



INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

**Sistema Inmune Artificial con Población
Reducida para Optimización Numérica**

T E S I S
QUE PARA OBTENER EL GRADO DE
DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN

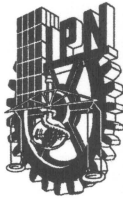
P R E S E N T A
Juan Carlos Herrera Lozada

Directores de Tesis:
Dr. Francisco Hiram Calvo Castro
Dra. Hind Taud



México, D.F.

Junio, 2011



INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 21 del mes de Junio de 2011 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

“SISTEMA INMUNE ARTIFICIAL CON POBLACIÓN REDUCIDA PARA OPTIMIZACIÓN NUMÉRICA”

Presentada por el alumno:

HERRERA

Apellido paterno

LOZADA

Apellido materno

JUAN CARLOS

Nombre(s)

Con registro:

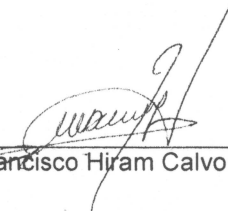
B	0	7	1	2	3	4
---	---	---	---	---	---	---

aspirante de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**


Después de intercambiar opiniones los miembros de la Comisión manifestaron **APROBAR LA TESIS**, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de tesis



Dr. Francisco Hiram Calvo Castro




Dra. Hind Taud



Dr. Jesús Guillermo Figueroa Nazuno

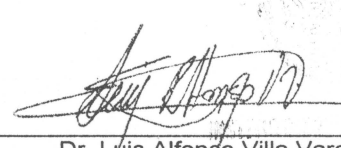


Dr. Marco Antonio Moreno Armendáriz



Dr. Edgar Alfredo Portilla Flores

PRESIDENTE DEL COLEGIO DE PROFESORES



Dr. Luis Alfonso Villa Vargas



INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de *México, D.F.* el día *23* del mes de *Junio* del año *2011*, el que suscribe *Juan Carlos Herrera Lozada*, alumno del Programa de *Doctorado en Ciencias de la Computación* con número de registro *B071234*, adscrito al *Centro de Investigación en Computación*, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del *Dr. Francisco Hiram Calvo Castro* y *Dra. Hind Taud*, cede los derechos del trabajo intitulado *Sistema Inmune Artificial con Población Reducida para Optimización Numérica*, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección *jlozada@ipn.mx*. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Juan Carlos Herrera Lozada

Resumen

En este trabajo de tesis se presenta un nuevo algoritmo, se trata de un micro-sistema inmune artificial (micro-SIA) basado en la teoría de la selección clonal para resolver problemas de optimización numérica. Para este estudio se utilizó el algoritmo CLONALG en su versión estándar que es un sistema inmune artificial ampliamente utilizado para resolver problemas de optimización y de reconocimiento de patrones. Durante la fase de clonación, CLONALG incrementa drásticamente el tamaño de su población, por lo que esta característica es atractiva para proponer una versión con una población de individuos reducida. La hipótesis que se establece es que al reducir el número de individuos en la población se reduce el número de evaluaciones a la función objetivo, con lo que fue posible incrementar la velocidad de convergencia y decrementar el uso de la memoria de datos. El micro-SIA propuesto usa una población de 5 individuos (anticuerpos) de los cuales se obtienen sólo 15 clones. En el proceso de maduración de estos clones, dos simples y rápidos operadores de mutación son diseñados y utilizados dentro de una convergencia nominal que trabaja en conjunto con un proceso de reinicialización para preservar la diversidad. El SIA no utiliza un operador de cruza.

Se realizaron dos versiones del micro-SIA, una que no maneja restricciones y otra que sí permite lidiar con éstas, considerando que en el mundo real la mayoría de los problemas de optimización tienen restricciones. También se presentan dos aproximaciones del micro-SIA embebidas en hardware que se ejecutan intrínsecamente para resolver el problema del conteo de unos, la primera de implemento en un microcontrolador comercial y la segunda en un dispositivo de lógica reconfigurable (FPGA).

Abstract

In this thesis, we present a new algorithm, this is a micro-artificial immune system (micro-AIS) based on the Clonal Selection Theory for solving numerical optimization problems. For our study we consider the algorithm named CLONALG, it is a widely used artificial immune system. During the process of cloning, CLONALG greatly increases the size of its population, so this feature is attractive to propose a version with a reduced population. Our hypothesis is that by reducing the number of individuals in a population will decrease the number of evaluations to the objective function, increasing the speed of convergence and reducing the use of data memory. Our proposal uses a population of 5 individuals (antibodies) which are obtained only 15 clones. In the maturation stage of the clones, two simple and fast mutation operators are used in a nominal convergence that works together with a reinitialization process to preserve the diversity. The SIA does not use a cross operator. To validate our algorithm, we use a set of test functions taken from the specialized literature to compare our approach with the standard version of CLONALG.

We made two versions of the micro-AIS, one that does not handle constraints and one that will allow them to deal with, whereas in the real world most optimization problems have constraints. It also presents two approaches of micro-AIS embedded hardware running intrinsically to solve the maxone problem, first implemented on a commercial microcontroller and the second in a reconfigurable logic device (FPGA).

Índice general

Resumen	i
Abstract	ii
Capítulo 1	1
Introducción	1
1.1 Algoritmo evolutivos y bioinspirados	2
1.2 Hardware evolutivo	6
1.3 Objetivo general	8
1.3.1 Objetivos particulares	8
1.4 Justificación	9
1.5 Contribuciones de la tesis	10
1.6 Organización de la tesis	11
Capítulo 2	13
Sistema Inmune Artificial	13
2.1 CLONALG	19
2.2 AiNet	14
2.3 Estado del arte	21
2.4 Problemática a resolver	26
Capítulo 3	28
Micro-Sistema Inmune Artificial (micro-SIA)	28
3.1 Propuesta generalizada	28
3.2 Micro-SIA para optimización sin manejo de restricciones	31
3.3 Micro-SIA para optimización con manejo de restricciones	34
3.4 Pruebas y resultados experimentales	35
3.4.1 Primera fase de experimentación	35
3.4.2 Segunda fase de experimentación	36

3.4.3 Tercera fase de experimentación	38
3.4.4 Cuarta fase de experimentación	39
Capítulo 4	41
Arquitectura del micro-SIA	41
4.1 Implementación en microcontrolador	43
4.2 Arquitectura en FPGA	46
Capítulo 5	50
Conclusiones y trabajo a futuro	50
Referencias	53
ANEXO A Funciones de prueba para el micro-SIA sin manejo de restricciones	57
ANEXO B Funciones de prueba para el micro-SIA con manejo de restricciones	59
ANEXO C Diseño de un micro-algoritmo de evolución diferencial	60
ANEXO D Una arquitectura bioinspirada para un módulo PWM utilizando la hibridación artificial de cadenas de ADN	67

Índice de figuras

Algoritmo 1.1. Algoritmo evolutivo estándar	4
Figura 1.1. Comportamiento de un algoritmo evolutivo estándar	4
Figura 1.2. Paradigma de la colonia de hormigas	6
Figura 2.1. Niveles de defensa del sistema inmune biológico	14
Figura 2.2. Principio de la selección clonal	16
Algoritmo 2.1. CLONALG.	18
Algoritmo 2.2. AiNet	20
Algoritmo 2.3. Micro-algoritmo genético	22
Figura 2.3 Micro-AG multi-objetivo	23
Figura 3.1 Esquema general de un micro-algoritmo bioinspirado	29
Figura 3.2 Micro-Sistema Inmune Artificial	31
Figura 4.1. Diagrama a bloques del micro-SIA implementado en hardware, para solucionar el problema del conteo de unos	45
Figura 4.2. Reconocimiento de caracteres implementado en el FPGA y que utiliza al micro-SIA para maximización	48

Índice de tablas

Tabla 3.1. Resultados experimentales del micro- sistema inmune artificial para problemas de optimización sin restricciones	36
Tabla 3.2. Resultados experimentales de la comparación entre el CLONALG en su versión estándar y el micro-SIA, ambos casos sin manejo de restricciones	37
Tabla 3.3. Resultados experimentales al variar los parámetros del micro-SIA	38
Tabla 3.4. Resultados experimentales del micro-SIA para problemas de optimización con manejo de restricciones	39
Tabla 4.1. Resultados experimentales del micro-SIA implementado en el microcontrolador para solucionar el problema del conteo de unos	46
Tabla 4.2. Resultados experimentales del micro-SIA implementado en el FPGA para solucionar el problema del conteo de unos	47
Tabla 4.3. Evolución de la solución para un caracter corrupto	48
Tabla 4.4. Resultados experimentales para 5 caracteres corruptos	49

Capítulo 1

Introducción

Los algoritmos con inspiración biológica, también conocidos como algoritmos bioinspirados, han adquirido gran importancia dentro del área de la inteligencia artificial debido a que han demostrado ser exitosos en la solución de ciertos problemas complejos de aprendizaje de máquina, reconocimiento de patrones, detección de fallas y optimización. El sistema inmune artificial (SIA), inspirado por algunos procesos observados principalmente en el sistema inmune biológico de los mamíferos, es un área emergente en los sistemas computacionales adaptativos que incluye algoritmos que han sido empleados satisfactoriamente, entre otros, a problemas de optimización global como combinatoria, tratándose de una técnica heurística relativamente nueva y que ha resultado altamente competitiva. Dentro del sistema inmune biológico, el principio de selección clonal es uno de los procesos más estudiados y que es abstraído directamente como un algoritmo particular del SIA, utilizándose principalmente para resolver problemas de optimización numérica y reconocimiento de patrones.

Al igual que sucede con los algoritmos evolutivos, el principio de selección clonal del SIA es *poblacional*, lo que significa que manipula simultáneamente un conjunto de soluciones potenciales del problema, implicando en la mayoría de los casos un tiempo de ejecución elevado y un amplio uso de la memoria de datos que se traduce en un alto costo computacional. Una alternativa para reducir tal costo ha sido el diseño de algoritmos con poblaciones

extremadamente pequeñas que realizan menos evaluaciones de la función objetivo utilizando un menor espacio en la memoria de datos del sistema de cómputo.

En este trabajo de tesis se presenta un algoritmo con un tamaño de población reducido, basado en el principio de selección clonal del sistema inmune artificial, para solucionar problemas de optimización numérica. El principio de selección clonal extrapolado del sistema inmune biológico, se caracteriza por aumentar el tamaño de la población durante la etapa de clonación y sólo permite aplicar operadores de mutación (no hay operador de cruza), por lo que mantener la diversidad de la población y asegurar la convergencia del algoritmo, son retos que hacen interesante el estudio de esta heurística bioinspirada. Se implementaron dos arquitecturas del micro-algoritmo diseñado, la primera en un microcontrolador comercial y la segunda en un dispositivo de lógica reconfigurable (FPGA). El hardware evolutivo puede ayudar a solucionar los problemas de diseño de circuitos, de tolerancia a fallos y adaptación a entornos cambiantes. Lo anterior se debe a que hardware evolutivo no usa reglas de diseño rígidas, sólo se guía por el comportamiento del circuito y por lo tanto el espacio de soluciones sobre el que trabaja es mayor, además un diseño complejo permite que el sistema se rediseñe de manera autónoma, cuando se presente una falla o bien cuando las características del entorno varíen.

1.1 Algoritmos Evolutivos y Bioinspirados

Las técnicas heurísticas con inspiración biológica, también referidas en la literatura especializada como algoritmos evolutivos y bioinspirados [10], [11], [12] son procedimientos de búsqueda, optimización y aprendizaje, cuyo modelo se obtiene del mecanismo de la selección natural y las teorías de la evolución, basadas en el paradigma *Neo-Darwiniano* que aplica sobre esquemas poblacionales. Estas técnicas son muy populares, debido principalmente a su alto

poder exploratorio y a su paralelismo implícito. El Neo-Darwinismo es el conjunto de teorías que explican el origen de las especies en el planeta y cómo sus procesos de reproducción, mutación, competencia y selección permiten la adaptación y evolución de las especies; las teorías involucradas en este modelo son: el seleccionismo de Weismann, el origen de las especies formulado por Darwin y la genética de Mendel [10]. Todos estos algoritmos tienen en común el hecho de utilizar una población o conjunto de soluciones potenciales, generada comúnmente de manera aleatoria, y someterla a un proceso iterativo utilizando diferentes esquemas, operadores y estrategias en función del tipo de algoritmo.

En la particularidad de los algoritmos evolutivos, los operadores genéticos básicos son tres [11]: la *selección*, la *cruza* (reproducción o recombinación) y la *mutación*. La selección consiste de un mecanismo (puede ser probabilístico o determinista) que permite elegir a los individuos que fungirán como padres de la siguiente generación o iteración. La *cruza* se refiere al intercambio de información (material genético) entre dos padres que han sido seleccionados con base en su aptitud de acuerdo a una función objetivo. El último de los operadores listados, la *mutación*, se encarga de realizar pequeñas perturbaciones o cambios mínimos a los individuos recién creados para la nueva población con la finalidad de explorar zonas del espacio de búsqueda que la *cruza* pudiera no alcanzar, manteniendo la diversidad de los individuos. La secuenciación de los pasos para un algoritmo evolutivo estándar se lista en el algoritmo 1.1.

1. *Inicialización*: Población inicial generada aleatoriamente.
2. *Generación*: Aplicar los siguientes procedimientos evolutivos de adaptación,
 - 2.1. *Selección*: Se seleccionan los individuos que sobrevivirán y se reproducirán, de acuerdo a su aptitud en base a la función objetivo.
 - 2.2. *Reproducción y Variación Genética*: Se crean nuevos individuos por medio de la recombinación y/o introduciendo variaciones genéticas

- (mutación) en individuos seleccionados.
- 2.3. *Evaluación de Aptitud*: evaluar nuevamente la aptitud de cada individuo modificado.
3. *Ciclo*: Repetir el paso 2 hasta que se alcance el criterio de convergencia.

Algoritmo 1.1. Algoritmo evolutivo estándar.

En la Figura 1.1 se describe gráficamente el comportamiento y secuenciación del algoritmo, obsérvese que los operadores genéticos modifican las características de los individuos en el modelo poblacional. Existen básicamente dos etapas de evaluación de la función objetivo, la primera en la *Selección* y posteriormente en la *Reselección* después de haber aplicado los operadores genéticos.

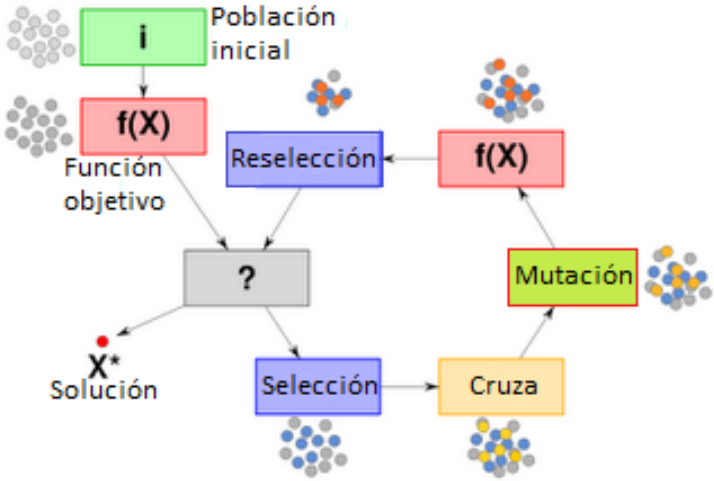


Figura 1.1. Comportamiento de un algoritmo evolutivo estándar.

Han surgido diferentes disciplinas que intentan imitar a través de medios computacionales, los procesos antes indicados, con la iniciativa de resolver problemas complejos NP, lineales y no lineales. Una de estas disciplinas es la denominada *Computación Evolutiva* [12] que conjunta a

su vez a la *Programación Genética*, a los *Algoritmos Genéticos* y a las *Estrategias Evolutivas*, que durante años han representado la viabilidad de utilizar este tipo de recursos para solucionar satisfactoriamente problemas de optimización a un coste computacional razonable. Fogel explica en [11] que los tres paradigmas antes mencionados tienen diferencias entre sí, específicamente en la importancia que cobran los operadores genéticos de cruza y mutación, sin embargo, comparten funciones comunes como la reproducción, la variación aleatoria, el ímpetu para simular la evolución, la competencia y la selección, además de ser algoritmos de naturaleza estocástica.

Otra disciplina que surge como una respuesta al análisis del comportamiento social y cultural de un sistema visto como una colectividad o colonia, es la denominada *Inteligencia de Enjambre* [14] que básicamente trata de explicar cómo se relacionan e intercambian información entre sí los individuos de estas sociedades, con la premisa de resolver una tarea común. La integración y la coordinación de tareas de los individuos no requiere de ningún supervisor, es decir, los individuos resuelven los problemas ligados a su supervivencia de una manera distribuida y paralela. En la optimización con este tipo de técnicas, la población demanda una memoria, refiriéndose al hecho de que el proceso de mejora se dirige y encauza influenciado por la historia grupal, por la memoria de cada individuo y por el estado presente en el que cada uno se encuentra. Aunque la inteligencia de enjambre es aplicable al comportamiento humano, se ha estudiado más a fondo sobre insectos. Los paradigmas más conocidos son la *Colonia de Hormigas* [15] y el *Cúmulo de Partículas* [16], este último inspirado en la coreografía de las parvadas de pájaros o los bancos de peces. Existen algunos otros de reciente creación y estudio, como la *Comunicación entre Luciérnagas* [17]. En la Figura 1.2 se ilustra el paradigma de la Colonia de Hormigas, en donde se emula el comportamiento de las hormigas para encontrar el camino o ruta más corta entre el alimento y su nido, apoyándose de la concentración de feromonas para guiar la búsqueda.

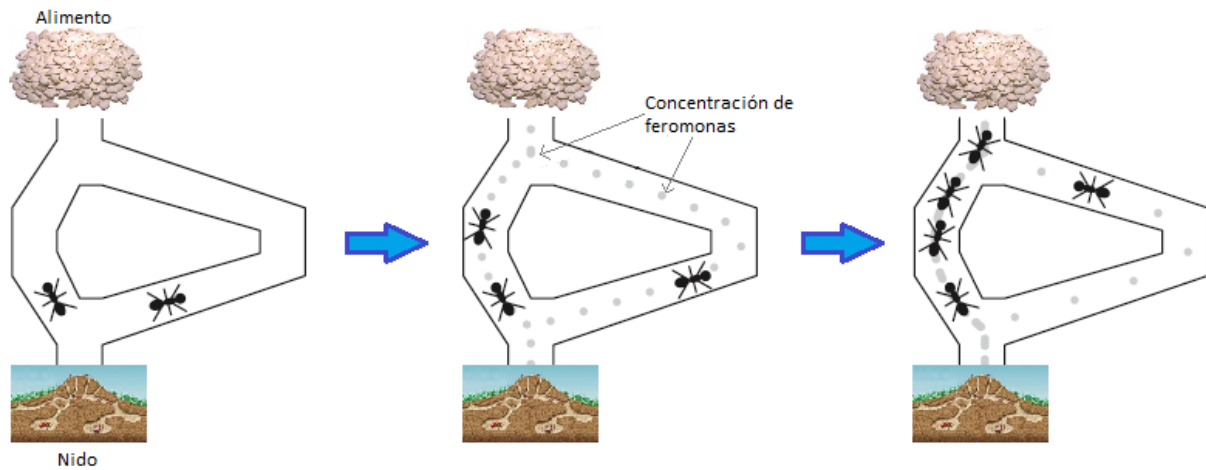


Figura 1.2. Paradigma de la colonia de hormigas.

Dentro de la literatura del área, se pueden encontrar otros modelos bioinspirados que no es sencillo clasificar, por ejemplo, la *Molécula de ADN* que se fundamenta de manera más formal en la llamada *Computación Molecular* y que básicamente ordena secuencias de cadenas de ADN para posteriormente compararlas buscando la hibridación de las mismas cuando sean complementarias [18]. Otro ejemplo son las *Redes Neuronales* que representan un modelo de aprendizaje y procesamiento automático que está inspirado en la forma en que funciona el sistema nervioso [19]. Para algunos autores, como Dasgupta en [20], los sistemas de procesamiento de información inspirados biológicamente pueden clasificarse de la siguiente manera: sistema nervioso cerebral (redes neuronales artificiales), sistemas genéticos (algoritmos evolutivos) y sistemas inmunes (sistema inmune artificial).

1.2 Hardware Evolutivo

Cuando se trasladan las técnicas evolutivas y bioinspiradas a alguna plataforma en Hardware, surge el término de *Hardware Evolutivo* [34, 35, 63]. Es posible percibir dos ramas principales,

concernientes a la operatividad del hardware, la primera de ellas es la reconfiguración automática que persigue una adaptación autónoma del hardware y, por otra parte, la implantación física de algoritmos evolutivos para acelerar algún proceso de búsqueda [64].

En la reconfiguración automática se pretende que un dispositivo electrónico pueda recibir datos del exterior, y de manera autónoma, pueda éste llevar a cabo una recombinación de su estructura interna para adaptarse al medio y mejorar su desempeño.

La implantación de algoritmos, que es la aplicación que se abordará en este trabajo de tesis, surge como una alternativa para acelerar los procesos internos y validar acciones que el hardware debe realizar [18, 36], mejorando considerablemente las soluciones realizadas en software y procesadas a través de una computadora con secuenciación de instrucciones. Las opciones de implantación conllevan aproximaciones electrónicas digitales y analógicas, y en algunos casos, los menos, híbridas que combinan ambas vertientes.

El hardware evolutivo se puede implementar de dos maneras sobre silicio [35]: extrínseca, con actualización fuera de línea, es decir, se simula todo el algoritmo y posteriormente se busca la manera de implantarlo en la plataforma hardware, o bien, de modo intrínseco en donde es posible integrar el algoritmo de forma embebida en el mismo dispositivo, lo cual repercute en una mayor velocidad de procesamiento y se puede hablar de reconfiguración y adaptación al medio de manera autónoma, aunque aún no se han logrado resultados totalmente independientes. Para algunos autores, es posible concebir estructuras híbridas [32].

Los dispositivos de lógica programable, con arquitecturas flexibles y robustas, como por ejemplo los FPGAs (Arreglos de Compuertas Programables en Campo) han sido durante años los recursos más utilizados para probar el funcionamiento de toda clase de algoritmos evolutivos y bioinspirados, en un circuito integrado digital [35, 57, 59, 64]. Lo anterior se debe a la disposición física de su arquitectura, que de manera natural está construida como un arreglo de bloques que pueden interconectarse de manera creciente entre sí para generar lógica

más compleja, aunado al gran número de pines de propósito general. Las aplicaciones relativas son muy variadas, encontrándose desarrollos en robótica [36, 41], en el diseño de filtros digitales [54, 55, 56], y en el reconocimiento de patrones [60], entre otras tantas referidas principalmente a la ingeniería.

La implementación analógica se realiza a nivel sustrato, proponiendo circuitos en base a transistores, resistencias y capacitores, en donde se busca evolucionar estos componentes para diseñar circuitos tolerantes a fallas y autoadaptativos [34, 64, 65].

1.3 Objetivo General

Diseñar un algoritmo basado en el sistema inmune artificial, con un tamaño de población reducido, para solucionar problemas de optimización numérica.

1.3.1 Objetivos Particulares

- Realizar un estudio de los diferentes micro - algoritmos evolutivos y bioinspirados existentes en la literatura especializada, para encontrar las características que permitan diseñar un sistema inmune artificial con población reducida.
- De los procesos biológicos del sistema inmune, se dirigirá el estudio al principio de selección clonal, considerando sus principales características: clonación y mutación.

- Diseñar los operadores de mutación que permitan el funcionamiento correcto del algoritmo.
- Realizar los análisis y pruebas pertinentes que permitan validar el desempeño del algoritmo.
- Dirigir los resultados hacia arquitecturas hardware y plantear posibles aplicaciones futuras.

1.4 Justificación

Existe un interés creciente por utilizar algoritmos evolutivos y bioinspirados para optimizar procesos. La adecuación de las soluciones antes exploradas en software y ahora trasladadas a hardware, deja entrever la inmediata ventaja de la velocidad de procesamiento. Considerando las herramientas actuales en el diseño electrónico, tanto analógico como digital, es posible realizar diversos estudios y abstracciones de las técnicas conocidas. El paralelismo inherente de los algoritmos ha sido ampliamente abordado en el diseño de *Hardware Evolutivo*, especialmente cuando se utiliza un dispositivo lógico programable en donde ha quedado de manifiesto que las velocidades de procesamiento y de convergencia son muy aceptables y no se escatima en la cantidad de recursos dentro de la arquitectura [1],[2],[3]; sin embargo, son muy pocos los estudios que pretenden reducir la arquitectura en función de los módulos lógicos, o bien, que involucren en estas tendencias a los dispositivos comerciales con restricciones de memoria y con ejecución procedural, como sucede con los microcontroladores comerciales.

La hipótesis planteada en esta tesis es que al disminuir la cantidad de individuos en una población, bajo algún criterio determinado, también se reduzca el número de evaluaciones de la función objetivo, permitiendo un uso más eficiente de la memoria de datos, sin afectar de manera importante el funcionamiento del algoritmo. Con base a esta suposición, la literatura especializada reporta algunos mecanismos que buscan reducir la cantidad de evaluaciones de la función objetivo caracterizando alguna variante en el manejo de la población, como sucede con el micro-algoritmo genético [6] que representa nuestra principal referencia y que como se indicará posteriormente utiliza una convergencia nominal, un proceso de reinicialización, considera elitismo y genera ruido estocástico al completar una nueva población con nuevos individuos obtenidos aleatoriamente. Con respecto al sistema inmune artificial para optimización, no existe referencia alguna de algoritmos con población reducida y en consecuencia tampoco se tienen evidencias de implementaciones en hardware, por lo que las aportaciones de esta tesis serán las primeras sobre el tema.

1.5 Contribuciones de la tesis

En este trabajo de tesis, las aportaciones entregadas se dividen en dos rubros: micro-SIA y arquitectura en hardware del micro-SIA.

Con respecto al micro-SIA, las aportaciones son:

Diseño e implementación en software de un micro-SIA, con una población de sólo 5 individuos, para resolver problemas de optimización numérica sin restricciones y con restricciones. Cabe mencionar que no existe en la literatura especializada ningún trabajo que aborde al SIA y que utilice una población reducida en tamaño. Se pudo demostrar que el

algoritmo propuesto presenta un desempeño correcto en comparación a la versión estándar, además se obtuvo un menor costo computacional al utilizar una cantidad menor de la memoria de datos al tratarse de una micro-población.

Con respecto a la arquitectura en hardware del micro-SIA, las contribuciones son:

Una arquitectura simplificada que se implementó intrínsecamente en un microcontrolador comercial y que resuelve satisfactoriamente el problema del conteo de unos (*maxone problem*) como un ejercicio de maximización. El carácter intrínseco está definido por la particularidad de que el algoritmo está embebido en el dispositivo y no sólo recibe datos provenientes de una PC que previamente ejecuta el algoritmo y entrega resultados para configurar el hardware, como sucedería con una implementación extrínseca.

Una arquitectura simplificada y modular que se implementó intrínsecamente en un dispositivo de lógica reconfigurable (FPGA) y que también resuelve satisfactoriamente el problema del conteo de unos. Esta misma arquitectura permitió la implementación de una aplicación de reconocimiento de caracteres en un FPGA.

1.6 Organización de la tesis

El presente documento está organizado en 5 capítulos.

En el Capítulo 1 se expone la introducción al trabajo realizado. También se hace mención de los objetivos general y particulares de la tesis.

El segundo capítulo del documento, Capítulo 2, versa en las generalidades del sistema inmune artificial y se presenta el estado del arte sobre micro-algoritmos.

El Capítulo 3 es en donde se explica nuestra propuesta y cómo funciona. Se hace hincapié en la manera en la que se concibió el micro-algoritmo y cuáles son sus características. En este mismo apartado se incluyen y discuten las pruebas y resultados experimentales.

En el Capítulo 4 se presentan las arquitecturas en hardware que se diseñaron e implementaron para el algoritmo en su versión de población reducida en tamaño. También se incluyen y discuten los resultados experimentales.

El último capítulo, Capítulo 5, es el propio para presentar las conclusiones derivadas de la investigación doctoral y se detallan las propuestas de trabajos a futuro que surgen a partir del desarrollo de esta tesis.

Finalmente se incluyen las referencias estudiadas y se presentan 4 anexos. Los dos primeros anexos, A y B, incluyen el listado de las funciones de prueba utilizadas para los experimentos del capítulo 3. Los anexos C y D presentan dos trabajos realizados durante el desarrollo de esta investigación doctoral y que tienen que ver con el tema: el anexo C muestra una aproximación a un micro-algoritmo de evolución diferencial que es otra heurística bioinspirada diferente al SIA y el anexo D presenta el diseño de una arquitectura bioinspirada para un módulo PWM (modulación por ancho de pulso) basada en la hibridación artificial de cadenas de ADN.

Capítulo 2

Sistema Inmune Artificial

El sistema inmune artificial (SIA) puede procesar información de manera muy significativa siendo un sistema adaptativo, distribuido, paralelo y descentralizado; algunas de sus principales características son: memoria, aprendizaje, robustez, tolerancia a fallas, diversidad y autoregulación [20], [21]. La principal encomienda del sistema inmune en los seres vivos, es mantener al organismo libre de agentes infecciosos extraños a él, así como reparar las células dañadas o eliminarlas en el momento que sea necesario. Los microorganismos *patógenos*, que en su superficie tienen *antígenos*, que penetran al organismo pueden resultar dañinos para éste. Los antígenos son moléculas que pueden ser reconocidas por el sistema inmune y que además son capaces de dar inicio a la respuesta inmune para eliminarlos, propiciando una serie de procesos que neutralizarán y acabarán a los invasores.

El sistema inmune tiene cuatro capas o niveles de defensa principales; cada uno de éstos antepone diferentes tipos de protección para la detección, reconocimiento y respuesta. La primera de estas capas es la *barrera física o anatómica*, como la piel y las mucosas. La segunda está conformada por las *condiciones fisiológicas* como la temperatura y el *ph* del órgano en que se encuentra el antígeno. La tercera capa es la *respuesta innata* (no específica) del sistema inmune, la cual no tiene memoria y permite actuar a las células macrófagas, las cuales ingieren a los antígenos para facilitar su eliminación. Si un antígeno sobrepasa la defensa innata, automáticamente se activa el cuarto nivel de defensa del sistema inmune, conocido como

respuesta adaptativa o aprendida (específica) y es el más estudiado para adecuar este modelo bioinspirado a un modelo computacional [22], [23]. Las células más representativas de esta respuesta son los *linfocitos B* y *T*. Los primeros maduran en la *médula ósea* y los segundos en el *timo*. Los linfocitos B secretan a su vez, otro tipo de células denominadas *anticuerpos*. Ante la presencia de un antígeno, únicamente los linfocitos con anticuerpos que posean la forma específica al antígeno, es decir los más afines, serán estimulados para proceder a la eliminación del antígeno. Los anticuerpos poseen en su estructura externa una región con una forma específica denominada *paratope* que se acopla con su contraparte *epitope* localizada en la estructura del antígeno, para reconocer al invasor. Cada linfocito B posee en su superficie anticuerpos con el mismo tipo de paratope y los antígenos poseen diferentes tipos de epitopes, de manera que un mismo antígeno puede activar varios linfocitos a la vez siendo reconocido por éstos. La Figura 2.1 ejemplifica los cuatro niveles de defensa del sistema inmune biológico.

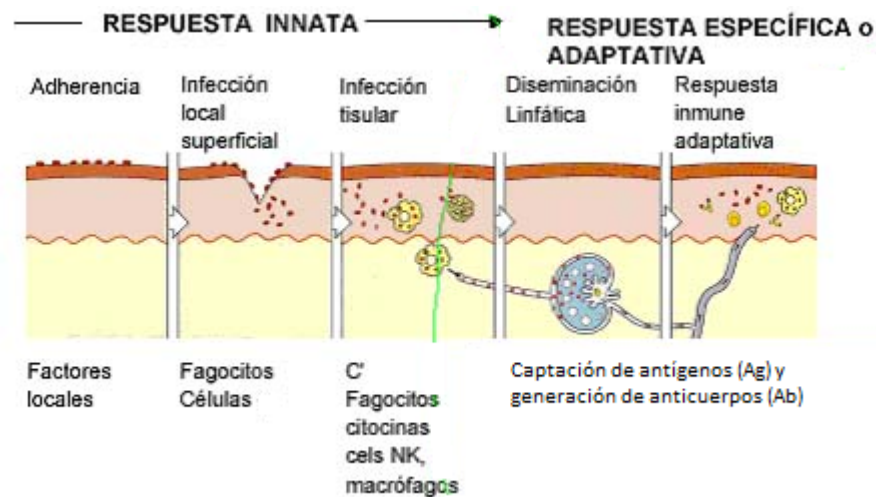


Figura 2.1. Niveles de defensa del sistema inmune biológico.

Nunes de Castro *et al* [21] y Dasgupta [22], coinciden en que debido a su gran complejidad, existen varios modelos del sistema inmune artificial que se abstraen directamente de los mecanismos de respuesta; sin embargo, son tres los modelos más estudiados: *modelos de médula*

ósea y de timo, modelo de la selección clonal y el *modelo de la red inmune*. Los dos últimos modelos, son los que se han utilizado mayormente para resolver problemas de optimización [23],[24], [25], [26], [27] y hacen uso del principio de *selección negativa* (también existe su complemento) que a grandes rasgos hace referencia a la propiedad de exponer a los linfocitos maduros ante un grupo de células propias del organismo, permitiéndoles sobrevivir a aquellos que no presenten ninguna reacción, y en contra sentido, se elimina a los que presenten alguna reacción.

El *modelo de médula ósea* infiere la creación y maduración de los linfocitos detectores del sistema inmune, aumentando su capacidad para poder distinguir entre los agentes externos. El *modelo de timo* se fundamenta en el proceso mediante el cual se crea la población de linfocitos como células detectoras.

El *modelo de selección clonal* emula el proceso mediante el cual el sistema inmune, ante la presencia de un antígeno específico, estimula únicamente a aquellos linfocitos que sean más afines, para después ser clonados y mutados. La Figura 2.2 muestra el principio de la selección clonal, en donde el *anticuerpo A* es el que tiene mayor afinidad con el antígeno, por lo que será clonado. Posteriormente, los nuevos clones sufrirán un proceso de mutación a gran escala. Una vez que los antígenos han sido eliminados del organismo, existirá un excedente de células (anticuerpos) que deben ser exterminados para que el sistema regrese a sus niveles normales, a este proceso se le denomina *autoregulación*; sin embargo, algunas de estas células se conservan circulando a través del organismo como células de memoria con la encomienda de que la próxima vez que el mismo tipo de antígeno sea reconocido (o uno similar), el sistema inmune utilizará sus células de memoria y su respuesta será cada vez más rápida y eficiente (*respuesta secundaria*). La teoría de la *red inmune* explica las interrelaciones que existen entre anticuerpos, aún en la ausencia de antígenos, considerando que un paratope de un anticuerpo puede ser estimulado por el paratope de otro anticuerpo, llegando a suprimirse.

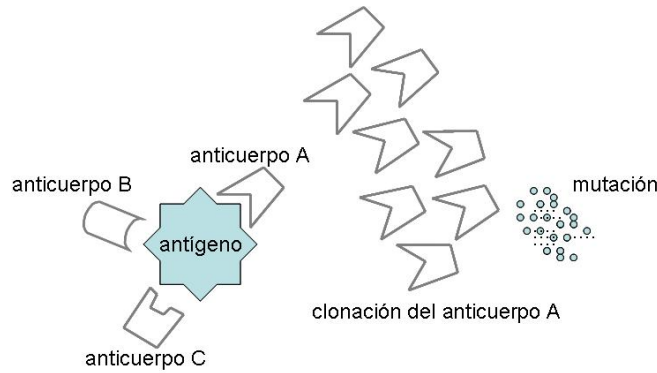


Figura 2.2. Principio de Selección Clonal.

En la terminología empleada en el área del SIA, entiéndase lo siguiente: un *antígeno* es el problema a resolver y los *anticuerpos* son las posibles soluciones, siendo estos últimos el equivalente a los *cromosomas* (individuos) en un algoritmo genético, así como la *afinidad* es el equivalente a la *aptitud*.

De acuerdo a Dasgupta en [22], el sistema inmune artificial tiene aplicaciones muy diversas en dependencia al modelo de funcionamiento biológico a emular y del dominio del problema a resolver. En lo general, hace referencia a aplicaciones de tolerancia a fallos, detección de fraudes, seguridad, reconocimiento de patrones, minería de datos, control adaptativo y optimización.

El problema que nos interesa solucionar, es el problema general de la optimización numérica global, que se define de la siguiente manera [23]:

$$\text{Encontrar } \vec{x} \text{ que optimice } f(\vec{x}) \quad (2.1)$$

suje to a:

$$g_i(\vec{x}) \leq 0, \quad i = 1, \dots, n \quad (2.2)$$

$$h_j(\vec{x}) = 0, \quad j = 1, \dots, p \quad (2.3)$$

donde \vec{x} es el vector de r variables del problema $\vec{x} = [x_1, x_2, \dots, x_r]^T$, n es el número de restricciones de desigualdad y p es el número de restricciones de igualdad; en ambos casos las restricciones pueden ser lineales o no lineales. Para resolver problemas de optimización, la literatura especializada reporta varios algoritmos; sin embargo, son dos los que se consideran básicos y a partir de modificar estos surgen otros: *Clonal Selection Algorithm (CLONALG)* y *Artificial Immune Network (AiNet)*.

2.1 CLONALG

CLONALG está basado en el *principio de la selección clonal* [24], [25] que se comentó con anterioridad y cuyo funcionamiento estima la respuesta inmune ante el estímulo de un antígeno. A grandes rasgos, el funcionamiento del algoritmo es el siguiente: dada una población de anticuerpos, donde cada uno de ellos tiene asociado un valor de afinidad (correspondiente al valor de la función objetivo), se seleccionan para ser clonados aquellos individuos con mejor afinidad.

Los anticuerpos seleccionados serán clonados proporcionalmente a su afinidad generando un repertorio de clones. Cada uno de los clones es hipermutado (mutación a gran escala) en forma inversamente proporcional a su afinidad. Para luego seleccionar los mejores individuos entre la población de clones y la población de anticuerpos. Por último se reemplazan los individuos de menor afinidad por individuos generados aleatoriamente. El algoritmo 2.1, listado a continuación, resume el proceso.

1. *Inicialización.* Generar la población inicial (anticuerpos) del algoritmo, aleatoriamente.
2. *Manejo de la Población.* Para cada antígeno hacer:
 - 2.1. *Selección.* Seleccionar a los anticuerpos con mayor afinidad con respecto al antígeno, considerando las soluciones como anticuerpos y el antígeno como la función objetivo.
 - 2.2. *Clonación y Variación Genética.* Clonación de los anticuerpos estimulados en forma directamente proporcional a la afinidad: el más alto en afinidad, clonará más. Cada uno de los clones es mutado en forma inversamente proporcional a su afinidad: el más alto en afinidad mutará menos.
 - 2.3. *Evaluación de la afinidad.* Evalúa la afinidad de cada anticuerpo mutado con el antígeno.
 - 2.4. *Autorregulación.* Una vez exterminados los antígenos, el SI debe regresar a sus valores normales, eliminando el exceso de anticuerpos.
3. *Ciclo.* Repetir el paso 2 hasta que se alcance el criterio de convergencia.

Algoritmo 2.1. CLONALG.

Inicialmente, CLONALG fue utilizado para resolver problemas de reconocimiento de patrones y optimización multimodal. En [23] fue extendido para trabajar con problemas de optimización con restricciones.

2.2 AiNet

El algoritmo AiNet está basado en la teoría de la red inmune que hace mención a la capacidad que tiene un anticuerpo de un linfocito de estimular a otro anticuerpo de otro linfocito diferente, de manera similar al reconocimiento de un antígeno [26], [27]. Se dice que existe un mecanismo de retroalimentación que mantiene la memoria del sistema inmune, recordando que estas células de memoria tienden a morir si no se estimulan constantemente; el resultado es una *red idiopática* de comunicación entre los linfocitos o células inmunes del sistema para reconocer los antígenos. El principio fundamental para lograr la estabilidad de la red, indica que cuando un linfocito reconoce a un antígeno o a algún otro linfocito, el primero es estimulado; sin embargo, cuando un linfocito es reconocido, éste es suprimido de la red. La suma de la estimulación y supresión recibida por la red de linfocitos, más la estimulación provocada por el reconocimiento de antígenos, corresponden al nivel de estimulación total S de una célula, tal y como lo describe la ecuación 2.4.

$$S = N_{st} - N_{sup} + A_s \quad (2.4)$$

donde N_{st} corresponde a la estimulación de la red, N_{sup} es la supresión de la red y A_s corresponde a la estimulación antigénica. En el algoritmo 2.2, se muestra la abstracción del algoritmo AiNet de acuerdo a [26].

1. *Inicialización.* Inicializar una red de células inmunes (linfocitos con sus anticuerpos) aleatoriamente.
2. *Manejo de la Población.* Para cada antígeno hacer:
 - 2.1. *Reconocimiento Antígena.* Relacionar la red de células con el antígeno.
 - 2.2. *Interacciones de la Red.* Relacionar las células de la red con otras células de la misma red.
 - 2.3. *Evaluación de la afinidad.* Introducir nuevas células dentro de la red y eliminar las menos útiles de acuerdo a algún criterio específico.
 - 2.4. *Nivel de Estimulación.* Evaluar el nivel de estimulación de la red, considerando el conteo de resultados de los pasos previos de acuerdo a la ecuación 3.4.
 - 2.5. *Dinámica de la Red.* Actualizar la estructura de la red y liberar los parámetros de acuerdo al nivel de estimulación de los anticuerpos individuales.
3. *Ciclo.* Repetir el paso 2 hasta que se alcance el criterio de convergencia.

Algoritmo 2.2. AiNet.

2.3 Estado del Arte

En [5] Goldberg realizó varios experimentos utilizando un Algoritmo Genético (AG) simple con representación binaria. Probó una población de sólo 3 individuos y afirmó que éstos eran suficientes para asegurar la convergencia del algoritmo sin importar la dimensión del cromosoma. Goldberg indicó que la mecánica consistía en aplicar los operadores genéticos hasta alcanzar una convergencia nominal. Esta convergencia nominal es un ciclo interno que concluye cuando los individuos son muy similares entre sí o cuando se alcanza cierto número predefinido de iteraciones. Al finalizar, se obtiene un nuevo individuo (el de mejor aptitud), para posteriormente generar de manera aleatoria los otros dos individuos que completarán la nueva población.

Bajo este esquema, Golberg definió un criterio para reinicializar el algoritmo al concluir la convergencia nominal, manteniendo al menos al mejor individuo (elitismo) y generando ruido estocástico al completar la nueva población con individuos generados aleatoriamente. En sus resultados, Goldberg indicó que manteniendo elitismo y con el ruido estocástico habría convergencia aunque el número de generaciones fuera muy grande. El algoritmo básico planteado por Goldberg se puede resumir como se indica en el algoritmo 2.3:

1. Generar aleatoriamente una μ -población de tres a cinco individuos (soluciones).
2. Determinar la aptitud de cada individuo y el mejor se mantiene para la próxima generación (estrategia de elitismo).
3. Los padres de los restantes individuos, menos aptos, se determinan utilizando alguna estrategia de selección, por lo general, torneo binario.
4. Se analiza la convergencia de la μ -población. Si la población converge, regresar al paso 1, manteniendo al mejor individuo y generando

aleatoriamente a los restantes. Si la población no converge, regresar al paso 2.

Algoritmo 2.3. Micro - Algoritmo Genético.

Krishnakumar en [6] diseñó un AG con una población reducida a sólo 5 individuos, a su algoritmo de representación binaria lo nombró Micro Algoritmo Genético (Micro-Genetic Algorithm). Al igual que Goldberg, Krishnakumar utilizó elitismo para preservar la mejor cadena encontrada al término de la convergencia nominal, como uno de los individuos obligatorios para la siguiente generación. Al comparar el desempeño del micro-AG contra un AG simple con una población de 50 individuos, se obtuvieron mejores resultados sobre funciones de un solo objetivo, además de que se comprobó que el AG de población reducida convergía más rápido.

Dozier et al en [7] presentaron dos aproximaciones basadas en el micro-AG que rápidamente encuentran soluciones para problemas de optimización con restricciones. Los autores aportaron algunas alternativas para el manejo de espacios restringidos bajo una técnica de no penalización.

Coello y Toscano, diseñaron un micro AG para resolver problemas de optimización de múltiples objetivos [8], aportando criterios para el manejo de restricciones de igualdad y desigualdad, además de proponer un esquema dominancia de pareto con un posicionamiento geográfico para mantener la diversidad y distribuir uniformemente las soluciones del frente de pareto. Este algoritmo trabaja con una población de 4 individuos y utiliza una memoria secundaria que almacena las soluciones potenciales a lo largo de la búsqueda.

El funcionamiento de este micro-AG multi-objetivo comienza generando aleatoriamente la población reducida que se almacena en una memoria interna dividida en dos partes: una de ellas se mantiene sin cambio durante todo el proceso, suministrando la diversidad; la otra parte de la memoria actualiza su contenido en cada ciclo del micro-algoritmo. Con cierta probabilidad, la población se conforma en cada ciclo tomando valores de ambas porciones de memoria y aplicando los operadores genéticos de manera convencional. Terminado un ciclo, bajo algún criterio específico, se seleccionan dos vectores no dominados de la población final, es decir, aquellos que presenten las mejores soluciones obtenidas hasta el momento y se comparan con el contenido de la memoria externa.

Al realizar la comparación, si uno de ellos o ambos siguen siendo no dominados, se incluyen en la misma memoria externa y se eliminan todos los individuos de menor aptitud o dominados. La figura 2.3, muestra una aproximación al diagrama de flujo del micro-AG multi-objetivo.

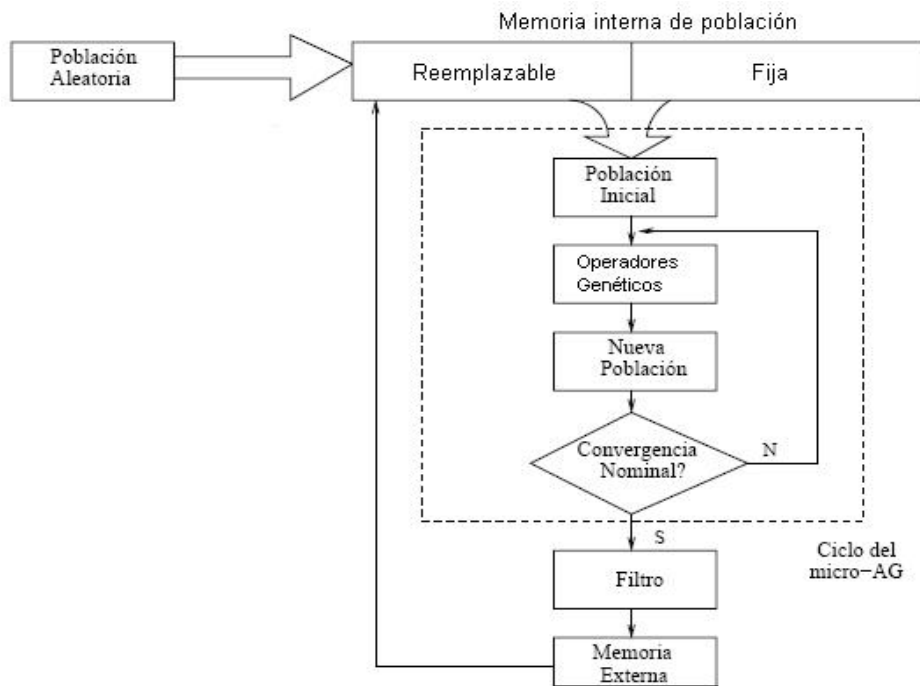


Figura 2.3. micro-AG multi-objetivo.

Recientemente, Fuentes y Coello en [9], diseñaron un micro algoritmo PSO (Particle Swarm Optimization – Optimización por Cúmulo de Partículas) para resolver problemas de optimización de un solo objetivo con manejo de restricciones. Utilizan 5 partículas (individuos) y se apoyan de una convergencia nominal.

En 1999 Harik *et al* [4] propusieron un *algoritmo genético compacto* (AGc). En lugar de trabajar con la población total, como lo hace un algoritmo genético simple, el AGc únicamente simula su existencia, es decir, representa la población mediante un vector de probabilidades de valores $p_i \in [0,1] \forall i = 1, \dots, l$, donde l es el número de elementos del alfabeto necesarios para representar las soluciones.

Cada valor p_i indica la proporción de individuos en la población simulada que tienen un cero (uno) en la posición i de su representación. Si estos valores se toman como probabilidades, se pueden generar nuevos individuos y actualizar el vector, favoreciendo a los mejores individuos para realizar este proceso. Los valores de las probabilidades p_i , se fijan inicialmente a 0.5 para representar una población generada aleatoriamente en el que el valor de cada alelo tiene la misma probabilidad de pertenecer a una solución. En cada iteración el AGc genera dos individuos, basándose en la representación elegida y compara los valores de sus funciones de coste. Se le denomina W a la representación del individuo con mejor coste y L a la del peor. Las representaciones de los competidores se usan para actualizar el vector de probabilidad de la iteración k a la $k + 1$ según la ecuación 2.5:

$$p_i^{k+1} = \begin{cases} p_i^k + \frac{1}{n} & \text{si } W_i = 1 \wedge L_i = 0 \\ p_i^k - \frac{1}{n} & \text{si } W_i = 0 \wedge L_i = 1 \\ p_i^k & \text{si } W_i = L_i \end{cases} \quad (2.5)$$

siendo n la dimensión de la población simulada y $W_i(L_i)$ el valor del alelo i ésimo de $W(L)$. El AGc finaliza cuando todos los valores del vector son iguales a 0 ó 1. En este momento, el propio vector p , representa la solución final. Para representar n individuos, el AGc actualiza el vector de probabilidad mediante un valor constante igual a $1/n$. De esta forma, sólo hacen falta $\log_2 n$ bits para almacenar cada valor de p_i . Por lo tanto, el AGc sólo necesita $\log_2 n * l$ bits con respecto a los $n * l$ bits necesarios de un AG convencional. De esta manera se pueden explorar poblaciones de una dimensión mayor sin un aumento significativo de la memoria utilizada, aunque se hace más lenta la convergencia del algoritmo.

Aunque este algoritmo plantea una modificación al manejo de una población estándar, no utiliza una micro – población. Sin embargo, se ha demostrado que la implementación en plataformas hardware del AGc es realizable. Apornetewan *et al* [28] y Gallagher *et al*[29], abordan la implementación de este tipo de algoritmos en dispositivos FPGA, en donde se establece claramente el alcance de esta técnica: sustituir la población actual por un vector de dimensión l , donde l se conoce como la longitud del cromosoma, lo que reduce la cantidad de bits necesarios para almacenar una población, aunque ésta, a diferencia de los micro-algoritmos genéticos que tienen una población de máximo 5 individuos, consta de 250 individuos para sus experimentos con funciones estándares de prueba.

Por su parte, Cupertino en [30] utilizó un AGc para controlar la velocidad y la posición de un motor de inducción, que en un esquema de trabajo normal, requiere controladores independientes y que no se pueden activar al mismo tiempo, es decir, primero se tiene que modificar la velocidad y posteriormente la posición del motor; el AGc permite optimizar los tiempos de conmutación, mejorando la linealidad del sistema.

Lo interesante de su realización es que él busca reducir el coste computacional sin demeritar el desempeño de su controlador, el cual fue implementado en un microcontrolador. Debido a las características de los algoritmos genéticos compactos, Cupertino obtiene ventaja de la

subpoblación que se evalúa menos veces en la función objetivo, pudiendo encontrar un buen desempeño comparado contra el propio del algoritmo genético normal. En esta realización se justifica la implementación debido a la proliferación de los sistemas de control embebidos en los cuales los actuadores vienen equipados con microcontroladores de muy bajo costo.

2.4 Problemática a resolver

Considerando el estado del arte en la realización de micro-algoritmos para resolver problemas de optimización numérica es posible advertir que sólo el algoritmo genético es ampliamente estudiado. El modelo del AG simple ha sido principalmente abordado por su sencillez de programación. Se tienen modelos estándares que facilitan la comparación en desempeño con los modelos de población de tamaño reducido. Los operadores genéticos de mutación, cruza y selección son estudiados utilizando métricas estadísticas, lo que repercute en una simplicidad en las pruebas experimentales.

También es posible aseverar, tras haber revisado el estado del arte, que existe un interés por realizar implementaciones en plataformas en hardware de estos algoritmos. Los principales esfuerzos en este rubro están dirigidos a lograr arquitecturas que de manera intrínseca puedan soportar la ejecución de algoritmos, como es el caso del algoritmo genético compacto, aunque como ya se indicó, éste no utiliza una población reducida sino un criterio que utiliza un vector de probabilidades como un registro lineal de datos binarios.

El planteamiento del problema que originó este trabajo de tesis fue el siguiente: ¿cómo implementar un algoritmo basado en el principio de selección clonal del SIA que utilice una población reducida drásticamente en su cantidad de individuos?

Si se reduce la población, ¿será posible realizar arquitecturas en hardware que de manera embebida ejecuten este micro-algoritmo?

En el algoritmo estándar del principio de selección clonal crece el número de individuos de la población durante la fase de clonación hasta en un 600% por lo que si se consideran poblaciones estándares de 20 individuos (en la práctica pueden ser hasta 100 o más, dependiendo del problema a resolver) se está infiriendo un número muy grande de datos que se tienen que manipular de manera simultánea. Nosotros propusimos utilizar sólo 5 individuos y mantuvimos un crecimiento fijo de sólo 15 individuos durante la fase de clonación, lo que permite realizar menos evaluaciones a la función objetivo y disminuir el uso de la memoria de datos. Las implementaciones en hardware se pudieron realizar para resolver un problema común en la práctica del reconocimiento de patrones.

Capítulo 3

Micro-Sistema Inmune Artificial

3.1 Propuesta generalizada

Realizando una revisión de los trabajos citados en el estado del arte, es posible encontrar similitudes en el diseño de un algoritmo con un tamaño de población reducido:

1. Población de 3 a 5 individuos.
2. Se requiere de una convergencia nominal y de un proceso de reinicialización.
3. Es necesario considerar elitismo para preservar, al menos, al mejor individuo obtenido al término de la convergencia nominal.

Otro punto en común que comparten estas realizaciones es que el micro-algoritmo que se utiliza para optimización mono-objetivo puede adecuarse para manejar restricciones y para optimización con múltiples objetivos. En base a lo anteriormente expuesto, presentamos el esquema general de la figura 3.1 para diseñar (o adaptar en su caso) un algoritmo estándar a un modelo de micro población. Se puede apreciar que existen dos ciclos de funcionamiento: uno interno que se ejecutará mientras no se haya

alcanzado la convergencia nominal y uno externo que se efectuará hasta que se alcance el criterio de paro del algoritmo.

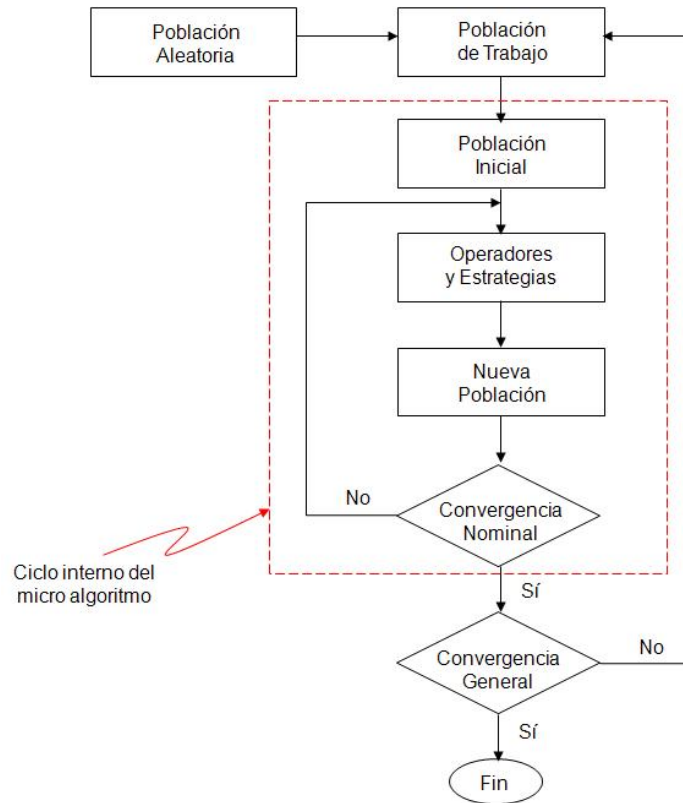


Figura 3.1. Esquema general de un micro algoritmo bioinspirado.

La metodología generalizada consta de 3 partes fundamentales:

1. **Definir el número de individuos de la población.** Resulta evidente que en dependencia a la complejidad del problema a resolver, será necesario realizar ajustes en el tamaño de la micro población, no obstante, para cumplir con la metodología que proponemos el número de 5 individuos es asequible para resolver las funciones que se probaron experimentalmente. En la generación inicial, los individuos de la población se obtienen de manera aleatoria, éstos se copian a

la población de trabajo y a la población inicial del ciclo interno del micro algoritmo (refiérase a la Figura 3). Durante el proceso de evolución, generación tras generación, la población se mantendrá con un número estático, es decir, el tamaño de la población no crece ni se disminuye de manera dinámica, aunque se debe considerar la inclusión de individuos nuevos en cada generación como se mencionará posteriormente.

2. **Definir el criterio de la convergencia nominal y aplicar los operadores y estrategias del algoritmo bioinspirado en su versión estándar.** El criterio más utilizado para la convergencia nominal es definir un número de generaciones máximas a realizar; en este rubro ejecutamos pruebas funcionales que permitieron precisar que 10 generaciones son suficientes para los experimentos con el micro algoritmo del sistema inmune artificial. Al alcanzar la convergencia nominal es necesario un proceso de reinicialización para conformar una nueva población de trabajo. En el ciclo interno controlado por la convergencia nominal se utilizan los operadores y estrategias particulares definidas para el algoritmo bioinspirado en su versión estándar siendo posible implementarlo sin cambios o en otro caso realizar modificaciones que mejoren el desempeño del algoritmo con respecto a un problema en particular. Debido a que la calidad de las soluciones generadas al término de la convergencia nominal depende de los operadores y estrategias definidas por cada algoritmo estándar, el número de generaciones para alcanzar la convergencia nominal seguramente será diferente al cambiar de un algoritmo bioinspirado a otro.

3. **Aplicar elitismo para garantizar la convergencia.** Algunos de los individuos obtenidos al término de la convergencia nominal, generalmente los mejores o al menos el mejor (elitismo) en base a la aptitud, deben ser copiados a la población de trabajo que se completa con individuos generados aleatoriamente (para mantener la diversidad) y se incrementa una generación más en el ciclo externo hasta que se alcance la condición de paro del algoritmo. Es significativo mencionar que depende mucho de la naturaleza de los algoritmos bioinspirados, con respecto a los operadores y estrategias internas, para determinar cuántos individuos deben ser copiados como parte del elitismo a la población de trabajo, además de que éste garantiza la convergencia del micro algoritmo.

3.2. Micro-Sistema Inmune Artificial para optimización sin manejo de restricciones

La propuesta presentada en este trabajo de tesis se muestra en el diagrama de la figura 3.2. Está basada en el principio de selección clonal del SIA y básicamente es una adaptación de CLONALG [24].

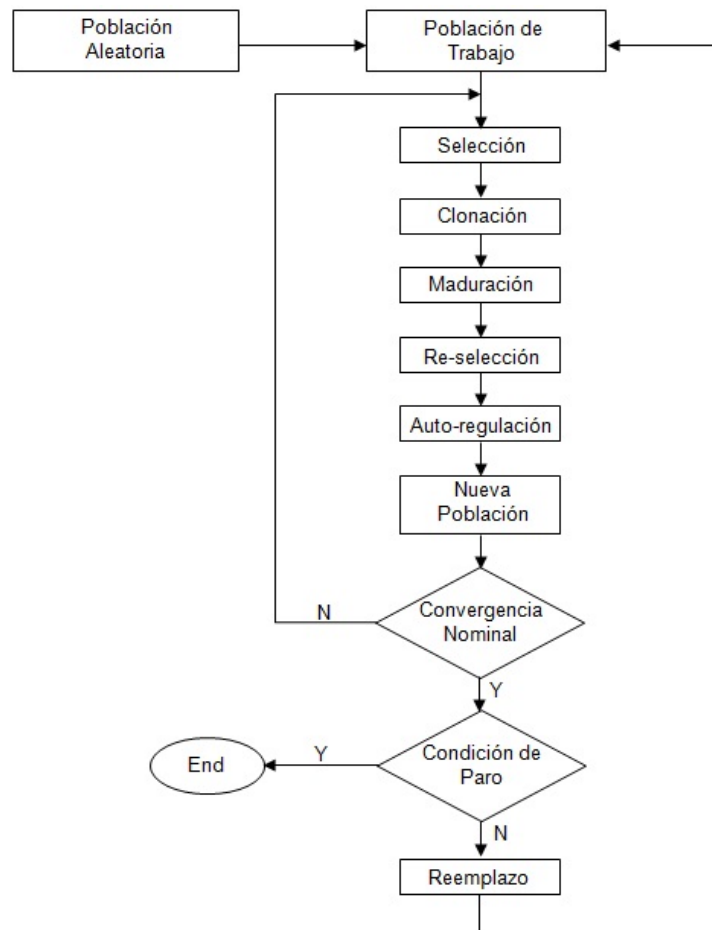


Figura 3.2. Micro – Sistema Inmune Artificial.

El algoritmo funciona de la siguiente manera:

1. Generar aleatoriamente una población de 5 anticuerpos (individuos). En la generación inicial estos anticuerpos se copian directamente a la población de trabajo para comenzar la convergencia nominal que está controlada por un número de generaciones igual a 10.
2. Utilizar una selección basada en ranking. El anticuerpo de mayor afinidad será el mejor individuo. En nuestro algoritmo a este individuo lo nombramos *BestAb*.
3. Realizar la clonación de todos los anticuerpos utilizando

$$N_C = \sum_{i=1}^n (n - (i - 1)) \quad (3.1)$$

en donde N_C es el número de clones que se generarán para cada anticuerpo, n es el número total de anticuerpos de la población, i es el anticuerpo corriente, comenzando con el de mayor afinidad (*BestAb*). Considerando una población de 5 anticuerpos obtendremos una población de 15 clones: el anticuerpo *BestAb* obtendrá 5 clones, el segundo anticuerpo del ranking clonará 4 veces y así sucesivamente hasta llegar al peor anticuerpo que sólo obtendrá un clon.

4. Realizar la maduración de los clones a través de un proceso de mutación. La probabilidad de mutación se fijará al principio de la convergencia nominal para cada grupo de clones obtenidos del mismo anticuerpo. Esta probabilidad se determina de manera proporcional a la afinidad del anticuerpo que originó los clones y disminuirá en cada generación, así el grupo de clones del anticuerpo *BestAb* mutará menos que los demás grupos de clones que se generaron de los restantes anticuerpos. El único clon que se obtuvo del peor anticuerpo tendrá la mayor posibilidad de mutar. Obsérvese la siguiente relación matemática

$$prob_mutation(i) = \frac{Aff(i)}{\sum_{i=1}^n Aff(i)} \quad (3.2)$$

en donde i es el anticuerpo que permitirá fijar la probabilidad de mutación para el grupo de clones que se obtuvieron de él mismo y n es el total de anticuerpos de la población. Para disminuir uniformemente la probabilidad de mutación utilizamos en cada generación dentro de la convergencia nominal utilizamos la siguiente ecuación:

$$if\ prob \leq \frac{prob_mutation(i)}{generation} \text{ then apply mutation} \quad (3.3)$$

en donde $random\ prob \in [0,1]$ y $generation$ es la generación corriente dentro de la convergencia nominal, es decir, $int\ generation \in [1,10]$ y no se desea dividir entre 0.

Para la variación de cada uno de los clones, presentamos dos operadores sencillos y que permiten explotar mayormente el espacio de búsqueda al realizar diferentes tamaños de paso en el proceso de mutación. Varios aspectos han sido considerados para implementar estos operadores: el número de clones, la generación corriente dentro de la convergencia nominal y el rango permisible de valores de las variables de decisión.

Utilizar con una probabilidad de 0.5 los siguientes operadores de mutación, que actúan sobre cada variable de decisión de un clon (en nuestro esquema, todo el vector de soluciones es mutado):

$$x' = x + \frac{(\alpha \cdot range \cdot generation)}{N_c} \quad (3.4)$$

o

$$x' = x + \frac{(\alpha \cdot range)}{(generation \cdot N_c)} \quad (3.5)$$

donde x' es la variable de decisión mutada, x es la variable de decisión a mutar, α es un número aleatorio con distribución uniforme donde $random\ \alpha \in [0,1]$, $generation$ es la generación corriente dentro de la convergencia nominal y N_c es el número total de clones. El valor de α se debe computar para cada variable de decisión del clon.

En el caso de los 5 clones obtenidos de $BestAb$, $range \in [LB, UB]$ es un número aleatorio entre el límite inferior (LB) y el límite superior (UB) de los valores que pueden tomar las variables de decisión y se mantiene constante para toda la dimensión del clon a mutar, es decir, $range$ será el mismo para todas las variables de decisión del clon.

Para los restantes clones que se obtuvieron de los 4 anticuerpos del ranking, $range$ es cualquier variable de decisión del anticuerpo $BestAb$, cuya posición dentro de la dimensión del vector se obtiene aleatoriamente.

5. Efectuar nuevamente una selección basada en ranking. Esta ocasión, se ordenarán los 15 clones con respecto a su afinidad. Los dos mejores clones se seleccionarán (elitismo) y la nueva población se completará con otros 3 clones seleccionados aleatoriamente de la población de clones maduros.

Los restantes clones se eliminarán, proveyendo una auto-regulación dentro de la convergencia nominal.

6. Si se alcanzan las 10 generaciones de la convergencia nominal, se mantendrán los dos mejores clones y se generarán otros tres anticuerpos de manera aleatoria para completar la nueva población de trabajo y comienza nuevamente la convergencia nominal hasta que el ciclo externo del algoritmo cumple con la condición de parada.

3.3 Micro – Sistema Inmune Artificial para optimización con manejo de restricciones

Para el manejo de espacios restringidos utilizamos dos parámetros para seleccionar a los mejores individuos: la afinidad con respecto a la función objetivo y la cantidad de restricciones violadas. Esta metodología fue propuesta por Deb en [31]. La adecuación para el micro – algoritmo aquí propuesto sólo considera en la etapa del primer ranking, en el ciclo de la convergencia nominal, selecciona al mejor individuo bajo las siguientes características: como primera opción se buscará al anticuerpo factible sobre el infactible; si fuera el caso que hubiera varios factibles, seleccionar al de la mayor afinidad. Para los infactibles, se seleccionará al que tenga el menor número de restricciones violadas y mayor afinidad.

Una vez realizada la selección inicial de anticuerpos (refiérase al paso 2 del algoritmo explicado en el subtema anterior), utilizamos los mismos criterios para la segunda selección después de haber realizado la mutación de los clones (refiérase al paso 5 del mismo algoritmo).

3.4 Pruebas y resultados experimentales

Los operadores de mutación aportados tienen características de uniformidad y permiten pasos de mutación pequeños y grandes. Notamos experimentalmente que estos operadores aceleran la convergencia y permiten una mayor exploración del espacio de búsqueda como se observará en los resultados reportados. Los primeros experimentos de nuestro algoritmo utilizaban el único operador de mutación no uniforme propuesto por Nunes de Castro et al en [24], sin embargo para el algoritmo con población reducida no entregó buenos resultados debido a que no converge al óptimo aún aumentando el número de individuos hasta 10.

3.4.1 Primera fase de experimentación

Para la primera experimentación y con respecto al micro-SIA sin manejo de restricciones, utilizamos las funciones citadas en [32]. Se trata de funciones con múltiples variables (dimensión 30) y óptimo en cero, a excepción de $f08$. Refiérase al anexo A para revisar estas funciones. La elección de este test de prueba se debe a que las funciones abordadas por el autor de la referencia se resuelven utilizando el algoritmo de evolución diferencial lo que presupone que las funciones son complejas y son un referente actual.

En la tabla 3.1 se aprecian los resultados obtenidos experimentalmente. Se muestran los resultados de 20 corridas para cada problema. Para todos los casos utilizamos una población fija de 5 anticuerpos y 10 generaciones para alcanzar la convergencia nominal. El equipo utilizado para las pruebas fue una PC *Quad Core* a 2.66 MHz, con 2MB de memoria. La intención de este experimento fue probar estadísticamente la convergencia del micro-SIA al óptimo.

Tabla 3.1. Resultados experimentales del micro- sistema inmune artificial para problemas de optimización sin restricciones.

Función	Ciclo Externo	Convergencia Nominal	Mejor	Peor	Media
<i>f01</i>	1000	10	0.0	0.000022	0.000009
<i>f02</i>	1000	10	0.0	0.000017	0.000008
<i>f03</i>	1000	10	0.0	0.000002	0.000001
<i>f04</i>	1000	10	0.0	0.000012	0.000005
<i>f05</i>	1000	10	0.0	0.000028	0.000012
<i>f06</i>	2000	10	0.0	0.000032	0.000015
<i>f07</i>	2000	10	0.0	0.000027	0.000013
	2000		-		
<i>f08</i>		10	12569. 5	-12569.57	-12569.496
<i>f09</i>	2000	10	0.0	0.000033	0.000013
<i>f10</i>	2000	10	0.0	0.000011	0.000007
<i>f11</i>	2000	10	0.0	0.000013	0.000004

Se aprecia que en todas las funciones utilizadas se convergió al óptimo. La medida del error promedio es mínima en todos los casos.

3.4.2 Segunda fase de experimentación

Continuando con respecto al micro-SIA sin manejo de restricciones, se procedió a realizar una comparación entre el SIA en su versión estándar y el micro-SIA. Sólo se utilizaron tres funciones de las citadas en el anexo A. Para realizar esta comparación se corrieron los algoritmos en la misma PC, respetando las características físicas antes expuestas en el numeral 3.5.1.

La intención de este experimento fue revisar el número de evaluaciones a la función objetivo y el tiempo de ejecución del algoritmo para converger al óptimo.

En la tabla 3.2 se listan los resultados experimentales obtenidos en esta fase.

Tabla 3.2. Resultados experimentales de la comparación entre el CLONALG en su versión estándar y el micro-SIA, ambos casos sin manejo de restricciones.

	Ab	Clones	Ciclo interno	Ciclo externo	Evaluaciones a la función objetivo	Tiempo (segundos)
<i>CLONALG</i>						
<i>f01</i>	50	256	0	1000	1,280,000	47.2
<i>f05</i>	70	312	0	1000	21,840,000	78.6
<i>f07</i>	70	312	0	1200	26,208,000	103.7
<i>Micro-SIA</i>						
<i>f01</i>	5	15	10	1000	750,000	14.8
<i>f05</i>	5	15	10	1000	750,000	14.2
<i>f07</i>	5	15	10	2000	1,500,000	48.3

Nótese en la tabla 3.2 que el número de anticuerpos (individuos) en la población inicial del algoritmo en su versión estándar para *f01* es de 50, mientras que para el micro-SIA en la misma función se utilizan sólo 5 individuos. Así mismo, en la etapa de clonación, para la misma *f01* en la versión estándar la población crece hasta 256 clones, mientras que en el micro-SIA se tiene un crecimiento controlado de 15 clones. Con lo anterior, el uso de la memoria de datos en el micro-SIA se reduce al 5.85% de la utilizada por la versión estándar de CLONALG. Para el caso de *f05* y *f07*, el micro-SIA utiliza el 4.80% de los 312 clones generados en la etapa de clonación.

Con respecto al número de evaluaciones a la función objetivo, comparando los resultados, se aprecia que el micro-SIA realiza menos evaluaciones para alcanzar el óptimo. Nótese que si se aumenta el ciclo externo

del micro-algoritmo también crece el número de evaluaciones a la función objetivo. Considerando $f07$, el micro-SIA ejecuta 1,500,000 evaluaciones a la función objetivo y en contraparte CLONALG verifica 26,208,000 la evaluación, lo que representa un menor coste computacional para la versión reducida. En este mismo sentido, el tiempo de ejecución del micro-SIA es también menor en todos los casos reportados.

3.4.3 Tercera fase de experimentación

El tercer experimento consistió en modificar el número de anticuerpos en la población inicial, modificando por ende el número de clones en la fase de clonación. También se procedió a realizar cambios en los ciclos interno y externo del micro-algoritmo. En la tabla 3.3 se aprecian los resultados obtenidos de este experimento.

Tabla 3.3. Resultados experimentales al variar los parámetros del micro-SIA.

	Ab	Clones	Ciclo interno	Ciclo externo	Mejor
$f01$	3	12	5	1000	0.000021
$f01$	10	25	10	1000	0.0
$f01$	15	35	20	500	0.000084
$f07$	3	12	5	2000	0.000009
$f07$	10	25	10	2000	0.000002
$f07$	15	35	20	1000	0.000040

En la Tabla 3.3 se aprecia que aumentar o disminuir el número de anticuerpos de la población inicial es un parámetro sensible que está relacionado con el número de clones que se obtendrán. Recuérdese que en la etapa de clonación los mejores individuos clonarán más y mutarán menos que los peores. El ciclo

interno obliga a que se modifique el externo para buscar la mejor manera de que funcione el algoritmo en términos de que converja al óptimo. Esta experimentación permitió definir que un número de 5 individuos iniciales, un crecimiento controlado de 15 clones y una convergencia nominal de 10, eran los parámetros que mejor se adaptaban para resolver las funciones propuestas a la vez que se generalizaron las características del micro-SIA.

3.4.4 Cuarta fase de experimentación

Debido a que los problemas de optimización en la vida real contienen restricciones de diferente índole, en este último experimento se decidió incluir una aproximación al manejo de restricciones para saber cómo se comporta el micro-SIA ante estas situaciones.

Se probaron dos funciones: $g01$ y $g02$, presentadas en [33]. Refiérase al anexo B para revisar estas funciones. La función $g01$ tiene 13 variables y la función $g02$ tiene 20 variables. El óptimo para $g01$ es -15.0. Cabe mencionar que para $g02$ el mejor óptimo encontrado (el óptimo es desconocido) es -0.803619. En la tabla 3.4 se muestran los resultados de 20 corridas.

Tabla 3.4. Resultados experimentales del micro-SIA para problemas de optimización con restricciones.

Función	Ciclo Externo	Convergencia nominal	Mejor	Peor	Media	Óptimo esperado	Error (considerando el mejor resultado)
$g01$	2000	10	-15.0	-14.978	-14.989	-15.0	0.0
$g02$	2000	10	-0.804090	-0.83023	-0.81025	-0.803619	0.000471

Revisando los resultados reportados para las funciones citadas podemos afirmar que el algoritmo diseñado tiene un desempeño satisfactorio comparando los valores obtenidos con el óptimo esperado.

Capítulo 4

Arquitectura del micro-SIA

Para las realizaciones en hardware se propuso resolver el problema del conteo máximo de unos (*maxone problem*) [34], también conocido como *counting ones problem*, que consiste en maximizar el número de bits puestos a uno en una cadena de bits. Matemáticamente el problema se describe como encontrar un vector $\vec{x} = \{x_1, x_2, \dots, x_N\}$, en donde $x \in \{0,1\}$ que maximice la ecuación

$$F(\vec{x}) = \sum_{i=1}^N x_i \quad (4.1)$$

En esta ecuación, N representa el número de bits de la cadena, por lo que el máximo se obtendrá cuando los N bits de la cadena sean unos.

Este problema no es sencillo de resolver en hardware, debido a que un número mayor que otro no necesariamente representa un mejor resultado, por ejemplo, el dato binario 1000 es peor solución que el dato 0011 debido a que su cadena de bits tiene menos unos. A medida que aumenta el número de bits de la cadena, la complejidad de la búsqueda también aumenta tal y como se comenta en [34].

El algoritmo funciona de manera general como se describe a continuación:

1. Generar aleatoriamente 5 anticuerpos de N bits, para el caso del microcontrolador se utilizaron 8 bits y para el FPGA 16. En la generación inicial estos anticuerpos se copian directamente a la población de trabajo para comenzar la convergencia nominal que está controlada por un número de generaciones igual a 5.
2. Utilizar una selección basada en ranking, tal y como se hizo con el micro-SIA. El anticuerpo de mayor afinidad será el mejor individuo.
3. Realizar la clonación de todos los anticuerpos utilizando la misma ecuación 3.1 del capítulo 3 de este mismo documento. De igual manera a la versión en software, considerando una población de 5 anticuerpos obtendremos una población de 15 clones.
4. Realizar la maduración de los clones a través de un proceso de mutación. En esta parte del algoritmo es importante mencionar que existen diferencias entre la versión del micro-SIA realizada en software con la aproximación en hardware. La principal diferencia es que en software se trabajó con una versión con números reales y para hardware sólo se trata de representaciones binarias.

La probabilidad de mutación se fijará al principio de la convergencia nominal para cada grupo de clones obtenidos del mismo anticuerpo. Esta probabilidad se determina de manera proporcional a la afinidad del anticuerpo que originó los clones y disminuirá en cada generación, así el grupo de clones del mejor anticuerpo, nombrado *BestAb* en la versión en software, mutará menos que los demás grupos de clones que se generaron de los restantes anticuerpos.

El único clon que se obtuvo del peor anticuerpo tendrá la mayor posibilidad de mutar. Para tal propósito se aplican las mismas ecuaciones 3.2 y 3.3 del capítulo 3. Con respecto a la variación que presenta cada uno de los clones al efectuar la mutación, se utilizó un operador sencillo que invierte un bit en determinada posición de la cadena. Para determinar qué bit debe mutarse se aplica una probabilidad de 0.5 considerando que un bit de una cadena que representa un individuo que es una buena solución tendrá menos posibilidades de cambiar que uno que pertenece a una mala solución.

5. Efectuar nuevamente una selección basada en ranking. Esta ocasión, se ordenarán los 15 clones con respecto a su afinidad. Los dos mejores clones se seleccionarán (elitismo) y la nueva población se completará con otros 3 clones seleccionados aleatoriamente de la población de clones maduros. Los restantes clones se eliminarán, proveyendo una auto-regulación dentro de la convergencia nominal.
6. Si se alcanzan las 10 generaciones de la convergencia nominal, se mantendrán los dos mejores clones y se generarán otros tres anticuerpos de manera aleatoria para completar la nueva población de trabajo y comienza nuevamente la convergencia nominal hasta que el ciclo externo del algoritmo cumple con la condición de parada.

4.1 Implementación en microcontrolador

Se decidió dirigir la implementación del diseño hacia un dispositivo comercial, en este caso un microcontrolador de 8 bits de bajo coste. Se utilizó un PIC16F628A que obliga a que se realice un procesamiento procedural de las instrucciones, además estamos restringidos por el número de pines de E/S, así como de la memoria de datos que es de sólo 224 bytes. Las etapas del diseño incluyen:

1. Generador de números aleatorios. Se utiliza el oscilador interno del microcontrolador (4MHz) y un push button para accionar el módulo generador que como elemento base tiene un contador binario. Este módulo genera 5 cadenas binarias (anticuerpos) para copiarlos a la población inicial.
2. Módulo de ordenación (ranking). Este módulo cuenta los unos de cada cadena y asigna una calificación a cada individuo. Posteriormente los ordena de mejor a peor considerando que los mejores individuos son los que tienen el mayor número de unos. Para contar los unos de una

cadena se utiliza un registro de corrimiento que con ayuda de una compuerta xor y un contador nos permita detectar la presencia de un uno y así incrementar el conteo. Esta tarea es el equivalente a la evaluación de la función objetivo. El mismo número de unos contados es el valor en calificación que se le asigna a cada individuo. Para ordenarlos se utiliza un comparador que detecta quién o quiénes tienen el mayor número de unos y los clasifica como los mejores individuos.

3. Módulo de clonación. Este módulo se encarga de generar 5 clones del mejor individuo, 4 del segundo mejor y así sucesivamente hasta obtener los 15 clones que caracterizan a nuestro algoritmo.
4. Módulo de mutación. Cada uno de los 15 clones tiene probabilidad de mutar, sin embargo, los 5 clones obtenidos del mejor individuo tienen menos probabilidad de mutar que el único clon obtenido del peor individuo, quien tiene la más alta probabilidad de mutar. Se utiliza un registro de corrimiento que permite cambiar el bit en la posición necesaria de acuerdo a la probabilidad definida en el algoritmo. Toda la cadena de bits de cada clon se recorre completamente para aplicar el operador de mutación en donde corresponda. Para los 5 clones del mejor anticuerpo se utiliza mutación de un punto, es decir, se invertirá un solo bit de la cadena. La posición de este bit se elige aleatoriamente. Para los 4 clones del segundo mejor anticuerpo se aplica mutación de dos puntos, es decir, se invertirán dos bits en dos posiciones elegidas aleatoriamente. Sucesivamente se utiliza este criterio hasta llegar al peor anticuerpo que sólo obtuvo un clon, a éste se le aplicará mutación en 5 puntos, lo que indica que se invertirán 5 bits de su cadena cuyas posiciones en ésta también se eligen aleatoriamente.
5. Módulo de ordenamiento de clones. Este módulo funciona de manera similar al primer módulo de ordenamiento para 5 anticuerpos, con la observación de que ahora debe clasificar 15 clones. Tras haber mutado, es posible que los individuos en el ranking hayan cambiado su calidad, por lo que se deben reclasificar y ordenar.

6. Módulo de remplazo. Este módulo se encarga de mantener a los dos mejores clones de los 15 generados y completa la población de trabajo con otros tres que se generan de manera aleatoria. Este módulo hace uso del módulo generador de números aleatorios para cumplir con su encomienda. Este mismo módulo lleva el control de la convergencia nominal por lo que al alcanzar las 10 iteraciones del ciclo interno. Mantendrá a los dos mejores individuos, que a su vez son las dos mejores soluciones y las copia a la población de trabajo para comenzar nuevamente un ciclo externo del micro-SIA.

En la figura 4.1 se pueden observar los módulos concebidos para la implementación embebida, que complementan la explicación anterior.

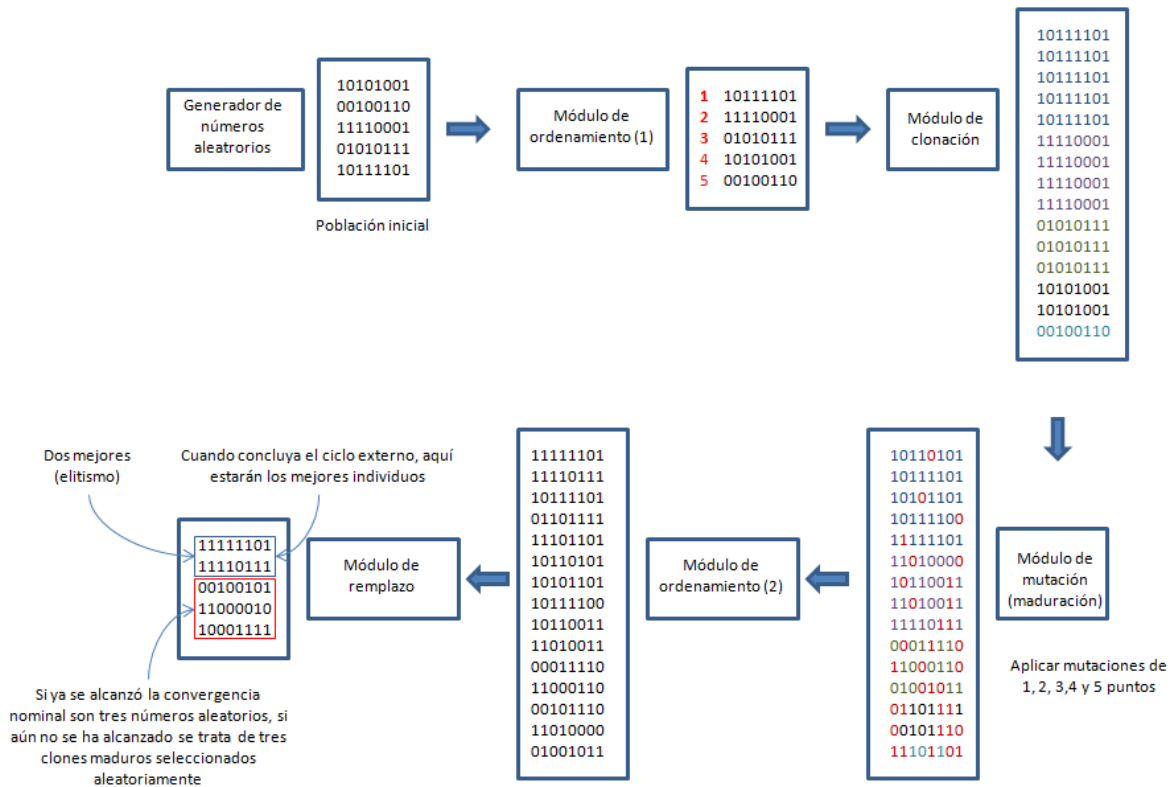


Figura 4.1. Diagrama a bloques del micro-SIA implementado en hardware, para solucionar el problema del conteo de unos.

La implementación embebida en el microcontrolador entregó los resultados listados en la tabla 4.1. Se realizaron 20 corridas del algoritmo para reportar los datos. La frecuencia de trabajo fue de 4MHz correspondiente al oscilador interno del dispositivo. Se puede observar cómo se maximiza el número de unos en el individuo reportado en cada generación. Para los experimentos con el microcontrolador, 40 ciclos externos fueron suficientes para obtener el máximo. El número de ciclos en la convergencia nominal (interno) se mantuvo en 5.

Tabla 4.1. Resultados experimentales del micro-SLA implementado en el microcontrolador para solucionar el problema del conteo de unos.

Longitud del individuo	1 Generación (salida)	5 Generaciones (salida)	10 Generaciones (salida)	20 Generaciones (salida)	40 Generaciones (salida)	Tiempo (segundos)
8 bits	00001101	00001111	00101111	01110111	11111111	132

4.2 Arquitectura en FPGA

La arquitectura propuesta se implementó en un FPGA Spartan 3A, con número de parte 3S700A del fabricante Xilinx. Los módulos diseñados son los mismos que para el microcontrolador con la observación de que en el FPGA es posible generar una arquitectura más flexible y óptima que en el microcontrolador comercial. El oscilador incorporado en la tarjeta de desarrollo es de 50MHz y fue el que se utilizó en el diseño. Los módulos fueron realizados en VHDL que es uno de los lenguajes descriptores de hardware más utilizados actualmente. Se utilizó el mismo problema de maximización del conteo de unos, con la diferencia de que los registros utilizados en el FPGA fueron de 16 bits, por lo que el óptimo real a obtener fue de 1111111111111111 que representa una afinidad (aptitud) de 16. En la Tabla 4.2 se listan los resultados obtenidos para la maximización de unos en el FPGA. Comparado con el microcontrolador el tiempo de obtención del resultado fue menor aún con un número mayor de bits en el registro. Lo anterior no sólo se debe a que el oscilador de la tarjeta de desarrollo es de 50MHz en contraparte con los 4MHz

del oscilador interno del microcontrolador, sino que en el FPGA es posible hacer comparaciones de datos en forma paralela acelerando la convergencia.

Tabla 4.2. Resultados experimentales del micro-SIA implementado en el FPGA para solucionar el problema del conteo de unos.

Longitud del individuo	1 Generación (salida)	10 Generaciones (salida)	20 Generaciones (salida)	40 Generaciones (salida)	Tiempo (segundos)
16 bits	0000001010100111	0001001111110111	0001011111110111	1111111111111111	23

Debido a la flexibilidad de los recursos del FPGA y para demostrar en forma práctica el uso del micro-SIA diseñado, se decidió implementar una aplicación que realizara el reconocimiento de caracteres. De manera cierta, ésta es una aplicación clásica que permitirá una ejecución intrínseca del algoritmo en el dispositivo de lógica reconfigurable. Se parte de una matriz de 7 filas y 5 columnas (35 puntos) para conformar los caracteres. El contenido de esta matriz se codificó en una cadena de 35 bits, por ejemplo, para el carácter corrupto colocado a la entrada de la Figura 4.2 se tendría la cadena 01110101011010100100001000010011100 en donde un 0 representa un punto de la figura sin marcar.

El carácter corrupto se compara paralelamente con 14 caracteres predefinidos: 9 dígitos (1 al 9) y 5 letras (vocales), todos éstos también codificados como cadenas de 35 bits. Con ayuda de una etapa de compuertas XOR y aplicando el criterio de la distancia de Hamming es posible obtener un patrón de comparación entre caracteres. Posteriormente con un módulo de conteo de unos se asigna una calificación a la calidad del individuo para saber cuál es la afinidad de cada uno de éstos. Debido a que se obtienen 14 individuos después de la comparación, al que tenga la mejor afinidad se le considera el elemento principal en nuestra población. Este individuo completará, con otros cuatro generados aleatoriamente, la población inicial del micro-SIA cuya función será maximizar el número de unos en la cadena que representa el resultado de la comparación.

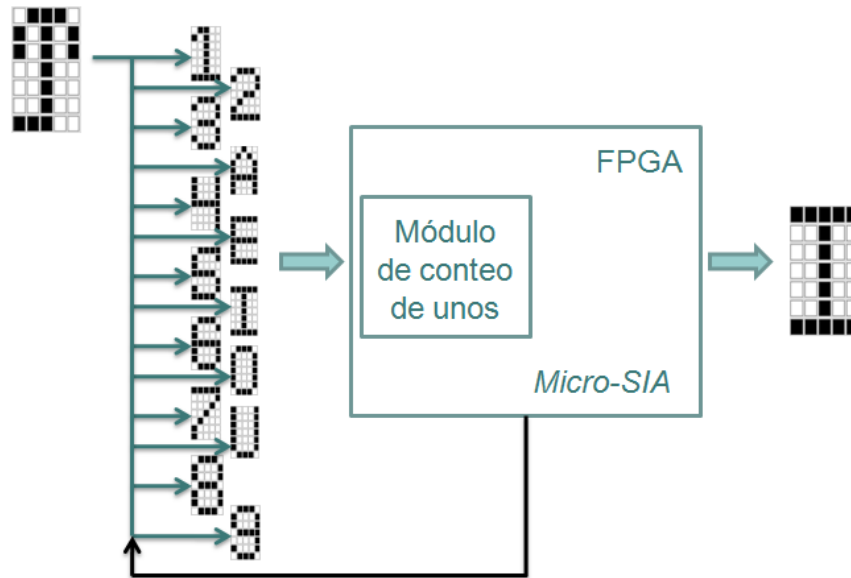


Figura 4.2. Reconocimiento de caracteres implementado en el FPGA y que utiliza al micro-SIA.

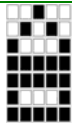
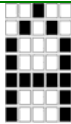
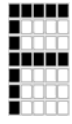
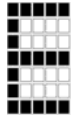
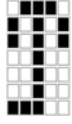
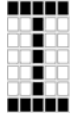
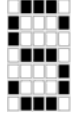
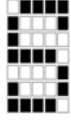
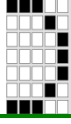
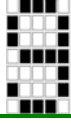
Se utilizaron 50 generaciones (ciclos externos del micro-SIA) para obtener el mejor resultado. Se definieron 5 ciclos internos para la convergencia nominal y el algoritmo se ejecutó 20 veces y en todas las corridas se obtuvieron los mismos resultados. En la Tabla 4.3 se muestra la salida obtenida para un caracter corrupto al paso de las generaciones.

Tabla 4.3. Evolución de la solución para un caracter corrupto.

Caracter corrupto	Generación 10	Generación 20	Generación 30	Generación 40	Generación 50 Salida

En la Tabla 4.4 se muestran los resultados obtenidos para los 5 caracteres corruptos indicados en la primera columna de la misma tabla.

Tabla 4.4. Resultados experimentales para 5 caracteres corruptos.

Carácter corrupto	Generación 50 Salida
	
	
	
	
	

Capítulo 5

Conclusiones

En este trabajo de tesis se diseñó y se implementó un algoritmo bioinspirado con un tamaño de población reducido, basado en el principio de selección clonal del sistema inmune artificial, para solucionar problemas de optimización numérica. El principio de selección clonal extrapolado del sistema inmune biológico, se caracteriza por aumentar el tamaño de la población durante la etapa de clonación y sólo permite aplicar operadores de mutación, por lo que mantener la diversidad de la población y asegurar la convergencia del algoritmo, han sido los principales retos que hacen interesante el estudio de esta heurística. Los elementos principales en nuestra metodología consisten en incorporar una convergencia nominal que se alcance después de un número fijo de iteraciones, un elitismo que preserve al mejor o mejores individuos encontrados al término de la convergencia nominal y un criterio para incorporarlos como parte de una nueva población al momento de reinicializar el algoritmo. Debido a que el proceso inmune simulado por este algoritmo no incluye un operador de cruce, la clonación con crecimiento controlado (fijo de 15 clones en nuestro diseño) y los operadores de mutación aportados son primordiales para mantener la diversidad y acelerar la convergencia.

Se probaron varios operadores de mutación, no obstante los mejores resultados se obtuvieron con los que aquí se presentaron y que involucran al número de clones, al rango de las variables y a la generación corriente o actual dentro de la ejecución del algoritmo. Experimentalmente se probaron poblaciones de 3 y hasta 5 anticuerpos, adaptando la clonación de manera proporcional siguiendo la misma metodología planteada y con 5 anticuerpos y 15 clones se obtuvieron los mejores resultados, por lo tanto el algoritmo se presenta con estas características.

En las funciones de prueba utilizadas, el elitismo designado (preservando a los dos mejores individuos) tanto en el ciclo interno de la convergencia nominal como en el ciclo externo, coadyuva a asegurar la convergencia del algoritmo. El número de generaciones para alcanzar la convergencia nominal se planteó de 10, debido a que menos generaciones no aceleran la búsqueda y más generaciones no mejoran los resultados.

De acuerdo a los resultados obtenidos, el algoritmo funciona correctamente para problemas de optimización mono-objetivo sin restricciones y con restricciones para las funciones presentadas en los benchmarks correspondientes. De acuerdo a [6] un micro – algoritmo debería ser más rápido que un algoritmo estándar; en las pruebas comparativas encontramos que efectivamente para las funciones utilizadas es más rápida la versión micro.

Está comprobados que al existir un menor número de individuos, se utiliza un espacio menor en la memoria de datos de cualquier plataforma en software o en hardware. El problema del conteo de unos es frecuentemente utilizado en el área del reconocimiento de patrones, especialmente cuando se desea realizar una comparación entre cadenas de bits.

El algoritmo del SIA en su versión de población con sólo 5 individuos se implementó con éxito, primeramente en un microcontrolador y posteriormente en un dispositivo de lógica programable (FPGA), para resolver este problema. La generación de números aleatorios no es trivial en hardware, la mayoría de los casos infieren una generación pseudoaleatoria de los individuos. En nuestro caso se logró con estas mismas características, pero que funcionó correctamente. El *speedup* presentado por las versiones en hardware, en comparación con su contraparte en software es mayor, tal y como se reportó en los resultados presentados en el Capítulo 4 de este mismo documento.

Trabajos a futuro

Es posible afirmar que el desempeño del micro algoritmo del SIA es tan bueno como el de su contraparte estándar, aunque con menores requerimientos de espacio en memoria de datos, lo que permite enfocar el diseño hacia aplicaciones con un bajo costo computacional o a implementaciones en circuitos embebidos refiriéndose al denominado hardware evolutivo. El manejo de espacios restringidos y la optimización con múltiples objetivos se pueden incorporar al esquema básico planteado, seguramente éste será un trabajo a futuro, al igual que explorar la adaptación de otros algoritmos bioinspirados comunes en la resolución de problemas de optimización.

La vertiente planteada está encamina a seguir trabajando con microcontroladores convencionales atacando problemas de decisión. También se está planteando la posibilidad de la implementación del algoritmo en dispositivos móviles tales como smartphones y PDAs, entre otros, sólo es cuestión de buscar las posibles aplicaciones. Con el uso mínimo de la memoria de datos para alojar individuos, por parte de nuestro algoritmo, es posible considerar esta propuesta.

Referencias

- [1] J. Koza and M. Keane. The importance of reuse and development in evolvable hardware. In Proceedings of 2003 NASA/DoD Conference on Evolvable Hardware. Los Alamitos, CA: IEEE Computer Society.
- [2] R. Zebulum and A. Stoica. Mixtrinsic evolution. In J. F. Miller, A. Thompson, P. Thomson, and T. C. Fogarty, editors, Proc. of Third International Conference on Evolvable System: From Biology to Hardware (ICES 2000), pages 208–217, Edinburgh, Scotland, April 2000. Springer-Verlag.
- [3] P.C. Haddow. An Evolvable Hardware FPGA for Adaptive Hardware. Congress on Evolutionary Computation pp. 553-560, 2000.
- [4] G. Harik, F. Lobo, D. Goldberg. The compact genetic algorithm. Evolutionary Computation, IEEE Transactions on Volume 3, Issue 4, Nov. 1999 Page(s):287 – 297.
- [5] David E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading, MA, 1989.
- [6] K. Krishnakumar. Micro-genetic algorithms for stationary and non-stationary function optimization. In SPIE Proceedings: Intelligent Control and Adaptive systems, pages 289–296, 1989.
- [7] G. Dozier, J. Bowen and D. Bahler. Solving Small and Large Scale Constraint Satisfaction Problems Using a Heuristic-Based Microgenetic Algorithm, in Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel and H. Kitano (editors), Proceedings of the First IEEE Conference on Evolutionary Computation (ICEC'94), pages 306-311.
- [8] C. A. Coello and G. Toscano. A Micro-Genetic Algorithm for multiobjective optimization. First International Conference on Evolutionary Multi-criterion Optimization. Springer – Verlag, Lecture Notes in Computer Science number 1993, 2001, pp. 126 – 140.

- [9] J. C. Fuentes and C. A. Coello. Handling Constraints in Particle Swarm Optimization Using a Small Population Size, in *Lecture Notes in Computer Science*, Springer. MICAI 2007: Advances in Artificial Intelligence, Volume 4827/2007.
- [10] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, New York, 1997.
- [11] D. B. Fogel. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*. The Institute of Electrical and Electronic Engineers, New York, 1995.
- [12] C. A. Coello. *Introducción a la computación evolutiva. Notas del curso*. CINVESTAV-IPN, Sección de Computación, Departamento de Ingeniería Eléctrica, 2003.
- [13] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69-93. Morgan Kaufmann Publishers, San Mateo, California, 1991.
- [14] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From natural to artificial systems*. Oxford University Press, 1999.
- [15] M. Dorigo & T. Stützle. *Ant Colony Optimization*, MIT Press, 2004.
- [16] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.
- [17] A. Tyrrell, G. Auer and C. Bettstetter. Bio-inspired networks and communication systems: Fireflies as role models for synchronization in ad hoc networks. December 2006. Proceedings of the 1st international conference on Bio inspired models of network, information and computing systems BIONETICS '06. Publisher: ACM.
- [18] A. Delgado. DNA chips as lookup tables for rule based systems. *IEE Computing and Control Engineering Journal*, Vol. 13, No. 3, pp. 113-119, 2002.
- [19] R. Sun and L. Bookman. *Computational Architectures Integrating Neural and Symbolic Processes*. Kluwer Academic Publishers, 1994.
- [20] D. Dasgupta and N. Attoh-Okine. Immunity-Based Systems: A Survey. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. October 1997.

- [21] L. N. de Castro and J. Timmis. An Introduction to Artificial Immune Systems: A New Computational Intelligence Paradigm. Springer-Verlag, 2002.
- [22] D. Dasgupta. Advances in artificial immune systems. Computational Intelligence Magazine, IEEE. Volume 1, Issue 4, Nov. 2006 Page(s):40 – 49.
- [23] N. Cruz. Sistema inmune artificial para solucionar problemas de optimización. Tesis Doctoral. CINVESTAV- IPN. México, 2004.
- [24] L. Nunes de Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comput., vol. 6, no. 3, pp. 239–251, Jun. 2002.
- [25] L. Nunes de Castro and F. J. Von Zuben. The clonal selection algorithm with engineering applications. Proceedings of Genetic and Evolutionary Computation Conference, Workshop on AISAA, pp. 36-37, July 2000.
- [26] L. Nunes de Castro. The immune response of an artificial immune network (aiNet). Evolutionary Computation, 2003. CEC '03. The 2003 Congress on Volume 1. Dec. 2003 Page(s):146 - 153 Vol.1.
- [27] L. Nunes de Castro and J. Timmis. An Artificial Immune Network for Multimodal Function Optimization. Proceedings of IEEE Congress on Evolutionary Computation (CEC'02), Hawaii, pp. 699- 674, 2002.
- [28] C. Apornthewan and P. Chongstitvatana. A hardware implementation of the Compact Genetic Algorithm. Evolutionary Computation, 2001. Proceedings of the 2001 Congress on Volume 1, 27-30 May 2001 Page(s):624 - 629 vol. 1.
- [29] J. Gallagher and G. Kramer. A family of compact genetic algorithms for intrinsic evolvable hardware. Evolutionary Computation, IEEE Transactions on Volume 8, Issue 2, April 2004 Page(s):111 – 126.
- [30] F. Cupertino and E. Mininno. Optimization of Position Control of Induction Motors using Compact Genetic Algorithms. IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on Nov. 2006. Page(s):55 – 60.
- [31] K. Deb. An efficient constraint handling method for genetic algorithms. Computer Methods in Applied Mechanics and Engineering, 186:311–338, 2000.

- [32] E. Mezura, J. Velázquez, C. A. Coello. A comparative study of differential evolution variants for global optimization. ACM, GECCO 2006: 485-492.
- [33] S. Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19-14, 1999.
- [34] J.D. Schaffer and L.J. Eshelman. On crossover as an evolutionary viable strategy. R.K. Belew and L.B. Booker, editors. *Proceedings of the 4th International Conference on Genetic Algorithms*. Page(s): 61-68, Morgan Kaufmann, 1991.

ANEXO A

Funciones de prueba para el micro-SIA sin manejo de restricciones

f01 – Sphere Model

$$f_1(x) = \sum_{i=1}^{30} (x_i)^2$$

$$-100 \leq x_i \leq 100$$

$$\min(f_1) = f_1(0, \dots, 0) = 0$$

f02 – Schwefel's Problem

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$$

$$-10 \leq x_i \leq 10$$

$$\min(f_2) = f_2(0, \dots, 0) = 0$$

f03 – Schwefel's Problem

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^i x_j \right)^2$$

$$-100 \leq x_i \leq 100$$

$$\min(f_3) = f_3(0, \dots, 0) = 0$$

f04 – Schwefel's Problem

$$f_4(x) = \max_i \{|x_i|, 1 \leq i \leq 30\}$$

$$-100 \leq x_i \leq 100$$

$$\min(f_4) = f_4(0, \dots, 0) = 0$$

f05 – Generalized Rosenbrock's Function

$$f_5(x) = \sum_{i=1}^{29} |100(x_{i+1} - x_i^2) + (x_i - 1)^2|$$

$$-30 \leq x_i \leq 30$$

$$\min(f_5) = f_5(1, \dots, 1) = 0$$

f06 – Step Function

$$f_6(x) = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2$$

$$-100 \leq x_i \leq 100$$

$$\min(f_6) = f_6(0, \dots, 0) = 0$$

f07 – Quartic Function with Noise

$$f_7(x) = \sum_{i=1}^{30} ix_i^4 + \text{random}[0,1]$$

$$-1.28 \leq x_i \leq 1.28$$

$$\min(f_7) = f_7(0, \dots, 0) = 0$$

f08 – Generalized Schwefel's Problem

$$f_8(x) = \sum_{i=1}^{30} (x_i \sin(\sqrt{|x_i|}))$$

$$-500 \leq x_i \leq 500$$

$$\min(f_8) = f_8(420.9687, \dots, 420.9687) = -12596.5$$

f09 – Generalized Rastrigin's Problem

$$f_9(x) = \sum_{i=1}^{30} [x_i^2 - 10 \cos(2\pi x_i) + 10]$$

$$-5.12 \leq x_i \leq 5.12$$

$$\min(f_9) = f_9(0, \dots, 0) = 0$$

f10 – Ackley's Function

$$f_{10}(x) = -20e \left(-0.2 \sqrt{\frac{1}{30} \sum_{i=1}^{30} x_i^2} \right) - e \left(\frac{1}{30} \sum_{i=1}^{30} \cos(2\pi x_i) \right) + 20 + e$$

$$-32 \leq x_i \leq 32$$

$$\min(f_{10}) = f_{10}(0, \dots, 0) = 0$$

f11 – Generalized Griewank's Function

$$f_{11}(x) = \frac{1}{4000} \sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

$$-600 \leq x_i \leq 600$$

$$\min(f_{11}) = f_{11}(0, \dots, 0) = 0$$

ANEXO B

Funciones de prueba para el micro-SIA con manejo de restricciones

g01

Min

$$f(x) = 5 \sum_{t=1}^4 x_t - 5 \sum_{t=1}^4 x_t^2 - \sum_{t=5}^{13} x_t$$

$$g(x)_1 = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0$$

$$g(x)_2 = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0$$

$$g(x)_3 = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0$$

$$g(x)_4 = -8x_1 + x_{10} \leq 0$$

$$g(x)_5 = -8x_2 + x_{11} \leq 0$$

$$g(x)_6 = -8x_3 + x_{12} \leq 0$$

$$g(x)_7 = -2x_4 - x_5 + x_{10} \leq 0$$

$$g(x)_8 = -2x_6 - x_7 + x_{11} \leq 0$$

$$g(x)_9 = -2x_8 - x_9 + x_{12} \leq 0$$

$$0 \leq x_t \leq 1 (i = 1, \dots, 9), 0 \leq x_t \leq 100 (i = 10, 11, 12) \text{ y } 0 \leq x_{13} \leq 1$$

$$\min(f) = f(1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1) = -15$$

g02

Max

$$f(x) = \left| \frac{\sum_{t=1}^n \cos^4(x_t) - 2 \prod_{t=1}^n \cos^2(x_t)}{\sqrt{\sum_{t=1}^n i x_t^2}} \right|$$

$$g(x)_1 = 0.75 - \prod_{t=1}^n x_t \leq 0$$

$$g(x)_1 = \sum_{t=1}^n x_t - 7.5n \leq 0$$

$$n = 20, 0 \leq x_t \leq 10 (i = 1, \dots, n)$$

$$\max(f) = f(x^*) = -0.803619$$

ANEXO C

Micro-algoritmo de evolución diferencial.

La Evolución Diferencial es un algoritmo evolutivo propuesto por Storn y Price [10] para resolver problemas de optimización principalmente en espacios continuos y en el cual las variables se representan mediante números reales. Actualmente se ha utilizado en problemas de alta complejidad con muy buenos resultados [11].

En la ED se genera de forma aleatoria una población inicial de individuos (vectores originales), de los cuales se seleccionan tres, también aleatoriamente y que sean distintos entre sí. A cada uno de estos individuos se le denomina vector objetivo y se denotan por $r1$, $r2$ y $r3$, en donde $r3$ es el vector base o principal. Con ayuda de un operador especial (F) se realiza una combinación lineal con las diferencias entre $r1$, $r2$ y $r3$, con la intención de generar nuevos vectores ruidosos, a esta etapa se le conoce como mutación. F es entonces, un factor que escala la diferencia entre vectores. Posteriormente se generan vectores de prueba a partir de la recombinación de vectores ruidosos y vectores originales, lo anterior se logra a través de un parámetro denominado tasa de recombinación (CR). Para finalizar el algoritmo se procede a seleccionar el vector que se copiará a la generación siguiente, comparando los vectores de prueba con los originales en base a su aptitud.

La versión más común de la ED es la denominada **DE/rand/1/bin**; que fue la que se abordó en este trabajo y se lista en Algoritmo 1, se recomienda referirse a [12] para mayores detalles del mismo. La cantidad de generaciones, el tamaño de la población y los parámetros CR y F , son definidos por el usuario.

Algoritmo 1. Evolución Diferencial estándar, versión DE/rand/1/bin

```

Begin
   $G = 0$ 
  Crear aleatoriamente la población inicial  $\vec{x}_G \forall i, i = 1, \dots, NP$ 
  Evaluar  $f(\vec{x}_G) \forall i, i = 1, \dots, NP$ 
  For  $G = 1$  to  $G_{max}$  Do
    For  $i = 1$  to  $NP$  Do
      Seleccionar aleatoriamente  $r_1 \neq r_2 \neq r_3$ 
       $j_{rand} = \text{randint}(1,D)$ 
      For  $j = 1$  to  $D$  Do
        If  $(\text{rand}[0,1] < CR \text{ or } j=j_{rand})$  then
           $u_{j,G+1}^i = x_{j,G}^{r_1} + F(x_{j,G}^{r_1} - x_{j,G}^{r_2})$ 
        Else
           $u_{j,G+1}^i = x_{j,G}^i$ 
        End if
      End for
      If  $(f(u_{G+1}^i) \leq f(x_G^i))$  then
         $\vec{x}_{G+1}^i = u_{G+1}^i$ 
      Else
         $\vec{x}_{G+1}^i = x_G^i$ 
      End if
    End For
     $G = G + 1$ 
  End For
End

```

La versión con población reducida de este mismo algoritmo se observa modularmente en la Figura 2. En nuestros experimentos con ED utilizamos una población de 5 individuos, ya que se comprobó funcionalmente que con un menor número de individuos se obtiene una convergencia prematura y utilizar más de 5 individuos no mejora los resultados. Se determinó que el criterio para alcanzar la convergencia nominal fuera al cabo de 5 generaciones, refiriéndose al ciclo interno del algoritmo. En este rubro también se realizaron pruebas con 10 generaciones, sin observarse alguna mejoría en los resultados.

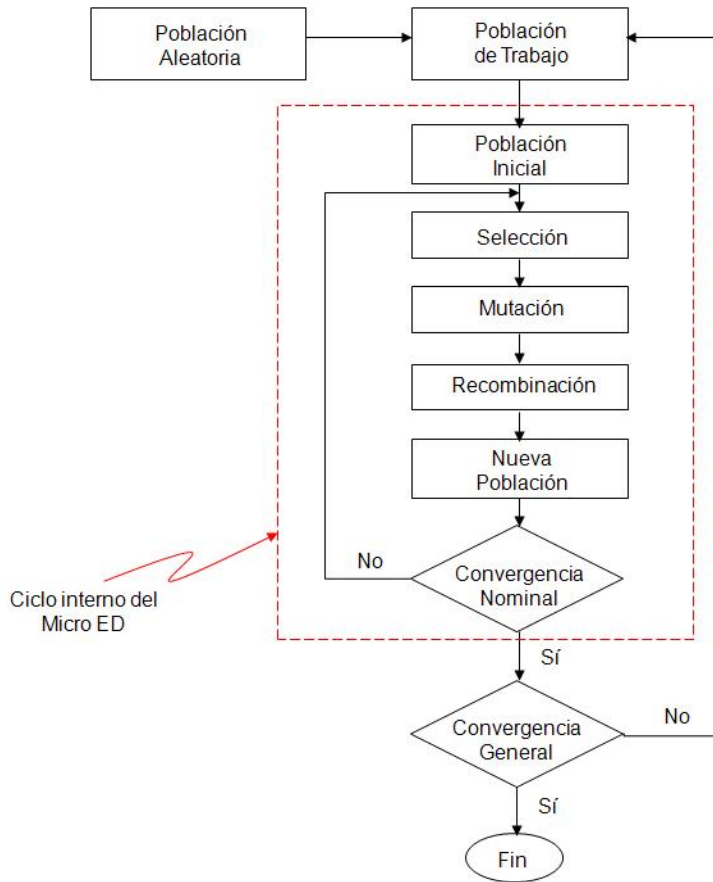


Figura 2. Diagrama a bloques del micro algoritmo de ED.

En la generación cero, utilizando una población aleatoria de 5 individuos, se procedió a copiar los mismos en la población de trabajo y en la población inicial. Nótese que para el ciclo interno en donde aplica la convergencia nominal, el algoritmo ED se implementa sin cambios en sus operadores y estrategias para evaluar la metodología de manera básica. Al alcanzarse la convergencia nominal es necesario determinar cuántos individuos se copiarán a la población de trabajo y aplicar el proceso de reinicialización. Después de una serie de experimentos, se decidió copiar los 4 mejores individuos y el restante se genera aleatoriamente. Los experimentos realizados contemplaron copiar 1, 2, 3 y 4 individuos con la mejor aptitud, sin embargo con 4 individuos se obtuvieron los mejores resultados. En este micro algoritmo la diversidad se mantiene gracias a las etapas de mutación y recombinación del propio algoritmo base de ED y al individuo aleatorio incorporado en la población de trabajo.

1 Experimentos y Resultados

Para los experimentos se utilizaron las siguientes funciones de alta dimensionalidad, referidas en [12]. $f1$ y $f2$ son funciones unimodales y separables. $f5$ es una función multimodal y no separable.

$f1$ – Esfera de De Jong

$$f(x) = \sum_{i=1}^{30} (x_i)^2 \quad (1)$$

$$-100 \leq x_i \leq 100$$

$$\min (f_1) = f_1(0, \dots, 0) = 0$$

$f2$ – Función techo

$$f(x) = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2 \quad (2)$$

$$-100 \leq x_i \leq 100$$

$$\min (f_2) = f_2(0, \dots, 0) = 0$$

$f5$ – Función generalizada de Rosenbrock

$$f(x) = \sum_{i=1}^{29} |100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2| \quad (3)$$

$$-30 \leq x_i \leq 30$$

$$\min (f_3) = f_3(1, \dots, 1) = 0$$

Las tres funciones fueron probadas en el micro algoritmos ED, realizando 20 ejecuciones para cada una y obteniendo los resultados que a continuación se discutirán. Se definieron los parámetros indicados en la Tabla 1, los cuales son sugeridos en [12]:

Tabla 1: Parámetros utilizados para el micro algoritmo de ED.

Función	CR	F
$f1$	0.9	$\in [0.3,0.9]$
$f2$	0.0	$\in [0.3,0.9]$
$f5$	0.0	$\in [0.3,0.9]$

En la Tabla 2 se indican las generaciones utilizadas para cada función, así como los resultados experimentales alrededor del valor óptimo.

Tabla 2: 20 ejecuciones del micro algoritmo de ED.

Función	Ciclo Externo	Ciclo interno	Mejor	Peor	Media
$f1$	300	5	0.0	0.0064	0.00010
$f2$	200	5	0.0	1.0	0.20
$f5$	300	5	0.0	0.0010	0.00012

En la Figura 3, se aprecia el comportamiento del algoritmo de ED estándar en su versión **DE/rand/1/bin** con respecto a la aptitud, para $f2$. Se utilizó una población de 60 individuos, 1000 generaciones y los mismos valores de CR y F listados en la Tabla 1.

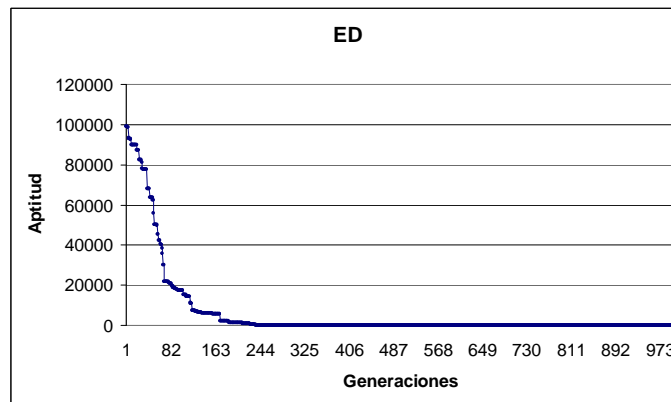


Figura 3. Aptitud para $f2$ con ED estándar.

Utilizando una población de 5 individuos y los parámetros indicados en la Tabla 2, se obtuvo la gráfica exhibida en la Figura 4, para $f2$ probada en el micro algoritmo de ED.

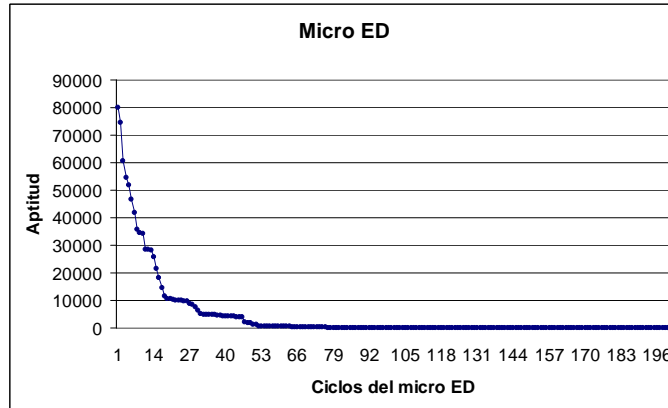


Figura 4. Aptitud para $f2$ con micro algoritmo de ED.

Es importante notar que para las funciones experimentales, el desempeño del algoritmo de población reducida es tan eficiente como el del algoritmo en su versión estándar. La cantidad de evaluaciones a la función objetivo es similar, considerando el ciclo interno del micro algoritmo.

2 Conclusiones

El micro algoritmo de ED (en su versión *DE/rand/1/bin*) generó buenos resultado con base en la metodología aplicada y con respecto a las funciones seleccionadas. En los resultados experimentales se obtuvo el óptimo para las tres funciones de prueba de alta dimensionalidad, lo que denota resultados alentadores para continuar con la investigación. Es posible afirmar que el desempeño del micro algoritmo de ED es tan bueno como el de su contraparte estándar, aunque con menores requerimientos de espacio en memoria de datos, lo que permite enfocar el diseño hacia aplicaciones con un bajo costo computacional o a implementaciones en circuitos embebidos (p. e. dispositivos de lógica programable y microcontroladores convencionales) refiriéndose al denominado *hardware evolutivo*. El manejo de espacios restringidos y la optimización con múltiples objetivos se pueden incorporar al esquema básico planteado, seguramente éste será un trabajo a futuro, al igual que explorar la adaptación de otros algoritmos bioinspirados comunes en la resolución de problemas de optimización.

Referencias

1. D. Ashlock. Evolutionary Computation for Modeling and Optimization. Springer, first edition, 2005.

2. M. Munetomo, Y. Satake. Enhancing Model-building Efficiency in Extended Compact Genetic Algorithms. *Systems, Man and Cybernetics*, 2006. ICSMC '06. IEEE International Conference on Volume 3, 8-11 Oct. 2006 Page(s):2362 – 2367.3.
3. J. Torresen. Possibilities and limitations of applying evolvable hardware to real-world application. In *Field-Programmable Logic and Applications: 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, volume 1896 of *Lecture Notes in Computer Science*, R. W. Hartenstein et al. (eds.), Springer-Verlag, 2000, pp. 230–239.
4. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989.
5. K. Krishnakumar. Micro-genetic algorithms for stationary and non-stationary function optimization. In *SPIE Proceedings: Intelligent Control and Adaptive systems*, pages 289–296, 1989.
6. G. Toscano, Carlos. A. Coello. A Micro-Genetic Algorithm for multiobjective optimization. *First International Conference on Evolutionary Multi-criterion Optimization*. Springer – Verlag, *Lecture Notes in Computer Science* number 1993, 2001, pp. 126 – 140.
7. J. C. Fuentes, C. A. Coello, Handling Constraints in Particle Swarm Optimization Using a Small Population Size, in *Lecture Notes in Computer Science*, Springer. *MICAI 2007: Advances in Artificial Intelligence*, Volume 4827/2007.
8. J. C. Fuentes. Un nuevo Algoritmo de optimización basado en optimización mediante cúmulos de partículas utilizando tamaños de población pequeños. Tesis de Maestría. CINVESTAV- IPN. México, 2008.
9. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, New York, 1997.
10. R. Storn, K. Price. *Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Technical Report TR-95-012, ICSI, March 1995, ftp.icsi.berkeley.edu. Accedido por última vez el 20 de junio de 2009.

ANEXO D

Bio-inspired architecture design for a PWM module

Abstract

This paper presents the analysis and design of a Pulse Width Modulation embedded unit as an alternative to convert digital signals to analog signals extrapolating the biological principle of DNA hybridization. Biological hybridization process is a binding event between two complementary DNA single strands and that leads to the formation of a double-stranded helix. The proposed technique to detect the hybridization is based on direct comparison of data, allowing a high degree of parallelism in the Bio-inspired architecture specially designed to regulate the light intensity of LED lamps as a home automation application.

1. Introduction

In digital electronic design, a PWM (Pulse Width Modulation) unit is basically a digital – analog converter (DAC) which can be used in solutions that do not require a high frequency of sampling, it converts a binary data into a series of pulses, so that the pulse duration is directly proportional to the value of binary data. PWM is

widely used in different areas of control systems, such as robotics, industrial process control, power control systems, and others [1].

With regard to home automation (domotics), PWM has been very useful in regulating the light intensity of lamps (LED, fluorescent and incandescent), as well as regulating valves and small engines that automate some process [2], [3].

When designing with programmable logic devices (FPGA, CPLD), embedded realization of such converters is common [4], so this paper proposes an alternative to design a PWM unit simulating the combination of DNA strands in order to regulate the light intensity of a LED lamp in an application for home automation and so contribute to energy savings. The artificial DNA hybridization suggests a simple mechanical that compare parallel binary strands stored in data registers looking for evidence that they are complementary to each other to validate a subsequent action as a response. This approach is

very similar to evaluation of inference rules in an expert system that tries to model the reasoning of a human operator [5].

2. Electronic detection of DNA hybridization

DNA (Deoxyribonucleic Acid) is a chemical substance that is found in the nucleus of cells, which stores the basic code of all life translated as biological instructions. In the molecular genetics theory about DNA hybridization, single strands of DNA from two different species are allowed to join together to form hybrid double helices, much like a twisted ladder. These hybrid segments of DNA are used to determine the evolutionary relatedness of organisms by examining how similar (or dissimilar) the DNA base pair sequences are; in other words, the degree of hybridization is proportional to the degree of similarity between the molecules of DNA from the two species.

The way to detect the hybridization of two single strands of DNA has been replicated in the field of electronics through DNA Array, also called DNA chip or Gene array [6]. This chip is a matrix structure on which are distributed DNA single strands that have been implanted into a

silicon base. These sequences have a simple fixed value that when they are incubated with strands injected up to the chip, generating helices of DNA that can be detected through electronic tools. Engineering has shown that DNA chips can be used to store and evaluate in parallel, Boolean or fuzzy rules [5], [7].

In electronic design, hybridization can be implemented using a reference register that will be compared in parallel with others test registers, looking for evidence that they are complementary. In Figure 1 test register represents the simple sequence of DNA with a fixed value and reference register is the DNA strand that is injected or introduced to chip. To maintain the analogy with biological hybridization, the reference register is unique and will be compared with all test registers in parallel form.

Note that to make this comparison, both registers must have the same size and only if they are complementary, the flag output will be high logical value. To verify that the bits of both registers are mutually complementary, XOR logic gates are used for bitwise comparison.

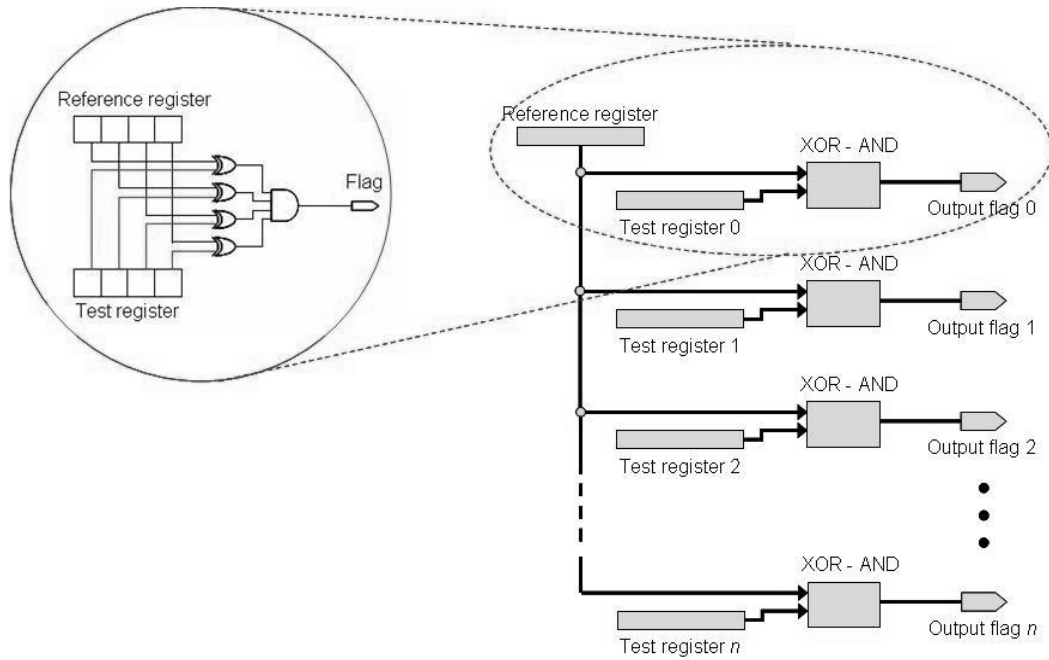


Figure 1. Standard model of artificial hybridization with digital electronic.

3. Pulse width modulation (PWM)

In digital electronic circuits a PWM embedded unit is usually accomplished by connecting a binary counter and a comparator circuit together; this last one will determine when data applied to the entrance of the unit is less than the value of binary counter constantly changing. At the output of the PWM unit, is necessary to connect a low-pass RC filter to determine the voltage of the analog signal equivalent to digital data entered [8]. The entire period of a PWM cycle is equal to the product of the period clock signal reference (system clock)

with 2^n , where n is the number of bits of the counter circuit proposed.

In the particularity of this work were considered 4-bit data registers, so in [8] is demonstrated that a converter of more than 8 bits does not get more benefits, so 16 different levels (at a rate of 2^4) of luminous intensity are adequate.

4. Bio-inspired architecture

We consider a modular design consisting of three parts. First part of this realization uses a reference register to be compared in parallel with every test registers, simultaneously.

According to the scheme established with 4-bit registers, 16 test registers were designed with fixed and different values, and a single reference register which establishes the digital data to convert to their analog equivalent in a range of 0 to 5 V, for example, a binary input equal to 0000 will get an analog voltage of 0V; in the same way, a binary input equal to 1111 will get the maximum analog value equal to 5V.

The fixed values of the test registers were assigned complementary in correspondence to the values that they can take at the rate of 2^n , in other words, for a binary entry 0000, stored in the reference register, the test register labeled as "0" will store a binary data 1111; for an entry 0001, it was assigned 1110 in the test register labeled as "1". Just the only one output from the module in Figure 1 is a data flag that indicates a successful hybridization through a high logical level.

In the second part of our design each flag from the previous stage activates individually a tri-state data register which is connected to a common output bus. This time were considered 16-bit tri-state registers to improve the resolution of the converter. In this scheme, only a single tri-state register may be enabled at a time. Each tri-state register has a fixed value assigned according to the binary value that will be delivered as result of modulation.

The third and final stage of architecture, is the stage of control, consisting of a 16-bit multiplexer that with help of a 4-bit binary counter, performs a binary sweep of the only tri-state register enabled, starting with the most significant bit. The same binary counter determines that after 16 pulses clock, when the unit PWM has finished data conversion, the reference register can receive a new data to process.

Figure 2 shows the schematic diagram of the complete unit designed with the three parts. All modules were described in VHDL for logic synthesis.

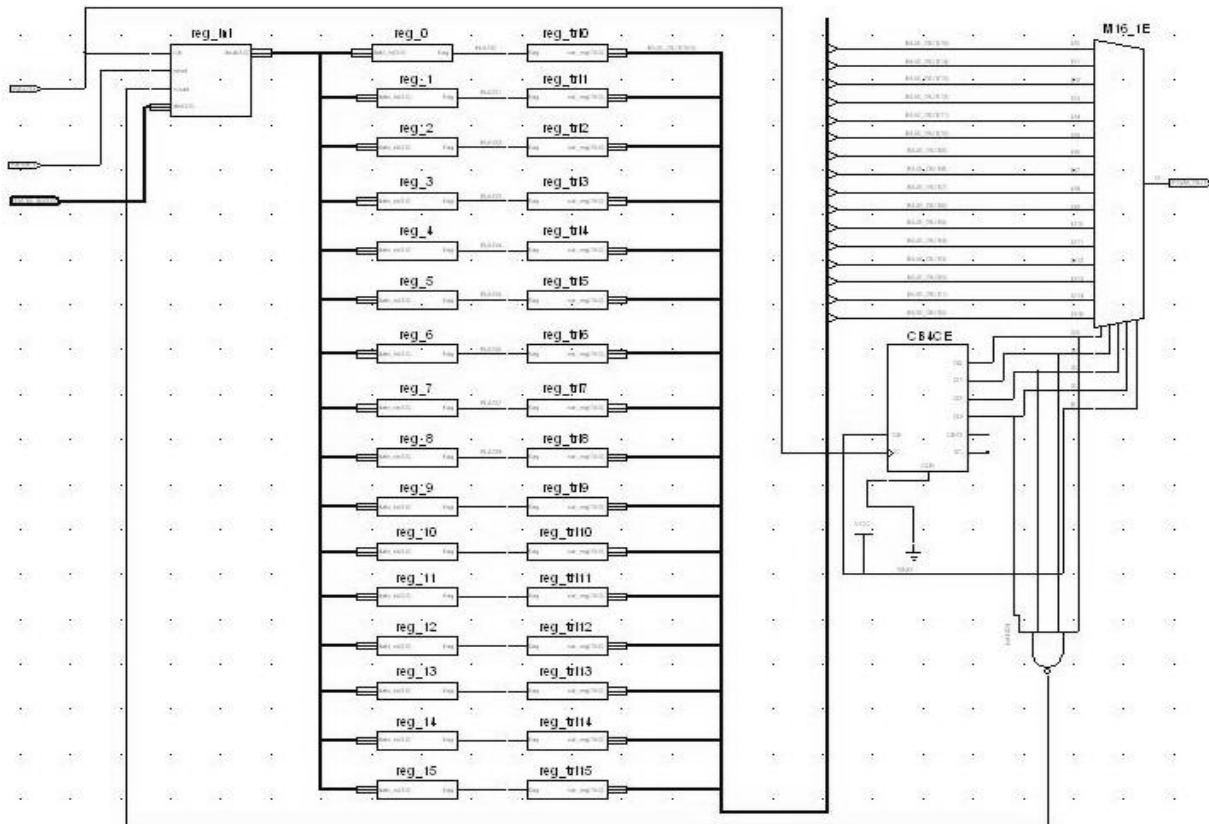


Figure 2. PWM parallel architecture using the principle of hybridization of DNA strands.

5. Results

The logic synthesis design was carried out using ISE Webpack v.8.1i, to configure a 3S200 Spartan 3 FPGA of vendor Xilinx. The PWM Bio-inspired unit implemented uses only 4% of the amount of internal resources of FPGA.

The PWM unit was tested at different frequencies, in each case calculating the values of RC filter. They were considering the following clock frequencies: 4MHz, 1MHz, 700Hz, 320Hz and 120Hz. Brief details of the conversion can be seen in Figure 3. It verifies the correct operation of PWM generator.

Tri-state registers	Binary data
reg_tri0	0000000000000000
reg_tri1	1000000000000000
reg_tri2	1100000000000000
reg_tri3	1110000000000000
reg_tri4	1111000000000000
reg_tri5	1111100000000000
reg_tri6	1111110000000000
reg_tri7	1111111000000000
reg_tri8	1111111100000000
reg_tri9	1111111110000000
reg_tri10	1111111111000000
reg_tri11	1111111111100000
reg_tri12	1111111111110000
reg_tri13	1111111111111000
reg_tri14	1111111111111100
reg_tri15	1111111111111110

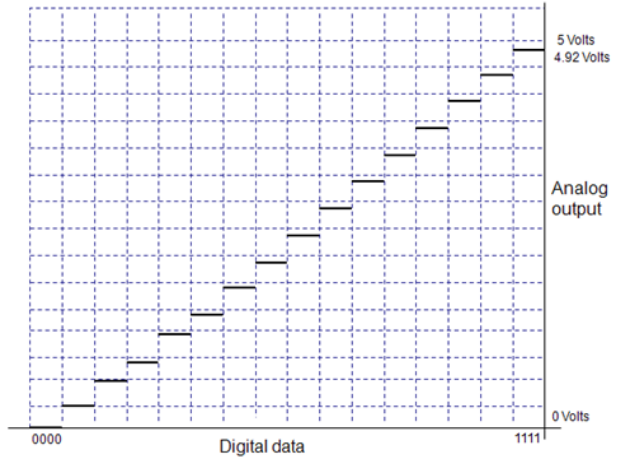


Figure 3. Details of the digital–analog conversion in practical experiments.

It was necessary to design an electronic power stage to attach the PWM output with the LED lamp used in laboratory. In Figure 4 we show the lamp used in our experiments. This lamp was designed with similar characteristics as commercial lamps with a panel of 18 x 18 high brightness LEDs.

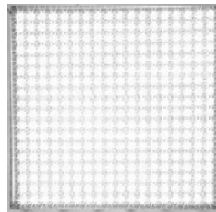


Figure 4. LED lamp used in laboratory.

6. Conclusions

The artificial hybridization of DNA single strands allows directing a design based on

inference rules, into a natural parallelization that can be explored through implementation in programmable logic devices. This technique transferred to Bio-inspired hardware has allowed the completion of a functional PWM generator as an alternative to convert digital signals to analog signals.

Each tri-state register enabled individually, as a result of a successful hybridization, stores a value that can be updated in real-time increasing the scope of this proposal. This will allow attacking the quantification problem, which directly affects the converter resolution and that a conventional PWM unit could not stand it.

This modular design supports changing the content of the registers according to the needs of modulation, being possible to increase or decrease the resolution of the conversion.

It will be interesting to explore the capabilities of parallel evaluation of inference rules in expert systems more complex; in future work we will apply this same technique to create intelligent systems that allow the automatic regulation of the luminous intensity in conventional lamps, focusing more specialized home automation applications, encouraging a systematic energy saving.

7. References

- [1] Zrilic, D. G.: Alternative Approach to Use of Pulse Width Modulation, Automation Congress, 2006. WAC '06. World IEEE, 24-26 July 2006, pp. 19–24.
- [2] Alonso J. M.; Cardesin, J.; Calleja, A. J.; Rico-Secades, M.; García, J.: A fluorescent lamp electronic ballast for railway applications based on low cost microcontroller, Industry Applications Conference, 2003. 38th IAS Annual Meeting IEEE. Conference Record of the Volume 1, Issue , 12-16 Oct. 2003, pp. 523 – 530.
- [3] Sandu, F.; Romanca, M.; Nedelcu, A.; Borza, P.; Dimova, R.: Remote and mobile control in domotics, Optimization of Electrical and Electronic Equipment, 2008. OPTIM 2008. 11th International Conference on 22-24 May 2008, pp. 225 – 228.
- [4] Dan, D.; Su, C.; Joss, G.: FPGA implementation of PWM pattern generators [for PWM invertors], Canadian Conference on Electrical and Computer Engineering IEEE, 2001, vol.1, pp. 225-230.
- [5] Delgado, A.: Rule base evaluation using DNA chips, Proceedings American Control Conference, pp. 3242 – 3245, Anchorage - Alaska, Mayo 8-10, 2002.
- [6] Moeller, R.; Fritzsche, W.: Chip-based electrical detection of DNA. Nanobiotechnology, IEE Proceedings - Vol 152, Issue 1, 4 Feb. 2005, pp. 47 – 51.
- [7] Delgado, A.: DNA chips as lookup tables for rule based systems. IEE Computing and Control Engineering Journal 2002, Vol. 13, No. 3, pp. 113-119.
- [8] Yu, Z.; Mohammed, A.; Panahi, I.: A Review of three PWM Techniques, American Control Conference, 1997, Proceedings of the 1997. Vol. 1, 4-6 June 1997, pp. 257 – 261 Khalil, Hassan K. Nonlinear Systems. Third Edition. Ed. Prentice Hall. 2002. ISBN 0-13-06-7389-7.

