New York, NY, USA: ACM.

Koza, J. R., Bennett III, F., Lohn, J., Dunlap, F., Keane, M. A., & Andre, D. (1997b). Automated synthesis of computational circuits using genetic programming. In *IEEE International Conference on Evolutionary Computation* (pp. 447–452).: IEEE.

Koza, J. R., Streeter, M. J., & Keane, M. A. (2008). Routine high-return human-competitive automated problem-solving by means of genetic programming. *Information Sciences*, 178(23), 4434–4452.

Koza, R. J., Keane, A. M., & Streeter, J. M. (2003). Routine automated synthesis of five patented analog circuits using genetic programming. *Soft Computing*, 8(5), 318–324.

Kunaver, M. (2020). Grammatical evolution-based analog circuit synthesis. *Informacije MIDEM*, 49(4), 229–240.

Lanchares, J., Garnica, O., Hidalgo, J. I., & Risco-Martín, J. L. (2013). Sumador evolutivo de 2 bits tolerante a fallos. In A. Alonso-Betanzos & others (Eds.), *Actas de Multiconferencia CAEPIA-13* (pp. 733–742).

Lanchares, J., Garnica, O., Risco-Martín, J. L., Hidalgo, J. I., Colmenar, J. M., & Cuesta-Infante, A. (2014). Real time evolvable hardware for optimal reconfiguration of cusp-like pulse shapers. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 763, 124–131.

Langdon, W. B. (2000). Size fair and homologous tree crossovers for tree genetic programming. *Genetic programming and evolvable machines*, 1(1-2), 95–119.

Lohn, J. D. & Colombano, S. P. (1999). A circuit representation technique for automated circuit design. *IEEE Transactions on Evolutionary Computation*, 3(3), 205–219.

Lourenço, N., Ferrer, J., Pereira, F. B., & Costa, E. (2017). A comparative study of

different grammar-based genetic programming approaches. In *European Conference on Genetic Programming* (pp. 311–325).: Springer.

Martens, E. & Gielen, G. (2008). Classification of analog synthesis tools based on their architecture selection mechanisms. *Integration, the VLSI Journal*, 41(2), 238–252.

Mattiussi, C. (2005). *Evolutionary synthesis of analog networks*. PhD thesis, Università degli Studi di Trieste.

Mattiussi, C. & Floreano, D. (2007). Analog genetic encoding for the evolution of circuits and networks. *IEEE Transactions on Evolutionary Computation*, 11(5), 596–607.

McClean, B. (2018). The McClean report. Recuperado de `http://www.icinsights.com/services/mcclean-report/`. Último acceso 26/04/2020.

McConaghy, T. & Gielen, G. (2006a). Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation* (pp. 855–862).: ACM.

McConaghy, T. & Gielen, G. (2006b). Genetic programming in industrial analog CAD: Applications and challenges. In T. Yu, R. Riolo, & B. Worzel (Eds.), *Genetic Programming Theory and Practice III* (pp. 291–306). Springer US.

McConaghy, T., Palmers, P., Gielen, G., & Steyaert, M. (2007). Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07 (pp. 944–947). New York, NY, USA: ACM.

McConaghy, T., Palmers, P., Steyaert, M., & Gielen, G. G. (2009). Variation-aware structural synthesis of analog circuits via hierarchical building blocks and structural homotopy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(9), 1281–1294.

Mckay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., & O'Neill, M. (2010). Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines*, 11(3-4), 365–396.

Miller, J. F. & Thomson, P. (2000). Cartesian genetic programming. In *European Conference on Genetic Programming* (pp. 121–132).: Springer.

Miller, J. F., Thomson, P., & Fogarty, T. (1997). *Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study*, volume 8. Wiley.

Millman, J. & Valls, E. B. (1981). *Microelectrónica: Circuitos y sistemas análogos y digitales*. Hispano Europea.

Mrazek, V. & Vasicek, Z. (2018). Evolutionary design of large approximate adders optimized for various error criteria. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 294–295).: ACM.

Mydlowec, W. & Koza, J. (2000). Use of time-domain simulations in automatic synthesis of computational circuits using genetic programming. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference* (pp. 187–197).

Nagel, L. W. & Pederson, D. O. (1973). *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California.

Nenzi, P. & Vogt, H. (2011). NGSPICE user's manual version 23.

Nicolau, M. & Dempsey, I. (2006). Introducing grammar based extensions for grammatical evolution. In *IEEE Congress on Evolutionary Computation* (pp. 648–655).: IEEE.

Nordin, P., Francone, F., & Banzhaf, W. (1995). Explicitly defined introns and destructive crossover in genetic programming. *Advances in genetic programming*, 2, 111–134.

O'Neill, M. & Brabazon, A. (2004). Grammatical swarm. In K. Deb (Ed.), *Genetic and Evolutionary Computation – GECCO 2004* (pp. 163–174). Berlin: Springer.

O'Neill, M. & Brabazon, A. (2006a). Grammatical differential evolution. In H. R. Arabnia (Ed.), *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, volume 1 (pp. 231–236). Las Vegas, Nevada, USA: CSREA Press.

O'Neill, M. & Brabazon, A. (2006b). Grammatical swarm: The generation of programs by social programming. *Natural Computing*, 5(4), 443–462.

O'Neill, M., Brabazon, A., Nicolau, M., Mc Garraghy, S., & Keenan, P. (2004). πGrammatical evolution. In *Genetic and Evolutionary Computation Conference* (pp. 617–629).: Springer.

O'Neill, M. & Ryan, C. (1999). Under the hood of grammatical evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2 (pp. 1143–1148).

O'Neill, M. & Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5, 349–358.

O'Neill, M. & Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.

O'Neill, M. & Ryan, C. (2004). Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. *Genetic Programming*, (pp. 138–149).

O'Neill, M., Ryan, C., Keijzer, M., & Cattolico, M. (2003). Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4, 67–93.

Pêcheux, F., Lallement, C., & Vachoux, A. (2005). VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multi-discipline systems. *IEEE transactions on Computer-Aided design of integrated Circuits and Systems*, 24(2), 204–225.

Povoa, R., Bastos, I., Lourenço, N., & Horta, N. (2016). Automatic synthesis of RF front-end blocks using multi-objective evolutionary techniques. *Integration, the VLSI Journal*, 52, 243–252.

Pressman, R. S. (1997). *Software Engineering: A Practitioner's Approach*. New York, NY, USA: McGraw-Hill.

Puhan, J., Tuma, T., & Fajfar, I. (1999). Optimisation methods in SPICE: a comparison. In *Proceedings of European Conference on Circuit Theory and Design. ECCTD'99. Vol*, volume 2 (pp. 1279–1282).

Rojec, Ž., Bűrmen, Á., & Fajfar, I. (2019). Analog circuit topology synthesis by means of evolutionary computation. *Engineering Applications of Artificial Intelligence*, 80, 48–65.

Rutenbar, R. (1993). Analog design automation: Where are we? Where are we going? In *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993* (pp. 13.1.1–13.1.7).

Rutenbar, R., Gielen, G., & Roychowdhury, J. (2007). Hierarchical modeling, optimization, and synthesis for system-level analog and RF designs. *Proceedings of the IEEE*, 95(3), 640–669.

Ryan, C. (2017). A rebuttal to Whigham, Dick, and Maclaurin by one of the inventors of grammatical evolution: Commentary on "On the mapping of genotype to phenotype in evolutionary algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin. *Genetic Programming and Evolvable Machines*, 18(3), 385–389.

Ryan, C., Collins, J. J., & O'Neill, M. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf, R. Poli, M. Schoenauer, & T. C. Fogarty (Eds.), *Proceedings of the First European Workshop on Genetic Programming*, volume 1391 of *LNCS* (pp. 83–96). Berlin: Springer-Verlag.

Sabat, S. L., Kumar, K. S., & Udgata, S. K. (2009). Differential evolution and swarm intelligence techniques for analog circuit synthesis. In *World Congress on Nature & Biologically Inspired Computing (NaBIC)* (pp. 469–474).

Sapargaliyev, Y. (2011). *Automatic design of analogue circuits*. PhD thesis, Brunel University School of Engineering and Design PhD Theses.

Sapargaliyev, Y. A. & Kalganova, T. G. (2012). Open-ended evolution to discover analogue circuits for beyond conventional applications. *Genetic Programming and Evolvable Machines*, 13(4), 411–443.

Scheible, J. & Lienig, J. (2015). Automation of analog IC layout: Challenges and solutions. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15 (pp. 33–40). New York, NY, USA: ACM.

Shannon, C. E. (1938). A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57(12), 713–723.

Slezák, J. & Petržela, J. (2014). Evolutionary synthesis of cube root computational circuit using graph hybrid estimation of distribution algorithm. *Radioengineering*, 23(1), 549.

Soderstrand, M. A. (2012). The moving interface between digital and analog circuits and its effect on future wireless systems. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on* (pp. 845–848).

Sorkhabi, S. E. & Zhang, L. (2017). Automated topology synthesis of analog and RF integrated circuits: A survey. *Integration*, 56, 128–138.

Sripramong, T. & Toumazou, C. (2002). The invention of CMOS amplifiers using genetic programming and current-flow analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11), 1237–1252.

Stoica, A., Zebulum, R., Guo, X., Keymeulen, D., Ferguson, M., & Duong, V. (2004). Taking evolutionary circuit design from experimentation to implementation: Some useful techniques and a silicon demonstration. *IEE Proceedings - Computers and Digital Techniques*, 151(4), 295–300.

Stoica, A., Zebulum, R., Keymeulen, D., Ferguson, M., & Guo, X. (2003). Scalability issues in evolutionary synthesis of electronic circuits: Lessons learned and challenges ahead. In *American association for artificial intelligence*.

Streeter, M. J., Keane, M. A., & Koza, J. R. (2002). Iterative refinement of computational circuits using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 877–884).: Morgan Kaufmann Publishers Inc.

Suganuma, M., Shirakawa, S., & Nagao, T. (2017). A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 497–504).: ACM.

Swafford, J. M., Hemberg, E., O'Neill, M., Nicolau, M., & Brabazon, A. (2011). A non-destructive grammar modification approach to modularity in grammatical evolution. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (pp. 1411–1418).: ACM.

Thompson, A. (1997). Artificial evolution in the physical world. *Evolutionary Robotics: From intelligent robotics to artificial life*.

Tlelo-Cuautle, E. & Duarte-Villaseñor, M. (2008). Evolutionary electronics: Automatic synthesis of analog circuits by GAs. In A. Yang, Y. Shan, & L. Bui (Eds.), *Success in Evolutionary Computation*, volume 92 (pp. 165–187). Springer Berlin / Heidelberg.

Trefzer, M. A. (2006). *Evolution of transistor circuits*. PhD thesis.

Varios (2013). ASCO (A Spice Circuit Optimizer). Recuperado de `http://asco.sourceforge.net/index.html`. Último acceso 26/04/2020.

Vasicek, Z. & Sekanina, L. (2011). Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3), 305–327.

Vasicek, Z. & Sekanina, L. (2016). Evolutionary design of complex approximate combinational circuits. *Genetic Programming and Evolvable Machines*, 17(2), 169–192.

Wagner, G. P. & Altenberg, L. (1996). Perspective: Complex adaptations and the evolution of evolvability. *Evolution*, 50(3), 967–976.

Wang, F., Li, Y., Li, K., & Lin, Z. (2008). A new circuit representation method for analog circuit design automation. In *IEEE Congress on Evolutionary Computation (CEC)* (pp. 1976–1980).: IEEE.

Wang, F., Li, Y., Li, L., & Li, K. (2007). Automated analog circuit design using two-layer genetic programming. *Applied Mathematics and Computation*, 185(2), 1087–1097.

Whigham, P. A., Dick, G., Maclaurin, J., & Owen, C. A. (2015). Examining the "best of both worlds" of grammatical evolution. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (pp. 1111–1118).: ACM.

Xiu, L. (2007). *VLSI circuit design methodology demystified: A conceptual taxonomy*. John Wiley & Sons.

Yan, X., Wei, W., Liu, R., Zeng, S., & Kang, L. (2006). Designing electronic circuits by means of gene expression programming. In *First NASA/ESA Conference on Adaptive Hardware and Systems* (pp. 194–199).

Yuan, H. & He, J. (2010). Evolutionary design of operational amplifier using variable-length differential evolution algorithm. In *International Conference on Computer Application and System Modeling (ICCASM)*, volume 4 (pp. 610–614).

Zebulum, R., Pacheco, M., & Vellasco, M. (1998a). Comparison of different evolutionary methodologies applied to electronic filter design. In *IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence* (pp. 434–439).

Zebulum, R., Pacheco, M. A., & Vellasco, M. (1998b). Synthesis of CMOS operational amplifiers through genetic algorithms. In *XI Brazilian Symposium on Integrated Circuit Design, 1998. Proceedings* (pp. 125–128).: IEEE.

Zebulum, R. S., Pacheco, M. A., & Vellasco, M. (1999). Artificial evolution of active filters: A case study. In *Proceedings of the 1st NASA/DOD Workshop on Evolvable Hardware* (pp. 66–75).: IEEE Computer Society.

Zebulum, R. S., Vellasco, M. S., & Pacheco, M. A. (2000). Variable length representation in evolutionary electronics. *Evolutionary Computation*, 8(1), 93–120.