Universidad del Papaloapan

Terra Uberrima, Mens Aperta

INGENIERÍA EN COMPUTACIÓN

Tesis

Diseño automatizado óptimo de circuitos lógicos utilizando algoritmos genéticos paralelos.

Que para obtener el grado de Ingeniero en Computación presenta:

Javier Vite Roldán

Director:

Dr. Sergio Ivvan Valdez Peña

Diciembre de 2011

"En lugar de intentar imitar aquello en lo que somos buenos, pienso que es mucho más fascinante investigar aquello en lo que nos desempeñamos pobremente, es insensato usar máquinas para imitar a los seres humanos, en tanto que las máquinas son realmente buenas siendo máquinas, y esto es algo en lo que los seres humanos somos malos. Cualquier proyecto de inteligencia artificial exitoso castra a la máquina por su propia naturaleza."

Edsger Dijkstra

Agradecimientos

Quiero hacer un agradecimiento muy especial a dos personas que formaron parte fundamental de mi vida y que ahora no están en este mundo: Mis abuelos.

A mis padres, que sin importar lo adverso de la situación han hecho un enorme esfuerzo por ver este sueño hecho realidad. Cecy y Mary, gracias por su paciencia y ayuda de toda una vida. Tia Ali, por ayudarme cuando más lo necesitaba. Adriana, por creer en mi y ayudarme a conseguir las metas trazadas. Frankie y Alfredo por su amistad invaluable. Dr. Ivvan por sus conocimientos, consejos, amistad y su apoyo incondicional. Doña Male, que sin conocerme confió en mi y abrió las puertas de su casa. Ing. Cesar, por todos sus consejos y su amistad. Mis revisores Mc. Rafa, Dr. Eduardo, Dra. Bety, por hacerlos trabajar a contra reloj.

Quiero agradecer a todas y cada una de las personas que han puesto su granito de arena en la realización de este trabajo, ya que han formado parte fundamental de mi desarrollo profesional y personal. A mis profesores, amigos y consejeros que durante 5 años orientaron este difícil camino, son tantas personas que me vienen a la mente que no alcanzaría este pequeño espacio para mencionarlos a todos, sin embargo, los tengo presentes en mi mente.

Agradecemos a PROMEP por el apoyo otorgado para la realización del presente proyecto *PROMEP/103.5/10/5489*

Infinitamente Gracias

Índice general

Índice de figuras

Indice de tablas

Capítulo 1

Introducción

La tarea de diseño, ha sido considerada a través de los años como una habilidad únicamente humana, razón por la cual surgieron algunas discrepancias acerca de la capacidad de invención de las computadoras a partir de reglas para la realización de actividades como la composición o el diseño. Con el surgimiento de la Inteligencia Artificial (IA) en 1956 [2] hubo una revolución en la forma de concebir cómo la computadora podía realizar tareas de forma "inteligente" y comenzó una era en la cual las computadoras fueron capaces de auxiliar en tareas que requerían de visión artificial [22, 3], diagnostico médico [14, 4], diseño de estructuras [28, 29, 30, 31], transformadores, diseño de circuitos [7, 32], alas de aviones, e incluso de obras consideradas artísticas como música o muebles.

En las décadas de 1950 y 1960 algunos investigadores como Rechenberg, Schwefel, Friedman, entre otros comenzaron a realizar investigaciones sobre sistemas evolutivos con el fin de proponer una nueva herramienta para ingeniería. Estos científicos notaron que había ciertas cosas que en la naturaleza sucedían con regularidad, tales como el movimiento de los planetas, eclipses solares y lunares, el desarrollo del lenguaje en un niño y muchos más [18], mismos que podían predecirse mediante la observación. La evolución que ha tenido la naturaleza es producto de miles de millones de años, este proceso es posible de imitar en algunos casos y tiende a ser mejor con el paso del tiempo.

John Holland fue inspirado por las teorías evolutivas de Charles Darwin, entre ellas

la "Selección Natural de las especies" [8] la cual sugiere que los padres heredan características a los hijos convirtiéndolos en individuos cada vez más aptos al entorno en que se encuentren, a través del tiempo únicamente los seres que obtengan la mejor combinación de características sobrevivirán a este proceso evolutivo.

Holland inició su investigación para proponer una herramienta que imite el proceso de selección natural, consiste en una serie de pasos ordenados (algoritmo) en los que se crea una población inicial aleatoria, compiten entre sí todos los individuos buscando la supervivencia de los más aptos, se cruzan entre ellos para preservar las características de aptitud y se incluye también una mutación, esto con el fin de explorar nuevas propiedades no existentes en la población actual. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos" [13], pero se hizo popular bajo el nombre de Algoritmos Genéticos (AG) en 1975.

Una definición formal de Algoritmo Genético es la siguiente:

> *"Algoritmos de búsqueda basados en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más aptos entre las estructuras de cadenas, con una estructura de información aleatorizada, que se intercambian para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana" [12].*

La optimización es una herramienta importante en la toma de decisiones y el análisis de sistemas físicos [19]. Para usarla, primero debemos identificar algún objetivo y una medida cuantitativa de los resultados del sistema bajo estudio. Este objetivo puede ser la utilidad, el tiempo, la energía potencial o cualquier cantidad o combinación de cantidades que pueden ser representadas por un número. La función objetivo depende de ciertas características del sistema, llamadas variables o incógnitas. Nuestro objetivo es encontrar valores de esas variables que optimicen la función objetivo.

El proceso de identificar el objetivo, variables y restricciones para un problema dado es conocido como modelado. Matemáticamente hablando, optimización es la minimización o maximización de una función sujeta a restricciones en sus variables.

En esta tesis se aborda el diseño de circuitos lógicos combinatorios de manera autónoma, modelado como un problema de optimización, en el cual deseamos obtener un circuito lógico que cumpla de manera satisfactoria con una tabla de verdad y que minimice la cantidad de componentes, además de reducir el tiempo que tomaría al experto humano en realizar la tarea de diseño se reduce el costo del proyecto.

Esta tarea generalmente es realizada por un humano que tiene amplios conocimientos del área, razón por la cual, sólo la puede llevar a cabo personal especializado, dichos conocimientos incluyen operaciones y expresiones booleanas, métodos de reducción de funciones lógicas tales como: Minitérminos, Maxitérminos, Mapas de Karnaugh, Quine-McCluskey, entre otros.

Finalmente, uno de los productos principales de esta tesis es una aplicación que sea capaz de diseñar de manera automatizada, rápida y eficiente circuitos lógicos combinatorios a partir de tablas de verdad y sin conocimiento a priori, utilizando únicamente Algoritmos Genéticos, siendo estos los algoritmos de optimización que según la literatura son los más consistentes en la solución exitosa del problema. Para disminuir el tiempo de búsqueda del circuito óptimo se utilizó una estrategia de cómputo paralelo en memoria distribuida.

1.1. Objetivo general

Realizar una aplicación capaz de diseñar circuitos lógicos combinatorios complejos de forma rápida, automatizada, eficiente, autónoma y sin conocimiento a priori, que cumpla con restricciones de funcionalidad dadas por una tabla de verdad y minimice el número de componentes del mismo.

1.2. Objetivos específicos

1. Desarrollo de un evaluador de circuitos lógicos. Determinar en base a una tabla de verdad la funcionalidad de un circuito lógico.

2. Desarrollo del optimizador de circuitos lógicos. Explorar mediante Algoritmos Genéticos diferentes circuitos lógicos que tengan un número reducido de compuertas.

3. Generación de conocimiento de una propuesta nueva de operador de selección en un Algoritmo Genético. Se pretende alcanzar el estado del arte actual y poder realizar una pequeña aportación.

4. Paralelización del algoritmo de optimización. Paralelizar las partes del algoritmo que más tiempo requieran.

5. Implementación de una aplicación paralela y multiplataforma para optimización de circuitos lógicos. Aplicación que genere de manera autónoma y sin conocimiento a priori circuitos lógicos combinatorios funcionales y de tamaño reducido.

1.3. Justificación

La tarea de diseño de circuitos lógicos combinatorios como se conoce actualmente es costosa, lenta y muy complicada de llevar a cabo sin la ayuda de un experto humano, existe una gran cantidad de posibles combinaciones para cada problema y no es factible explorarlas todas. Algunos programas como WorkBench®, Proteus®, Qucs®, KtechLab®, entre otros, reducen el tiempo que toma al diseñador realizar su trabajo, puesto que permiten simular el comportamiento de un circuito y en función de ciertas entradas observar las posibles salidas.

Para facilitar la tarea del diseño de circuitos lógicos se propone el desarrollo de una aplicación que hace uso de la Inteligencia Artificial para construir soluciones en base a las restricciones definidas para el problema, dicha aplicación es capaz de emular a un humano en la tarea del diseño, explora las mejores combinaciones de elementos y

conexiones, de manera que en un lapso muy corto de tiempo se puede tener un resultado, reduciendo los costos de desarrollo del proyecto.

Nuestra aplicación puede ser utilizada por aquellas personas que desean encontrar y simplificar una función lógica a partir de información binaria de entradas y salidas (tabla de verdad), la función lógica está representada por operadores booleanos, a partir de los cuales se puede construir un circuito lógico combinatorio.

La aplicación será una propuesta para el diseño de circuitos lógicos, capaz de realizar las mismas tareas que un humano (diseño de circuitos) de forma autónoma y sin conocimiento a priori, es decir, sin saber ningún método de síntesis de circuitos lógicos, tales como Quine-McCluskey, los mapas de Karnaugh o el método algebraico; por ser una aplicación paralela, nuestra propuesta de solución es extrapolable a otros problemas similares en el mundo real de forma eficiente.

El problema de diseño de circuitos lógicos que intentamos resolver ha sido abordado desde diferentes metodologías y algoritmos como: los Algoritmos Genéticos simples (AGs) con cardinalidad binaria y entera, y los Algoritmos de Estimación de Distribución(EDAs), entre otros. Consiguiendo con ellos resultados favorables.

El conocimiento adquirido y las habilidades desarrolladas me permitirán aplicar herramientas para la solución eficaz y eficiente de problemas de éste tipo en la vida profesional, así como la obtención del grado de Ingeniero en Computación.

1.4. Alcances y limitaciones

La aplicación que se propone pretende realizar de manera autónoma el diseño de los circuitos lógicos sin ningún tipo de conocimiento previo acerca de simplificación de funciones booleanas, requiere únicamente de una tabla de verdad proporcionada por el usuario; la aplicación realizará el diseño del circuito en base a las características introducidas. La propuesta de diseño que proporcionará la aplicación reducirá la cantidad de

componentes con el fin de entregar una implementación más sencilla del circuito lógico.

La presente propuesta es una implementación del conocimiento existente en la literatura científica y puede competir con otras existentes en el estado del arte, ya que se desarrolla una aplicación que sea capaz de satisfacer las mismas restricciones y alcanzar los mismos resultados que otras reportadas.

La aplicación es una implementación de un Algoritmo Genético simple, como se muestra en la tabla 4.1, además, se propondrá un nuevo método de selección para el AG con el cual se espera mejorar significativamente los resultados.

1.4.1. Alcances:

- Una aplicación que sea capaz de diseñar de forma autónoma circuitos lógicos presentes en la literatura.
- Una propuesta de método de selección que mejore de manera evidente un Algoritmo Genético.
- Una propuesta de paralelización que reducirá en más del 60 % el tiempo de cómputo.
- Mejora de los circuitos diseñados por humanos reportados en la literatura.

1.4.2. Limitaciones:

- Equipo de cómputo disponible.
- Tiempo reducido para el desarrollo del proyecto.
- Dado que la propuesta es el método de selección solamente es posible que no se supere a los “mejores” resultados reportados.

Capítulo 2

Marco de Referencia

2.1. Antecedentes

Un sistema digital es una combinación de dispositivos, diseñado para manipular cantidades físicas o información que estén representadas en forma digital; es decir, que sólo puedan tomar valores discretos [26]. Un circuito binario o lógico es aquel en el cual puede haber una o varias variables de entrada, cada una de las cuales puede ser 1 o 0 y una o varias variables de salida, cada una de las cuales puede ser 1 o 0. La electrónica digital involucra circuitos y sistemas en los que hay sólo dos estados posibles. Estos estados se representan mediante dos niveles distintos de voltaje: uno alto y uno bajo. En sistemas digitales, se utilizan combinaciones de los estados, llamadas códigos, para representar números, símbolos, caracteres alfabéticos y otros tipos de información. El sistema numérico de dos estados se llama binario y sus dígitos el 0 y el 1. A un dígito binario se le llama bit [11]. La compatibilidad del sistema binario con otros elementos usados en la Electrónica Digital es total, puesto que todos trabajan con dos estados opuestos, asimilables al 0 y el 1 binarios [27].

A mediados del siglo pasado, la necesidad de automatizar procesos se hizo presente con la creciente industrialización, surgieron algunos mecanismos que utilizaban componentes electrónicos para automatizar el control en la mayoría de los procesos. George Boole estableció una serie de postulados y operaciones tendentes a resolver automa-

tismos o procesos a ejecutar, obteniendo un conjunto de ecuaciones que deberían ser traducidas y llevadas a cabo por elementos mecánicos, hidráulicos, neumáticos, eléctricos o electrónicos, siendo estos últimos los que han sido elegidos para el desarrollo de esta tesis [27].

El álgebra de Boole es un sistema matemático usado para representar mediante símbolos el objeto de un circuito lógico, de forma que su estado pueda ser equivalente a un circuito real. A causa de la estrecha semejanza entre el comportamiento de los circuitos binarios y la lógica de preposiciones (Algebra de Boole), al circuito binario se le conoce también como circuito lógico [25]. En otras palabras, un circuito lógico es la representación gráfica que se da a una función lógica utilizando compuertas.

El álgebra booleana se utiliza para expresar los efectos que los diversos circuitos digitales ejercen sobre las entradas lógicas y para manipular variables lógicas con objeto de determinar el mejor método de ejecución de cierta función de un circuito. En el álgebra booleana solo existen tres operaciones básicas, denominadas operaciones lógicas: OR, AND, NOT. A partir de ellas es posible construir circuitos digitales llamados compuertas lógicas, los cuales a través de diodos, transistores y resistencias conectados de cierta manera hacen que la salida del circuito sea el resultado de una operación lógica básica (OR, AND, NOT) sobre la entrada.

Las tablas de verdad son un medio para describir cómo la salida de un circuito lógico depende de los niveles presentes en las entradas [26], expresan en forma tabular la correspondencia entre la variable de salida y cada combinación posible de valores de sus variables de entrada [24]. Un ejemplo de tabla de verdad se muestra en la Figura 2.1, n_e y n_s indican el número variables de entrada y salida respectivamente, l_t indica la longitud de la tabla de verdad.

En la literatura se encuentran algunas herramientas de síntesis de circuitos lógicos como el método algebraico, mapas de Karnaugh o Quine-McCluskey, que ayudan a reducir de forma manual las funciones booleanas de tamaño relativamente pequeño, para visualizar resultados y simular su funcionamiento se utilizan aplicaciones como WorkBench®,

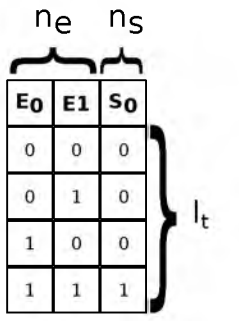

Figura 2.1: Elementos que conforman una Tabla de Verdad

Proteus®, Qucs®, KtechLab® entre otros programas para computadora que actualmente están disponibles en la red.

Existen también en la literatura artículos de investigación de propuestas exitosas de algoritmos para el diseño de artefactos, tales como: diseño de estructuras [28, 29, 30, 31], transformadores, diseño de circuitos [7, 32], alas de avión, e incluso de obras consideradas artísticas como música o muebles que han sido llevados a cabo de manera autónoma o semiautónoma por una computadora.

Las aplicaciones mencionadas en el párrafo anterior fueron realizadas mediante algoritmos evolutivos, entre los cuales destacan los Algoritmos Genéticos (AG's) y los Algoritmos de Estimación de Distribución (EDA's).

Dentro del ámbito específico del diseño de circuitos lógicos, se encuentran algunos trabajos de representación binaria y representación entera utilizando EDA's y AG's [10, 17, 21, 7, 32], en los cuales se ha abordado el problema de manera exitosa.

Para la realización de esta tesis, se ha tomado como referencia principal el Capítulo 12 del libro: Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering [32], por ser un trabajo reciente que condensa y compara las propuestas en el estado del arte de este problema. Así al utilizar esta información, estamos realmente considerando las propuestas exitosas más

relevantes. En este trabajo se presenta un conjunto de problemas de prueba representativos del estado del arte, y las mejores soluciones conocidas para dichos problemas, se hacen comparaciones de eficacia (circuitos factibles diseñados y su número de compuertas), eficiencia (número de evaluaciones de la función objetivo, que está relacionada con el esfuerzo computacional realizado) y escalabilidad (como se incrementan los requerimientos de cómputo en función del tamaño del problema).

2.2. Optimización Global

La optimización se refiere a encontrar una o más soluciones factibles, las cuales corresponden a los extremos de una o más funciones objetivo, también conocidas como función de aptitud. La necesidad de encontrar las soluciones óptimas en un problema proviene principalmente de la aplicación para la que se requiere un valor extremo de una función o de encontrar soluciones de diseño para minimizar posibles costos de fabricación, para aumentar la exactitud o cualquier otro propósito. Los métodos de optimización son de gran importancia en la práctica, particularmente en ingeniería de diseño, experimentos científicos y toma de decisiones en los negocios [9].

El proceso de identificar el objetivo, variables y restricciones para un problema dado es conocido como modelado. Matemáticamente hablando, optimización es la minimización o maximización de una función sujeta a restricciones en sus variables [19].

Un ejemplo de definición matemática de optimización para un problema de minimización es el que se muestra en la Ecuación 2.1

$$\underset{x \in R^n}{\mathbb{M}in}\ f(x)\ sujeto\ a \begin{cases} c_i(x) = 0, & i \in A \\ c_i(x) \geq 0, & i \in B \end{cases} \tag{2.1}$$

Donde f y C_i son funciones con valores reales, A y B son conjuntos de índices.

Una vez que el modelo ha sido formulado, un algoritmo de optimización puede ser usado para encontrar la solución. Usualmente, el algoritmo y el modelo tan complicados que es necesario usar la computadora para implementar el proceso. No hay un algoritmo

de optimización que funcione de manera universal. Mejor dicho, hay numerosos algoritmos, cada uno de los cuales está hecho a la medida de un tipo particular de optimización. A menudo es responsabilidad del usuario elegir un algoritmo que sea el apropiado para esa aplicación específica. La elección es lo más importante; eso determina si el algoritmo es capaz de encontrar la solución y de ser así, que sea de forma rápida o lenta.

Los algoritmos de optimización son iterativos. Comienzan con un valor inicial propuesto y generan una secuencia de mejoras estimadas hasta que alcanzan una solución. La estrategia usada para pasar de una iteración a la siguiente distingue a un algoritmo de otro. Algunos algoritmos acumulan información recopilada de iteraciones previas, mientras otros usan solo información local del punto actual. Todos los algoritmos deberían poseer las siguientes características: robustez, eficiencia y exactitud.

Los algoritmos de optimización más rápidos buscan únicamente una solución local, un punto en el cual el valor de la función objetivo es más pequeña que todos los puntos factibles en su vecindad. Un caso importante y muy especial es la programación convexa, en la cual todas las soluciones locales son también las globales. Los problemas de programación líneal caen en la categoría de programación convexa. Sin embargo, los problemas no líneales pueden poseer soluciones locales [19].

En la Figura 2.2 se muestra una función que posee máximos locales y globales, un máximo local es una solución parcial existente en un intervalo muy pequeño, razón por la cual es muy común caer en el error de tomar un máximo local en lugar de uno global, los algoritmos de programación líneal tienden a tomar máximos locales como globales debido a que buscan únicamente valores mayores en cada iteración y al encontrar un punto de inflexión en la función se detienen y toman el máximo encontrado como global; un máximo global es el valor más alto que puede tomar la función y es mayor a todos los máximos locales existentes, el máximo global es también un máximo local, pero no todos los máximos locales son globales; los algoritmos genéticos son estrategias que evitan tomar un máximo local en lugar de uno global.

Debido a que la función objetivo que modela el problema de optimización de esta tesis

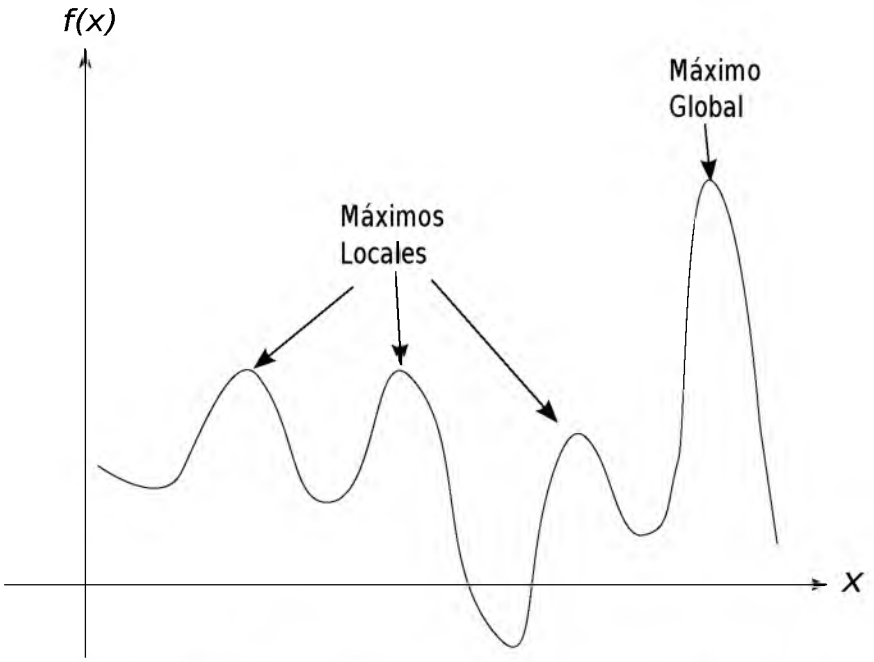

Figura 2.2: Óptimos locales y globales en una función

no es resoluble por los métodos clásicos de optimización que suponen convexidad y un solo máximo global, el problema se aborda utilizando Algoritmos Genéticos.

2.3. Algoritmo Genético Simple

Un Algoritmo Genético simple es un algoritmo matemático altamente paralelizable, que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo, usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las cuales destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud física que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con cierta función matemática que refleja su aptitud [16].

El Algoritmo Genético es un método de optimización comúnmente utilizado por la facilidad de su implementación; el funcionamiento de un Algoritmo Genético simple se representa en la Figura 2.3.

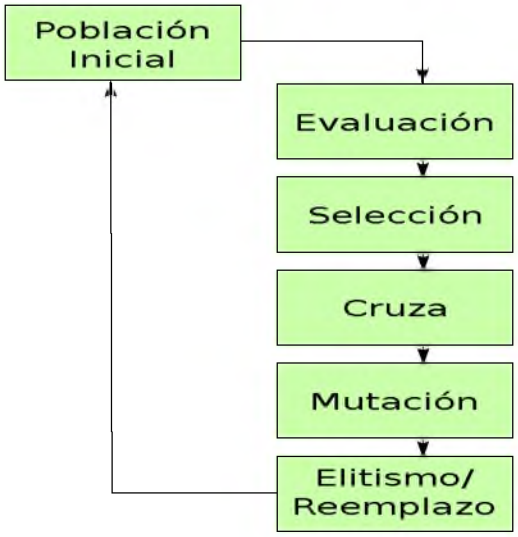

Figura 2.3: Diagrama a bloques de un Algoritmo Genético simple

En los Algoritmos Genéticos, una población es un conjunto de individuos de la misma especie con características en común; un individuo es una propuesta de solución a un problema en particular, la información genética del individuo se expresa como un cromosoma, el cual está formado por un conjunto de genes.

Para el tipo de problema que se aborda en esta tesis, un cromosoma es una cadena de números enteros (codificación entera) que forma un circuito lógico combinatorio mediante conexiones y compuertas (genes). Los operadores genéticos son: selección, cruza, mutación y elitismo, mismos que se describen a continuación.

Funcionamiento del Algoritmo Genético:

- Se ***genera una población inicial*** completamente aleatoria, formada por un conjunto de cromosomas que contienen la información genética de cada individuo de

la población.

- La ***evaluación*** se realiza a cada uno de los individuos de la población para obtener su valor de aptitud, dicho valor se obtiene a partir de la Función Objetivo del problema a resolver, su finalidad es saber qué tan buena es una solución respecto a las demás.
- En la ***selección*** se elige a los individuos más aptos en base a la función de aptitud definida previamente, la selección puede realizarse por truncamiento, rueda de la fortuna, torneo binario, entre otros métodos. La selección es uno de los pasos más importantes del algoritmo ya que en ella se determinan los individuos que heredarán características a las nuevas generaciones.
- La ***cruza*** se realiza intercambiando material cromosómico de dos individuos formando uno nuevo, el objetivo de la cruza es producir una nueva población de individuos presumiblemente más aptos que su generación predecesora y con ello explorar otras posibles soluciones.
- Adicionalmente, existe un operador de ***mutación*** que modificará de manera aleatoria un gen en el cromosoma de un individuo elegido aleatoriamente. Este operador garantiza la introducción de nuevo material cromosómico no existente en el individuo y probablemente ni en sus antecesores, su finalidad es asegurar la interconexión total del espacio de búsqueda.
- Una forma de preservar al individuo con mayor aptitud es el ***elitismo***, esto consiste en introducir a la población al individuo más apto encontrado durante la evaluación. El Algoritmo Genético se ejecuta un número predeterminado de generaciones, o hasta que la población se haya estabilizado (i.e., cuando todos o la mayoría de los individuos tengan la misma aptitud).

Dentro de los Algoritmos Evolutivos, los Algoritmos Genéticos son los que más utilizados en la resolución exitosa de problemas de optimización, a continuación se listan algunas ventajas:

- No requieren una función explícita.

- No requieren que la función sea convexa.
- No requieren que los máximos sean únicos.
- Poco desarrollo analítico.
- Son altamente paralelizables.

2.4. Cómputo Paralelo

Existen en la actualidad dos paradigmas de paralelización en aplicaciones de cómputo, estos son: memoria compartida (por ejemplo: OpenMP) y memoria distribuida (por ejemplo: OpenMPI).

OpenMP es una Interfaz de Programación de Aplicaciones (API) de memoria compartida en múltiples plataformas, cuyas características están comprometidas en disminuir esfuerzos para facilitar la programación paralela en memoria compartida [6]. Se basa en el modelo fork-join, paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (fork) con menor peso, para "recolectar" sus resultados al final y unirlos en un solo resultado (join).

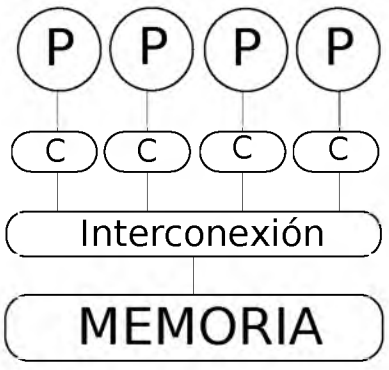

Figura 2.4: Arquitectura de memoria compartida OpenMP

En una arquitectura de memoria compartida como la que muestra en la figura 2.4 hay usualmente un proceso que contiene un grupo de hilos con los cuales comparte un mis-

mo espacio de memoria, por ello el nombre de memoria compartida.

MPI es una interfaz de paso de mensajes. Está escrita por el MPI Forum, un grupo formado por investigadores de universidades, laboratorios y empresas involucrados en la computación de altas prestaciones. OpenMPI es una API estándar utilizada para la computación paralela o distribuida. El estándar MPI comprende dos documentos: MPI-1 (1994) y MPI-2 (1996) [15] y es considerada de facto estándar pero no de manera oficial por la IEEE [20]. OpenMPI es una implementación de código abierto de los estándares MPI-1 y MPI-2 y se encuentra disponible para varios lenguajes de programación. Entre sus características está la distribución de procesos de manera dinámica, alto rendimiento en todas las plataformas, administración de trabajos rápida y fiable, tolerancia a fallos de red y procesos, muchos planificadores de trabajo soportados, portable y mantenible, modificable por los instaladores y usuarios finales, entre otras características [5].

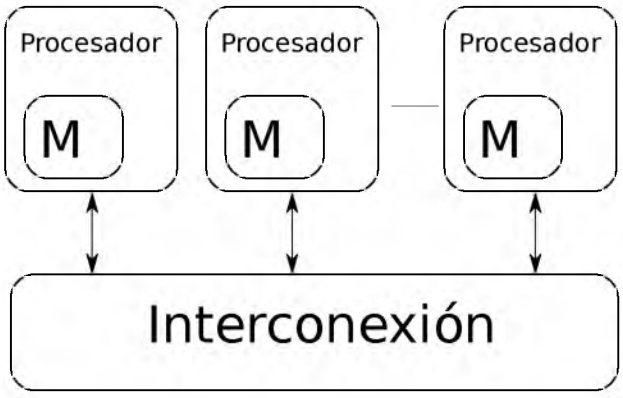

Figura 2.5: Arquitectura de memoria distribuida OpenMPI

En una arquitectura de memoria distribuida como la que se muestra en la Figura 2.5 cada proceso no comparte el mismo espacio de memoria que los demás ya que es muy posible que se ejecuten en maquinas diferentes, esto significa que un proceso no puede ver las variables de sus compañeros, para hacerlo necesita enviar mensajes a los demás procesos y así poder sincronizar sus valores. Una librería de MPI como lo es OpenMPI opera como un intermediario para facilitar el intercambio de mensajes entre procesos.

2.5. Tecnologías de Software para implementación

Para facilitar la codificación y reducir el tiempo del desarrollo de este proyecto se ha elegido Python, un lenguaje de programación interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses "Monty Python" [1].

Python no se compila en la computadora en que se ejecutará, es una desventaja en velocidad de ejecución pero es un lenguaje multiplataforma, lo cual lo hace sumamente portable; podemos implementar nuestra aplicación en diferentes sistemas operativos fácilmente utilizando librerías estándar. El intérprete de Python está disponible en multitud de plataformas como son UNIX, Solaris, Linux, DOS, Windows, OS/2, MacOS, entre otras[1].

La exploración exhaustiva de posibles soluciones de un algoritmo genético convencional puede consumir gran cantidad de tiempo, es necesario implementar una variante de los mismos denominada Algoritmos Genéticos Paralelos; la paralelización de los Algoritmos Genéticos puede llevarse a cabo utilizando estándares de software para desarrollo de cómputo paralelo con un alto grado de madurez como son OpenMP para memoria compartida y OpenMPI para memoria distribuida.

PyPar es un modulo eficiente y fácil de usar que permite que programas escritos en Python puedan ejecutarse en paralelo sobre múltiples procesadores utilizando paso de mensajes. Pypar provee enlace a un subconjunto de instrucciones de la interfaz de paso de mensajes estándar MPI. Entre sus características destacan la flexibilidad de permitir la comunicación general de cualquier objeto de Python sin importar su tipo, es una API intuitiva, eficiente, ligera y no se necesita modificar el interprete original de Python.

Dados los recursos de cómputo disponibles, se eligió el estándar de memoria distribuido OpenMPI en una de sus implementaciones para el lenguaje Python como lo es Pypar.

Capítulo 3

Planteamiento del problema

Diseñar un circuito lógico es una tarea difícil de realizar por un humano, el proceso de búsqueda de los elementos idóneos y su mejor colocación para que cumpla con una tabla de verdad requiere de muchas horas de arduo trabajo, la garantía de que el circuito encontrado sea el óptimo radica principalmente en la experiencia del diseñador.

Existen miles de combinaciones posibles de circuitos lógicos funcionales y el diseñador humano no es capaz de analizar todas éstas combinaciones, además, el costo de tiempo sería demasiado alto y quizás no se encuentre el óptimo; para este tipo de problemas la computadora es una herramienta de gran utilidad ya que puede ayudar a buscar posibles soluciones en base a reglas y procedimientos bien definidos en un tiempo relativamente corto.

Convertimos el problema de diseño en un problema de optimización, en el cual ya no solo nos interesa encontrar una circuito factible, sino que también sea óptimo, para ello definimos el problema como maximizar el número de salidas correctas de la tabla de verdad y minimizar el número de componentes.

Las compuertas que se han elegido para abordar este problema se muestran en la Figura 3.1, se incluyen las que realizan las operaciones lógicas elementales más un pequeño grupo de compuertas que ayuden a simplificar las funciones lógicas y con ello

el tamaño del circuito.

#	Fig	Nombre
0		AND
1		OR
2		XOR
3		NOT
4		NAND
5		NOR
6		WIRE

Figura 3.1: Compuertas seleccionadas para diseñar circuitos lógicos

Las variables de decisión están representadas por una matríz de valores enteros, a esta representación se denomina codificación entera, cada valor dentro de la matríz representa una conexión o una compuerta lógica. Un ejemplo de codificación entera se muestra en la Figura 3.2 en la que se observa a un individuo de tamaño $3(n_r)(n_c)$, donde n_r es el número de renglones y n_c el número de columnas de la matríz, $(n_r)(n_c)$ es la cantidad que indica el tamaño máximo de compuertas que pueden formar a los circuitos lógicos que competirán para ser el circuito final; para este ejemplo en particular se muestra un circuito con un tamaño máximo de $(n_r = 2)(n_c = 2) = 4$ compuertas, es decir, la solución al problema puede tener desde una hasta cuatro compuertas.

Las columnas están divididas en 3 bloques, los dos primeros corresponden a las entradas de la compuerta y el tercero corresponde al tipo de compuerta. En los bloques que corresponden a las entradas para la columna 0, las entradas pueden tomar valores de 0 hasta $n_e - 1$, para las demás columnas esos valores dependen de las salidas de la columna anterior, es decir, toman valores desde 0 hasta $n_r - 1$. El tercer bloque puede tomar valores comprendidos entre 0 y 6, número que corresponde a las compuertas utilizadas, estas son: AND, OR, XOR, NOT, NAND, NOR, WIRE, ver Figura 3.1.

E0	E1	S0
0	0	0
0	1	0
1	0	0
1	1	1

Tabla de verdad para compuerta AND

E0	E1	S0
0	0	0
0	1	1
1	0	1
1	1	1

Tabla de verdad para compuerta OR

E0	E1	S0
0	0	0
0	1	1
1	0	1
1	1	0

Tabla de verdad para compuerta XOR

E0	S0
0	1
1	0

Tabla de verdad para compuerta NOT

E0	E1	S0
0	0	1
0	1	0
1	0	0
1	1	0

Tabla de verdad para compuerta NAND

E0	E1	S0
0	0	1
0	1	0
1	0	0
1	1	0

Tabla de verdad para compuerta NOR

E0	S0
0	0
1	1

Tabla de verdad para compuerta WIRE

Tabla 3.1: Tablas de verdad de las compuertas lógicas propuestas para los experimentos

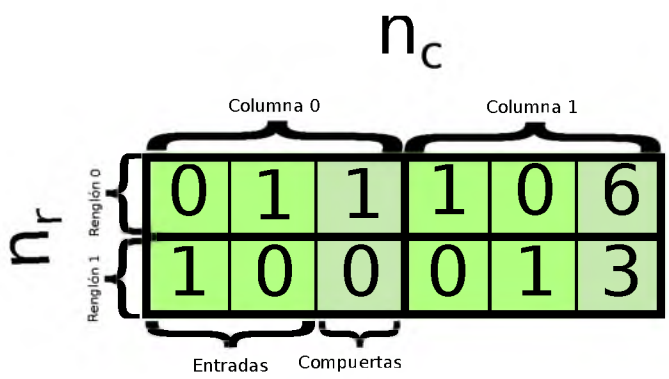

Figura 3.2: Representación Matricial de un individuo utilizando codificación entera

La representación matricial es una forma de abstraer un circuito lógico, la Figura 3.3 muestra la equivalencia de ambas representaciones, la matricial con codificación entera y la del circuito lógico.

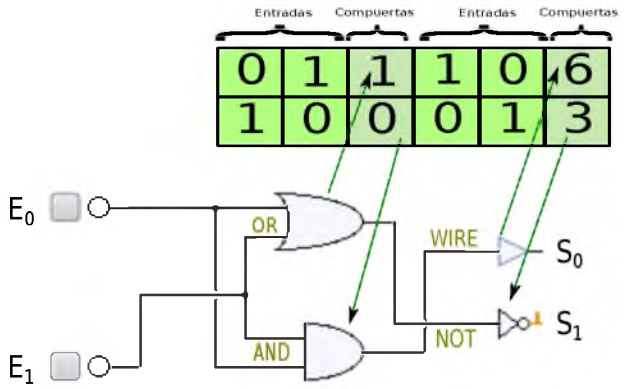

Figura 3.3: Equivalencia entre la representación física del circuito lógico y su correspondiente representación matricial

La Función Objetivo para este problema queda definida por la Ecuación 3.1:

Función objetivo

Maximizar

$$f(x) = C \cdot g(x) + N(x) \tag{3.1}$$

donde:

$g(x)$: es el número máximo de coincidencias existentes entre las salidas del circuito y la tabla de verdad.

C : es una constante.

$N(x)$: es el número de wires existentes en el circuito evaluado.

Es posible construir una función que evalúe todas las propuestas de diseño que se generen mientras se encuentra la mejor solución, la cual permita cuantificar la eficiencia y eficacia del circuito evaluado. Esta función se denomina Función Objetivo $f(x)$ o Función de aptitud y se forma con la suma de las salidas correctas $g(x)$ que entregue el circuito comparadas con las de la tabla de verdad, y del número de wires existentes en el circuito $N(x)$, es decir, suma las coincidencias que existen entre las salidas del circuito y las salidas de la tabla de verdad del problema a resolver. Un wire representa la ausencia de una compuerta, entre más wires tenga el circuito mejor es la solución. No basta con encontrar circuitos lógicos que entreguen a la salida todas las combinaciones que exija la tabla de verdad del problema, se busca optimizar el tamaño del circuito reduciendo la mayor cantidad posible de componentes, sin que esto implique perder al circuito factible. $N(x)$ sirve para diferenciar dos circuitos que tienen la misma funcionalidad y asigna un valor mayor a aquel que tiene un menor tamaño, la optimización comienza desde la exploración de soluciones candidatas. El valor de C se obtiene mediante $(n_r)(n_c)+1$ y varia de acuerdo al tamaño de la tabla de verdad del problema que se este resolviendo.

Las variables x de entrada pueden ser de dos tipos: conexiones y compuertas. La función $N(x)$ regresa el número de wires encontrados. La función $f(x)$ regresa el número de coincidencias existentes entre las salidas del circuito y la tabla de verdad.

Capítulo 4

Propuesta de solución

Con el propósito de aproximar la solución al problema de optimización que se plantea en el *Capítulo 3* es necesario el uso de algún método de optimización tal como un Algoritmo Genético (AG).

En este trabajo se propone aproximar la solución por medio de un Algoritmo Genético simple usando codificación entera, que maximiza la función 3.1. El pseudo-código del AG utilizado en esta tesis se describe en la Tabla 4.1.

El AG parte del principio Darwiniano de Selección Natural, en el cual sobrevive el individuo más apto, para ello se requiere una población inicial X_0 de tamaño $npop = 2n$ donde: $n \in \mathbb{N}$. Para el problema en cuestión un ejemplo de población se muestra en la Figura 4.1.

La población está formada por un grupo de individuos con características en común, tales como los mostrados en las Figuras 3.2 y 3.3, para este problema las características de los individuos corresponden a las conexiones y compuertas, que a su vez son las variables de decisión del problema de optimización.

A la población se le aplican las siguientes operaciones: selección, cruza, mutación y elitismo, lo cual generará una población que remplazará a la existente formando la

Parámetros de usuario:	
$npop$: Tamaño de la población X_t	
pc : Probabilidad de cruza $0 \leq pc \leq 1$	
pm : Probabilidad de mutación $0 \leq pm \leq 1$	
ng : Número de generaciones	
np : Número procesos ejecutados	
Notación:	
X_t : Matriz de individuos	
F_t : Vector de valores de aptitud de cada individuo evaluado	
S_t : Subconjunto de individuos seleccionados a partir de X_t	
$\hat{X}_t$: Matriz de individuos después de la cruza	
X_{t+1} : Matriz de individuos después de la mutación	
X_{mejor} : Individuo con mayor calor de aptitud	
Algoritmo:	
1	$t = 0$
2	Generar población inicial X_t
3	Repetir:
4	$F_t \leftarrow evaluar(X_t)$
5	$S_t \leftarrow seleccion(X_t, F_t)$
6	$\hat{X}_t \leftarrow cruza(S_t)$
7	$X_{t+1} \leftarrow mutacion(\hat{X}_t)$
8	$[x_{mejor}, X_{t+1}] \leftarrow elitismo(X_t, F_t)$
9	Hasta que se cumpla criterio de paro
10	Regresar x_{mejor}

Tabla 4.1: Algoritmo Genético simple.

siguiente generación t de individuos. Se espera que a través de las generaciones se obtenga un mayor valor de aptitud, tanto para la población como para el individuo elite. Los operadores genéticos aplicados se describen a continuación.

La *Evaluación* (ver línea 4 de la Tabla 4.1)se realiza obteniendo el valor de la función objetivo $f(x)$ de cada elemento de la población X_t, y devuelve los valores de la evaluación en un arreglo F_t de la misma longitud que X_t, es decir $npop$, F_t guarda el valor de aptitud de cada uno de los individuos existentes en la población. En la evaluación se aplica a cada una de las compuertas que forman al individuo su correspondiente tabla de

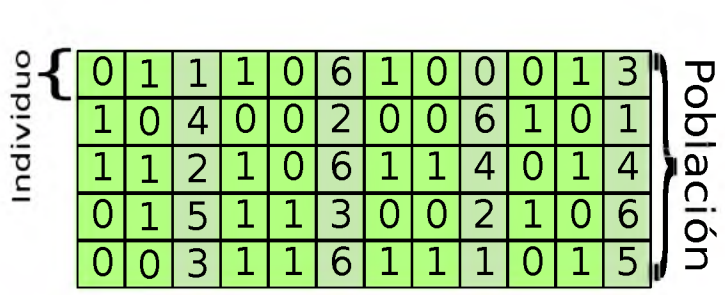

Figura 4.1: Ejemplo de una población de circuitos lógicos utilizando codificación entera

verdad, mismas que se muestran en la Tabla 3.

La *Selección* (ver línea 5 de la Tabla 4.1) se realiza por tres métodos diferentes: Truncamiento, Torneo Binario y Proporcional. Este operador nos permite elegir $npop/2$ elementos existentes en la población X_t de entre los más aptos. Los diferentes tipos de selecciones utilizados en esta tesis son:

- En la selección por *Truncamiento* se ordena el vector de valores de aptitud F_t, ver Ecuación 4.1 y se obtiene su mediana θ; quedan seleccionados todos aquellos individuos S_t cuyo valor de aptitud sea mayor o igual a la mediana, tal como se indica en la Ecuación 4.2.

$$\begin{gathered} I = \{0, 1, 2, 3, ..., n\} \\ Donde: \\ I: \ son\ los\ valores\ de\ indices\ del\ vector\ F\ ordenados \\ F_{I_1}: \ es\ el\ valor\ menor \\ F_{I_n}: \ es\ el\ valor\ mayor \\ \theta = F_{I_{n/2}} \end{gathered} \tag{4.1}$$

$$Truncamiento \begin{cases} Si\ f(x_i) > \theta, entonces\ Individuo\ i\ es\ seleccionado \\ Si\ f(x_i) \leq \theta, entonces\ Individuo\ i\ no\ es\ seleccionado \end{cases} \tag{4.2}$$

- En la selección por *Torneo Binario* compiten entre si dos subconjuntos de elementos del vector F de tamaño $npop/2$, elegidos y comparados de forma aleatoria. Después de haberlos comparado se seleccionan los individuos ganadores S_t, ver Ecuación 4.3.

$$Torneo \begin{cases} Si\ f(x_i) > f(x_j), entonces\ el\ Individuo\ i\ es\ seleccionado \\ De\ lo\ contrario\ el\ Individuo\ j\ es\ seleccionado \end{cases} \tag{4.3}$$

- En la selección *Proporcional* o de ruleta, se asigna una probabilidad P_i a cada individuo, esa probabilidad se calcula mediante la Ecuación 4.4. Utilizando esta selección se pretende que los mejores individuos S_t sean elegidos con una mayor probabilidad.

$$Proporcional \begin{cases} P_i = \dfrac{f(x_i)}{\sum_{j\geq 1}^{n} f(x_j)} \end{cases} \tag{4.4}$$

Únicamente se puede elegir una selección a la vez, cada una de ellas tiene efectos diferentes en el comportamiento de la población durante el algoritmo, en el *Capítulo 5* se muestran resultados obtenidos con los 3 tipos de selección implementados (Truncamiento, Torneo Binario y Proporcional).

La *Cruza* (ver línea 6 de la Tabla 4.1) se realiza después de haber elegido a los S_t elementos representativos de la población X_t por cualquiera de los 3 tipos de selección mencionados anteriormente. Este operador realiza una combinación de características (conexiones y compuertas). La idea principal es crear hijos a partir de los elementos seleccionados, los cuales formarán una población $\hat{X}_t$, la forma de elegir que elementos se cruzan depende de una probabilidad de cruza pc y está definida por el usuario.

La cruza que se ha implementado se denomina cruza de n puntos, para ello se requiere de un número aleatorio $r \in M^n donde : M = \{x | x \in \mathbb{N}, x < nc \cdot nr\}$. En la Figura 4.2 se muestra como ejemplo una cruza de $n = 1$ punto, es decir, ambos individuos son

divididos por un solo punto r cuyo posible valor está en: $0 < r < 4$; esto indica que para poder efectuar la cruza se debe tomar al menos una compuerta con sus respectivas conexiones de cada uno de los padres.

En la Figura 4.2 se distinguen 2 padres divididos en 4 segmentos, r indica el punto donde se realiza el corte para dividir a los padres y formar a los hijos. Para ese ejemplo el corte se realiza exactamente a la mitad del individuo, es decir en $r = 2$, así, el Padre 1 hereda al Hijo 1 la mitad de su información genética ubicada en r_0 y r_1 y al Hijo 2 su información restante ubicada en r_2 y r_3; de igual manera sucede con el Padre 2, quien hereda información a sus dos hijos, nótese la diferencia de colores existente entre padres e hijos.

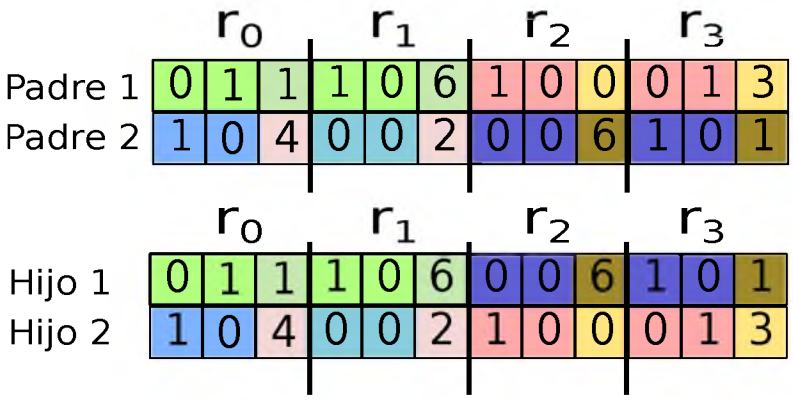

Figura 4.2: Ejemplo de Cruza

La *Mutación* (ver línea 7 de la Tabla 4.1) introduce a la nueva población formada en la cruza $\dot{X}_t$ características que no existían en los padres, para este caso la mutación consiste en alterar una compuerta y sus respectivas conexiones de manera aleatoria, la forma de elegir que elementos mutan depende de una probabilidad de mutación pm y está definida por el usuario.

En la Figura 4.3 se muestra un ejemplo de mutación, en la parte superior se observa un individuo tomado de la población $\dot{X}_t$, al que de acuerdo a un valor $r = 2$ se le modifican sus conexiones y su compuerta. Nótese el cambio de color en ambos individuos, esta diferencia de color indica cuales compuertas y conexiones han sido cambiadas. El

propósito de la mutación es explorar otras características inexistentes en la población para formar nuevas soluciones candidatas.

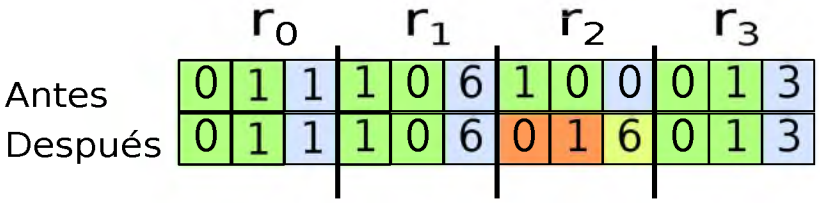

Figura 4.3: Ejemplo de Mutacion

El *Elitismo* (ver línea 8 de la Tabla 4.1) consiste en elegir al elemento con mayor valor de aptitud x_{mejor} para introducirlo en la nueva población X_{t+1}, este individuo es el mejor circuito encontrado durante la exploración de soluciones candidatas; es necesario conservarlo debido a que durante la cruza y la mutación pudieron haberse formado circuitos que no necesariamente son mejores que él y se corre el riesgo de perderlo, para que lo anterior no suceda, se introduce x_{mejor} en un lugar de la población elegido de forma aleatoria.

El *Reemplazo* no es más que sustituir a la población X_t por la X_{t+1} después de haber aplicado los operadores genéticos correspondientes.

El algoritmo se detiene cuando se cumple el criterio de paro, es decir, cuando se han alcanzado todas las generaciones ng que el usuario a definido, el mejor circuito encontrado es x_{mejor}.

La evaluación de la función de aptitud consume aproximadamente el 90 % del tiempo que tarda en explorarse cada generación, lo cual representa un problema en tiempo de ejecución del algoritmo; una de las características más importantes de los AGs es que son altamente paralelizables, esto facilita la implementación de un Algoritmo Genético Paralelo, cuyo pseudo-código se muestra en la Tabla 4.2.

Se ha decidido paralelizar únicamente la evaluación por ser la parte del algoritmo que

más recursos de cómputo consume, debido a que la forma de paralelizar es en memoria distribuida por medio de paso de mensajes (OpenMPI) no es factible paralelizar el algoritmo completo, el intercambio de mensajes entre procesos tiene un costo computacional alto y solo debe usarse cuando este costo sea mayor que realizar la misma operación de manera serial. El algoritmo está diseñado para trabajar con un número de procesos $2 \leq np \leq npop$.

A diferencia del AG simple, el AG Paralelo necesita una semilla para sincronizar la función que genera valores pseudoaleatorios en todos los procesos (ver líneas 2, 4, 20 y 21 de la Tabla 4.2), razón por la cual el proceso 0 envía a cada uno de los demás procesos la semilla que ha generado (ver línea 6 de la Tabla 4.2), cada proceso debe generar la misma secuencia de números aleatorios.

En todos los procesos se crea la misma población inicial, el proceso 0 se encarga de distribuir al resto de los procesos su porción de índices $J_{t,i}$ que debe evaluar (ver línea 10 de la Tabla 4.2), F_t es un vector que se crea a partir de las evaluaciones realizadas en $F_{t,i}$, las cuales son devueltas al proceso 0 para realizar la selección (ver línea 26 de la Tabla 4.2). Para que los demás procesos sepan cuales son los individuos seleccionados S_t se les envía un vector de los índices L_t (ver línea 15 de la Tabla 4.2), esto con el fin de enviar la menor cantidad de información posible entre procesos.

En resumen, todos los procesos excepto el proceso 0 hacen evaluación (ver línea 25 de la Tabla 4.2), este se encarga de hacer la selección (ver línea 13 de la Tabla 4.2) y absolutamente todos realizan cruza, mutación, elitismo y reemplazo.

4.1. Propuesta de modificación al método de Selección

Con la finalidad de mejorar los resultados que proporciona un Algoritmo Genético simple, se propone una forma distinta de realizar la selección basada en el AG simple de la Tabla 4.2, esta consiste en realizar un remuestreo sobre la población original una cantidad rm veces, donde $rm \in \mathbb{N}$, es decir, se obtiene una población de tamaño $npop \cdot rm$

para aplicarle los operadores genéticos (cruza, mutación, etc). Se evalúa la población para obtener un vector F_t de valores de aptitud (ver línea 10 de la Tabla 4.4), para realizar la cruza y la mutación se toma la población de tamaño $npop \cdot rm$ (ver líneas 13 y 14 de la Tabla 4.4), con ello se obtiene una nueva población $\hat{Y}_t$.

La idea de esta modificación es utilizar los operadores de cruza y mutación para explorar el espacio de búsqueda y posteriormente aplicar el sesgo para mantener las soluciones en las regiones más prometedoras del espacio de búsqueda conocidas. En contraste con la selección anterior, aquí primero se perturba la población y luego se realiza la selección, las soluciones seleccionadas con mayor probabilidad serán las vecinas de las mejores soluciones conocidas, la vecindad la define la similaridad de conexiones y compuertas. Observe que la población después de perturbada debe ser de mayor tamaño ya que si las poblaciones son del mismo tamaño no se aplica ningún sesgo.

Ese esquema es muy parecido a un tipo de algoritmo denominado estrategias evolutivas [18, 9]. La diferencia principal con este tipo de algoritmos es la forma en que se seleccionan los individuos, primero estimando el valor de la función objetivo de la población perturbada sin necesidad de ser evaluada, con el valor estimado de la función objetivo se calcula un valor de probabilidad con el cual se realiza la selección. Se ha demostrado que si una población infinita se distribuye de acuerdo a las probabilidades utilizadas en la Tabla 4.3 se puede garantizar la convergencia al óptimo [33].

La selección se realiza eligiendo a los individuos de acuerdo a una probabilidad p^S [23], esta varia de acuerdo al tipo de selección utilizada (Truncamiento, Torneo Binario o Proporcional) ver Tabla 4.3, de igual forma que en el AG simple, se realiza el elitismo y remplazo de la población al final de cada generación. En la Tabla 4.4 se muestra el pseudocódigo de la propuesta de modificación al algoritmo de la Tabla 4.2.

Se espera mediante esta propuesta encontrar de una forma eficaz y eficiente la solución al problema que se ha abordado, utilizando para ello una cantidad menor de evaluaciones de las que necesita el AG simple.

Parámetros de usuario:

$npop$: Tamaño de la población X_t
pc : Probabilidad de cruza $0 \leq pc \leq 1$
pm : Probabilidad de mutación $0 \leq pm \leq 1$
ng : Número de generaciones
np : Número procesos
$proceso$: Proceso en ejecución

Notación:

X_t : Matriz de individuos
F_t : Vector de valores de aptitud de cada individuo evaluado
S_t : Subconjunto de individuos seleccionados a partir de X_t
$\hat{X}_t$: Matriz de individuos después de la cruza
X_{t+1} : Matriz de individuos después de la mutación
X_{mejor} : Individuo con mayor valor de aptitud
$J_{t,i}$: Vector de índices de la Población X_t que evalúa el proceso i
L_t : Vector de índices seleccionados del subconjunto S_t
$F_{t,i}$: Vector de valores de aptitud correspondiente a los índices $J_{t,i}$

Algoritmo:

```
1   t = 0
2   Si proceso = 0
3       semilla ← generarSemilla()
4       pseudoAleatorioInit()
5       Para i desde 1 a np − 1 :
6           enviar(semilla, i, etiqueta)
7       Generar población inicial X_t
8           Para k desde 0 a ng − 1 :
9               Para i desde 1 a np − 1 :
10                  enviar(J_{t,i}, i, etiqueta)
11              Para i desde 1 a np − 1 :
12                  F_{t,i} ← recibir(i, etiqueta)
13              S_t ← seleccion(X_t, F_t)
14              Para i desde 1 a np − 1 :
15                  enviar(L_{t,i}, i, etiqueta)
16              X̂_t ← cruza(S_t)
17              X_{t+1} ← mutacion(X̂_t)
18              [x_mejor, X_{t+1}] ← elitismo(X_t, F_t)
19  De lo contrario:
20      semilla ← recibir(i, etiqueta)
21      pseudoAleatorioInit()
22      Generar población inicial X_t
23      Para k desde 0 a ng − 1 :
24          J_{t,i} ← recibir(0, i, etiqueta)
25          F_{t,i} ← evaluar(X_t, J_{t,i})
26          enviar(F_{t,i}, i, etiqueta)
27          L_{t,i} ← recibir(0, i, etiqueta)
28          X̂_t ← cruza(S_t)
29          X_{t+1} ← mutacion(X̂_t)
30          [x_mejor, X_{t+1}] ← elitismo(X_t, F_t)
31  Regresar x_mejor
```

Tabla 4.2: Algoritmo Genético Paralelo.

Probabilidades para la Selección
Truncamiento $p^S(x,t) = \begin{cases} \frac{p(x,t)}{\alpha(t)} & \text{if } f(x) \geq \theta_t \\ 0 & \text{otherwise} \end{cases}$
Proporcional $p^S(x,t) = \frac{f(x)p(x,t)}{E(t)}$
Torneo Binario $p^S(x,t) = 2p(x,t) \int_{f(y) \leq f(x)} p(y,t)dy$

Tabla 4.3: Probabilidades para los diferentes tipos de Selección implementados

Parámetros de usuario:

$npop$: Tamaño de la población X_t
pc : Probabilidad de cruza $0 \leq pc \leq 1$
pm : Probabilidad de mutación $0 \leq pm \leq 1$
ng : Número de generaciones
np : Número procesos
rm : Remuestreo, $rm \in \mathbb{N}$
$proceso$: Proceso en ejecución

Notación:

X_t : Matriz de individuos
Y_t : Matriz de individuos con remuestreo
F_t : Vector de valores de aptitud de cada individuo evaluado
S_t : Subconjunto de individuos seleccionados a partir de Y_t
$\hat{Y}_t$: Matriz de individuos después de la cruza
Y_{t+1} : Matriz de individuos después de la mutación
X_{mejor} : Individuo con mayor valor de aptitud
$J_{t,i}$: Vector de índices de la Población X_t que evalúa el proceso i
L_t : Vector de índices seleccionados del subconjunto S_t
$F_{t,i}$: Vector de valores de aptitud correspondiente a los índices $J_{t,i}$

Algoritmo:

```
1   t = 0
2   Si proceso = 0
3       semilla ← generarSemilla()
4       pseudoAleatorioInit()
5       Para i desde 1 a np − 1 :
6           enviar(semilla, i, etiqueta)
7       Generar población inicial X_t
8           Para k desde 0 a ng − 1 :
9               Para i desde 1 a np − 1 :
10                  enviar(J_{t,i}, i, etiqueta)
11              Para i desde 1 a np − 1 :
12                  F_{t,i} ← recibir(i, etiqueta)
13              Y_t ← cruzaConRemuestreo(X_t)
14              Ŷ_t ← mutacion(Y_t)
15              F̂_{y_t} ← estimacionFunObj(Y_t, X_t)
16              X_{t+1} ← seleccion ∗ (Ŷ_t, F_{y_t})
17              Para i desde 1 a np − 1 :
18                  enviar(L_{t,i}, i, etiqueta)
19              [x_mejor, X_{t+1}] ← elitismo(X_t, F_t)
20  De lo contrario:
21      semilla ← recibir(i, etiqueta)
22      pseudoAleatorioInit()
23      Generar población inicial X_t
24      Para k desde 0 a ng − 1 :
25          J_{t,i} ← recibir(0, i, etiqueta)
26          F_{t,i} ← evaluar(X_t, J_{t,i})
27          enviar(F_{t,i}, i, etiqueta)
28          Y_t ← cruzaConRemuestreo(X_t)
29          Ŷ_t ← mutacion(Y_t)
30          L_{t,i} ← recibir(0, i, etiqueta)
31          [x_mejor, X_{t+1}] ← elitismo(X_t, F_t)
32  Regresar x_mejor
```

Tabla 4.4: Propuesta de Algoritmo Genético Paralelo con modificación en la Selección.

Capítulo 5

Experimentos y resultados

Como se ha mencionado en Capítulo 2, esta tesis utiliza los mismos experimentos que se usan en el libro: Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering [32], por ser un trabajo reciente que condensa y compara las propuestas en el estado del arte de este problema.

Para realizar toda la experimentación se tomaron como referencia 11 problemas, cuyas tablas de verdad se muestran en las Tablas 5.1, 5.2 y 5.3.

Se ha ejecutado la aplicación 1215 veces, utilizando las 3 selecciones (Truncamiento, Torneo Binario y Proporcional) con probabilidades de Cruza y Mutación como: $pc = \{0.1, 0.2, 0.3, ..., 0.9\}$ y $pm = \{0.1, 0.2, 0.3, ..., 0.9\}$, buscando un circuito de tamaño máximo 16 compuertas $nc = 4 \cdot nr = 4$, un población $npop = 500$ y $ng = 800$ generaciones; se realizaron 5 corridas para cada de las 243 posibles combinaciones de parámetros. A partir de lo anterior se obtuvieron las combinaciones de parámetros que se muestran en la Tabla 5.4, con estas se han realizado todos los experimentos de la tesis.

E0	E1	E2	E3	S0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Experimento 1

E0	E1	E2	E3	S0
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Experimento 2

E0	E1	E2	E3	E4	S0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Experimento 3

E0	E1	E2	E3	S0	S1
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	1
1	1	1	1	0	1

Experimento 4

Tabla 5.1: Tablas de verdad para los Experimentos 1-4

E0	E1	E2	E3	S0	S1	S2
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Experimento 5

E0	E1	S0	S1	S2	S3
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Experimento 6

E0	E1	E2	E3	S0
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Experimento 7

E0	E1	E2	E3	S0
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Experimento 8

Tabla 5.2: Tablas de verdad para los Experimentos 5-8

E0	E1	E2	S0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0
Experimento 9			

E0	E1	E2	E3	S0
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0
Experimento 10				

E0	E1	E2	E3	S0
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1
Experimento 11				

Tabla 5.3: Tablas de verdad para los Experimentos 9-11

Selección	Probabilidad de Cruza	Probabilidad de Mutación
Truncamiento	0.5	0.9
Torneo Binario	0.4	0.9
Proporcional	0.5	0.9

Tabla 5.4: Parámetros elegidos para la experimentación

Con los parámetros de la Tabla 5.4 se ha realizado toda la experimentación, en la cual se obtuvieron los 11 circuitos lógicos que aparecen en las Figuras 5.1, 5.2, 5.3 utilizando un Algoritmo Genético Paralelo de cardinalidad n.

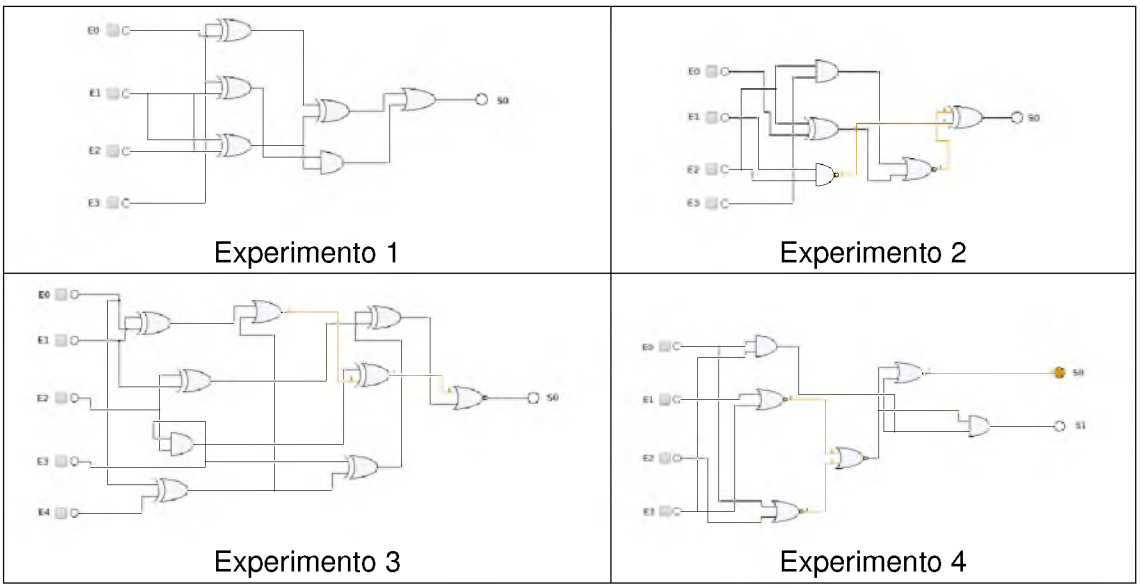

Figura 5.1: Mejores circuitos obtenidos en los experimentos 1-4

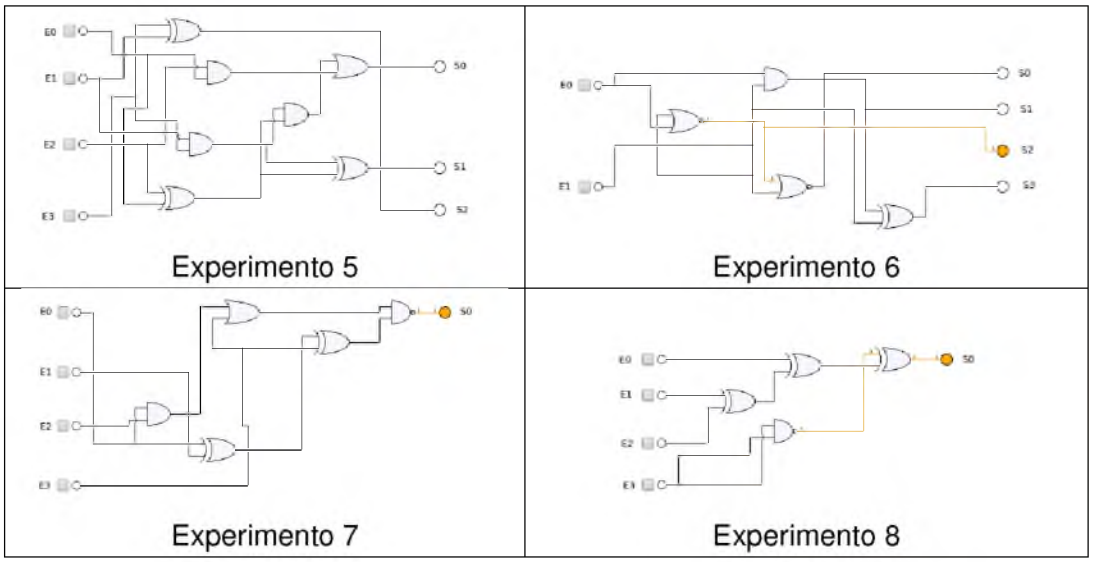

Figura 5.2: Mejores circuitos obtenidos en los experimentos 5-8

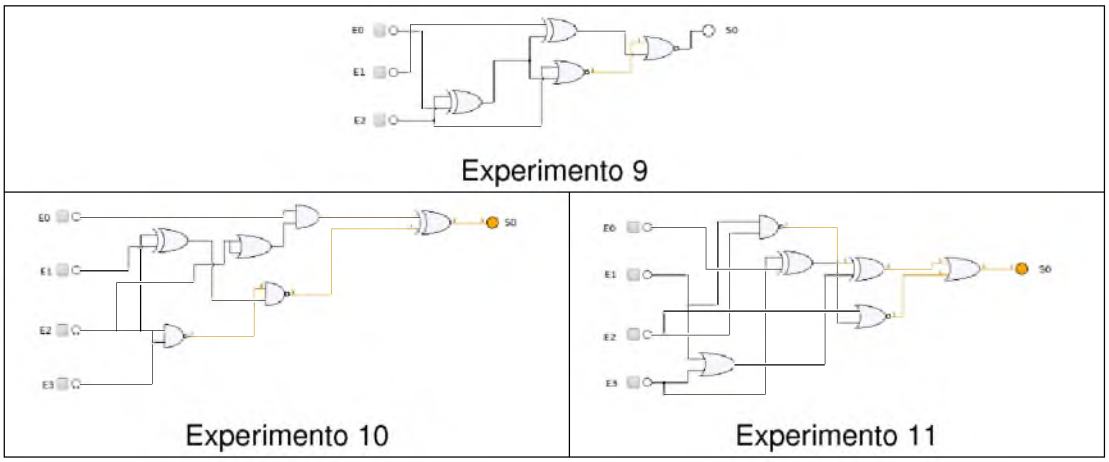

Figura 5.3: Mejores circuitos obtenidos en los experimentos 9-11

La Tabla 5.5 es una compilación de resultados existente en la literatura [32], utilizando las tablas de verdad de los 11 experimentos mostrados en las Tablas 5.1, 5.2 y 5.3. Proporciona una visión general de cómo un mismo problema puede ser resuelto utilizando diferentes algoritmos, cada uno de los cuales intenta encontrar las mejores soluciones. Los resultados obtenidos en esta tesis se muestran en la columna NGA*; nótese que para los experimentos 1, 2, 5, 8 y 9 se logró igualar los resultados que proporcionaron otros algoritmos y para el resto de experimentos se consiguió mejorar los resultados existentes en la literatura.

Function	Bayesian	Polytree	PSO	Ant System	NGA	NGA*	BGA	HD
1	6	6	6	UN	UN	6	UN	UN
2	5	5	5	UN	UN	5	UN	UN
3	10	12	0*	UN	UN	9	UN	UN
4	7	7	7	UN	UN	6	UN	UN
5	8	7	7	UN	UN	7	UN	UN
6	5	6	UN	5	UN	4	UN	UN
7	6	6	UN	7	UN	5	UN	UN
8	4	4	UN	4	UN	4	UN	UN
9	4	4	UN	4	4	4	6	6
10	7	UN	UN	UN	8	6	8	11
11	8	UN	UN	UN	7	6	8	9

Tabla 5.5: Número de compuertas de las mejores soluciones obtenidas en cada uno de los algoritmos (* = inalcanzable, UN = No disponible, Bayesian = Redes Bayesianas, PSO = Enjambre de partículas, Ant System = Colonia de hormigas, NGA = Algoritmos Genéticos de cardinalidad n, NGA* = Algoritmos Genéticos de cardinalidad n (esta tesis), BGA = Algoritmos genéticos de codificación binaria, HD = Diseñador humano)

La eficacia de la aplicación se puede observar en la Tabla 5.6, en ella se expresan los porcentajes de las corridas realizadas para cada una de las selecciones implementadas y en términos generales, para este problema eficacia es equivalente a encontrar un circuito que satisface una determinada tabla de verdad sin importar que tenga un número elevado de compuertas, la eficiencia puede tomarse como el circuito con el menor número posible de compuertas y que además se encuentre en pocas generaciones.

% / Experimento	FTrunc	OTrunc	FTor	OTor	FProp	OProp	FAlg	OAlg
Experimento 1	100	60	100	50	100	50	100	53.3
Experimento 2	90	30	100	30	60	15	83.3	25
Experimento 3	5	0	5	5	0	0	3.3	1.6
Experimento 4	100	65	100	45	55	0	85	36.6
Experimento 5	15	5	35	10	0	0	16.6	5
Experimento 6	100	55	100	45	100	5	100	35
Experimento 7	55	20	75	50	40	5	56.6	25
Experimento 8	100	100	100	100	100	90	100	96.6
Experimento 9	100	100	100	100	100	70	100	90
Experimento 10	25	5	15	5	5	0	15	3.3
Experimento 11	30	15	55	25	25	5	36.6	15
Totales	65.45	41.36	71.36	42.27	53.18	21.81	63.33	35.15

Tabla 5.6: Resultados de la experimentación expresado en porcentajes(FTrunc = Circuitos factibles usando Truncamiento, OTrunc = Circuitos óptimos usando Truncamiento, FTor = Circuitos factibles usando Torneo Binario, OTor = Circuitos óptimos usando Torneo Binario, FProp = Cicuitos factibles usando Proporcional, OProp = Circuitos óptimos usando Proporcional, FAlg = Circuitos factibles encontrados por la aplicación, OAlg = Circuitos óptimos encontrados por la aplicación)

Analizando los porcentajes de la Tabla 5.6 es posible observar que el Experimento 8 ha sido el más fácil de resolver, se encontró el circuito factible el 100 % de las veces y el circuito óptimo fue hallado un 96.6 % de veces. Por otra parte, el Experimento 3 ha sido el más difícil de resolver, se encontró el circuito factible 3.3 % de veces y el circuito óptimo un 1.6 % de las veces que se ejecutó la aplicación.

También es posible observar que la Selección por Torneo Binario es la que mejores resultados ha registrado, la diferencia entre ésta y la Selección por Truncamiento es mínima, pero entre ambas y la Selección Proporcional si es considerable.

5.1. Resultados preliminares de la modificación al operador de selección

Utilizando la modificación que se realizó al operador de selección, se espera que se igualen o mejoren los resultados mostrados en la Tabla 5.6. Debido al tiempo disponible para realizar la fase de experimentación, solo se ha podido mostrar los resultados obtenidos para el Experimento 1, los cuales se muestran en la Tabla 5.7.

% / Experimento	FTrunc	OTrunc	FTor	OTor	FProp	OProp	FAlg	OAlg
Experimento 1	100	100	100	40	100	20	100	53.3

Tabla 5.7: Resultados preliminares de la experimentación utilizando la propuesta de selección expresado en porcentajes(FTrunc = Circuitos factibles usando Truncamiento, OTrunc = Circuitos óptimos usando Truncamiento, FTor = Circuitos factibles usando Torneo Binario, OTor = Circuitos óptimos usando Torneo Binario, FProp = Cicuitos factibles usando Proporcional, OProp = Circuitos óptimos usando Proporcional, FAlg = Circuitos factibles encontrados por la aplicación, OAlg = Circuitos óptimos encontrados por la aplicación)

Si se comparan los resultados del Experimento 1 de la Tabla 5.6 y la Tabla 5.7 se observa que utilizando la selección por Truncamiento mejoró considerablemente la obtención del circuito óptimo, pasó de un 60 % a un 100 %, para Torneo binario y Proporcional no hubo mejora, puesto que bajó el porcentaje un 10 % y 30 % respectivamente. Los circuitos factibles fueron hallados el 100 % de las veces con las 3 selecciones.

Cabe mencionar que esto es únicamente para un experimento, hay que realizar el resto de la experimentación para ver el comportamiento del algoritmo.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

Durante el desarrollo de la tesis se ha buscado cumplir todos y cada uno de los objetivos propuestos, afortunadamente se han alcanzado todos los objetivos satisfactoriamente.

Después de haberse realizado la experimentación es posible concluir que se ha logrado reproducir lo que actualmente existe en estado del arte. Los resultados indican que la aplicación realizada es capaz de diseñar circuitos lógicos combinatorios de una tamaño igual o menor que los que se han reportado en la literatura, lo anterior se indica en la columna NGA* de la Tabla 5.5.

Los circuitos que se encontraron fueron simulados utilizando KtechLab® cumpliendo de manera satisfactoria todas las tablas de verdad, lo cual comprueba la eficacia de la aplicación realizada.

Debido a que Python es un lenguaje interpretado la aplicación requiere demasiado tiempo de ejecución si se compara con otros lenguajes no interpretados como el lenguaje C, sin embargo, Python permitió desarrollar el proyecto en un tiempo relativamente corto. El poder trabajar con un lenguaje de código abierto y que permita desarrollar software

libre, entendiendo lo anterior como la libertad que tienen los demás usuarios y programadores para analizar, modificar y compartir lo que hemos estado realizando. Esto nos permite producir tecnología y ponerla al alcance de todos los interesados sin ninguna finalidad de lucro, con la única intención de compartir conocimiento y generar tecnología propia.

6.2. Trabajo futuro

- Implementar un modulo que a partir de los resultados de la aplicación dibuje el circuito lógico que se ha encontrado. La aplicación actual entrega el resultado en modo texto, los circuitos que se muestran en el Capitulo 5 fueron realizados con KtechLab®, lo ideal es poder hacer este proceso de forma automatizada.
- Realizar una interfaz gráfica mediante la cual el usuario pueda introducir los parámetros de una manera sencilla. La aplicación que se ha realizado se ejecuta en modo consola, desde ahí se introduce cada uno de los parámetros requeridos por el algoritmo, una interfaz gráfica simplificaría el uso de la aplicación.
- Concluir la fase de pruebas para todos los experimentos utilizando la propuesta de selección. Debido al tiempo disponible para la realización de la tesis no se pudo completar la experimentación de los 11 problemas aplicando la propuesta de selección, hasta ahora solo conocemos resultados preliminares.
- Realizar una prueba de escalabilidad. Este tipo de pruebas se necesita para verificar el alcance del algoritmo, con ella podremos saber los limites en los cuales la aplicación encontrará soluciones para problemas de mayor tamaño.
- Utilizar la aplicación realizada para resolver otro tipo de problemas. Existe una gran cantidad de problemas que pueden representarse mediante una función lógica. Desde una clave, un imagen, una canción, etc., todo problema que pueda abstraerse hacia una tabla de verdad es resoluble mediante esta aplicación.

Bibliografía

[1] ANGEL, G. D. M. *Python para todos*. 2009.

[2] BONDEN, M. A. *Artificial Intelligence*. Academic Press Inc., 2006, p. 376.

[3] BONNOT, N., SEULIN, R., AND MARIENNE, F. Machine vision system for surface inspection on brushed industrial parts. *Proc. SPIE 5303* (1978), 126–143.

[4] BUCHANAN, B., AND SHORTLIFFE, E. *Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984.

[5] CENTRO INFORMÁTICO CIENTÍFICO DE ANDALUCÍA, CONSEJERIA DE ECONOMÍA, INNOVACIÓN Y CIENCIA. Open mpi, 2011.

[6] CHAPMAN, B., JOST, G., AND PAS, R. *Using OpenMP: portable shared memory parallel programming*. No. v. 10 in Scientific and engineering computation. MIT Press, 2007.

[7] COELLO, C., CHRISTIANSEN, A., AND HERNÁNDEZ, A. Towards Automated Evolutionary Design of Combinational Circuits. *Computers & Electrical Engineering 27*, 1 (2000), 1–28.

[8] DARWIN, C., AND LEVINE, G. *The Origin of Species*. Fine Creative Media, Inc., 2006, p. 480.

[9] DEB, K. *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester, Chichester, West Sussex UK , 2001, 2001, ch. 5.

[10] DELGADO, C. A. Síntesis de funciones lógicas mediante un eda poliárbol. Master's thesis, Centro de Investigación en Matemáticas A.C, Octubre 2007.

[11] FLOYD, T. *Fundamentos De Electronica Digital/ Digital Electronic Fundamentals*. Editorial Limusa S.A. De C.V., 1996.

[12] GOLDBERG, D. *Genetics Algorithms in Search, Optimization and Machine Learning.* Addison Wesley., 1989.

[13] HOLLAND, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press., 1975.

[14] JACKSON, P. *Introduction to expert systems*. International computer science series. Addison-Wesley, 1990.

[15] JEFF SQUYRE. What is Open MPI?, 2008.

[16] KOZA, J. *On the programming of computers by means of natural selection*. A Bradford book. MIT Press, 1996.

[17] LUNA, E. H. Diseño de circuitos lógicos combinatorios usando optimización mediante cúmulos de partículas. Master's thesis, Centro de investigación y de estudios avanzados del Instituto Politécnico Nacional, Febrero 2004.

[18] MITCHELL, M. *An Introduction to Genetic Algorithms*. The MIT Press, 1996.

[19] NOCEDAL, J., AND WRIGHT, S. J. *Numerical Optimization*. 1999.

[20] PABLO MUÑOZ. Presentación de openmpi, 2010.

[21] PÉREZ, J. C. Plataforma en java para el diseño de circuitos lógicos combinatorios, experimentando con diferentes operadores genéticos. Master's thesis, Universidad Tecnológica de la Mixteca, Julio 2003.

[22] PERKINS, W. A. A model-based vision system for industrial parts. *IEEE Trans. Computers* (1978), 126–143.

[23] PEA, S. I. V., AGUIRRE, A. H., AND RIONDA, S. B. Approximating the search distribution to the selection distribution in edas. In *GECCO'09* (2009), pp. 461–468.

[24] SANTAMARÍA, T. *Electrónica digital*. Textos docentes / Prensas Universitarias de Zaragoza. Prensas Universitarias de Zaragoza, 2007.

[25] SU, K. *Introducción al estudio de circuitos*. Reverté, 1979.

[26] TOCCI, R., WIDMER, N., AND CÁRDENAS, J. *Sistemas digitales: principios y aplicaciones*. Pearson Education, 2003.

[27] USATEGUI, J. *Electronica Digital Moderna*. Electrónica general. Paraninfo, 1996.

[28] VALDEZ, S., BOTELLO, S., AND HERNÁNDEZ, A. Multiobjective Shape Optimization using Estimation of Distribution Algorithms and Correlated Information. In *Third International Conference in Evolutionary Multicriterion Optimization (EMO 2005)* (2005), Springer-Verlag, pp. 664–676.

[29] VALDEZ, S., BOTELLO, S., AND HERNÁNDEZ, A. Multiobjective Shape Optimization with Contraints based on Estimation Distribution Algorithms and Correlated Information. In *Genetic and Evolutionary Computation Conference (GECCO 2005)* (June 2005), vol. 1, ACM Press, pp. 749–750.

[30] VALDEZ, S., BOTELLO, S., AND HERNÁNDEZ, A. *Numerical Modeling of Coupled Phenomena in Science and Engineering*. Taylor & Francis, Routledge, USA, 2008, 2008, ch. 5.

[31] VALDEZ, S., HERNÁNDEZ, A., AND BOTELLO, S. Estimation of distribution algorithm based on bayesian network for topological structure optimization. In *COMCEV 2008: Avances en Computación Evolutiva: Cuarto Congreso Mexicano de Computación Evolutiva* (2008), pp. 129–134.

[32] VALDEZ, S., HERNÁNDEZ, A., AND BOTELLO, S. *Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering.* IGI Global, E. Cholate Evenue, Suite 200, Hershey PA 17033-1240,USA, 2009, ch. 12: Combinational Circuit Design with Estimation of Distribution Algorithms.

[33] ZHANG, Q., AND MÜHLENBEIN, H. On the Convergence of a Class of Estimation of Distribution Algorithms. *IEEE Transactions on Evolutionary Computation 8*, 2 (April 2004), 127–136.