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][22].

Para Nunes De Castro y Timmis en [21], los sistemas inmunes artificiales son sistemas adaptativos, inspirados en la teoría inmunológica, funciones, principios y modelos estudiados, los cuales son aplicados a la solución de diversos problemas. En la actualidad existen varios modelos algorítmicos del sistema inmune artificial y diversas aplicaciones de éste, por lo mismo el SIA no tiene un algoritmo habitual único (como sería el caso de un algoritmo genético). Los mismos autores sugieren en [21] que para diseñar un SIA es admisible considerar un esquema similar al que utiliza cualquier otro sistema bioinspirado, es decir, se requiere una representación predefinida de los componentes que intervienen en el sistema (simbólica, real o binaria), un conjunto de mecanismos que permitan evolucionar a éstos (midiendo la afinidad o aptitud) y un proceso que controle las dinámicas del sistema (el algoritmo).

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 (Ag)*, 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 biológico tiene cuatro capas o niveles de defensa principales; cada uno de éstas 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 (Ab)*. 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. La figura 2.1 muestra los cuatro niveles de defensa del sistema inmune biológico.

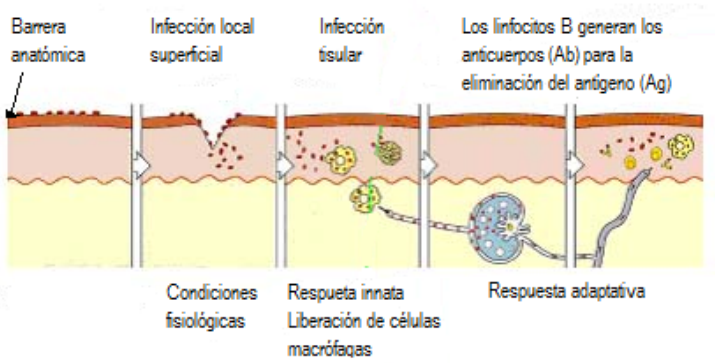

Figura 2.1. Niveles de defensa del sistema inmune biológico.

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. En la figura 2.2 se aprecia el acoplamiento entre un epitope y un paratope.

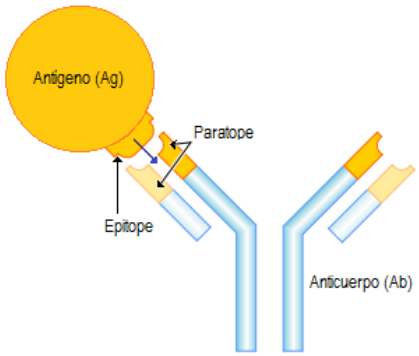

Figura 2.2. Acoplamiento del epitope de un antígeno con el paratope de un anticuerpo.

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 adaptativa; 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 2.3 muestra el principio de la selección clonal, en donde el *anticuerpo B* 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 (maduración).

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 de 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*).

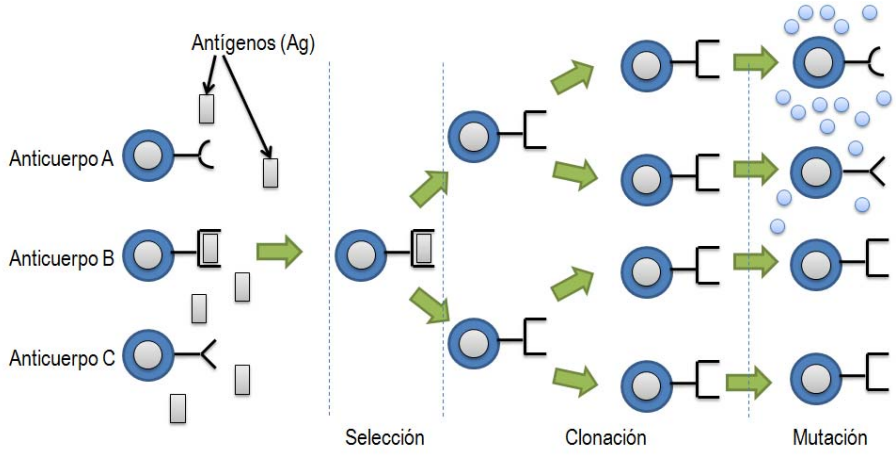

Figura 2.3. Principio de Selección Clonal.

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.

En la terminología empleada en el área emergente del SIA, entiéndase lo siguiente: un *antígeno* es la función objetivo del problema a resolver y los *anticuerpos* son las posibles soluciones al mismo, siendo estos últimos el equivalente a los *cromosomas* (individuos) en un algoritmo genético. La calidad de un individuo se determina evaluándolo en la función objetivo, por lo que a esta medida se le conoce como *aptitud* o *fitness* cuando se está trabajando con un algoritmo genético, en contraparte, para el SIA se le conoce como la *afinidad* entre los anticuerpos y el antígeno.

Para resolver problemas de optimización numérica y de reconocimiento de patrones, la literatura especializada reporta varios algoritmos derivados de los procesos observados en el sistema inmune biológico, sin embargo, son dos los que se consideran básicos y a partir de éstos surgen otros: *Clonal Selection Algorithm (CLONALG)* y *Artificial Immune Network (AiNet)*.

2.1 CLONALG

CLONALG está basado en el *principio de selección clonal* [24], [25] que se comentó al principio de este capítulo y cuyo funcionamiento considera la respuesta inmune ante el estímulo de un antígeno. 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 obtenido de la evaluación en la función objetivo, representada por el antígeno), 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 somáticamente (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. En la figura 2.4 se lista el algoritmo CLONALG propuesto por Nunes y Von Zuben [24]. Originalmente fue diseñado para aplicaciones de reconocimiento de caracteres y posteriormente lo modificaron para que pudiera resolver problemas de optimización numérica multimodal (las funciones presentan varios mínimos locales). En [23] fue extendido para trabajar con problemas de optimización con restricciones.

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*. Evaluar la afinidad de cada anticuerpo mutado con el antígeno.

   2.4 *Autorregulación*. Una vez exterminados los antígenos, el sistema inmune artificial 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.

Figura 2.4. Algoritmo CLONALG.

En la figura 2.5 se muestra el diagrama de flujo de CLONALG. En este diagrama, C representa la población temporal de clones y C^* es la población de clones que mutó (maduró). Para obtener el número de clones de cada anticuerpo de la población ya seleccionada por ranking, se utiliza:

$$Nc = \sum_{i=1}^{n} round\left(\frac{\beta . N}{i}\right) \qquad (2.1)$$

en donde Nc es el número de clones, N es el número de anticuerpos de la población, β es un factor multiplicativo (generalmente igual a 1), y finalmente, i es la posición del anticuerpo en el ranking, siendo $i=1$ el mejor de los anticuerpos.

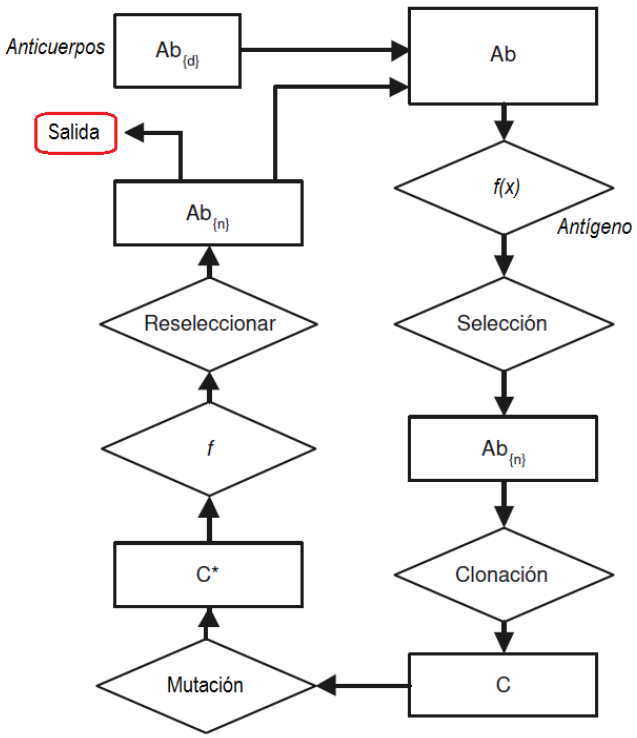

Figura 2.5. Diagrama de flujo de CLONALG.

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 idiotípica* 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 2.2.

$$S = N_{st} - N_{\sup} + A_s \qquad (2.2)$$

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 antígena. En la figura 2.6 se explica 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 Antígeno*. 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.

Figura 2.6. Algoritmo AiNet.

2.3 Estado del Arte

Los problemas de optimización del mundo real presentan frecuentemente restricciones que generalmente resultan difíciles de satisfacer. Considerando que matemáticamente el problema de minimizar $f(\vec{x})$ es equivalente al de maximizar $-f(\vec{x})$, el problema general de la optimización numérica global se define de la siguiente manera [23]:

Encontrar $\vec{x}$ que optimice (maximice o minimice) $f(\vec{x})$ (2.3)
sujeto a:

$$g_i(\vec{x}) \leq 0, \quad i = 1,\ldots,n \tag{2.4}$$

$$h_j(\vec{x}) = 0, \quad j = 1,\ldots,p \tag{2.5}$$

donde $\vec{x}$ es el vector de r variables del problema $\vec{x} = [x_1, x_2, \ldots, x_r]^T$, n es el número de restricciones de desigualdad $g_i(\vec{x})$ y p es el número de restricciones de igualdad $h_j(\vec{x})$; en ambos casos las restricciones pueden ser lineales o no lineales. En la figura 2.7 se aprecia el espacio de búsqueda en un problema de optimización global. Una solución factible es aquella que satisface todas las restricciones o condiciones específicas del problema, es decir, si cumple las ecuaciones (2.4) y (2.5). Es mejor solución aquella que se encuentre dentro de la región factible sin importar su valor en la función objetivo. Se les denomina *restricciones activas* a aquellas que cumplen con la igualdad cuando alcanzan el valor óptimo de la función.

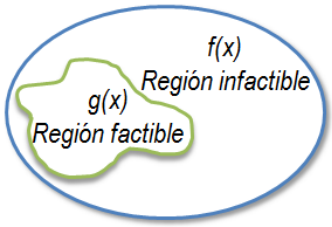

Figura 2.7. Espacio de búsqueda en un problema con restricciones.

En [5] Goldberg realizó varios experimentos utilizando un *algoritmo genético simple* (*AG*) con representación binaria. Probó una población de sólo 3 individuos, cromosomas propiamente, 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 esta convergencia limitada, 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 precisó 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 concebidos aleatoriamente. De esta manera se incrementa una iteración del ciclo externo del algoritmo y se utiliza la nueva población. 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 muestra en el algoritmo de la figura 2.8.

1. *Generar aleatoriamente una micro-población*. Considerar tres individuos (cromosomas).

2. *Aptitud y elitismo.* Determinar la *aptitud* de cada individuo y el mejor se mantiene para la próxima generación (estrategia de *elitismo*).

3. *Selección.* Los padres de los restantes individuos, menos aptos, se determinan utilizando alguna estrategia de selección, por lo general, torneo binario.

4. *Analizar la convergencia de la micro-población*. Si la población converge (haciendo referencia a la convergencia nominal), regresar al paso 1, manteniendo al mejor individuo y generando aleatoriamente a los restantes. Si la población no converge, regresar al paso 2.

Figura 2.8. Micro - Algoritmo Genético propuesto por Goldberg.

Existe una población aleatoria de tres cromosomas que se copia a la población inicial. Después de haber aplicado los operadores genéticos se itera el procedimiento de evolución de acuerdo a la convergencia nominal. Si no se ha concluido la convergencia nominal, la población de trabajo siguen siendo los tres cromosomas que se están evolucionando, si ya concluyó la convergencia, el mejor cromosoma evolucionado se copia a la población inicial y ésta se completa con dos cromosomas generados aleatoriamente. En la figura 2.9 se aprecian los ciclos interno y externo del micro-AG; el ciclo interno acaba cuando se alcanza la convergencia nominal y el ciclo externo concluirá hasta que se haya alcanzado la convergencia general.

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-AG).* 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 algunos mecanismos para el manejo de espacios restringidos bajo una técnica de no penalización.

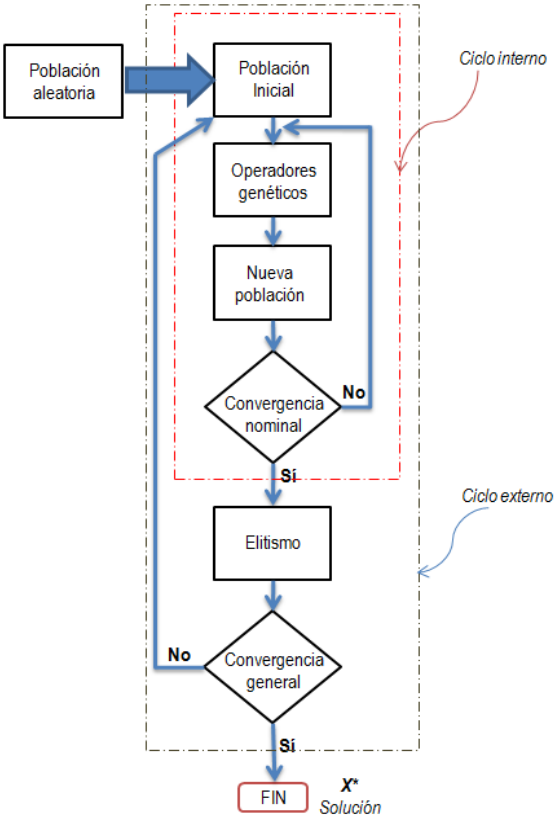

Figura 2.9. Ciclos interno y externo del Micro - Algoritmo Genético propuesto por Goldberg.

Coello y Toscano [8], diseñaron un micro AG para resolver problemas de optimización de múltiples objetivos, aportando criterios para el manejo de restricciones de igualdad y desigualdad, además de proponer un esquema de *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 únicamente 4 individuos (cromosomas) 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 micro-población 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 2.10, muestra una aproximación al diagrama de flujo del micro-AG multi-objetivo.

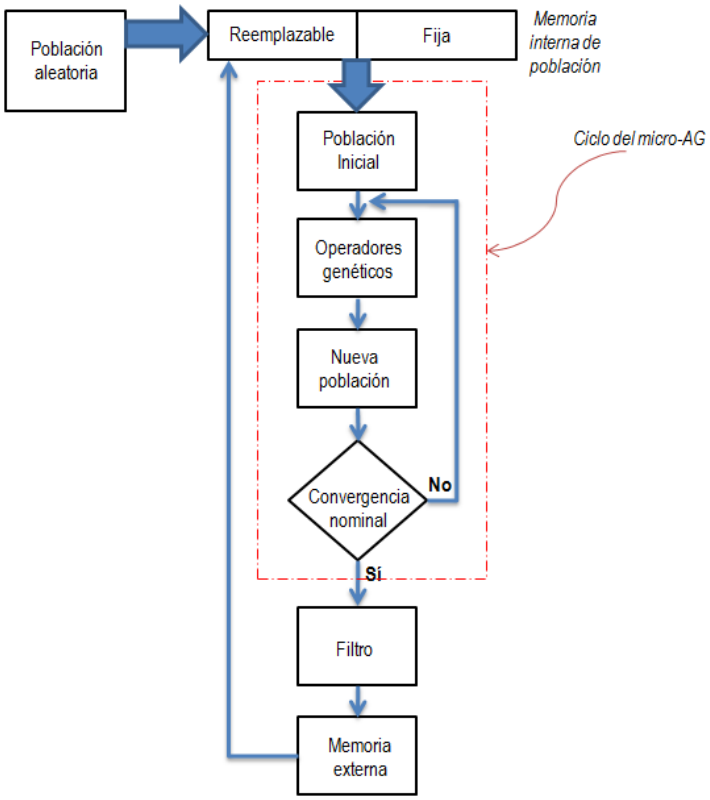

Figura 2.10. Micro-AG multi-objetivo propuesto por Toscano.

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 vez 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 $p_i \in [0,1] \ \ \forall i = 1,...,l$, 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 se generan dos individuos, basándose en la representación elegida y compara los valores de sus funciones de aptitud. Se le denomina *W* a la representación del individuo con mejor aptitud 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 2.6:

$$p_{i_i}^{k+1} = \begin{cases} p_i^k + 1/n & si \ \ W_i = 1 \wedge L_i = 0 \\ p_i^k - 1/n & si \ \ W_i = 0 \wedge L_i = 1 \\ p_i^k & si \ \ W_i = L_i \end{cases} \qquad (2.6)$$

siendo n la dimensión de la población simulada y $W_i(L_i)$ el valor del alelo *iésimo* 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. Aporntewan *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 esta realización es que Cupertino busca reducir el coste computacional sin demeritar el desempeño de su controlador, el cual fue implementado en un dispositivo 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 este trabajo 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 ha sido 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 define un vector de probabilidades como un registro lineal de datos binarios.

Entonces, 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 cantidad de individuos de dicha 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 ó más, dependiendo del problema a resolver) se está infiriendo un número muy grande de datos que se tienen que manipular de manera simultánea. En este trabajo de tesis se propuso utilizar sólo 5 anticuerpos y mantener un crecimiento fijo de sólo 15 clones durante la fase de clonación, lo que permitió realizar menos evaluaciones a la función objetivo. Con ayuda de los operadores de mutación diseñados también fue posible acelerar la convergencia del micro-algoritmo en comparación con la versión estándar del mismo.

La micro-población facilita la migración del algoritmo para optimización numérica implementado y probado inicialmente en software, a una plataforma en hardware, pudiéndose realizar aplicaciones clásicas en el reconocimiento de patrones, como la maximización de unos en una cadena finita y el reconocimiento de caracteres.