UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

CLASIFICACIÓN DE PATRONES DE CIRCUITOS DE INTERCONEXIÓN

TESIS

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:

ING. JULIETA AYALA RODRÍGUEZ

DIRECTORA DE TESIS:

DRA. CRISTINA VERDE RODARTE
INSTITUTO DE INGENIERÍA

Ciudad de México a 24 de noviembre de 2019

Agradecimientos

Gracias a *Powerbeam* por sembrar la idea de que era posible encontrar patrones en el diseño electrónico desde una perspectiva diferente a la teoría de circuitos y además proporcionar los recursos necesarios para probarla. Sin ellos, este proyecto no hubiera sido posible.

Gracias a la Dra. María Cristina Verde Rodarte por creer en el éxito de este proyecto desde el primer día. Sin su guía y comprensión no habría podido comprobar la hipótesis sobre la cual se cimienta esta tesis.

Resumen

La verificación de esquemáticos consiste en revisar los diseños electrónicos componente por componente con el objetivo de detectar errores antes de manufacturar el circuito. Realizar esta tarea de forma manual es ineficiente y está propensa a errores por lo que se busca un método sistemático que la simplifique. En esta tesis se propone el *Sistema Clasificador de Patrones de Subcircuitos de Interconexión* (*SCPSI*) basado en la teoría de circuitos, la teoría de grafos y la teoría de clasificación, el cual permita detectar potenciales conexiones inadecuadas en proyectos de diseños electrónicos. De forma general, las tareas involucradas son:

- Obtener la información de interconexión de los diseños electrónicos.
- Transformarla a estructuras de datos del lenguaje Python para su manipulación.
- Extraer los subcircuitos de interconexión y transformarlos a grafos de interconexión.
- Caracterizar los subcircuitos extraídos con base en la teoría de grafos.
- Diseñar el clasificador secuencial.
- Evaluar el sistema para encontrar los patrones.

A partir del desarrollo y evaluación del *SCPSI* se determinó que es posible abordar el análisis estructural de circuitos electrónicos desde una perspectiva de aprendizaje de máquina y sin tomar en cuenta características eléctricas.

Se confirmó la hipótesis que dio origen al *SCPSI*, es decir que existen patrones que prevalecen en el diseño electrónico independientemente del propósito del circuito. Esto abre las puertas a un nuevo tipo de herramienta auxiliar en el diseño de hardware: la detección de subcircuitos de interconexión potencialmente erróneos dadas caracterizaciones no conocidas previamente por el *SCPSI*.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Planteamiento del Problema

Gracias a la automatización de la manufactura de placas de circuito impreso, hoy en día el orden de magnitud de componentes en proyectos de diseño de hardware es de 10^3 (Dato obtenido a partir de proyectos reales proporcionados por la empresa PowerBeam Inc). El tipo de componentes que conforman un diseño electrónico varía entre simples o de dos terminales (capacitores, resistores, inductores y diodos), de tres terminales (transistores principalmente), unidades de procesamiento (componentes complejos) e interfaces hacia el exterior. En general, el propósito de los componentes simples es formar subcircuitos de interconexión entre las unidades de mayor complejidad, como los enmarcados en rojo en la Figura 1.1.

Una tarea muy importante del proceso de diseño es la verificación de esquemáticos (*schematic checking*) la cual consiste en revisar el diseño componente por componente con el objetivo de detectar errores antes de manufacturar el circuito. El método tradicional consiste en imprimir el esquemático del proyecto, el cual puede consistir en hasta decenas de páginas, comparar el nombre o número de pin de la hoja de datos (*datasheet*) con el nombre o número de pin del esquema y con el nombre de la pista y buscar a simple vista los errores más comunes (resistores en *pull up* o *pull down* faltantes, señales invertidas, componentes no aterrizados). En resumen, verificar un esquemático es una tarea manual y por ende ineficiente y propensa a errores. De ahí el interés de empresas dedicas al diseño y validación de circuitos electrónicos de contar con métodos sistemáticos que simplifiquen la tarea. Por lo expuesto, se propone un *Sistema Clasificador de Patrones de Subcircuitos de Interconexión*, de ahora en adelante referido como *SCPSI*, que permita detectar potenciales conexiones inadecuadas

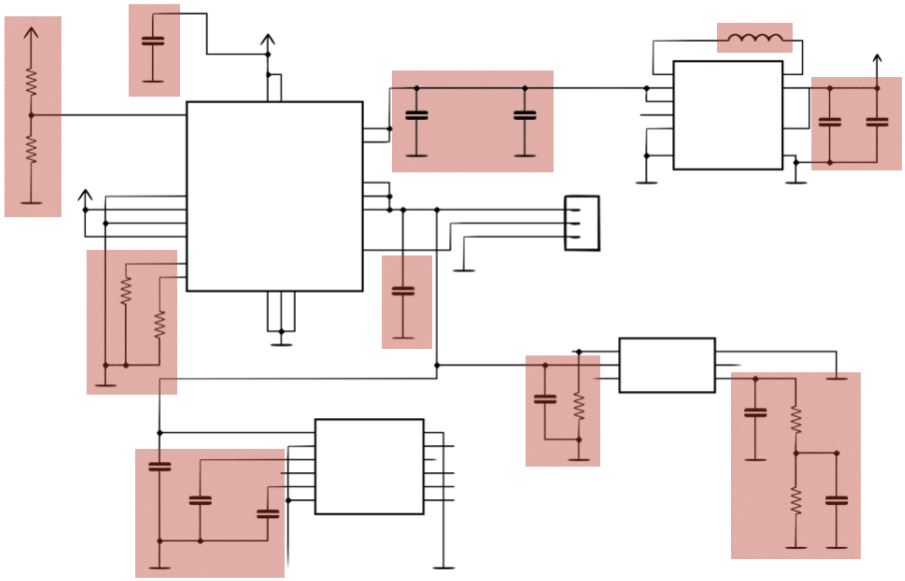


Figura 1.1: Circuitos de interconexión de un esquemático

en proyectos de diseño electrónico. Es decir, se pretende apoyar al ingeniero electrónico con una aplicación automatizada que identifique conexiones inusuales dentro de los diseños. Para ello, el sistema analiza circuitos de interconexión con base en las teorías de reconocimiento de patrones, grafos y circuitos.

1.1.1. Hipótesis

Como idea general se piensa que existen familias de subcircuitos de interconexión que prevalecen a través de los diseños electrónicos independientemente de su aplicación; es decir, que hay patrones de componentes simples que vinculan unidades de procesamiento más complejas que aparecen recurrentemente a través de proyectos cuyos propósitos son diferentes.

De forma puntual, se cree que es posible encontrar dichos patrones a través de la técnica de clasificación no supervisada y además que es posible lograr dicha clasificación de subcircuitos de interconexión con el menor número posible de características eléctricas.

1.2. Antecedentes

Existen técnicas para identificar potenciales problemas a través del análisis de diseño. Se apoyan de herramientas cuya perspectiva está centrada en la seguridad y rendimiento del producto para

evitar que ocurran situaciones no deseadas o no intencionadas; sobretodo situaciones cuya posibilidad de ocurrencia no se explica ni se identifica [Scappaticci et al., 2016].

Un circuito escurridizo es una ruta no intencionada en una red que puede causar acciones no deseadas. El análisis de circuitos de fuga (*Sneak Circuit Analysis*) es una de dichas técnicas analíticas empleadas por ingenieros de diseño para identificar rutas latentes que causan la aparición de funciones no deseadas o que inhiben las funciones deseadas en sistemas electrónicos y electromecánicos [Miller, 1989]. SCA, por sus siglas en inglés, ayuda a los ingenieros a garantizar que el diseño funcionará como se desea en cada modo y/o fase de operación. La información necesaria para un análisis SCA incluye conocimiento detallado del sistema, obtenido de las especificaciones del sistema y los esquemáticos, un extenso análisis de modo de falla y sus efectos (FMEA, *Failure Mode and Effects Analysis*) y acceso a los ingenieros de diseño o a los expertos en la materia que estén más familiarizados con el diseño y operación del sistema [Scappaticci et al., 2016].

Se descubrió que las causas frecuentes de los circuitos de fuga estaban asociadas con distintos patrones topológicos en los diagramas de circuitos por lo que a partir de la identificación de dichos patrones y el registro de los atributos del circuito de cada patrón fue posible llevar a cabo un análisis basado en la experiencia. Este enfoque condujo al desarrollo de métodos semiautomáticos de aislamiento de los patrones topológicos en circuitos relé y a la generación de listas de pistas aplicables a cada tipo de patrón topológico. Los patrones más significativos se muestran en le Figura 1.2 [Miller, 1989].

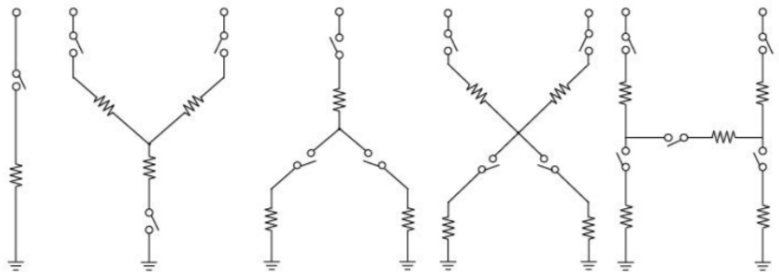


Figura 1.2: Patrones topológicos básicos de circuitos de fuga, de [Chen, 2009]

El análisis SCA se compone de cuatro fases [Scappaticci et al., 2016]:

- División del circuito: Separar el circuito en funciones llamadas particiones.

- Árboles de red: Desarrollar árboles de red para cada partición identificada.
- Cálculo de caminos: Descubrir rutas no intencionadas y determinar si dichas rutas podrían producir un circuito o condición de fuga.
- Pistas de fuga: Documentar las respuestas a las preguntas surgidas al examinar los patrones topológicos de circuitos de fuga identificados en los árboles de red.

El análisis de circuitos de fuga es de interés para el desarrollo de esta tesis pues sus dos primeras fases son tareas que también se llevan a cabo en el SCPSI, con algunas variaciones especificadas más adelante. En el caso del SCA, la información de interconexión del circuito se divide para construir árboles de red y filtrar así los datos no relevantes del esquemáticos para generar una representación visual simplificada del circuito. La topología de los árboles se analiza para identificar la aparición de los patrones básicos [Miller, 1989].

En 1989, Jeff Miller escribió un reporte llamado *Sneak Circuit Analysis for the Common Man* [Miller, 1989] en el cual explicaba que SCA era una tarea que requería mucha mano de obra e importantes recursos informáticos, por lo tanto, únicamente se aplicaba en áreas críticas de los sistemas. Gracias al crecimiento en la capacidad de procesamiento de las computadoras, hoy en día los recursos informáticos no son una limitante; sin embargo, la información de interconexión de los circuitos sigue siendo bastante compleja pues se encuentra repartida a través de muchos esquemáticos en un mismo proyecto. Se han desarrollado técnicas automatizadas para capturar circuitos electrónicos y generar datos de interconexión de arboles de red las cuales han demostrado ser indispensables para un análisis eficiente, preciso y exhaustivo de grandes sistemas. El software para realizar el procesamiento de datos del circuito y la generación de árboles se considera altamente patentado por aquellos contratistas que han desarrollado capacidad de SCA. Además, se requiere que un equipo de analistas especialmente capacitados aplique listas de pistas de fuga a cientos de árboles de red que se generan por proyecto. Por estos motivos, el rendimiento del análisis se limita a los contratistas de SCA en todos los casos, excepto en los más simples [Miller, 1989].

En 2009, Bolin Chen publicó un artículo titulado *Topological Patterns Identification for Sneak Circuit Analysis* [Chen, 2009] donde propuso asociar grafos con signos a las redes de circuito representando las terminales de los elementos con los vértices y los elementos con las aristas de tal forma que el grafo con signo contenía toda la información topológica de la red. Entonces, transformó el

problema de identificar patrones topológicos en redes de circuitos al de generar todos los subgrafos homeomorfos de los cinco arboles básicos con signo en los grafos con signo de los circuitos. Dos grafos son homeomorfos si ambos pueden obtenerse a partir de un mismo grafo por una sucesión de subdivisiones elementales de aristas. En el mismo artículo explica que la identificación de patrones topológicos es muy importante para hacer SCA, sin embargo, la mayoría de los métodos de identificación se mantienen en secreto.

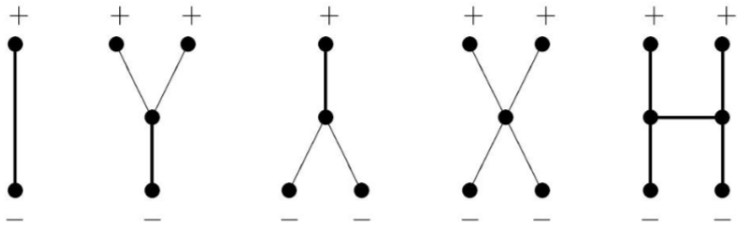


Figura 1.3: Patrones topológicos básicos con signo, de [Chen, 2009]

1.3. Objetivo

Con base en lo planteado, el objetivo de esta tesis es diseñar y desarrollar un sistema clasificador de patrones de subcirucitos de interconexión de proyectos de gran escala y que ayude a detectar potenciales errores en el diseño electrónico. Este sistema se compone de las siguientes subtareas:

- Formular el problema de clasificación de circuitos de interconexión en el contexto de teoría de grafos.
- Transformar la información de conectividad y de los componentes del esquemático a representaciones computacionales de grafos.
- Identificar las particularidades de los grafos asociados a los circuitos de interconexión.
- Seleccionar las características con mayor relevancia de los mismos.
- Seleccionar el criterio de similitud para el agrupamiento.
- Diseñar el algoritmo no supervisado de clasificación en función de las características y el criterio de similitud seleccionados.

1.4. Contribución

El desarrollo de este proyecto creará una herramienta más para el proceso de verificación de circuitos, el cual es una actividad cotidiana en las empresas dedicadas al diseño y desarrollo de hardware. Específicamente, es una problemática actual de la empresa PowerBeam dedicada al diseño electrónico pues manufacturar una placa de circuito impreso con errores conlleva un costo tanto económico como temporal.

1.5. Metodología

Para cumplir con el objetivo planteado y sus particularidades se ejecutaron las siguientes tareas:

- Estudio preliminar de:
  - Teoría de grafos
  - Aplicación de la teoría de grafos a circuitos
  - Reconocimiento de patrones
  - Técnicas de agrupamiento y clasificación
- Diseño de algoritmos para la extracción de circuitos de interconexión
- Caracterización de grafos de interconexión
- Diseño del clasificador de grafos
- Evaluación del sistema

En primer lugar se estudió a profundidad la teoría de grafos y su relación con la teoría de circuitos, lo que a su vez condujo al análisis del isomorfismo en grafos. Posteriormente se analizó la teoría de clasificadores con el objetivo de aterrizar la búsqueda de patrones a través de esta técnica. Una vez sentadas las bases teóricas, se procedió a la implementación de dos algoritmos de extracción de subcircuitos de interconexión. El primero, llamado algoritmo empírico, se basa en la búsqueda por profundidad para la reconstrucción de los subcircuitos mientras que el algoritmo formal se vale de operaciones de adición, eliminación y división de nodos. En el tercer paso se da lugar la caracterización basada en cualidades propias de la teoría de grafos. A continuación se seleccionan los parámetros del clasificador secuencial para finalizar con la evaluación del sistema.

En el primer capítulo se planteó el problema, su contexto y algunos antecedentes que nutren este proyecto desde diversos enfoques. En el capítulo 2 se desarrollan los fundamentos teóricos sobre los cuales está basado el SCPSI: la teoría de circuitos, de grafos y de clasificadores. El capítulo 3 explica cómo se extrajo la información de interconexión de los diseños electrónicos y su transformación a grafos mediante dos métodos: el empírico y el formal. En el capítulo 4 se lleva a cabo la caracterización de los grafos mientras que en el capítulo 5 se describe el proceso de clasificación. Los resultados de la aplicación del clasificador especificado en el capítulo 5 a los datos generados en el capítulo 4 se muestran en el capítulo 6. Finalmente, en el capítulo 7 se presentan las conclusiones del *Sistema Clasificador de Patrones de Subcircuitos de Interconexión.*

Capítulo 2

Marco teórico

El proyecto está basado en tres áreas de conocimiento particulares: la teoría de grafos, la teoría de circuitos y el reconocimiento de patrones. A continuación se expone la teoría mínima necesaria de cada campo de conocimiento para entender el desarrollo del SCPSI.

2.1. Teoría de grafos

Narsingh Deo en su libro *Graph theory with applications to engineering and computer science* [Deo, 1974] define al *grafo* $\mathscr{G} = (V, E)$ como un conjunto de vértices o nodos $V = \{v_1, v_2, ...\}$ y un conjunto de arístas $E = \{e_1, e_2, ...\}$ donde cada arista e_k se identifica con un par no ordenado de vértices (v_i, v_j). A los vértices v_i, v_j asociados con la arista e_k les llama vértices terminales de e_k. La representación más común de un grafo es mediante un diagrama en el cual los vértices se visualizan como puntos y cada arista como un segmento de línea que une su vértices terminales.

La definición planteada permite que más de una arista esté asociada a un mismo par de vértices (aristas paralelas) y que una arista tenga un mismo vértice como terminales (bucles). Un grafo que no tiene lazos ni aristas paralelas se denomina grafo simple [Deo, 1974].

Al número de aristas incidentes en un vértice v_i, contando a los bucles dos veces, se le llama grado, $d(v_i)$, del vértice v_i. Por ejemplo, el la Figura 2.1 se observa un grafo no simple con cinco nodos, siete aristas, dos aristas paralelas y un bucle. Sus grados son los siguientes: $d(v_1) = d(v_3) = 2$, $d(v_2) = 4$, $d(v_4) = 3$ y $d(v_5) = 3$. El grado máximo del grafo es 4 correspondiente al vértice 2.

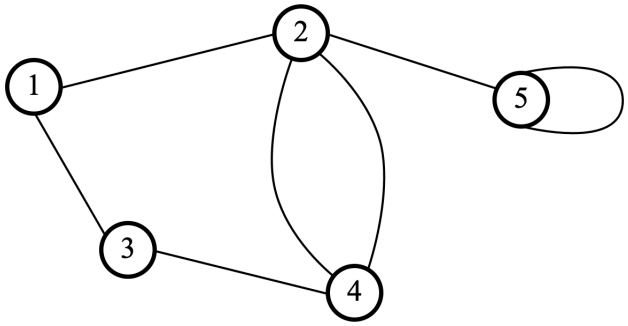


Figura 2.1: Grafo con cinco vértices y siete aristas.

2.1.1. Isomorfismo

Continuando con las definiciones de Deo en [Deo, 1974], dos grafos son equivalentes si tienen un comportamiento idéntico en términos de propiedades de la teoría de grafos; a este tipo de grafos se les conoce como *isomorfos*. Con más precisión, dos grafos $\mathscr{G}$ y $\mathscr{G}'$ son isomorfos si hay una correspondencia uno a uno entre sus vértices y entre sus aristas de tal manera que la relación de incidencia se mantiene.

> *Teóricamente, siempre es posible determinar si dos grafos G_1 y G_2 son isomorfos manteniendo G_1 fijo y reordenando los vértices de G_2 para verificar si sus matrices de adyacencia se vuelven idénticas. Este proceso puede requerir $n!$ reordenaciones y comparaciones, siendo n el número de vértices. Un procedimiento así de ineficiente, ya que el tiempo de ejecución crece factorialmente en función de n, es de uso limitado en problemas prácticos.* [Deo, 1974]

Por la definición de isomorfismo parecería evidente que dos grafos isomorfos deben tener:

1. El mismo número de nodos (v_i)
2. El mismo número de aristas (b_k)
3. Un número igual de nodos con un grado dado.

Sin embargo, estas condiciones no son suficientes. Encontrar un criterio simple y eficiente para la detección del isomorfismo es un problema aún no resuelto de la teoría de grafos. No obstante, existe un procedimiento heurístico [Deo, 1974] para determinar si dos grafos arbitrarios (G_1 y G_2) son isomorfos el cual consiste en demostrar que no lo son haciendo el siguiente tipo de preguntas:

1. ¿$\mathscr{G}_1$ y $\mathscr{G}_2$ tienen el mismo número de nodos?

2. ¿$\mathscr{G}_1$ y $\mathscr{G}_2$ tienen el mismo número de aristas?

3. ¿Es el número de nodos n_i con grado i el mismo en ambos grafos?

4. ¿Son los polinomios característicos de las matrices de adyacencia ($X(\mathscr{G}_1)$ y $X(\mathscr{G}_2)$) iguales?

Si la respuesta a cualquiera de estas preguntas es no, los grafos G_1 y G_2 no son isomorfos; sin embargo, una respuesta positiva a todas las preguntas no garantiza un isomorfismo.

Como explica Leon O. Chua en su libro *Linear and Nonlinear Circuits* [Chua et al., 1987], la teoría de grafos tiene una gama muy amplia de aplicaciones en ingeniería, en ciencias físicas, sociales y biológicas, en lingüística y en muchas otras áreas ya que es posible representar mediante un grafo casi cualquier situación física que involucre objetos discretos y una relación entre ellos.

2.2. Teoría de circuitos

Citando a Charles Alexander y Matthew Sadiku en su libro *Fundamentos de circuitos eléctricos* [Alexander et al., 2013]:

> *En ingeniería eléctrica a menudo interesa comunicar o transferir energía de un punto a otro. Hacerlo requiere una interconexión de dispositivos eléctricos. A tal interconexión se le conoce como circuito eléctrico, y a cada componente del circuito como elemento.*

Deo explica que las propiedades de una red o circuito eléctrico dependen únicamente de dos factores [Deo, 1974]:

1. La naturaleza y el valor de los elementos que forman la red, como resistencias, inductores, transistores, etc.

2. La forma en que estos elementos están conectados entre sí, es decir, la topología de la red.

Ya que solo existe una pequeña cantidad de elementos eléctricos diferentes, las variaciones en las redes son principalmente topológicas. Por ejemplo, la topología de una red se estudia mediante su grafo del circuito dado que las propiedades de interconexión se muestran mejor a través de él. Al dibujar el grafo de una red eléctrica las uniones (nodos) se representan mediante vértices y las ramas (elementos eléctricos) mediante aristas independientemente de la naturaleza y el tamaño de

los mismos. Así, el grafo conserva todas las propiedades de interconexión pero suprime la información sobre los elementos, como se muestra en la Figura 2.2.

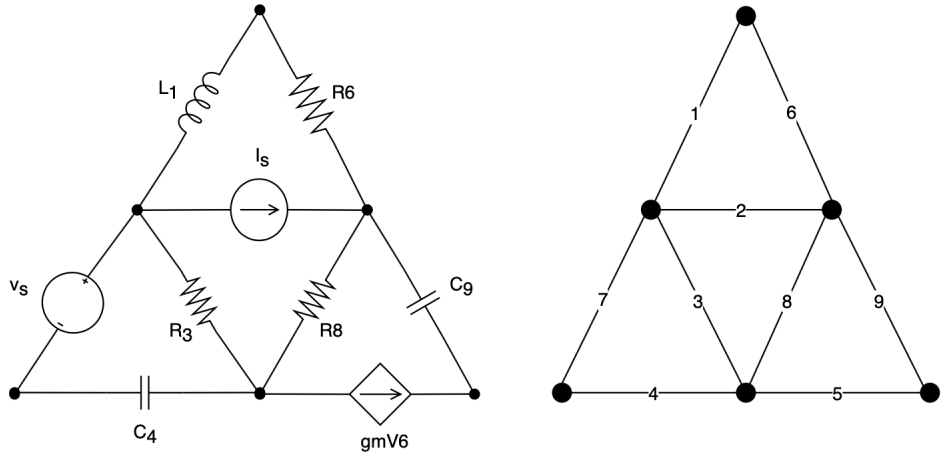


Figura 2.2: Circuito eléctrico y su representación gráfica

2.3. Teoría de clasificadores

Con base en el libro *Pattern Recognition* de Sergios Theodoridis, se puede decir que el objetivo del reconocimiento de patrones es la clasificación de objetos en un número determinado de categorías o clases. Dependiendo la aplicación los objetos a clasificar pueden ser imágenes, señales, cadenas de caracteres, etc. En general, cada uno de los objetos es un *patrón* con cantidades medibles llamadas *características*. Al conjunto de características de un patrón se le conoce como *vector de características* el cual lo identifica de forma única. Pero, ¿cómo se eligen las características y cuál es el mejor número de características a usar? Depende del problema y se trata en las etapas de generación y selección de características. El tipo de nolinearidad que se debe adoptar y el criterio de optimización se eligen en la etapa de diseño del clasificador. Finalmente, ¿cómo se puede evaluar el rendimiento del clasificador? Esa es la tarea de la etapa de evaluación del sistema.

Las etapas del diseño de un sistema de clasificación no son independientes, como lo muestran las flechas de retroalimentación de la Figura 2.3, tomada de [Theodoridis and Koutroumbas, 2009]. Al contrario, están interrelacionadas y dependiendo de los resultados es posible rediseñar las etapas anteriores para mejorar el rendimiento general.

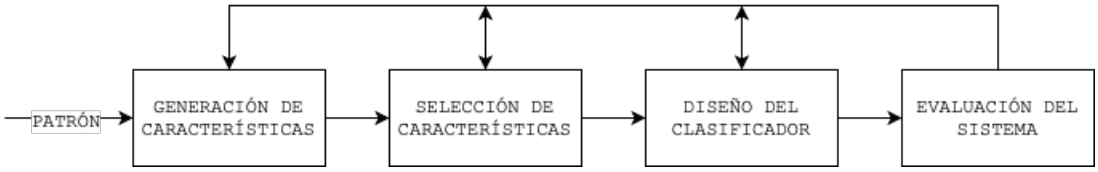


Figura 2.3: Etapas del diseño de un sistema de clasificación

Dado que se desconocen los patrones de los subcircuitos de interconexión, la primera tarea consiste en encontrar agrupamientos de los mismos. Para ello se utilizan técnicas de agrupamiento no supervisadas donde la principal preocupación es descubrir la organización de los patrones en grupos que permitan revelar similitudes y diferencias entre los mismos y derivar conclusiones útiles sobre ellos. Para desarrollar una tarea de agrupación, Theodoridis resalta los siguientes pasos:

1. Seleccionar las características que más información contienen.
2. Determinar la *medida de proximidad* que cuantifica qué tan similares o distintos son dos vectores de características.
3. Determinar el *criterio de agrupamiento* en forma de una función de costo o de alguna otra regla.
4. Seleccionar el algoritmo que descifre la estructura del agrupamiento del conjunto de datos.
5. Validar los resultados del algoritmo para verificar la precisión.
6. Interpretar los resultados para obtener conclusiones acertadas.

Diferentes combinaciones de características, medidas de proximidad, criterios de agrupamiento y algoritmos de agrupamiento pueden llevar a agrupamientos totalmente diferentes.

El agrupamiento es una herramienta con muchas aplicaciones las cuales toman cuatro direcciones principalmente: reducción de datos, generación de hipótesis, prueba de hipótesis y predicción basada en grupos. En la predicción basada en grupos se aplican las técnicas de agrupamiento al conjunto de datos disponibles y los grupos resultantes se caracterizan en función de las características de los patrones por los que se forman. En lo subsecuente, dado un patrón desconocido podemos determinar el grupo al que es más probable que pertenezca [Theodoridis and Koutroumbas, 2009]. Dado el objetivo de la aplicación, la predicción basada en grupos es la más apropiada.

Por otro lado, las características son categorizadas con base en la importancia relativa de los valores que toman [Jain and Dubes, 1988]. Existen cuatro categorías: *nominal*, *ordinal*, *intervalo de escala* y *de relación de escala*. La categoría nominal incluye características cuyos posibles valores codifican estados y la comparación cuantitativa entre dichos valores carece de significado (e.g., en una característica correspondiente al sexo de un individuo los valores posibles son 1 para un hombre y 0 para una mujer). La categoría ordinal incluye características cuyos valores pueden ordenarse significativamente pero la diferencia entre dos valores sucesivos no tiene importancia cuantitativa significativa (e.g, los valores 4, 3, 2, 1 que corresponden a las calificaciones *excelente*, *muy bueno*, *bueno* y *no bueno*). Si para una característica la diferencia entre dos valores es significativa mientras que su relación carece de sentido, entonces se trata de una característica de escala de intervalo (i.e., grados de temperatura, años, millas). Finalmente, si la relación entre dos valores de una característica es significativa, entonces esta es una característica de relación de escala (i. e. peso, altura, precios, edad). Al clasificar los tipos de características en *nominal*, *ordinal*, *de intervalo de escala* y *de relación de escala* es perceptible que cada tipo de característica posee todas las propiedades de los tipos anteriores [Theodoridis and Koutroumbas, 2009]. De forma general, las l características de una muestra x se agrupan en el llamado *vector de características*: $x = [x_1, x_2, ..., x_l]$.

Dado un conjunto de muestras $X = x_1, x_2, x_3, ..., x_N$, Theodoridis y Koutroumbas definen el *m-agrupamiento* de X como la partición de X en m conjuntos $(C_1, ..., C_m)$ de tal forma que las siguientes tres condiciones se cumplen:

- $C_i \neq \emptyset, i = 1, ..., m$
- $\cup_{i=1}^{m} C_i = X$
- $C_i \cap C_j = \emptyset, i \neq j, i, j = 1, ..., m$

Los vectores contenidos en el grupo C_i son más similares entre sí y menos similares a los vectores de características del resto de los grupos. La similitud y la disimilitud son dos tipos de medidas entre vectores. La disimilitud d es una función que indica la distancia entre dos vectores x y y de X. Cumple la desigualdad del triángulo y el mínimo valor de disimilitud posible se alcanza cuando $x = y$. Formalmente, la disimilitud se define como:

$$d : X \times X \to R$$

donde R es el conjunto de los números reales, tal que:

$$\exists d_0 \in R : -\infty < d_0 \leq d(x, y) < +\infty, \forall x, y \in X$$
$$d(x, x) = d_0, \forall x \in X$$
$$d(x, y) = d(y, x), \forall x, y \in X$$
$$d(x, y) = d_0 \text{ si y solo si } x = y$$
$$d(x, z) =\leq d(x, y) + d(y, z), \forall x, y, z \in X$$

Por otro lado, una medida de similitud s en X se define como:

$$s : X \times X \to R$$

tal que:

$$\exists s_0 \in R : -\infty < s(x, y) \leq s_0 < +\infty, \forall x, y \in X$$
$$s(x, x) = s_0, \forall x \in X$$
$$s(x, y) = s(y, x), \forall x, y \in X$$
$$s(x, y) = s_0 \text{ si y solo si } x = y$$
$$s(x, y)s(y, z) \leq [s(x, y) + s(y, z)]s(x, z), \forall x, y, z \in X$$

Por ejemplo, la distancia euclidiana

$$d_2(x, y) = \sqrt{\sum_{i=1}^{l} (x_i - y_i)^2}$$

donde $x, y \in X$ y x_i, y_i son las i-ésimas coordenadas de x y y respectivamente, es una medida de disimilitud en X con $d_0 = 0$; es decir, la mínima distancia posible entre dos vectores de X es 0, al igual que la distancia entre un vector y él mismo. También se observa que $d_2(x, y) = d_2(y, x)$ y que la desigualdad del triángulo se sostiene para este caso en particular. Por lo tanto, la distancia euclidiana es una medida de disimilitud.

En muchos esquemas de agrupamiento, un vector x se asigna a un clúster C tomando en cuenta la proximidad entre x y C, de ahora en adelante designada como $\mathscr{P}(x, C)$. Hay dos opciones generales

en la definición de $\mathscr{P}(x, C)$. En la primera todos los puntos de C contribuyen a $\mathscr{P}(x, C)$ mientras que en la segunda se elige un representante de C y la proximidad entre x y C se mide como la proximidad entre x y el representante de C. Algunos ejemplos de la primera estrategia son:

- La función de proximidad máxima:

$$\mathscr{P}_{max}(x, C) = max_{y \in C} \mathscr{P}(x, y)$$

- La función de proximidad mínima:

$$\mathscr{P}_{min}(x, C) = miny \in C \mathscr{P}(x, y)$$

- La función de proximidad promedio:

$$\mathscr{P}_{avg}(x, C) = \frac{1}{nC} \sum_{y \in C} \mathscr{P}(x, y)$$

  donde n_C es el número de elementos en C.

Cabe resaltar que en los ejemplos mencionados, $\mathscr{P}(x, y)$ puede ser cualquier medida de proximidad entre dos vectores.

Respecto a las categorías de algoritmos de agrupamiento, Theodoridis y Koutroumbas explican: *Los algoritmos de agrupación se pueden ver como esquemas que nos proporcionan agrupaciones sensibles al considerar solo una pequeña fracción del conjunto que contiene todas las particiones posibles de X. El resultado depende del algoritmo específico y los criterios utilizados. Por lo tanto, un algoritmo de agrupación es un procedimiento de aprendizaje que intenta identificar las características específicas de las agrupaciones subyacentes al conjunto de datos.* [Theodoridis and Koutroumbas, 2009]

Las categorías más generales son:

- Algoritmos secuenciales: Son métodos sencillos y rápidos que producen un solo agrupamiento. En la mayoría de ellos, los vectores de características se presentan al algoritmo una o varias veces y el resultado final suele depender del orden en que se le presentaron.

- Algoritmos de agrupamiento jerárquico: Estos esquemas se dividen en algoritmos aglomerantes

y algoritmos divisivos. Los primeros tiene un enfoque *de abajo hacia arriba*, es decir, inicialmente cada muestra comienza en su propio grupo para que posteriormente los pares de grupos se fusionen a medida que sube la jerarquía. Los segundos tienen un enfoque *de arriba a abajo*, es decir, todas las muestras comienzan en un grupo y las divisiones se realizan recursivamente a medida que la jerarquía desciende.

- Algoritmos de agrupamiento basados en la optimización de la función de costo: En esta categoría la *sensibilidad* se cuantifica mediante una función de costo $\boldsymbol{J}$. La mayoría de estos algoritmos utilizan conceptos de cálculo diferencial y producen agrupamientos sucesivos mientras intentan optimizar $\boldsymbol{J}$. Terminan cuando se determina un óptimo local de $\boldsymbol{J}$.

- Otros: Esta última categoría contiene algunas técnicas especiales de agrupamiento que no encajan bien en ninguna de las categorías anteriores. Por ejemplo:

  - Algoritmos de agrupamiento de rama y límite
  - Algoritmos de agrupamiento genético
  - Métodos de relajación estocásticos
  - Algoritmos de agrupamiento de búsqueda de valle
  - Algoritmos de aprendizaje competitivo
  - Algoritmos basados en técnicas de transformación morfológica
  - Algoritmos basados en densidad
  - Algoritmos de agrupamiento subespacial

Particularmente, los algoritmos secuenciales están basados en el *esquema algorítmico secuencial básico*. La idea general del algoritmo es que a medida que se considera cada vector nuevo, se asigna a un *cluster* existente o a uno recién creado, dependiendo de su distancia de los ya formados. Sea $d(x, C)$ la distancia (o disimilitud) entre un vector de características x y un cluster C y m el número de agrupaciones que el algoritmo ha creado hasta ahora. Entonces el esquema algorítmico puede ser enunciado como en la Figura 2.4, tomada del libro [Theodoridis and Koutroumbas, 2009].

- $m = 1$
- $C_m = \{\boldsymbol{x}_1\}$
- For $i = 2$ to N
  - Find C_k: $d(\boldsymbol{x}_i, C_k) = \min_{1 \le j \le m} d(\boldsymbol{x}_i, C_j)$.
  - If $(d(\boldsymbol{x}_i, C_k) > \Theta)$ AND $(m < q)$ then
    - $m = m + 1$
    - $C_m = \{\boldsymbol{x}_i\}$
  - Else
    - $C_k = C_k \cup \{\boldsymbol{x}_i\}$
    - Where necessary, update representatives[2]
  - End {if}
- End {For}

Figura 2.4: Esquema algorítmico secuencial básico

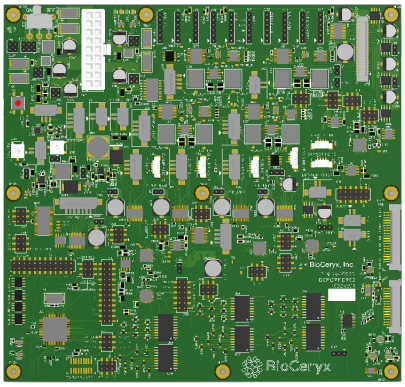
Los parámetros definidos por el usuario requeridos por el esquema algorítmico son el umbral de disimilitud Θ y el número máximo permitido de agrupaciones q. Diferentes elecciones de $d(x, C)$ conducen a diferentes algoritmos; un ejemplo de d es la distancia euclidiana explicada previamente. El orden en el que se presentan los vectores al esquema juega un papel importante en los resultados del agrupamiento. Un orden de presentación diferente puede llevar a resultados de agrupación totalmente diferentes, en términos del número de grupos así como los grupos mismos. Otro factor importante que afecta el resultado del algoritmo es la elección del umbral Θ. Este valor afecta directamente el número de agrupaciones formadas. Si es demasiado pequeño, se crearán agrupaciones innecesarias; en cambio, si es demasiado grande, se creará un número de clusters más pequeño que el adecuado. Por otro lado, no restringir el número máximo de grupos es permitir que el algoritmo decida el número apropiado de clusters. Sin embargo, la restricción de q es necesaria cuando se trata de implementaciones donde los recursos computacionales disponibles son limitados.

A partir de lo expuesto respecto a las teorías de grafos y de circuitos, un diseño electrónico se puede descomponer en un conjunto de subgrafos y por tanto la tarea de diseñar un clasificador de circuitos se puede ver como la tarea de clasificar grafos con algoritmos no supervisados. En la clasificación no supervisada, la etiqueta de la clase de los patrones de entrenamiento no es conocida. Por lo tanto, el objetivo principal es descubrir su organización en grupos sensibles, lo que permite exhibir similitudes y diferencias entre patrones y derivar conclusiones útiles al respecto.

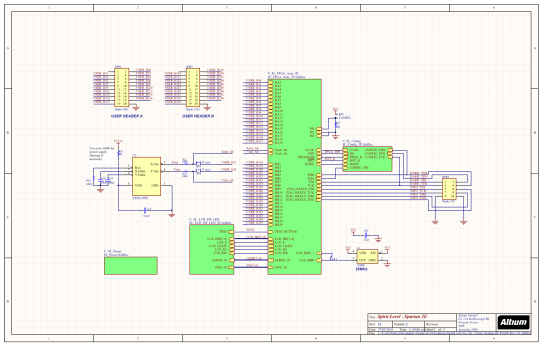
Capítulo 3

Extracción de los circuitos de interconexión

Hoy en día el ingeniero electrónico se apoya en herramientas de software para diseñar esquemáticos y tarjetas de circuito impreso; algunos ejemplos de este tipo de herramientas son *Altium Designer*, *EAGLE* y *OrCAD*. Los diseños de la empresa *Powerbeam* están desarrollados en la plataforma *Altium Designer*, motivo por el cual la obtención de la información es específica para dicho software.



(a) Tarjeta de circuito impreso



(b) Un esquemático

Figura 3.1: Proyecto BioCeryx

3.1. Obtención y transformación de la información

Como explica la documentación oficial de Altium [Altium-LLC, 2017], es posible escribir secuencias de comandos (scripts) en alguno de los lenguajes de script (DelphiScript, EnableBasic,

VB Script o JavaScript) para automatizar tareas repetitivas o mejorar alguna función en Altium Designer. La automatización de comandos resulta conveniente pues dependiendo de la complejidad del proyecto puede estar compuesto de uno o hasta decenas de esquemáticos.

3.1.1. Fuentes de información

Dada la condición de gran escala de los proyectos analizados, se programaron tres scripts con el objetivo de recorrer los esquemáticos de los proyectos para extraer su información y almacenarla en archivos de texto plano. A continuación se precisa el tipo de información contenida en cada archivo resultante. Los cuadros de texto 3.1, 3.2 y 3.3 son una muestra de cada tipo.

1. Archivo de voltajes y tierras: Precisa los nombres de los nodos de voltaje y tierra.

2. Archivo de pines: Especifica el nombre y número de pines de cada componente del esquemático y de los nodos de tierra y voltaje.

3. Archivo de conexiones: Relaciona los componentes electrónicos con los nodos a los que están conectados. Su propósito fundamental es transmitir información de conectividad.

Código 3.1: Voltajes y tierras

```
G GND
V VCC3V3
G GND
G GND
V VADJ_FPGA
V VCC2V5
V VCCAUX_IO
G XADC_AGND
V VCC2V5
V VCC3V3
V VCC2V5_FPGA
V VCC2V5_FPGA
G GND
```

Código 3.2: Pines

```
Designator: R330
N: 2 D: 2
N: 1 D: 1
Designator: R296
N: 2 D: 2
N: 1 D: 1
Designator: U48
N: DIR D: 5
N: VCCB D: 6
N: B D: 4
N: VCCA D: 1
N: GND D: 2
N: A D: 3
```

Código 3.3: Conexiones

```
NetC177_1
C177.1 R222.1 R223.2 U33.1
NetC590_1
C590.1 U71.5
NetC591_2
C591.2 R425.2 R426.1 U71.6
NetDS24_1
DS24.1 Q16.3
NetDS24_2
DS24.2 R379.2
NetR326_1
R326.1 R331.2 U47.4
```

3.1.2. Transformación de la información

Las tres fuentes de información se introducen en programas informáticos que analizan cadenas de símbolos de acuerdo a ciertas reglas de gramática mejor conocidos como analizadores sintácticos o parseadores. El análisis sintáctico convierte el texto de entrada en otras estructuras más útiles para el análisis posterior [Grune and Jacobs, 2008]. En este caso, el parseador está programado en el lenguaje Python, lo que implica que el texto es transformado en estructuras de datos de dicho lenguaje.

Archivo de voltajes y tierras El archivo de voltajes y tierras es el primero en ser transformado. La estructura de cada línea de texto se muestra en el código 3.4 mientras que en el código 3.5 se observan un ejemplo de la misma.

Código 3.4: Estructura

```
tipo-nodo nombre-nodo
tipo-nodo nombre-nodo
tipo-nodo nombre-nodo
tipo-nodo nombre-nodo
...
```

Código 3.5: Ejemplo

```
V VCC3V3
V VADJ-FPGA
V VCCAUX-IO
G GND
...
```

A partir del archivo se obtienen dos listas, una con los nombres de los nodos de voltajes y otra con los nombres de los nodos de tierra, Figuras 3.2a y 3.2b. Para ello, cada línea del archivo de texto se separa en dos partes: cabeza y cola; la cabeza contiene el tipo de nodo (`V` para voltajes o `G` para tierras) y la cola el nombre del mismo. Dependiendo el tipo de nodo se almacena el nombre en la lista correspondiente, como se muestra en la Figura 3.3.

Index ▲	Type	Size	Value
0	str	1	VCC3V3
1	str	1	VADJ_FPGA
2	str	1	VCC2V5
3	str	1	VCCAUX_IO
4	str	1	VCC2V5_FPGA
5	str	1	VCCAUX
6	str	1	VADJ
7	str	1	VCC1V5_FPGA
8	str	1	VTTVREF
9	str	1	MGTAVTT
10	str	1	VCCINT_FPGA
11	str	1	MGTVCCAUX
12	str	1	MGTAVCC
13	str	1	VCCBRAM

(a) Lista de voltajes

Index ▲	Type	Size	Value
0	str	1	GND
1	str	1	XADC_AGND
2	str	1	VIDEO_GND
3	str	1	INPUT_GND
4	str	1	DGND1_9248
5	str	1	AGND1_9248
6	str	1	AGND_VCCINT
7	str	1	ANGND_VADJ
8	str	1	AGND2_9248
9	str	1	DGND2_9248
10	str	1	AGND2
11	str	1	AGND3_9248
12	str	1	DGND3_9248
13	str	1	AGND_MGTVCCAUX

(b) Lista de tierras

Figura 3.2: Listas en Python

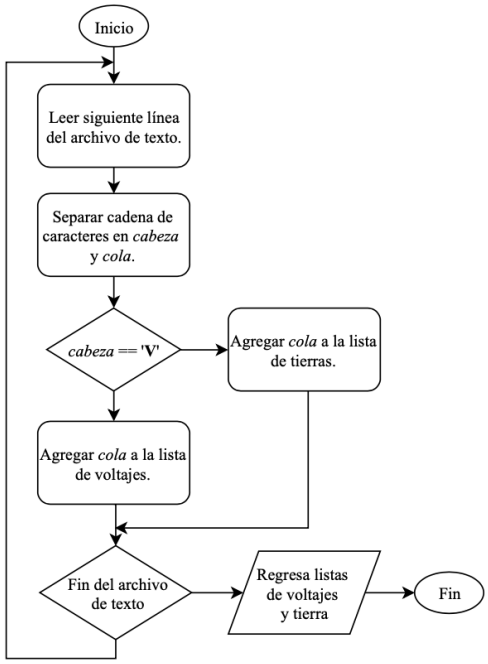


Figura 3.3: Algoritmo para procesamiento de archivo de voltajes y tierras

Archivo de pines La lista de pines es la segunda en ser transformada pues se complementa con las listas de voltajes y tierras para generar diccionarios de componentes. La estructura del archivo de texto y un ejemplo se muestra en los códigos 3.6 y 3.7, respectivamente.

Código 3.6: Estructura

```
Designator: componente-1
N: nombre-pin-1 D: numero-pin-1
N: nombre-pin-2 D: numero-pin-2
Designator: componente-2
N: nombre-pin-1 D: numero-pin-1
N: nombre-pin-2 D: numero-pin-2
Designator: componente-3
N: nombre-pin-1 D: numero-pin-1
N: nombre-pin-2 D: numero-pin-2
N: nombre-pin-3 D: numero-pin-3
N: nombre-pin-4 D: numero-pin-4
N: nombre-pin-5 D: numero-pin-5
N: nombre-pin-6 D: numero-pin-6
...
```

Código 3.7: Ejemplo

```
Designator: R330
N: 2 D: 2
N: 1 D: 1
Designator: R296
N: 2 D: 2
N: 1 D: 1
Designator: U48
N: DIR D: 5
N: VCCB D: 6
N: B D: 4
N: VCCA D: 1
N: GND D: 2
N: A D: 3
...
```

Si la línea a procesar contiene la palabra `Designator` entonces es dividida en dos partes: cabeza y cola. Si la cola no es un nodo de tierra ni de voltaje, se agrega como una llave nueva al diccionario general de componentes, si sí lo es, se ignora. Si la línea no contiene la palabra `Designator`, se agrega el nombre del pin (etiquetado con *N:*) a la lista de valores del último componente del diccionario. Los ejemplos del código 3.7 se transforman en entradas del diccionario como se muestra en la Figura 3.4.

Key ▲	Type	Size	Value
R296	list	2	['2', '1']
R330	list	2	['2', '1']
U48	list	6	['DIR', 'VCCB', 'B', 'VCCA', 'GND', 'A']

Figura 3.4: Diccionario general de componentes en Python

Finalmente, cada par *llave-valor* del diccionario general de componentes es valorado para ser clasificado en el grupo de componentes correspondiente. Primero, la llave es evaluada con expresiones regulares para determinar el tipo de componente a partir del nombre del mismo. Si la evaluación

es positiva, se contabilizan el número de elementos en el valor, el cual corresponde al número de terminales del componente, para precisar que se trata de un componente de dos terminales. El código 3.8 ejemplifica la valoración para el caso específico de los resistores. En la linea 1 se lleva a cabo la búsqueda de la expresión regular en la llave y en la linea 2 la comparación del número de terminales del componente. En caso de aprobar ambas condiciones, el componente se agrega al diccionario de resistores; en caso de aprobar únicamente la primera condición, el componente entra en la categoría *otros*; en caso de no aprobar ninguna, pasa a ser evaluada en las expresiones regulares del resto de los componentes relevantes (capacitores, inductores y diodos). El diagrama de la Figura 3.6 sintetiza el algoritmo para el procesamiento de listas de terminales; la abreviación *e.r.* denota expresión regular.

Código 3.8: Validación de resistores

```
if (re.match(^[R][1-9],key)):
        if (len(value) == 2):
                resistors[key] = value
        else:
                other[key] = value
```

Key ▲	Type	Size	Value
R1	list	2	['2', '1']
R10	list	2	['2', '1']
R103	list	2	['2', '1']
R104	list	2	['2', '1']
R105	list	2	['2', '1']
R106	list	2	['2', '1']
R107	list	2	['2', '1']
R108	list	2	['2', '1']
R109	list	2	['2', '1']
R11	list	2	['2', '1']

(a) Resistores

Key ▲	Type	Size	Value
C1	list	2	['CAP2', 'CAP1']
C10	list	2	['CAP2', 'CAP1']
C100	list	2	['CAP2', 'CAP1']
C101	list	2	['CAP2', 'CAP1']
C102	list	2	['CAP2', 'CAP1']
C103	list	2	['1', '2']
C104	list	2	['CAP2', 'CAP1']
C105	list	2	['CAP2', 'CAP1']
C106	list	2	['CAP2', 'CAP1']
C107	list	2	['CAP2', 'CAP1']

(b) Capacitores

Key ▲	Type	Size	Value
L1	list	2	['1', '2']
L14	list	2	['1', '2']
L17	list	2	['1', '2']
L19	list	2	['1', '2']
L2	list	2	['1', '2']
L3	list	2	['1', '2']
L30	list	2	['1', '2']
L31	list	2	['1', '2']
L32	list	2	['1', '2']
L33	list	2	['1', '2']

(c) Inductores

Key ▲	Type	Size	Value
D12	list	2	['ANODE', 'CATHODE']
D13	list	2	['ANODE', 'CATHODE']
D14	list	2	['ANODE', 'CATHODE']
D15	list	2	['ANODE', 'CATHODE']
D16	list	2	['ANODE', 'CATHODE']
D17	list	2	['ANODE', 'CATHODE']
D18	list	2	['ANODE', 'CATHODE']
D19	list	2	['ANODE', 'CATHODE']
D20	list	2	['ANODE', 'CATHODE']
D21	list	2	['ANODE', 'CATHODE']

(d) Diodos

Key ▲	Type	Size	Value
J56	list	2	['P1', 'P2']
J6	list	9	['GND', 'SHLD4', 'VBUS', 'D_N', 'D_P', 'SHLD1', 'SHLD2', 'SHLD3', 'ID' ...
J60	list	14	['VSENSE', 'VREF', 'VPRG', 'TCK', 'TMS', 'TDI', 'INIT', 'TDO', 'PGND', ...
J61	list	3	['P1', 'P2', 'P3']
J62	list	2	['P1', 'P2']
J63	list	2	['P1', 'P2']
J65	list	2	['P1', 'P2']
J7	list	9	['P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9']
J8	list	5	['SIG', 'GND1', 'GND4', 'GND3', 'GND2']
J9	list	5	['SIG', 'GND1', 'GND4', 'GND3', 'GND2']

(e) Otros componentes

Figura 3.5: Diccionarios de componentes en Python

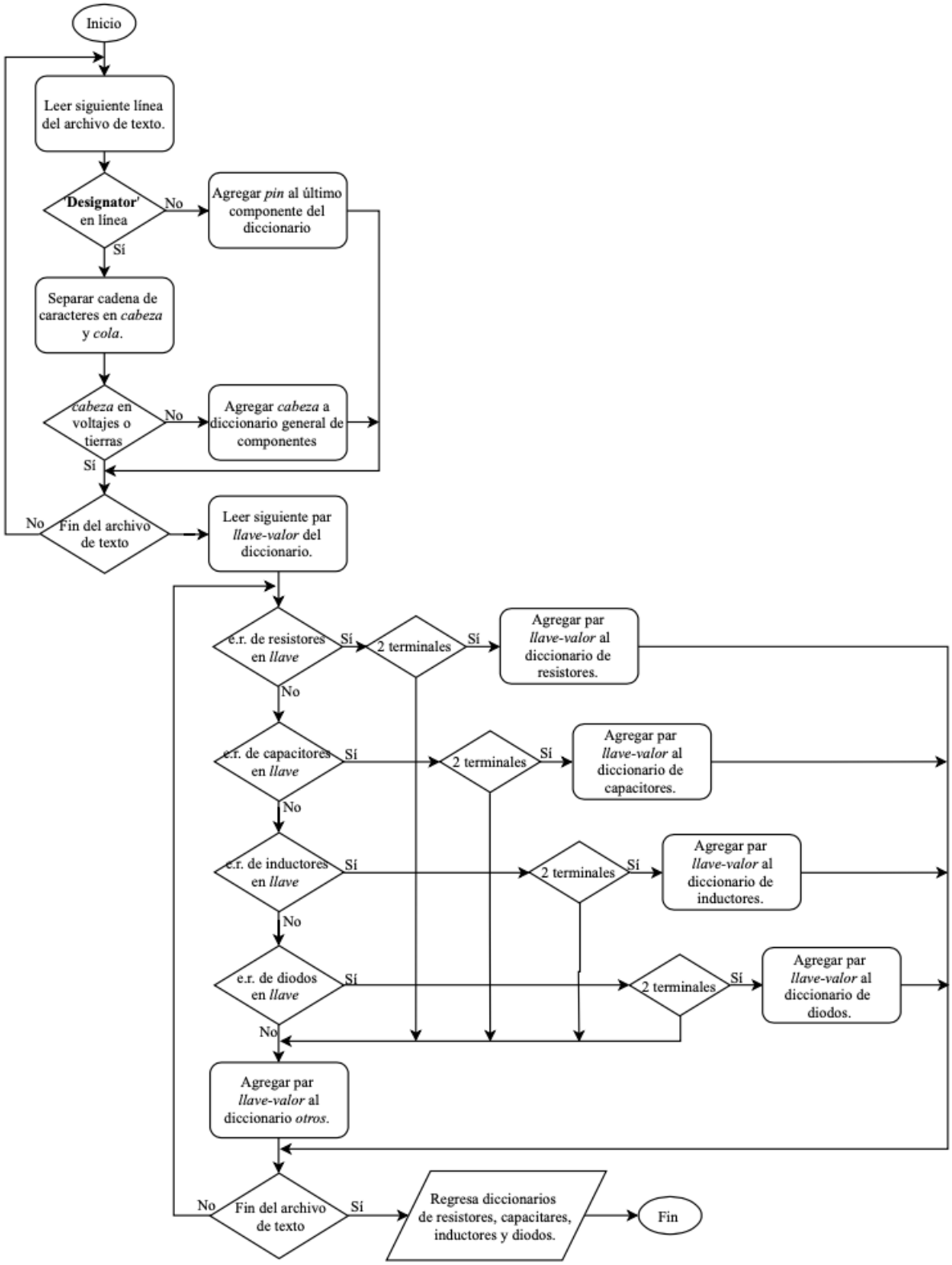


Figura 3.6: Algoritmo para procesamiento de archivo de pines

Archivo de conexiones La unidad fundamental de la estructura que se muestra en el código 3.9 está compuesta por dos líneas, la primera contiene el nombre del nodo y la segunda los componentes conectados a él, especificando el número de terminal. El código 3.10 es un ejemplo de ello.

El primer paso del algoritmo es convertir el archivo de texto en dos listas de cadenas de caracteres donde una contiene únicamente los nombres de los nodos y la otra conserva tanto los nombres de los nodos como los componentes adheridos a ellos. En lo que resta de la sección, a la primera se le denominará *lista de nodos* y a la segunda *lista de conexiones*. Dado que la unidad fundamental del archivo consta únicamente de dos líneas, es posible diferenciar aquella que contiene el nombre del nodo (cuyo índice siempre es par) de aquella que contiene los componentes (cuyo índice siempre es impar) mediante la operación `módulo 2 (%2)`. Siguiendo con el ejemplo del código 3.10, se obtiene la lista de nodos de la Figura 3.7a al únicamente agregar las cadenas de caracteres de las líneas cuyo resultado de la operación `índice%2` es 0. La lista de conexiones contiene exactamente la misma información que el archivo, la diferencia es que cada línea ahora es un elemento tipo *string*, Figura 3.7b.

Código 3.9: Estructura

```
nodo1
comp1.designator comp2.designator comp3.designator
nodo2
comp1.designator comp2.designator comp3.designator
nodo3
comp1.designator comp2.designator comp3.designator
nodo4
comp1.designator comp2.designator comp3.designator
...
```

Código 3.10: Ejemplo

```
0  NetC177_1
1  C177.1 R222.1 R223.2 U33.1
2  NetC590_1
3  C590.1 U71.5
4  NetC591_2
5  C591.2 R425.2 R426.1 U71.6
6  NetDS24_1
7  DS24.1 Q16.3
8  NetDS24_2
9  DS24.2 R379.2
10 ...
```

La lista de conexiones se transforma en un diccionario agregando cada nodo como una llave con los componentes adheridos a él como valores de la misma. Además, cada componente de dos terminales se añade como llave y los nodos a los que se encuentra conectado se le agregan como valores. De esta forma se conserva toda la información de conectividad del diseño original. El algoritmo de la Figura 3.9 muestra el procedimiento así como la Figura 3.8 es el ejemplo del diccionario de conexiones de un proyecto.

Index ▲	Type	Size	Value
0	str	1	NetC177_1
1	str	1	NetC590_1
2	str	1	NetC591_2
3	str	1	NetDS24_1
4	str	1	NetDS24_2
5	str	1	NetR326_1
6	str	1	NetR403_1
7	str	1	NetR428_2
8	str	1	VCC12_P_PWRCTL3_VCC3A
9	str	1	PWRCTL3_VCC3_SENSE_P
10	str	1	PWRCTL3_VCC3_SENSE_N

(a) Nodos

Index ▲	Type	Size	Value
0	str	1	NetC177_1
1	str	1	C177.1 R222.1 R223.2 U33.1
2	str	1	NetC590_1
3	str	1	C590.1 U71.5
4	str	1	NetC591_2
5	str	1	C591.2 R425.2 R426.1 U71.6
6	str	1	NetDS24_1
7	str	1	DS24.1 Q16.3
8	str	1	NetDS24_2
9	str	1	DS24.2 R379.2
10	str	1	NetR326_1

(b) Conexiones

Figura 3.7: Listas de nodos y conexiones en Python

Key	Type	Size	Value
AGND2_9248	list	2	['C113', 'C149', 'C513', 'C520', 'C528', 'C533', 'C534', 'C535', 'C536 ...
AGND3_9248	list	2	['C606', 'C607', 'C614', 'C671', 'C674', 'C675', 'C676', 'C724', 'R475 ...
AGND_MGTVCCAUX	list	2	['U99', 'Z24']
AGND_VCCINT	list	2	['U21', 'Z3']
ANGD_VADJ	list	2	['U17', 'Z5']
BRT	list	2	['J5', 'R9', 'R77']
BRT-PWM	list	2	['R7', 'R77', 'U1']
BRT-V	list	2	['R9', 'R133', 'R135']
C1	list	2	['CMLOUT_TP_N', 'CMLOUT_P']
C10	list	2	['NetC10_2', 'GND']
C100	list	2	['PWRCTL1_VCC3V3A', 'AGND1_9248']
C101	list	2	['NetC11_2', 'GND']
C102	list	2	['VADJ_FILT', 'GND']

Figura 3.8: Diccionario de conexiones

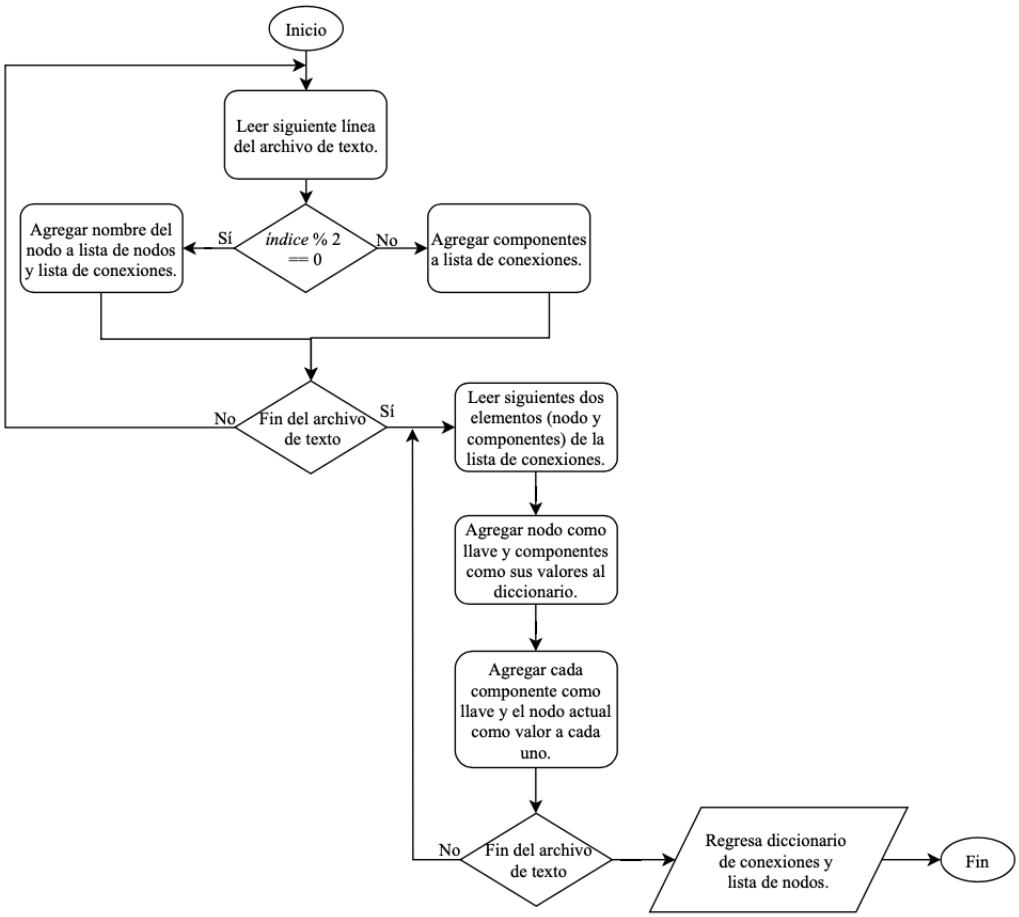


Figura 3.9: Algoritmo para procesamiento de archivo de conexiones

3.2. Algoritmo empírico

Como se explicó en la sección *Teoría de circuitos* del marco teórico, para que el grafo conserve todas las propiedades de interconexión del circuito, los nodos se convierten en vértices y los componentes en aristas. A partir del proceso de transformación de la información explicado en la sección anterior se obtienen dos diccionarios los cuales contienen la información necesaria para la construcción de los grafos de los circuitos de interconexión. El diccionario de conexiones engloba todos los vínculos del circuito mientras que el diccionario del proyecto es una recopilación de los diccionarios de componentes (D_r: diccionario de resistores, D_c: diccionario de capacitores, D_l: diccionario de inductores y D_o: diccionario de otros) y las listas nodos (L_v: lista de voltajes, L_t: lista de tierras y L_n: lista de nodos).

Antes de ahondar en el algoritmo empírico para la construcción de los grafos es importante hablar sobre la herramienta computacional sobre la cual se fundamentan. *Networkx* es una librería de Python concebida para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas [NetworkX-Developers, 2019a]. Entre las características que hacen de la librería una herramienta sumamente útil para el desarrollo del proyecto destacan las siguientes:

- Estructuras de datos para grafos, dígrafos y multigrafos.
- Muchos algoritmos estándar de teoría de grafos.
- Medidas de estructura y análisis de red.
- Los nodos pueden ser cualquier tipo de dato (p.e., texto, imágenes, archivos XML).
- Las aristas pueden contener información arbitraria (p.e., pesos, series de tiempo).

Dado que los circuitos electrónicos pueden tener componentes en configuración paralela, la estructura de datos idónea para contener objetos de este tipo es *MultiGraph*. *MultiGraph* es una clase de grafos no dirigidos que puede almacenar aristas múltiples entre dos nodos; además, cada arista puede contener atributos independientes [Networkx, 2019].

A continuación se describen los pasos para construir cada circuito intermedio. Mientras haya nodos en la lista de nodos del proyecto (L_n):

1. Crear un multigrafo vacío (G_a).

2. Crear una lista temporal de nodos (N_t) cuyo único elemento sea el siguiente nodo de L_n.

3. Mientras haya nodos en N_t:

   3.1. Agregar el nodo actual (n_a) a G.

   3.2. Obtener la lista de componentes L_c conectados a n_a.

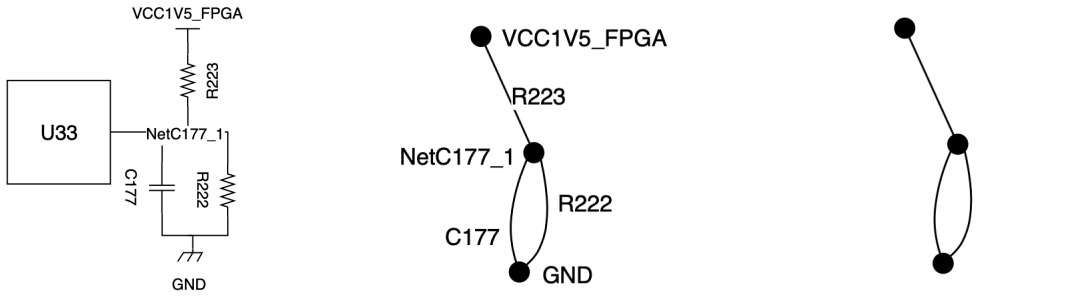
   3.3. Para cada componente c_i de L_c:

   - Si $c_i \in D_o$, no agregar a G_a pues el componente es complejo (es decir, tiene más de dos terminales).
   - Si $c_i \notin D_o$:
     - Obtener el otro nodo (n_o) de c_i (recordar que únicamente se trabaja con componentes de dos terminales).
     - Agregar n_o a G_a.
     - Agregar una arista entre n_a y n_o cuyo atributo *tipo* sea el tipo de componente de c_i.
     - Si $n_o \notin L_v$ y $n_o \notin L_t$, agregar n_o a N_t (los nodos de voltaje y tierra no se añaden a N_t ya que se consideran nodos terminales del circuito).

   3.4. Eliminar n_a de N_t y de L_n.

4. Agregar G_a a la lista de grafos de interconexión L_G.

La Figura 3.10 muestra un circuito de interconexión tomado de uno de los proyectos analizados. El circuito corresponde a los componentes circundantes al primer nodo de L_n y está formado por dos resistencias, un capacitor, un componente complejo y tres nodos. Para la construcción del grafo del circuito primero se crea el multigrafo actual G_a, en segundo lugar la lista temporal de nodos $N_t = [NetC177_1]$ y finalmente se agrega $NetC177_1$ a G_a. La lista de componentes adjacentes al nodo actual es $L_c(n_a) = L_c(NetC177_1) = [U33, R223, R222, C177]$. *U33* es un componente complejo por lo que no se agrega a G_a, tanto *R223* como *R222* y *C177* son componentes simples por lo que se obtienen los nodos del otro lado de los mismo, los cuales son $VCC1V5_FPGA$, *GND* y *GND* respectivamente, y se agregan a G_a. Entre ellos y el nodo $NetC177_1$ se agrega una arista cuyo atributo *tipo* corresponde al tipo de componente que los une, en este caso *resistor*, *resistor* y *capacitor*, respectivamente. La representación final de G_a se muestra en la Figura 3.10c. G_a se agrega a G y el proceso se repite hasta que L_n se vacía.



(a) Circuito de interconexión (b) Grafo intermedio (c) Grafo de interconexión

Figura 3.10: De circuito de interconexión real a grafo

La Figura 3.11 muestra algunos grafos de interconexión resultantes de aplicar el algoritmo empírico a los proyectos de hardware. Los grafos tienen diversas topologías al igual que diferente número de elementos.

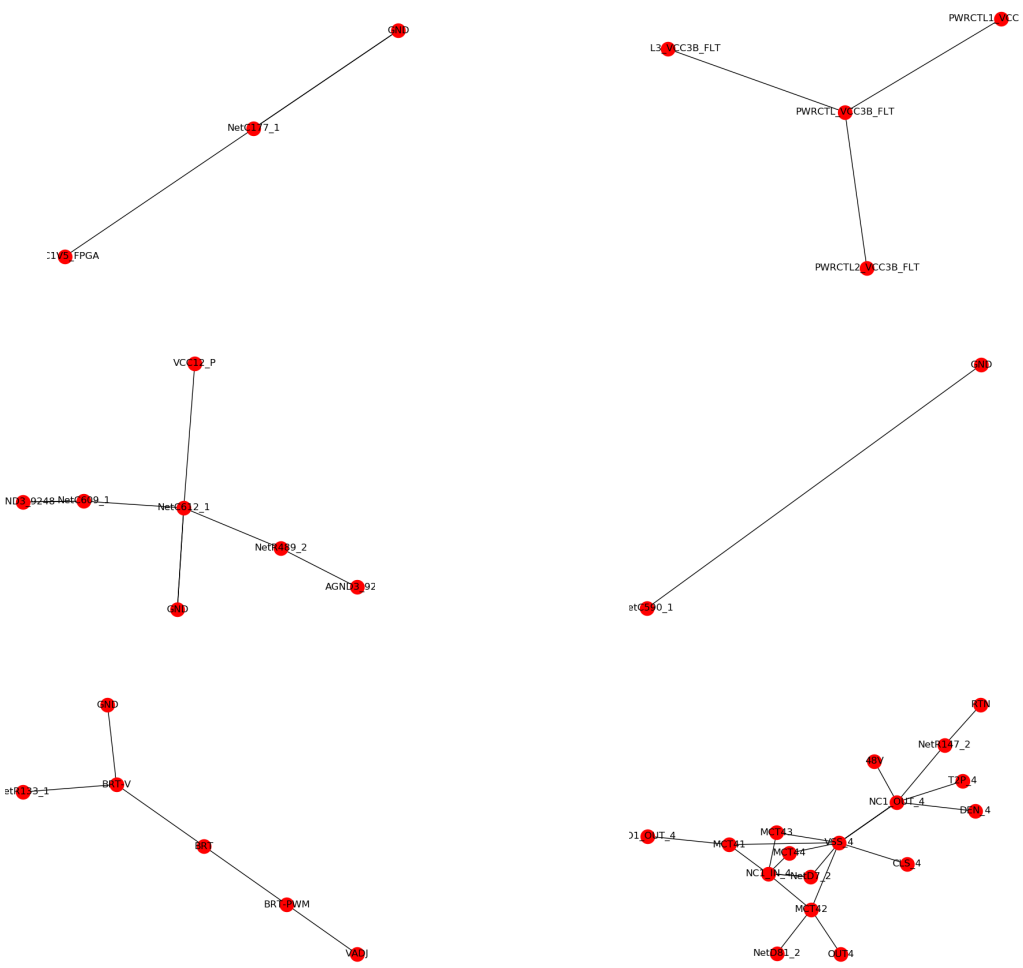


Figura 3.11: Algunos resultados del algoritmo empírico.

3.3. Algoritmo formal

Al igual que en algoritmo empírico, el desarrollo del algoritmo formal se hace sobre la librería *Networkx* de Python, descrita en la sección 3.2. El algoritmo formal se puede describir en términos de capas; es decir, cada paso agrega nodos y aristas ordenadamente de tal forma que el circuito entero es reconstruido como un grafo antes de descomponerlo en los subgrafos de interconexión. A partir de la creación del multigrafo vacío que contendrá todo el circuito (G_c), los pasos mencionados se precisan a continuación:

1. Capa uno: Agregar a G_c cada componente complejo en D_o como un nodo.

2. Capa dos: Agregar a G_c cada terminal de cada componente complejo (también almacenadas en D_o) como un nodo y conectar cada componente complejo con sus terminales a través de una arista.

3. Capa tres: Agregar los nodos intermedios, de voltaje (L_v) y de tierra (L_t) a G_c.

4. Capa cuatro: Agregar las conexiones entre los nodos de G_c. No todas las conexiones son necesariamente componentes. Puede haber conexiones directas entre terminales de componentes complejos o entre terminales y nodos de voltaje y tierra. En esta capa se tiene la reconstrucción completa del circuito.

5. Capa cinco: Eliminar de G_c únicamente los nodos que representan componentes complejos, más no los nodos que representan las terminales de los componentes complejos. Como consecuencia de eliminar el nodo central de un componente complejo también se eliminan las aristas incidentes (que no son componentes) que lo unían con los nodos de sus terminales.

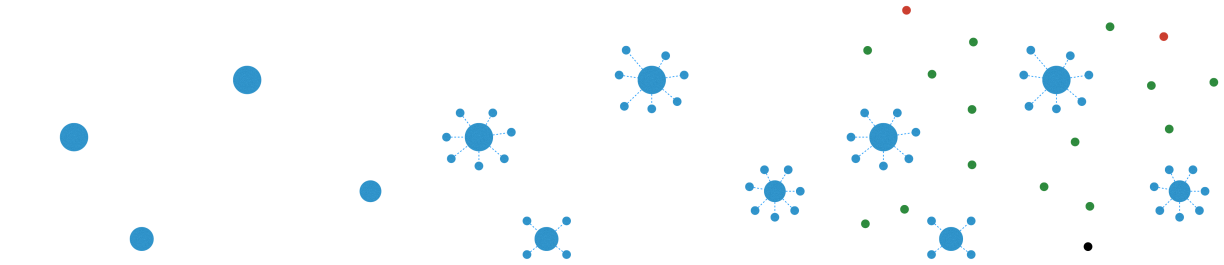
6. Capa seis: Eliminar de G_c las aristas que no son componentes.

7. Capa siete: Eliminar de G_c los nodos flotantes resultado de eliminar aristas que no eran componentes.

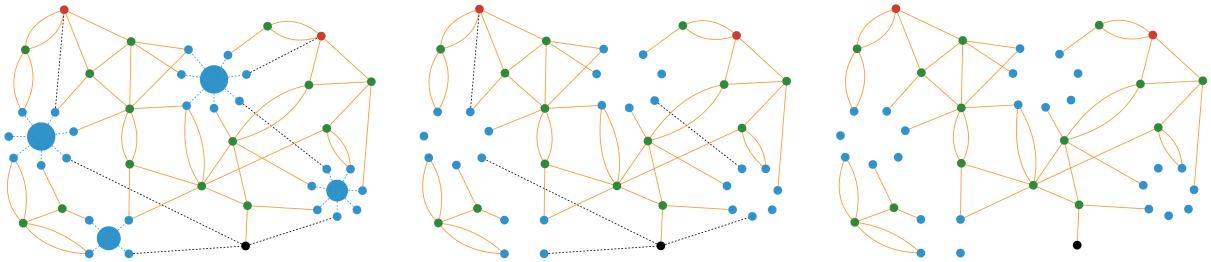
8. Capa ocho: Dividir los nodos de voltaje y tierra de G_c en función del número de aristas incidentes a cada uno.

9. Agregar cada componente de G_c a la lista de grafos de interconexión L_G.

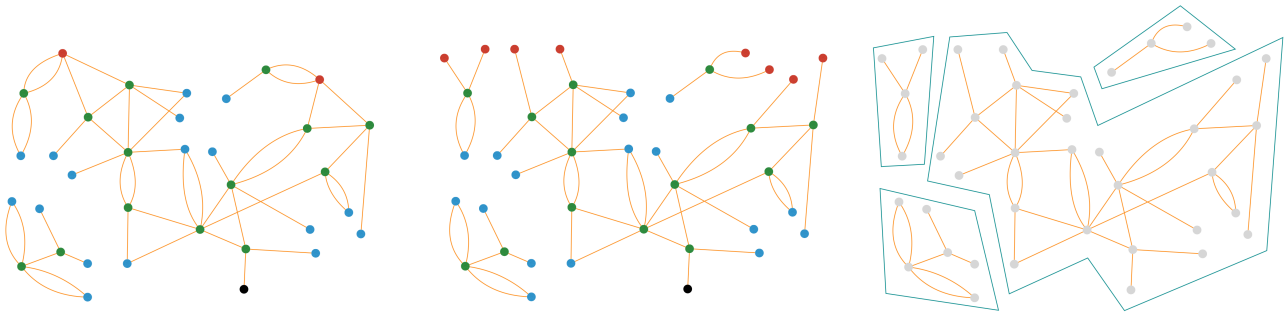
La Figura 3.12 ejemplifica el desarrollo del algoritmo. Los nodos azules de la capa uno representan los núcleos de cuatro componentes complejos. En la capa dos, a los núcleos se les agregan sus terminales en forma de pequeños nodos azules. Las terminales están conectadas a sus respectivos núcleos a través de aristas sin componentes representadas por líneas punteadas del mismo color. En la capa tres se agregan tres tipos de nodos: de voltaje (de color rojo), de tierra (de color negro) e intermedios (verdes). En la capa cuatro se conectan los nodos agregados en la capa tres entre sí y con las terminales de los componentes complejos a través de dos tipos de aristas: con componentes (líneas naranjas) y sin componentes (líneas punteadas negras). En esta capa el diseño electrónico ha sido reconstruido en su totalidad y a partir de este punto comienza la eliminación de elementos. La capa cinco muestra el circuito sin los núcleos de los componentes complejos. Recordemos que, como se mencionó en la sección 2.1 del marco teórico, eliminar un nodo implica eliminar las aristas incidentes al mismo, por consiguiente, eliminar los núcleos de los componentes complejos conlleva la eliminación de las aristas que los conectan con sus terminales. La capa seis elimina las aristas que conectan nodos a través de pistas y no de componentes, lo que genera nodos flotantes, mismos que son eliminados en la capa siete. Finalmente, en la capa ocho se dividen los nodos de voltaje y tierra lo que da origen a los grafos de interconexión.



(a) Capa 1 (b) Capa 2 (c) Capa 3



(d) Capa 4 (e) Capa 5 (f) Capa 6



(g) Capa 7 (h) Capa 8 (i) Grafos de interconexión

Figura 3.12: Ejemplificación del algoritmo formal

La Figura 3.13 muestra algunos grafos de interconexión resultantes de aplicar el algoritmo formal a los proyectos de hardware.

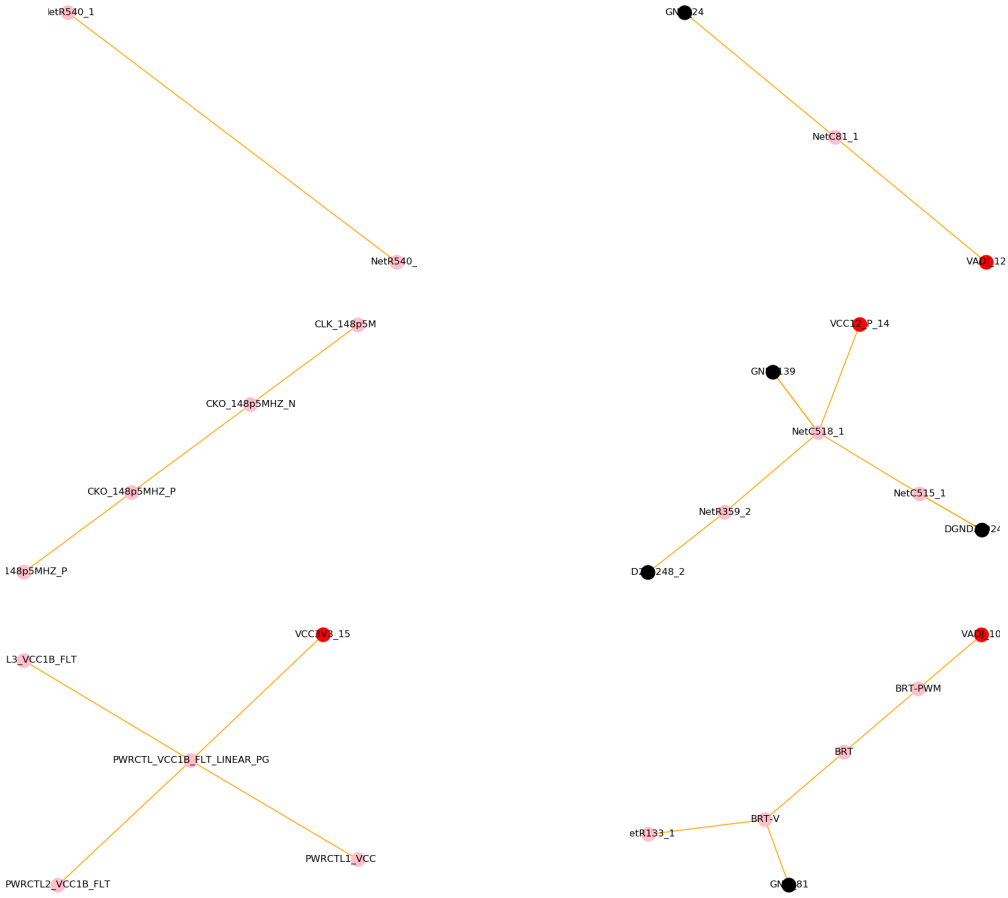


Figura 3.13: Algunos resultados del algoritmo formal

Capítulo 4

Diseño del clasificador

Como se explicó en el marco teórico, los algoritmos secuenciales están basados en el esquema secuencial básico especificado en la Figura 2.4. Dicho esquema es implementado como una función cuyos parámetros son el *Dataframe* que contiene los vectores de características x_i de los grafos de interconexión, el umbral de disimilitud Θ, el número máximo de agrupaciones q y un vector de pesos que pondera la relevancia de las características.

En particular, el umbral Θ está estrechamente relacionado con la medida de disimilitud adoptada. En esta implementación del esquema se utilizan en conjunto la distancia euclidiana y la función de proximidad mínima entre un punto y un conjunto, ambas definidas en la sección 2.3, para determinar la separación de un vector de características nuevo respecto a un cluster en particular. Una vez calculada, esta medida es comparada con Θ para determinar su pertenencia o no al cluster. Puede decirse que Θ es la distancia mínima necesaria para pertenecer o no a un cluster.

La idea general del algoritmo es que a medida que se evalúa cada vector nuevo, este se asigna a un grupo existente o se crea un nuevo grupo en función de la distancia respecto a los ya formados y del umbral de aceptación o rechazo Θ. Específicamente, el algoritmo en la Figura 2.4 comienza con la creación del primer grupo (C_1) el cual contiene únicamente el primer vector de características x_1. Para cada vector restante (x_i) se calculan todas las distancias entre x_i y los grupos existentes (C_j) y se elije la mínima (correspondiente al k-ésimo grupo). Si $d(x_i, C_k)$ es mayor a Θ y el número de grupos existentes aún no supera q, entonces se crea un nuevo grupo cuyo primer elemento es x_i. En caso de que alguna de las dos condiciones no se cumpla, x_i se integra al grupo más próximo (C_k).

La agrupación se hace en dos etapas. Primero se toma únicamente en cuenta la característica *nodos* y al aplicar el esquema secuencial básico se genera un conjunto inicial de grupos (G). Posteriormente a cada grupo de G se le vuelve a aplicar el algoritmo considerando simultáneamente las características *aristas* y *máximo grado*, lo que genera subgrupos H en los grupos G. Por ejemplo, en la Figura 4.1 se observan los mismos tres grafos de interconexión caracterizados en la Tabla **??**. Al aplicar el agrupamiento por nodos, se crean dos grupos: G_1 y G_2. Los grafos dentro de G_1 comparten el mismo número de nodos (4), el cual es diferente al del grupo G_2 (6). Al aplicar el segundo agrupamiento, G_1 se divide en dos subgrupos (H_1 y H_2) pues sus características de aristas y de máximo grado son diferentes. En otras palabras, H_1 y H_2 son parte de una agrupación más amplia la cual contempla grafos de interconexión con cuatro nodos; sin embargo, sus particularidades hacen que la agrupación más general pueda subdividirse y así diferenciar unas topologías de otras.

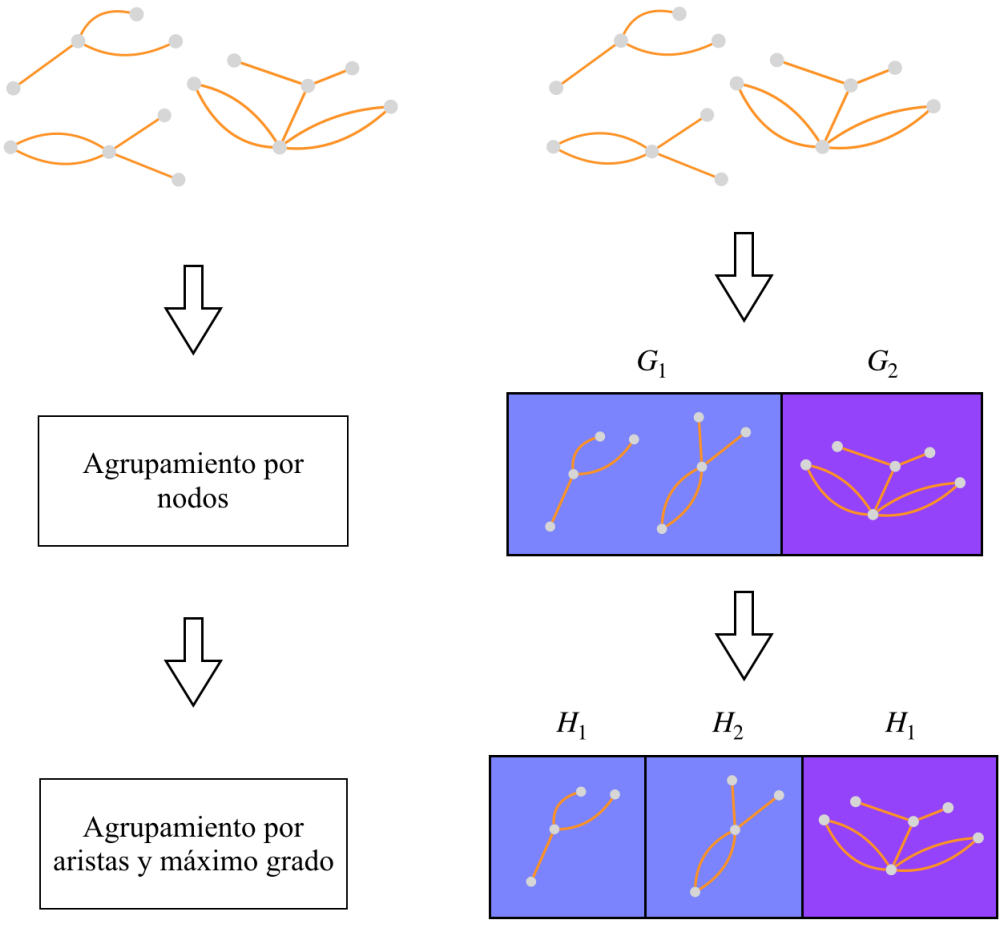


Figura 4.1: Secuencia de clasificación de los grafos de interconexión

Capítulo 5

Evaluación del sistema

La evaluación del sistema consiste en aplicar los pasos descritos en los capítulos anteriores a proyectos de hardware reales. La empresa *Powerbeam* proporcionó tres proyectos de diseño con diferentes características con el objetivo de validar el sistema propuesto. En primer lugar, la información de nodos, conectividad y componentes de los proyectos diseñados en *Altium Designer* es extraída a través de los tres scripts mencionados en la seccción 3.1.1. Los archivos de texto de cada proyecto generados en el proceso de extracción son transformados en estructuras de datos del lenguaje *Python* a través de *parseadores*. A continuación, los grafos de interconexión de los tres proyectos son extraídos mediante dos procesos: el algoritmo empírico (3.2) y el algoritmo formal (3.3). Dada la naturaleza de cada algoritmo, cada uno genera un número diferente, de grafos; a través del método empírico se obtienen 867 subcircuitos mientras que el método formal genera 933. Posterior a la extracción de los grafos, estos se caracterizan generando un *Dataframe* de datos donde cada fila representa un grafo y cada columna una característica. Dada la caracterización, se aplica el esquema secuencial básico en dos etapas para generar grupos en función la distancia entre las características de los grafos. La Figura 5.1 resume el proceso para ambos métodos.

5.1. Evaluación con método formal

La aplicación del algoritmo formal genera 933 subcircuitos de interconexión, mismos que al ser sometidos al agrupamiento secuencial tomando únicamente en cuenta la característica *nodos* generan 7 grupos, cada uno representado por un color en la Tabla 5.1. A cada grupo se le aplica el esquema secuencial considerando las características *nodos* y *máximo grado*, lo que genera un número diferente de subgrupos en cada caso. El número total de grafos que conforman cada grupo se especifica a la

derecha de la misma tabla. Por ejemplo, el grupo 1 es el predominante pues 645 de los 933 grafos comparten la misma característica de nodos. A su vez, el grupo 1 se divide en 18 subgrupos donde cada uno tiene una combinación diferente de aristas y máximo grado.

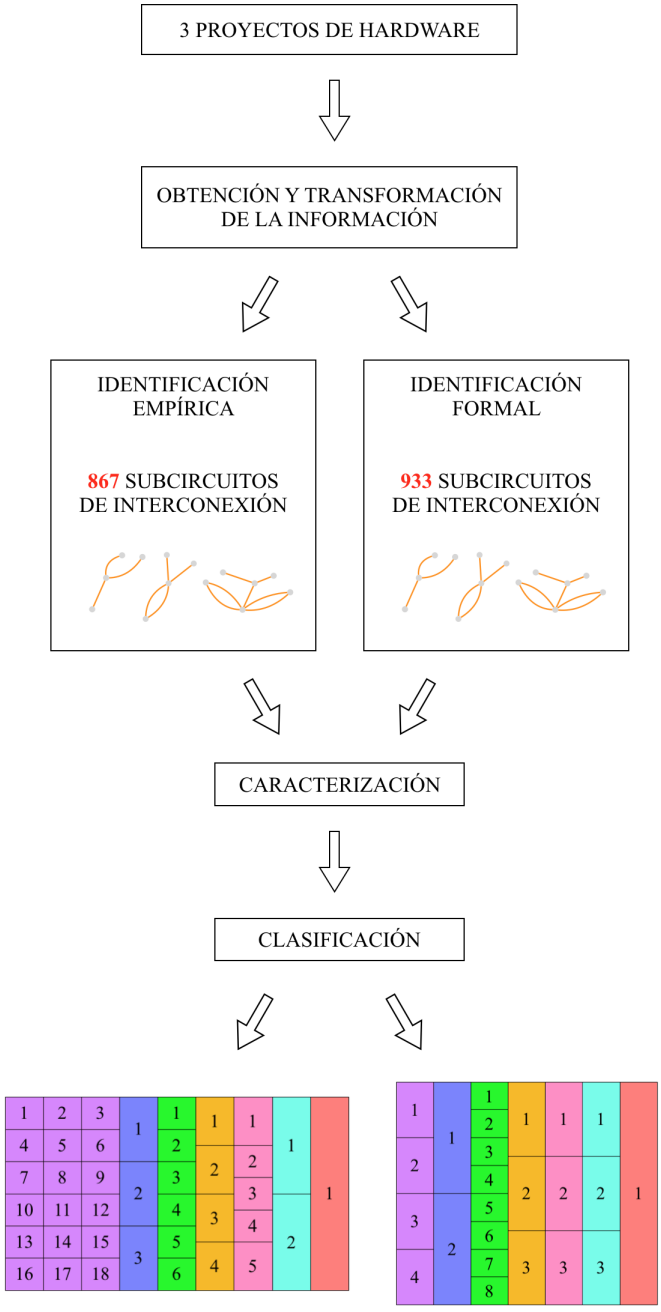


Figura 5.1: Secuencia de evaluación del sistema

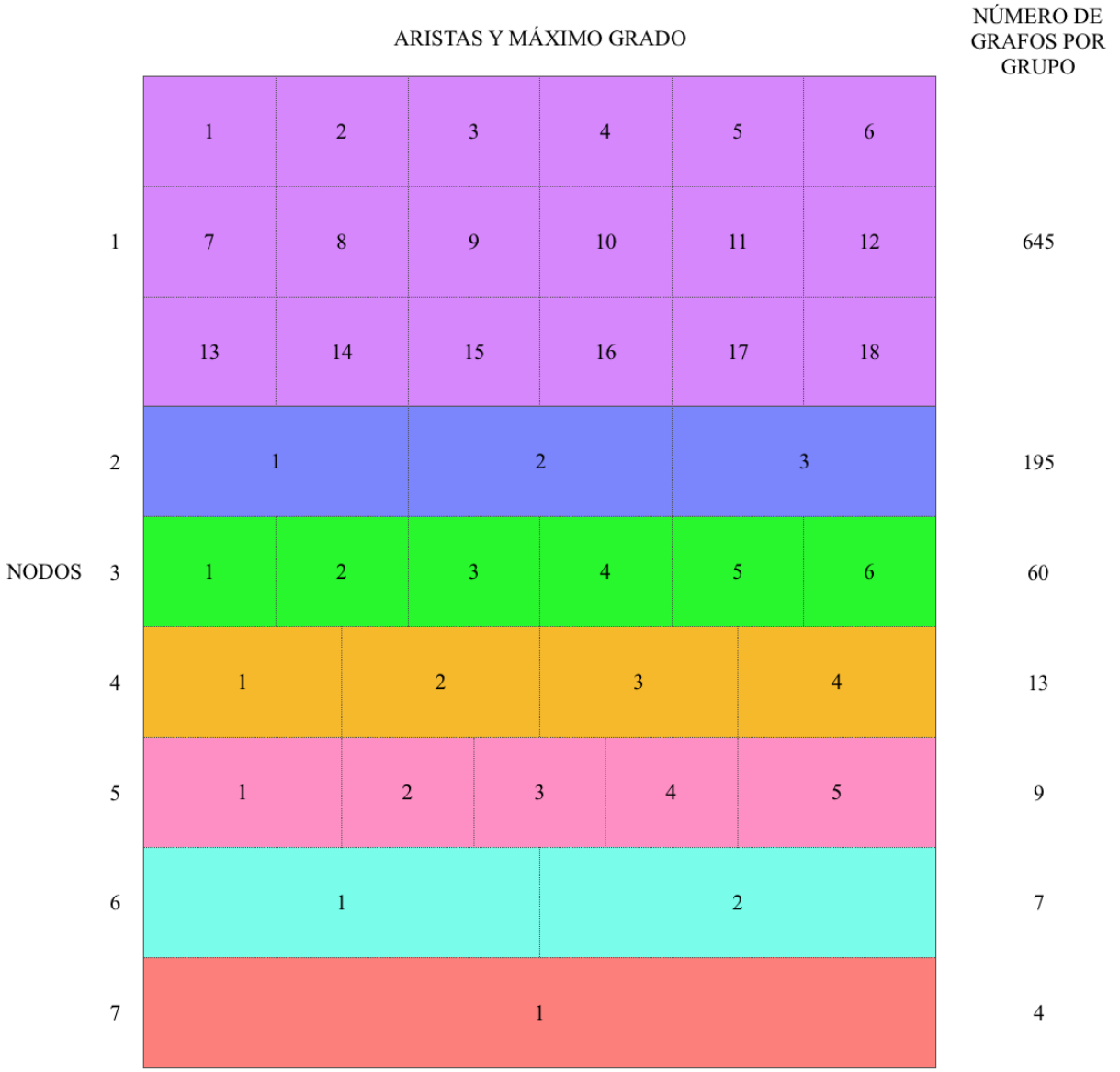


Tabla 5.1: Agrupaciones de grafos usando método formal

La Tabla 5.2 muestra la misma división en grupos y subgrupos que la Tabla 5.1 pero en ella cada casilla contiene el grafo representativo de los subgrupos donde es posible observar las particularidades de cada conjunto. Es decir, ahora además de observar que el grupo 1 está formado por 18 subconjuntos es posible percibir que dicho grupo contiene únicamente grafos con dos nodos en 18 variantes. Así mismo, los grupos 2, 3 ,4, 5, 6 y 7 contemplan grafos con 3, 4, 7, 5, 6 y 17 nodos, respectivamente.

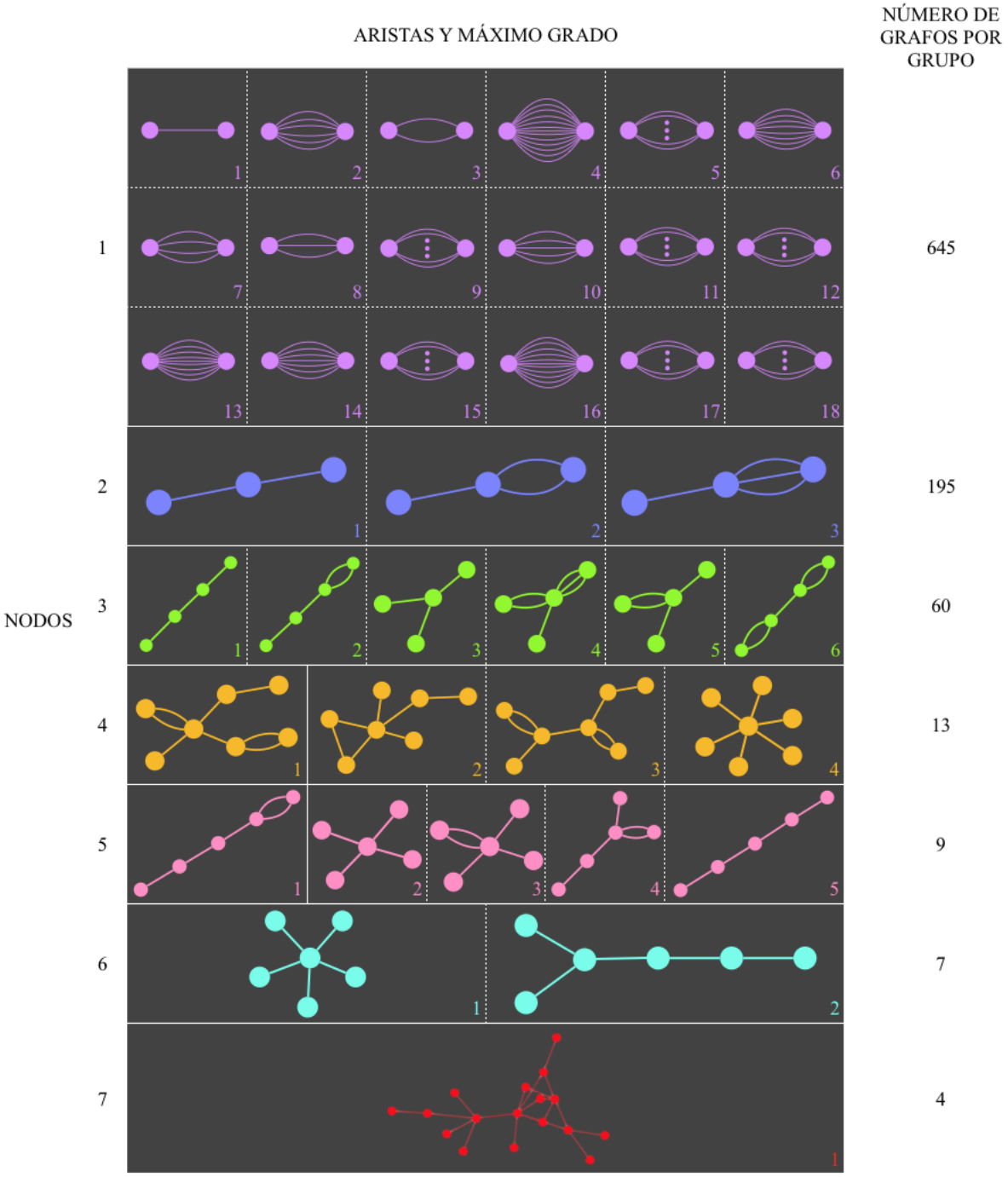
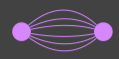
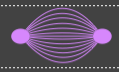
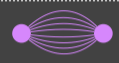
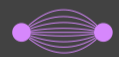
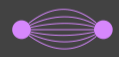
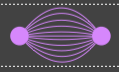


Tabla 5.2: Mapa de grafos por subgrupos del método formal

PATRÓN			GRAFO
NÚM. NODOS	NÚM. ARÍSTAS	MÁX. GRADO	
2	1	1	
	6	6	
	2	2	
	13	13	
	20	20	
	8	8	
	4	4	
	3	3	
	11	11	
	5	5	
	15	15	
	16	16	
	9	9	
	7	7	
	28	28	
	10	10	
	26	26	
	45	45	
6	5	3	
	5	5	

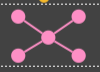
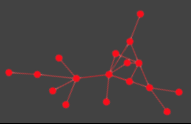
PATRÓN			GRAFO
NÚM. NODOS	NÚM. ARÍSTAS	MÁX. GRADO	
3	2	2	
	3	3	
	4	4	
4	3	2	
	4	3	
	3	3	
	6	6	
	4	4	
	5	3	
7	8	5	
	7	5	
	8	4	
	6	6	
5	4	4	
	5	3	
	5	5	
	4	2	
	5	4	
17	23	9	

Tabla 5.3: Patrones de grafos de interconexión del método formal

Por otro lado, la Tabla 5.3 puntualiza los patrones de características de los grafos de interconexión. Cada patrón está conformado por tres números, de izquierda a derecha: número de nodos, número de aristas y el máximo grado del grafo. Cabe mencionar que el color de los grupos, patrones y grafos es consistente a través de las tres Tablas, es decir, los grafos y patrones de las Tablas 5.2 y

5.3 con determinado color corresponden a los grupos de la Tabla 5.1 del mismo color.

Desde una perspectiva estadística, el 69 % de los grafos de interconexión están formados por dos nodos, independientemente de las posibles variantes de aristas, el 21 % tiene 3 nodos y el 6 % tiene 4 nodos. Cada grupo con más de 4 nodos apenas alcanzan el 1 % del total de circuitos de interconexión.

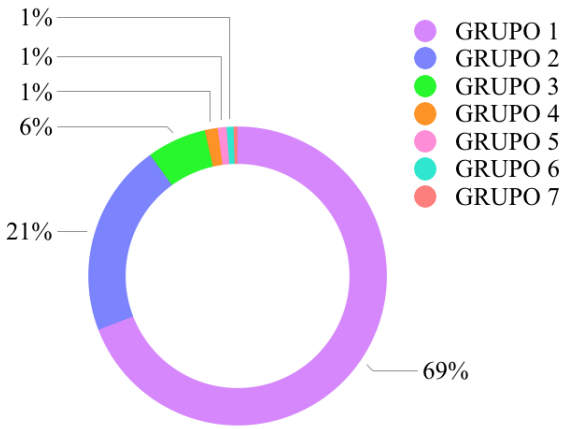


Figura 5.2: Volumen de grafos por grupo en método formal

5.2. Evaluación con método empírico

La aplicación del algoritmo empírico genera 867 subcircuitos de interconexión. Al someterlos al agrupamiento secuencial tomando únicamente en cuenta la característica *nodos* se generan 7 grupos (Tabla 5.4), al igual que con el prodecimiento formal. A cada grupo se le aplica el esquema secuencial considerando las características *nodos* y *máximo grado*, lo que origina un número de subgrupos en cada caso. A excepción de los grupos 3 y 6, el número de subgrupos en el método empírico es menor que en el método formal.

Cabe mencionar que en el método formal, lo grupos 1 y 2 corresponden a grafos con 2 y 3 nodos respectivamente; en cambio, las agrupaciones 1 y 2 del método empírico comprenden grafos con 3 y 2 nodos. El orden en el que las muestras le son presentadas al algoritmo de clasificación determina el orden final de los grupos. En otras palabras, la primera muestra del *Dataframe* del algoritmo formal es la caracterización de un grafo de 2 nodos mientras que la primera muestra del *Dataframe* del algoritmo empírico es de un grafo de 3 nodos, lo que provoca que el primer grupo del método

formal contenga grafos con 2 nodos mientras que el primer grupo del algoritmo empírico contiene grafos de 3 nodos.

Tabla 5.4: Agrupaciones de grafos usando método empirico

Una vez establecido esto es posible afirmar que, a pesar de que tanto el número total de subcircuitos de interconexión como el ordenamiento de los grupos en función del número de nodos son diferentes entre ambos métodos, el volumen de las agrupaciones cuya característica de nodos es la misma en ambos algoritmos es es similar. Es decir, el tamaño de la agrupación que contiene grafos de dos nodos en el método formal es similar a su contraparte en el método empírico: 69 % y 65 %, respectivamente. Cabe señalar que el color de las agrupaciones de las tablas de ambos métodos no está en función del número de nodos sino del número de grupo.

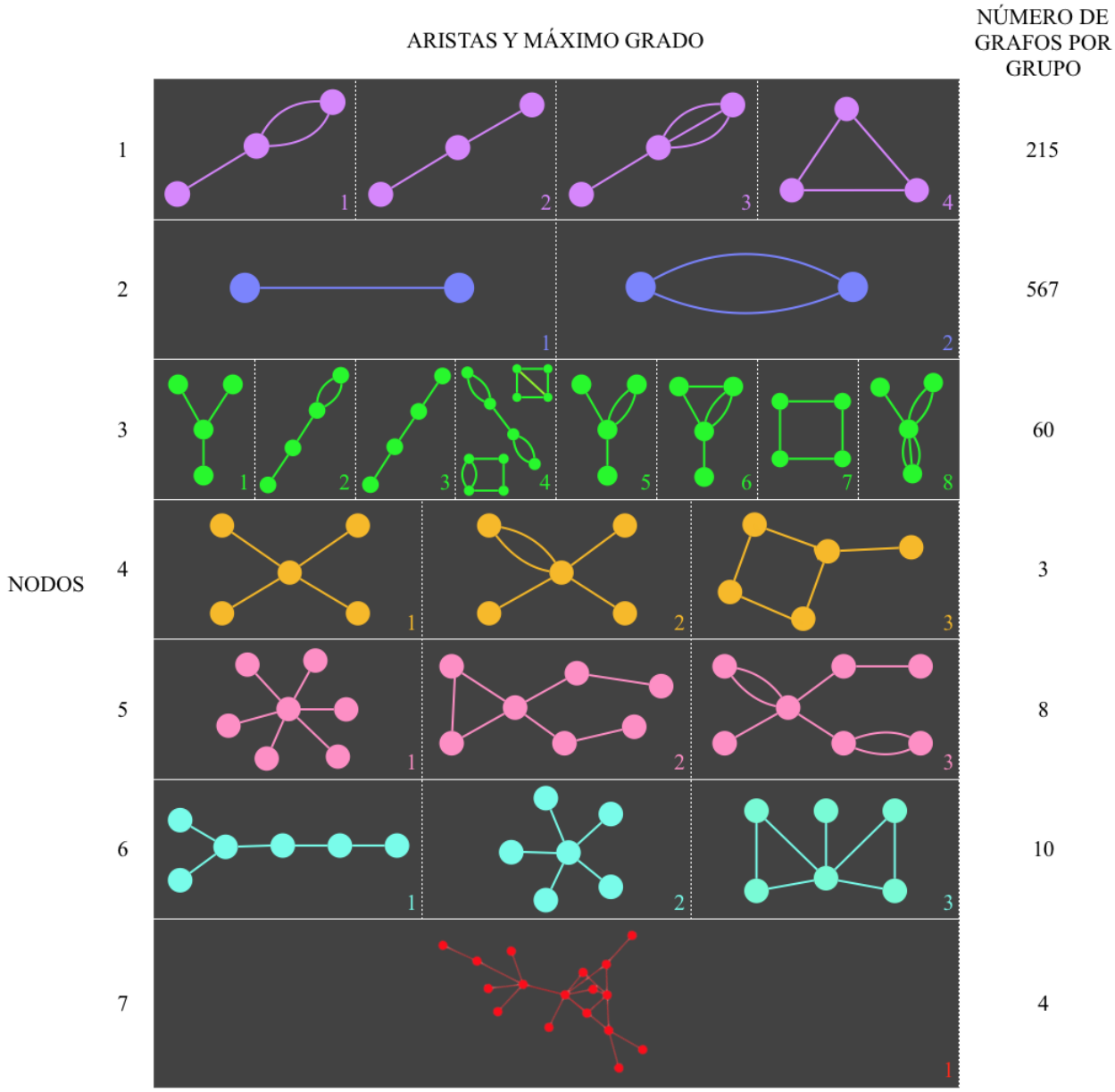
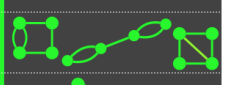
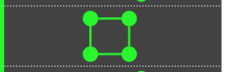
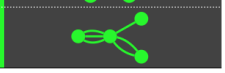


Tabla 5.5: Mapa de grafos por subgrupos del método empírico

PATRÓN			GRAFO
NÚM. NODOS	NÚM. ARÍSTAS	MÁX. GRADO	
2	1	1	
	2	2	
3	3	3	
	2	2	
	4	4	
	3	2	
4	3	3	
	4	3	
	3	2	
	5	3	
	4	4	
	5	4	
	4	2	
	6	6	

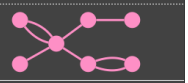
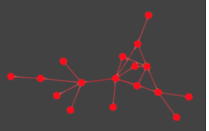
PATRÓN			GRAFO
NÚM. NODOS	NÚM. ARÍSTAS	MÁX. GRADO	
5	4	4	
	5	5	
	7	4	
7	6	6	
	7	4	
	8	5	
6	5	3	
	5	5	
	7	5	
17	23	9	

Tabla 5.6: Patrones de grafos de interconexión del método empírico

Al igual que en el método formal, los subgrupos del método empírico están descritos a través de patrones de tres características. Con el método empírico se obtienen 24 patrones capaces de describir y diferenciar completamente los circuitos de interconexión de los proyectos estudiados.

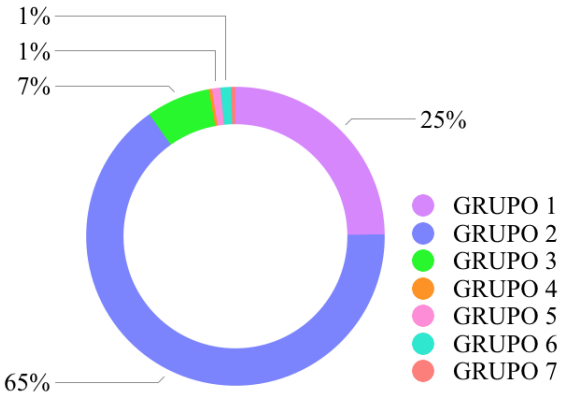


Figura 5.3: Volumen de grafos por grupo en método empírico

5.3. Comparación de métodos

Una vez evaluados ambos métodos procedemos a la comparación de sus resultados con el objetivo de determinar si existe una diferencia significativa en los resultados del *SCPSI* dada la elección del algoritmo de extracción.

La primera diferencia entre ambos algoritmos es el número de grafos de interconexión que generan. La extracción formal produce 933 subcircuitos de interconexión mientras que la identificación empírica únicamente 867; entre ambas hay 66 subcircuitos de diferencia. ¿Son estos 66 subcircuitos un factor determinante? Se analizará a continuación.

Al agrupar los 933 subcircuitos del algoritmo formal con base en el número de nodos se obtienen 7 grupos, al igual que al agrupar los 867 subcircuitos del algoritmo empírico. Es importante mencionar que pese a que el número de grupos es el mismo en ambas clasificaciones, el orden no necesariamente lo es. Por ejemplo, el grupo 1 del método formal comprende a los grafos de dos nodos mientras que en el grupo 1 del método empírico incluye los grafos de tres nodos. En el procedimiento formal el grupo 4 contiene grafos de siete nodos y el grupo 5 grafos de cinco nodos mientras que en el procedimiento empírico las características están invertidas. En ambos casos los grupos 3, 6 y 7 reúnen el mismo tipo de grafos, en términos de número de nodos. La equivalencia entre grupos en función del número de nodos se muestra en las primeras tres columnas de la tabla 5.7.

La topología de grafos que incluye cada agrupación lo determinó el orden de las muestras, pues como se mencionó en el marco teórico: El orden en el que se presentan los vectores al esquema juega un papel importante en los resultados del agrupamiento. Un orden de presentación diferente puede llevar a resultados de agrupación totalmente diferentes, en términos del número de grupos así como los grupos mismos.

A pesar de que el ordenamiento es diferente, el método utilizado para la extracción de los subcircuitos no influye en las agrupaciones por nodos pues ambos procedimientos derivan en el mismo número de grupos con dicha característica idéntica. Más aún, a pesar de la diferencia de 66 subcircuitos mencionada, el número de grafos por grupo es proporcionalmente equivalente. En otras palabras, en el algoritmo formal el grupo con mayor número de subcircuitos es el 1 con 69 %, corres-

NÚM. DE NODOS	GRUPO DEL MÉTODO		% DE GRAFOS DEL MÉTODO	
	FORMAL	EMPÍRICO	FORMAL	EMPÍRICO
2	1	2	69%	65%
3	2	1	21%	25%
4	3	3	6%	7%
5	5	4	< 1%	< 1%
6	6	6	< 1%	> 1%
7	4	5	> 1%	1%
17	7	7	< 1%	< 1%

Tabla 5.7: Porcentajes por grupo en función del número de nodos

pondiente a grafos de dos nodos; el grupo equivalente en las agrupaciones del método empírico es el 2 con 65 % de las muestras. Las columnas 5 y 6 de la tabla 5.7 indican el porcentaje de muestras obtenidas mediante cada método en función del número de nodos.

La segunda diferencia radica en el número de subgrupos generados a partir de la segunda clasificación basada en el número de aristas y el máximo grado de los grafos. Las agrupaciones equivalentes con la mayor diferencia de subgrupos son aquellas que contienen grafos con dos nodos pues en el método formal el grupo 1 consta de 18 subgrupos mientras que en el método empírico el grupo 2 sólo se subdivide únicamente en 2. En general, la variación de los subcircuitos de interconexión formados por dos nodos depende únicamente del número de aristas paralelas entre ellos pues no existe otra forma de conectarlos nodos mas que directamente uno con el otro. En ese sentido, los 18 grupos del método formal y los 2 del método empírico son extensiones del patrón más simple (presente en ambos métodos): 2 – 1 – 1.

Respecto al resto de las agrupaciones, a continuación se enumeran las diferencias más marcadas:

- El grupo de grafos con 3 nodos del método empírico presenta un subgrupo más que su contraparte en el método formal; dicho subgrupo es representado por estructuras triangulares. El resto de los subgrupos exhiben patrones y estructuras idénticas.
- Los subgrupos de grafos con 4 nodos del método empírico presentan algunas estructuras

cuadrangulares, a diferencia de los subgrupos del método empírico en el cual las estructuras de 4 nodos son rectas o en forma de *Y*.

- El método formal presenta una estructura con 7 nodos más que el método empírico; sin embargo, el resto de las estructuras son similares.
- El método empírico presenta una estructura de 6 nodos más que el método empírico. Las otras dos estructuras son idénticas a las del método formal.

Después de comparar los resultados de ambos métodos se determinó que la diferencia en cantidad de subcircuitos de interconexión extraídos de los proyectos de hardware no es relevante a la hora de clasificar por número de nodos pues que las agrupaciones de ambos métodos compartan el mismo número de grupos y particularmente el mismo número de nodos demuestra que los resultados de los procedimientos son similares, al menos al nivel de la primera clasificación. A nivel de subgrupos las diferencias son más evidentes; sin embargo, a pesar de las disparidades estructurales de los subgrupos entre ambos métodos cada uno está determinado por una única tercia de números de tal forma que sin importar el método de extracción un patrón no describe más que a un grafo en particular.

Capítulo 6

Conclusiones

Para abordar el problema de clasificación de patrones en circuitos electrónicos fueron necesarios una serie de pasos, mismos que a grandes rasgos se resumen a continuación:

- Estudiar la teoría de circuitos, la teoría de grafos y la teoría de clasificadores para trasladar el problema de identificación de subcircuitos de interconexión al contexto de clasificación no supervisada de grafos.
- Extraer la información de interconexión de los diseños con el objetivo de manipularla fuera el ambiente de Altium Designer.
- Transformar la información extraída a objetos del lenguaje de programación Python para construir los grafos de interconexión.
- Determinar las características más relevantes de los grafos.
- Definir los parámetros del clasificador secuencial.
- Evaluar el sistema clasificador de patrones de subcircuitos de interconexión.

La ejecución de cada uno de estos pasos condujo a resoluciones que vale la pena resaltar:

- Es posible abordar el problema de detección de segmentos de circuitos desde una perspectiva diferente a la teoría clásica de circuitos.
- Existe más de un método para reconstruir los subcircuitos de interconexión a partir de las fuentes de información extraídas de los proyectos diseñados en *Altium Designer*.

- La ventaja del método de extracción formal sobre el método empírico recae en que el primero hace una reconstrucción total del circuito en forma de grafo, lo que garantiza la inclusión de todos componentes, nodos y terminales.
- Transformar subcircuitos de interconexión a grafos de interconexión permite una caracterización más amplia pues está basada en propiedades de grafos.
- No es necesario limitar el número de agrupaciones (q) en el algoritmo de clasificación secuencial pues la segmentación no es desmesurada.

El desarrollo hecho hasta este punto sólo contempla un conjunto de descriptores (vértices, aristas y grado máximo) que en principio parecería arbitrario; sin embargo, estas características fueron seleccionadas a partir de un proceso de prueba y error. Inicialmente se consideraron algunas otras (tomadas desde una perspectiva de circuitos eléctricos) como el número de puertos, el tipo de componentes, el tipo de nodos y el número de elementos en paralelo. La clasificación con dichas características no derivó en resultados tan satisfactorios como cuando se abordó con descriptores tomados de la teoría de grafos. Para continuar con el desarrollo desde esta perspectiva se propone explorar otros conjuntos de características y determinar si la clasificación presenta una mayor precisión o no con el fin de encontrar el grupo con mejor desempeño. Como ideas preliminares de posibles nuevos descriptores están el diámetro y k-conexidad en lugar de grado máximo.

A partir del desarrollo y evaluación del *SCPSI* se determinó que es posible abordar el análisis estructural de circuitos electrónicos desde una perspectiva de aprendizaje de máquina y sin tomar en cuenta características eléctricas como voltajes, corrientes, impedancias, etc. Además, se confirmó la hipótesis que dio origen al *SCPSI*: existen patrones que prevalecen en el diseño electrónico independientemente del propósito del circuito. Esto abre las puertas a un nuevo tipo de herramienta auxiliar en el diseño de hardware: la detección de subcircuitos de interconexión potencialmente erróneos dadas caracterizaciones no conocidas previamente por el *SCPSI*.

Esta tesis de ninguna forma se proclama como un clasificador absoluto de circuitos sino inaugura un nuevo método de análisis; mismo que queda en espera de futuras mejoras y propuestas siendo las inmediatas incrementar el catálogo de patrones conocidos a través del procesamiento de más proyectos de diseño electrónico y la variación de los descriptores seleccionados para la clasificación.

Bibliografía

[Alexander et al., 2013] Alexander, C. K., Sadiku, M. N. O., Cordero Pedraza, C. R., Villagómez Velázquez, H., and Alexander, C. K. (2013). *Fundamentos de circuitos eléctricos.* McGraw-Hill education. México : McGraw-Hill Interamericana, c2013.

[Altium-LLC, 2017] Altium-LLC (2017). Scripting. `https://techdocs.altium.com/display/SCRT/Scripting`. Accessed: 2019-02-26.

[Chen, 2009] Chen, B. (2009). Topological patterns identification for sneak circuit analysis. *2009 8th International Conference on Reliability, Maintainability and Safety, Reliability, Maintainability and Safety, 2009. ICRMS 2009. 8th International Conference on*, page 133.

[Chua et al., 1987] Chua, L. O., Desoer, C. A., and Kuh, E. S. (1987). *Linear and nonlinear circuits.* McGraw-Hill series in electrical engineering: Circuits and systems. New York ; México : McGraw-Hill, [1987].

[Deo, 1974] Deo, N. (1974). *Graph theory with applications to engineering and computer science.* Englewood, cliffs : Prentice Hall, 1974.

[Grune and Jacobs, 2008] Grune, D. and Jacobs, C. J. H. (2008). *Parsing techniques: A practical guide.* New York: Springer.

[Jain and Dubes, 1988] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data.* Prentice Hall.

[Miller, 1989] Miller, J. (1989). Sneak circuit analysis for the common man.

[Networkx, 2019] Networkx (2019). Multigraph—undirected graphs with self loops and parallel edges. `https://networkx.github.io/documentation/stable/reference/classes/multigraph.html`. Accessed: 2019-04-01.

[NetworkX-Developers, 2019a] NetworkX-Developers (2019a). Networkx. https://networkx.github.io. Accessed: 2019-04-01.

[NetworkX-Developers, 2019b] NetworkX-Developers (2019b). networkx.multigraph.degree. https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.MultiGraph.degree.html. Accessed: 2019-06-07.

[NetworkX-Developers, 2019c] NetworkX-Developers (2019c). networkx.multigraph.edges. https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.MultiGraph.edges.html. Accessed: 2019-06-07.

[NetworkX-Developers, 2019d] NetworkX-Developers (2019d). networkx.multigraph.nodes. https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.MultiGraph.nodes.html. Accessed: 2019-06-07.

[Pandas, 2019a] Pandas (2019a). pandas.dataframe. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html. Accessed: 2019-06-06.

[Pandas, 2019b] Pandas (2019b). Python data analysis library. https://pandas.pydata.org. Accessed: 2019-06-06.

[Scappaticci et al., 2016] Scappaticci, A., Benson, R., Foley, D., and Kellner, D. (2016). Sneak circuit analysis: Lessons learned for beginners based on a successful application. *2016 Annual Reliability and Maintainability Symposium (RAMS), Reliability and Maintainability Symposium (RAMS), 2016 Annual*, page 1.

[Theodoridis and Koutroumbas, 2009] Theodoridis, S. and Koutroumbas, K. (2009). *Pattern recognition.* San Diego, California : Academic, c2009.