Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Ingeniería Eléctrica

Sección de Computación

Optimización global en espacios restringidos mediante un sistema inmune artificial.

Tesis que presenta

Daniel Trejo Pérez

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

con Opción en Computación

Director de la Tesis

Dr. Carlos A. Coello Coello

México, D.F. Agosto de 2005

Este trabajo esta dedicado a:
Toda mi familia. Muy en especial a mis padres.

Este trabajo de tesis se derivó del proyecto NSF-CONACyT titulado “Artificial Immune Systems for Multiobjective Optimization” (Ref. 42435-Y), cuyo responsable es el Dr. Carlos A. Coello Coello.

Gracias al Dr. Carlos A. Coello Coello por su valiosa dirección.

Gracias a la Dra. Nareli Cruz Cortés y al Dr. Efrén Mezura por su apoyo y orientación.

Gracias al Dr. Arturo Díaz y al Dr. Francisco José Rodriguez por sus comentarios acerca de este trabajo.

Gracias al Consejo Nacional para la Ciencia y Tecnología (CONACYT) por la beca que hizo posible mis estudios.

Un agradecimiento al personal administrativo de las distintas bibliotecas de la institución. en especial al personal de la biblioteca de Ingeniería Eléctrica.

Gracias al Centro de investigación y estudios avanzados (CINVESTAV).

Resumen

Los sistemas inmunes artificiales se han establecido como una nueva disciplina de estudio dentro de las heurísticas bio-inspiradas, atrayendo una enorme atención de investigadores de todas partes del mundo. Los sistemas inmunes artificiales se basan en los componentes y funciones del sistema inmune humano y de algunos animales, utilizando sus principios y teorías para desarrollar algoritmos que puedan resolver problemas de distintos tipos.

En años recientes, el uso de los sistemas inmunes artificiales se ha extendido a una amplia gama de disciplinas científicas, industriales y de ingeniería. Sin embargo, de entre estas diversas aplicaciones de los sistemas inmunes artificiales, la optimización global ha permanecido como un área muy poco explorada.

En esta tesis se plantea el uso de un sistema inmune basado en el principio de selección clonal para resolver problemas de optimización global. Inicialmente, se explora la optimización de funciones sin restricciones y posteriormente se aborda la optimización con restricciones. El algoritmo propuesto se basa en el principio de selección clonal y su validación se realiza adoptando funciones de prueba estándar y comparando resultados contra algoritmos representativos del estado del arte en el área.

Los resultados obtenidos indican que el uso de sistemas inmunes artificiales en optimización global es una alternativa viable.

Abstract

Artificial immune systems have established themselves as a new area of study within the bio-inspired heuristics, attracting a lot of attention from researchers all over the world. Artificial immune systems are based on components and functions from both the human and some animal's immune system, using its principles and theories to develop algorithms that can solve different types of problems.

In recent years, the use of artificial immune systems has extended to a wide variety of scientific, industrial and engineering disciplines. However, from within these diverse applications of the artificial immune systems, global optimization has remained as a very little explored area.

In this thesis, we propose the use of an artificial immune system based on the clonal selection principle to solve global optimization problems. Initially, we explore the optimization of unconstrained functions. Then, we focus on constrained optimization. The proposed algorithm is based on the clonal selection principle and its validation is performed adopting standard test functions and comparing results with respect to algorithms representative of the state-of-the-art in the area.

The results obtained indicate that the use of artificial immune systems in global optimization is a viable alternative.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El manejo de restricciones es un tema muy importante en optimización global dado que en el mundo real los problemas tienen por lo general restricciones de diferentes tipos (lineales y no lineales, de igualdad y desigualdad, etc.). En los últimos años se han realizado muchos esfuerzos en torno a desarrollar esquemas para optimización con restricciones utilizando diversas meta-heurísticas debido principalmente a que estas técnicas no necesitan que la función objetivo sea diferenciable o continua, es decir, no tienen muchas de las limitaciones de las técnicas clásicas de programación matemática.

Una meta-heurística (relativamente reciente) conocida como sistema inmune artificial ha tenido excelentes resultados en problemas de reconocimiento de patrones y seguridad informática. A pesar de la creciente popularidad del sistema inmune artificial en la solución de problemas de distintos tipos, no existen muchos estudios en torno al uso de esta meta-heurística en optimización global.

Por lo anterior, esta tesis tiene como objetivo principal desarrollar un algoritmo basado en algún esquema del sistema inmune artificial que sea competitivo con algoritmos representativos del estado del arte en problemas de optimización global con restricciones. Para completar dicho objetivo es necesario: Estudiar los modelos del sistema inmune artificial y definir cuál de ellos puede convertirse en buen optimizador. Estudiar y elegir un mecanismo de manejo de restricciones para adaptarlo al algoritmo. Llevar a cabo un estudio empírico de la implementación y compararla contra otras técnicas representativas del estado del arte usando funciones de prueba estándar reportadas en la literatura especializada.

En base a la metodología propuesta se desarrolló un algoritmo capaz de lidiar con problemas con restricciones, el cual resulta competitivo con respecto a dos algoritmos representativos del estado del arte en el área. El primero, conocido como jerarquías

estocásticas, es considerado como el mejor método de manejo de restricciones para algoritmos evolutivos disponible a la fecha. El segundo, llamado algoritmo evolutivo basado en la evolución de anticuerpos y autoanticuerpos (EAA) utiliza un esquema del sistema inmune como en nuestro caso por lo que lo usamos como referencia a fin de validar el desempeño de nuestra propuesta con respecto a otra previamente diseñada para la misma meta-heurística adoptada como nuestro motor de búsqueda. Adicionalmente se hace notar que el EAA fue producto de una tesis doctoral

La tesis está organizada de la siguiente manera: En el capítulo 1 se presenta el marco teórico del trabajo. Se plantea el problema y se describen algunos algoritmos evolutivos vistos como técnicas para resolver problemas de optimización. En el capítulo 2 se presenta una introducción al sistema inmune artificial, describiendo sus fundamentos biológicos y definiendo sus componentes computacionales. En el capítulo 3 se desarrollan los componentes del algoritmo propuesto tomando en cuenta los mecanismos de manejo de restricciones necesarios para resolver problemas de optimización global con restricciones. En el capítulo 4 se presentan los resultados obtenidos por el algoritmo en veintitrés funciones de prueba sin restricciones que nos ayudan a probar las capacidades de nuestra propuesta para este tipo de problemas. Finalmente el capítulo 5 cuenta con los resultados para trece funciones de prueba con restricciones. En los capítulos 4 y 5 se lleva a cabo un análisis estadístico de resultados para demostrar que los resultados obtenidos por nuestro algoritmo son representativos de su comportamiento promedio.

Capítulo 2

Marco teórico

2.1. Planteamiento del problema

Optimización es la acción de obtener el mejor resultado bajo ciertas circunstancias de diseño, construcción y mantenimiento de cualquier sistema de ingeniería [33]. El objetivo de la optimización es minimizar el esfuerzo requerido para maximizar los beneficios obtenidos. Tanto el beneficio como el esfuerzo pueden ser expresados en función de ciertas variables de decisión, por lo que optimización se puede definir como el proceso de encontrar las condiciones que maximicen o minimicen el valor de una función [33]. Cuando sólo tenemos una función a la cual queremos optimizar decimos que estamos llevando a cabo optimización global o mono-objetivo. Cuando contamos con dos o más funciones objetivo, realizamos optimización multiobjetivo.

A los métodos de búsqueda de un óptimo se les denomina técnicas de programación matemática. Aquellos métodos que se utilizan en problemas en los cuales alguna de las funciones que participan, ya sea la objetivo o alguna restricción, son no lineales se conocen como técnicas de programación matemática no lineal. Estas técnicas se suelen estudiar en investigación de operaciones que es la rama de las matemáticas que se ocupa de técnicas aplicables a problemas de toma de decisiones, así como el establecimiento de las mejores soluciones posibles (o sea, óptimas) [33].

El problema general de optimización no lineal, que es el que nos interesa para fines de esta tesis, se define como sigue [33]:

Encontrar:

$$X = \left\{ \begin{array}{c} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{array} \right\} \text{ que minimice (maximice) } f(X)$$

sujeto a las restricciones:

$$g_j(X) \leq 0, \ j = 1, 2, ..., m$$

$$h_j(X) = 0, \ j = 1, 2, ..., p$$

donde X es un vector de dimensión n llamado vector de variables de decisión, $f(X)$ es la función objetivo la cual puede ser lineal o no lineal, $h_j(X)$ representa las restricciones de igualdad (por ejemplo, $x^2+y^2+z^2 = 1$) y $g_j(X)$ representa de desigualdad (por ejemplo, $x^2 + y^2 + z^2 \leq 0.5$). Ambos tipos de restricciones pueden ser lineales o no lineales.

Podría pensarse que si el espacio de búsqueda se reduce con las restricciones entonces se puede llegar a una solución de manera más rápida pero esto no necesariamente es cierto. De hecho, la presencia de restricciones en un problema puede alterar de tal forma las características del espacio de búsqueda que su exploración se torna, en consecuencia, sumamente difícil.

Para el problema de optimización lineal existen técnicas de programación matemática como el método simplex [33] que lo resuelven satisfactoriamente. Sin embargo, aunque existen técnicas de este tipo que intentan resolver el problema de optimización no lineal, sus alcances son muy limitados debido a las propiedades que tienen las funciones no lineales. Algunas técnicas requieren que la función a optimizar sea diferenciable o continua, entre otras características. Por otro lado, muchas de estas técnicas necesitan una solución inicial y es a partir de ella que comienza la búsqueda, esto trae complicaciones; por ejemplo, en muchas ocasiones los procesos pueden quedar atrapados en óptimos locales por lo que se necesitan varias ejecuciones del algoritmo con distintos puntos iniciales. Evidentemente, este tipo de algoritmos dependen, en buena medida, de la experiencia del usuario. Lo anterior explica el uso de la computación evolutiva para resolver los problemas de optimización no lineal ya

que los algoritmos no imponen restricciones a las propiedades de las funciones. Las soluciones iniciales adoptadas por los algoritmos evolutivos son generadas de manera aleatoria y no son dependientes de la experiencia del usuario.

Se hace notar que aunque existen condiciones necesarias y suficientes de optimalidad para problemas no lineales, éstas resultan sumamente difíciles de verificar en la práctica. De ahí que la solución del problema general de optimización no lineal se considere como un área abierta de investigación [33].

2.2. Algoritmos evolutivos

Algoritmo evolutivo es un término que engloba a un conjunto de técnicas heurísticas de búsqueda que se basan en el principio de la supervivencia del más apto.

El funcionamiento básico de estas técnicas es el mismo: se comienza con un conjunto de elementos, conocido como población de individuos, el ambiente causa una selección de éstos (supervivencia con base en aptitud), la cual a su vez causa el incremento de la aptitud de la población [17].

Hablando en términos de optimización la aptitud se define en términos de la función objetivo. Normalmente, la población inicial se genera de manera aleatoria. Con base en aptitud, se selecciona a los mejores individuos para aplicarles operadores que puedan generar nuevos individuos (a los que se llama hijos). Los principales operadores son: mutación y recombinación. La mutación se aplica a un individuo y genera uno nuevo, y la recombinación (también conocida como cruza) se aplica a dos o más individuos para generar uno o más nuevos. Más tarde se hace una selección de los mejores individuos que sobrevivirán y formarán una nueva población. Este proceso se repite hasta cumplir una condición de terminación, la cual generalmente es un número máximo de iteraciones (o generaciones).

El pseudocódigo general de un algoritmo evolutivo se presenta a continuación:

```
BEGIN
  Inicializar la población de manera aleatoria.
  REPEAT UNTIL (una condición de terminación se cumpla) DO
    1 Evaluar la aptitud de cada individuo;
    2 Seleccionar y recombinar individuos;
    3 Mutar a los individuos obtenidos en el paso anterior;
    4 Seleccionar a los mejores individuos de la población;
  OD
END
```

Los componentes más importantes de un algoritmo evolutivo son [17]:

- **Representación:** Se trata de relacionar al "mundo real" con el espacio donde trabajan los algoritmos evolutivos. En el "mundo real" la representación de soluciones se conoce como fenotipo, y a su codificación para ser utilizadas por el algoritmo se le conoce como genotipo. La representación realiza el mapeo entre estos dos conjuntos.
- **Función de evaluación:** Conocida también como función de aptitud nos define que tan buena es una solución.
- **Población:** Es el conjunto de los individuos que nos proveen una solución al problema.
- **Mecanismo de selección:** Sirve para definir qué individuos deben pasar por los operadores evolutivos.
- **Operadores evolutivos:** Los típicos son la mutación y la recombinación, los cuales permiten a la población generar nuevas soluciones.
- **Mecanismo de supervivencia:** Se utiliza para elegir qué individuos deben mantenerse para continuar el proceso evolutivo.
- **Proceso de inicialización:** Se debe contar con un mecanismo para obtener una población inicial de individuos. Esta población inicial se suele generar aleatoriamente.
- **Condición de terminación:** Es la manera de terminar la ejecución del algoritmo. Generalmente, la condición de terminación es el número máximo de generaciones, aunque hay otros criterios de paro posibles.

Existen tres esquemas básicos de algoritmos evolutivos:

1. **Algoritmos genéticos:** Fueran inicialmente concebidos por John Holland en la década de los 60's [26]. Han sido ampliamente utilizados en problemas de optimización aunque el mismo Holland no los utilizó para eso sino para clasificación y aprendizaje de máquina. Un algoritmo genético clásico es de representación binaria, su selección es proporcional con respecto a la aptitud, tiene bajos porcentajes de mutación y enfatiza el uso de la recombinación.

2. **Estrategias evolutivas:** Fueron desarrolladas a principios de la década de los 60's por Ingo Rechenberg y Hans-Paul Schwefel en una aplicación de ingeniería [35]. Su uso es bastante extendido y ha reportado excelentes resultados en optimización no lineal. Se basa en mutación y usa representación mediante números reales. Suelen contar con mecanismos de auto-ajuste de parámetros (a lo que se denomina "áuto-adaptación"). Existen varias versiones del mismo algoritmo donde lo que cambia es la cantidad de padres e hijos y la competencia entre ellos.

3. **Programación evolutiva:** Fue desarrollada a principios de los 60's por Lawrence Fogel [19] para simular la evolución como un proceso de aprendizaje con el objetivo de resolver problemas de predicción. Utiliza vectores de números reales como representación, su selección es determinista, no utiliza recombinación y su mutación está basada en perturbaciones gaussianas.

Existen otros esquemas como el de programación genética que utiliza como individuos estructuras de árbol [29]. Se utiliza principalmente para aprendizaje de máquina y las operaciones de mutación y recombinación se llevan a cabo entre los árboles mismos.

2.3. Funciones de penalización

Los algoritmos evolutivos, en su forma original, no pueden resolver problemas de optimización con restricciones, por lo que es necesario implementarles un mecanismo de manejo de restricciones.

Las funciones de penalización son las más utilizadas en computación evolutiva para manejo de restricciones. [4], debido a su sencillez conceptual. En optimización, se utilizan dos tipos de funciones de penalización: exterior e interior. En el primer caso comenzamos con soluciones fuera de la zona factible e intentamos llevarlas a ella. En el segundo caso se busca un factor lo suficientemente pequeño que nos permita generar, a partir de una solución factible, soluciones que siempre lo sean. Las funciones de penalización más utilizadas en algoritmos evolutivos son las exteriores, debido a que no requieren soluciones factibles para comenzar la búsqueda, y existen problemas donde encontrar dichas soluciones factibles es muy complicado.

Las funciones de penalización exterior tienen la siguiente forma general:

$$\phi(X) = f(\vec{x}) \pm \left[\sum_{i=1}^{n} r_i \times G_i + \sum_{j=1}^{p} c_j \times L_j \right]$$

donde $\phi(X)$ es la nueva función objetivo (sin restricciones) a optimizarse, G_i y L_j son las restricciones violadas de desigualdad ($g_i(X)$) e igualdad ($h_j(X)$), respectivamente y r_i y c_j son constantes positivas llamadas "factores de penalización".

Las formas más comunes de G_i y L_j son las siguientes:

$$G_i = max[0, g_i(X)]^{\beta}$$
$$L_j = |h_j(X)|^{\gamma}$$

donde tanto β como γ son instanciadas en 1 o 2.

Existen dificultades para definir factores de penalización para un problema cualquiera. Si los factores de penalización son muy grandes, éstos podrían llevarnos rápidamente a la zona factible, pero no se explorará de manera correcta el espacio no factible, lo que resultará particularmente malo en algunos problemas (por ejemplo, en aquellos con zonas factibles disjuntas ya que no podremos movernos entre ellas a menos de que estén muy cerca las unas de las otras).

Por otro lado, si nuestra penalización es muy ligera podríamos tardar demasiado en llegar a la zona factible ya que nos mantendríamos mucho tiempo explorando la zona infactible. Este problema es muy complicado de resolver en una implementación basada en algoritmos evolutivos debido a que las penalizaciones dependen enteramente del problema, es decir, si obtenemos los factores de penalización óptimos para alguna función no necesariamente nos servirán para otro problema.

Hay al menos tres principales formas de definir la relación entre individuos infactibles y la zona factible en un problema de optimización [10]:

1. Un individuo puede ser penalizado por violar un restricción sin tomar en cuenta la "cantidad" de violación.

2. La "cantidad" de infactibilidad puede ser utilizada para determinar la penalización.

3. El esfuerzo por reparar la solución, puede usarse para definir la penalización

Existen diversas posibilidades para definir una función de penalización. A continuación se presentan algunos ejemplos:

1. **Penalización estática:** Los factores de penalización no dependen de la iteración o generación en la que se encuentre el algoritmo; es decir, los factores de penalización se mantienen constantes durante todo el proceso evolutivo.

2. **Penalización dinámica:** En este caso, los factores de penalización dependen de la generación en la que se encuentre el algoritmo. Los factores pueden aumentar o disminuir conforme las iteraciones y por lo tanto no son constantes.

3. **Penalización adaptativa:** En este tipo de penalización se extrae información del proceso de búsqueda para definir los factores de penalización. Por ejemplo, se puede modificar una variable dependiendo del número de generaciones consecutivas en que se han encontrado soluciones factibles.

Finalmente hay que mencionar que las funciones de penalización pueden lidiar tanto con restricciones de igualdad como con las de desigualdad.

Para lidiar con restricciones de igualdad lo más común es transformarlas a restricciones de desigualdad de la siguiente forma:

$$|h_j(X)| - \epsilon \leq 0$$

donde ϵ es la tolerancia permitida (un valor muy pequeño) de violación de la restricción. Este valor se puede variar. De tal manera que en la primera generación se cuenten con soluciones factibles y durante el proceso evolutivo ϵ se va reduciendo hasta llegar a un valor muy pequeño. El valor de ϵ que suele adoptarse en la literatura de algoritmos evolutivos es 1×10^{-4} [34].

Aunque existen muchos otros esquemas de manejo de restricciones, no los discutiremos aquí dado que el énfasis de esta tesis es el uso de un sistema inmune artificial como optimizador. El lector interesado puede consultar otras fuentes para más información al respecto (ver por ejemplo [4]).

Capítulo 3

El sistema inmune artificial

3.1. Fundamentos del sistema inmune

El contenido de esta sección está basado en las referencias [31] y [15].

La inmunología tiene como objetivo el estudio de los fenómenos relacionados con la inmunidad. Esta palabra, de origen latino, significa etimológicamente privilegio, exención, es decir, liberar a alguien de algo, especialmente de una obligación o culpa. En la edad media el término "inmunidad"se refería a una exención de cargos fiscales. Cuando la medicina adoptó este término, lo hizo en principio para caracterizar una exención de enfermedad, un estado de resistencia frente a ciertos agentes infecciosos. Muy pronto se estableció la distinción entre la resistencia natural, disponible inmediatamente después del primer contacto con el agente infeccioso, y la resistencia adquirida, que sólo existe si ha habido una exposición anterior o una vacuna.

Cuando un elemento extraño ha podido penetrar la piel, que es el primer resguardo inmunológico, se producen unos fenómenos vasculares y celulares característicos de la reacción inflamatoria, cuyo último objetivo es captar y eliminar los agentes extraños por un fenómeno llamado fagocitosis, que consiste básicamente en "digerir" al agente extraño.

En el suero o sangre, el germen extraño debe enfrentarse todavía con numerosas sustancias dotadas de una actividad bactericida directa. Se trata esencialmente de los anticuerpos llamados naturales. Se denomina así a los anticuerpos que existen en el suero antes de todo contacto manifiesto con el agente extraño contra el que son activos.

Una vez superada la barrera de la piel por el agente exterior extraño al organismo, y sin importar si fueron o no eficaces los medios de defensa espontáneos, al cabo de unos días el organismo atacado adquiere unos medios de defensa suplementarios. A estos medios suplementarios se les denomina reacciones inmunitarias, y son de utilidad eventual frente a la agresión que ha tenido lugar, pero sobre todo frente a agresiones posteriores idénticas. Entre las muchas células que participan en la captación de las sustancias extrañas, la mayoría fagocitan y degradan indiscriminadamente a los agentes reconocidos como extraños sin respetar los determinantes antigénicos, es decir, las estructuras necesarias para la inducción de la respuesta inmunitaria. Los macrófagos son las células encargadas de capturar al antígeno y de transmitir el mensaje antigénico a otras células.

Las poblaciones celulares que originan las reacciones inmunitarias derivan todas del mismo precursor celular llamado células madre. Se encuentran en el hígado durante la vida fetal y posteriormente en la médula ósea. Parte de estas células conocidas también como linfocitos se someten luego a la influencia del timo, bien al final de la vida fetal o en el período neo natal: entonces se convierten en linfocitos T, es decir timo dependientes. Los linfocitos T, que en general tienen un promedio grande de vida, suelen ser móviles y se encuentran en la sangre, la linfa y en ciertas regiones de otros órganos.

Ante la presencia de un antígeno, se transforman en etapas sucesivas en linfocitos sensibilizados a ese antígeno. La otra fracción de linfocitos no se somete a la acción del timo; en las aves, hacia el final de la vida fetal o en el período neo natal, estos linfocitos sufren la influencia de un órgano llamado Bolsa de Fabricio, situado al extremo del tubo digestivo, por debajo de la cloaca, que es un orificio común de las vías urinarias. De ahí la denominación de linfocitos B o bolsa dependiente, aunque no se conozca el equivalente de la Bolsa de Fabricio en los mamíferos: probablemente se trate del tejido intestinal. Estos linfocitos parecen menos móviles que los T y se sitúan en zonas timo independientes de los órganos. Ante una estimulación antigénica apropiada, se transforman en células que realizan en etapas sucesivas la síntesis y secreción de anticuerpos específicos: la reacción inmunitaria obtenida es de tipo humoral, o sea, caracterizada por la existencia de anticuerpos circulantes. Existen, además, fenómenos de cooperación entre linfocitos T y linfocitos B, siendo los linfocitos T los que estimulan y controlan la producción de anticuerpos por los linfocitos B.

Actualmente, con el término de inmunógeno se tiende a designar toda sustancia que, inyectada a un organismo, es capaz de inducir una reacción inmunitaria. Con el término de antígeno, clásicamente utilizado, más bien se tiende a señalar la pro-

piedad que tiene una sustancia para reaccionar específicamente con los productos de esta reacción inmunitaria, tanto anticuerpos (reacción humoral) como células (reacción celular). La amplia utilización del término antígeno de manera errónea explica el hecho de que todavía se utilice en lugar de inmunógeno.

En la médula ósea los linfocitos representan una fracción considerable. En ella se encuentran las células que originarán los linfocitos T, es decir, células madre que deben abandonar la médula para adquirir, por intervención del timo, su carácter de linfocitos timo dependientes. Igualmente se hallan en la médula las células que originan los linfocitos B, cuyos caracteres específicos se adquirirán en la bolsa de Fabricio en el caso de las aves (B = bolsa dependientes) o en un órgano equivalente mal precisado en los mamíferos (quizá sea sencillamente en la propia médula ósea: B = bone-marrow dependiente).

Existen dos teorías principales en inmunología con respecto a las relaciones existentes entre anticuerpos y antígenos. La primera se conoce como informativa o instructiva. En ella el antígeno por sí mismo es quien determina la estructura del anticuerpo. La segunda se conoce como selectiva y en ella el antígeno escoge entre los patrones de síntesis que ya están contenidos en el genoma, o conjunto genético del individuo.

La selección clonal pertenece a esta última teoría. En ella se dice que existen inicialmente células capaces de sintetizar anticuerpos correspondientes a cualquier antígeno. Esta hipótesis permite también explicar los fenómenos de tolerancia inmunológica, en particular los constituyentes del propio organismo, inducida durante la vida fetal.

Los siguientes antecedentes sobre inmunología pueden ser encontrados en [13]. La inmunología es una ciencia relativamente nueva. Sus orígenes se atribuyen a Edward Jenner quien, en 1796, se percató que una variante de la viruela, pero que aqueja a las vacas, ejercía un efecto inmunitario en el ser humano al ser introducida en su organismo. Este proceso fue llamado vacunación y sigue siendo utilizado en nuestros días.

Llegamos luego hasta Luis Pasteur quien, entre 1880 y 1889, diseñó la primera vacuna exitosa para uso humano y planteó algunas ideas sobre los mecanismos que forman parte del sistema inmune humano. Por ejemplo, él pensaba que las vacunas provocaban que el organismo que las recibía perdiera nutrientes esenciales para la proliferación de los agentes causantes de una enfermedad. Más tarde esta misma idea fue refutada por Emil Von Behring y Shibasaburo Kitasato, el primero de ellos obtuvo

el primer Nobel en medicina en 1901. Ambos esclarecieron el panorama con sus ideas sobre el funcionamiento del sistema inmune.

La primera controversia de importancia en inmunología se dio cuando Elie Metchnikoff, demostró en 1882, primero en invertebrados y posteriormente en mamíferos, que algunas células eran capaces de "comer" microorganismos. Además sugirió que los anticuerpos, que son "células protectoras" del organismo, jugaban un rol muy pequeño en el sistema inmune y, eran las células descubiertas por él, el principal componente. Más tarde Wright y Denys terminaron con la polémica colocando a los anticuerpos en un escenario principal.

Paul Ehrlich, ganador del Nobel junto con Metchnikoff en 1908, estableció los mecanismos y funciones del sistema inmune con base en las relaciones entre antígenos, que son las moléculas que pueden ser reconocidas por el sistema inmune, y los anticuerpos que actúan en presencia de los primeros. Su teoría fue conocida como selectiva dado que planteaba que las células B, que son células blancas de la sangre encargadas de producir anticuerpos, eran seleccionadas y estimuladas para producir anticuerpos con ayuda del genoma, es decir, por la colección de genes.

Entre 1914 y 1955 se pensó que la teoría selectiva podría ser incorrecta y se planteó la teoría de las interacciones idiotipo anti-idiotipo, donde se expresa la idea de que son los antígenos quienes llevan información a las células B para que éstas completen la estructura de los anticuerpos.

Las luchas entre las dos teorías presentadas continúa hasta nuestros días. Niel K. Jerne revivió en la década de 1950 la teoría selectiva definiendo más tarde la teoría de red inmune, que trata de las relaciones que se producen entre los mismos anticuerpos aún en la ausencia de antígenos. Por otra parte Burnet estableció el principio de selección clonal que no es otra cosa que la idea de que las células B generan anticuerpos de manera aleatoria para tener diversidad.

En la figura 3.1 puede verse el funcionamiento básico del sistema inmune humano.

3.2. Aproximación computacional al sistema inmune

El sistema inmune artificial es una aproximación computacional al sistema inmune humano, no es entonces de extrañar que los estudios inmunológicos sirvan de base para su desarrollo.

El sistema inmune artificial es un sistema adaptativo, inspirado por la teoría in-

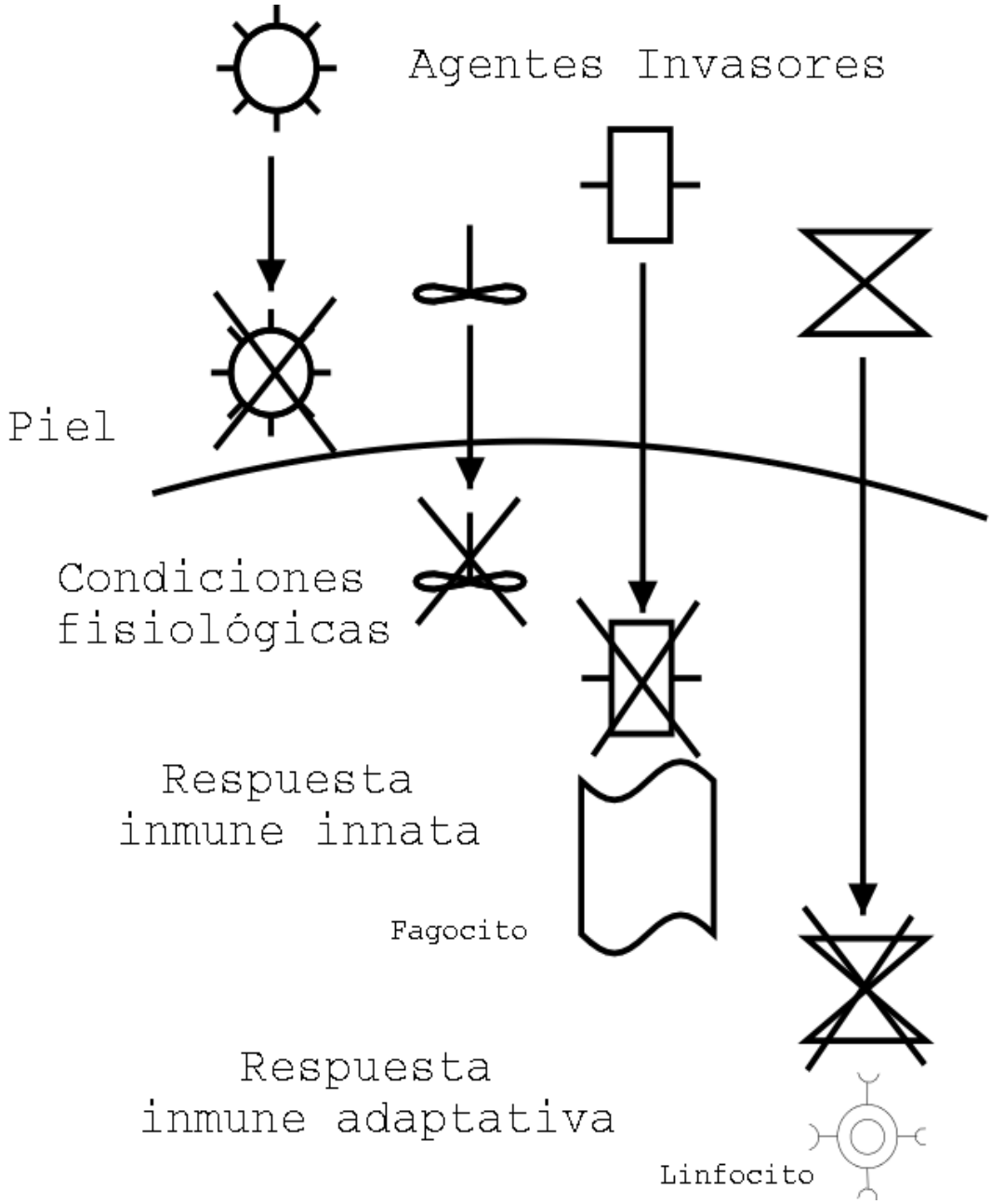

Figura 3.1: Sistema inmune humano. Entra en acción cuando detecta a un agente invasor (antígeno) y tiene cuatro distintos niveles: la piel, las condiciones fisiológicas (por ejemplo, calor corporal), la respuesta inmune innata mediante los fagocitos y la respuesta inmune adaptativa mediante los linfocitos.

munológica y las funciones, principios y modelos observados por la misma, utilizados para resolver problemas [13].

En el caso específico de algunos modelos de sistema inmune artificial se cuenta con operadores que permiten llevar a cabo interacciones entre soluciones ya obtenidas con anterioridad durante el proceso y generar nuevas soluciones que puedan superar la mejor solución obtenida hasta el momento. Es decir, el sistema inmune artificial trabaja estableciendo búsquedas locales. Es importante hacer notar que los sistemas inmunes artificiales pueden usarse como técnicas de optimización pero para problemas sin restricciones por lo que es necesario agregar un mecanismo que pueda manejarlas [13]. En [4] y [32] se presentan varios mecanismos que permiten llevar a cabo el manejo de restricciones en problemas de optimización global.

Es difícil definir quién fue el primero que acuñó el término sistema inmune artificial. En [13] se menciona que fue H. Sieburg y sus colaboradores en un artículo donde se realiza una simulación de la infección del virus de inmunodeficiencia humana [36]. Cabe señalar que dicho artículo fue escrito en un contexto biológico. Por otro lado es necesario mencionar que para describir esta área no sólo se utiliza el término sistema inmune artificial, sino también otros como computación inmunológica, inmunocomputación, inmunología computacional o sistemas inmuno-basados los cuales son utilizados como sinónimos de sistema inmune artificial.

Se dice en [13] que el área se creó a partir de dos artículos: el desarrollado por J. D. Farmer y sus colaboradores [18] y el de G. W. Hoffmann [25], ambos publicados en 1986.

El primer trabajo [18] establece componentes importantes del sistema inmune artificial que se mantienen hasta nuestros días. Se propone un modelo basado en ecuaciones diferenciales el cual es dinámico y cuya forma básica puede ser vista en otros sistemas biológicos. Se realiza una comparación del modelo frente a los sistemas de clasificadores encontrando una gran cantidad de similitudes. Una aportación clave de este trabajo es la representación a través de arreglos de bits tanto de antígenos como de anticuerpos. Finalmente se propone el uso del esquema para problemas de reconocimiento de patrones.

En el artículo presentado por Hoffmann [25] en 1986 el autor explora las similitudes y diferencias entre el sistema inmune y el nervioso de tal manera que formula un modelo neuronal. La gran diferencia de este trabajo con el anterior es que no tuvo muchas repercusiones en el campo de inteligencia artificial aunque motivó estudios posteriores relacionados con el sistema inmune y nervioso.

La investigación en el área continuó con el uso de algoritmos genéticos para estudiar las capacidades de reconocimiento de patrones de los modelos de sistema inmune artificial. Este trabajo fue desarrollado por Forrest y sus colegas [20].

El crecimiento en la utilización de modelos basados en el sistema inmune desarrollaron el uso, por ejemplo, de sistemas computacionales que tratan de aproximar el funcionamiento inmunológico para proteger computadoras y redes de invasiones de virus e intrusos. Lo anterior condujo al desarrollo de software anti-virus basado en modelos del sistema inmune. El primer trabajo al respecto fue desarrollado por Forrest y sus colaboradores en 1994 para IBM [21].

Se ha desarrollado una gran cantidad de aplicaciones de los sistemas inmunes artificiales en los últimos años. El paradigma del sistema inmune artificial continúa tomando fuerza y además ha demostrado capacidad para resolver problemas de optimización (por ejemplo véase [9] y [8]).

3.2.1. Modelos del sistema inmune artificial

El área de sistema inmune artificial ha tenido tan sólo dieciocho años de desarrollo pero en ese tiempo varias aplicaciones han sido producidas [16]. Algunas áreas en las que se ha trabajado son: reconocimiento de patrones, seguridad, y aprendizaje de máquina.

Hasta el momento existen cuatro modelos del sistema inmune los cuales se han aplicado en distintas áreas [13]:

1. **Modelos de médula espinal:** La médula espinal se encarga de la generación de células y moléculas. El modelo computacional más simple genera una cadena de atributos de tamaño L. Otra aproximación está basada en bibliotecas de las cuales se obtienen componentes que en conjunto forman la cadena de atributos que definen a un anticuerpo.

2. **Modelos de timo:** Se basa en el modelo inmunológico que explica la generación de repertorios de células y moléculas capaces de reconocer elementos externos. La glándula timo es la encargada de llevar a cabo este proceso en el sistema inmune humano. Existen dentro de este modelo los algoritmos de selección positiva y negativa. Los primeros básicamente tratan de que los anticuerpos alcancen un umbral de afinidad (qué tan parecido es un anticuerpo

con un antígeno dado). El segundo algoritmo busca que dicha afinidad sea menor al umbral establecido. Los anticuerpos que cumplan con la restricción se mantienen en la población, los otros son sustituidos.

3. **Algoritmos de selección clonal:** Usados para controlar a los componentes del sistema inmune que interactúan con los antígenos. Básicamente lo que se realiza aquí es establecer las afinidades de los anticuerpos con respecto a cierto antígeno para posteriormente realizar copias de aquellos que presenten mayor afinidad (clonación) y finalmente llevar a cabo mutaciones con porcentajes altos para obtener nuevos anticuerpos.

4. **Modelos de red inmune:** Se usan para simular la estructura y dinámica de la red inmune que consiste básicamente de las relaciones entre anticuerpos tanto en presencia como en ausencia de antígenos.

Los dos primeros modelos se basan en componentes principales del sistema inmune. Los dos últimos son básicamente esquemas de su funcionamiento.

Un algoritmo básico del sistema inmune artificial en su versión para problemas de optimización mono-objetivo utilizando el esquema de selección clonal maneja a los anticuerpos y antígenos en una sola población. Dicho algoritmo tiene la siguiente estructura [13]:

```
Entrada: Función objetivo (afinidad), tamaño de población de anticuerpos.
Salida:  Anticuerpo con mayor afinidad.

PASO 1    Generar al azar la población de anticuerpos.
PASO 2    Hasta que una condición de terminación se cumpla realizar:
PASO 2.1       Calcular la afinidad de los anticuerpos.
PASO 2.2       Seleccionar a los más afines.
PASO 2.3       Clonación de los anticuerpos seleccionados.
PASO 2.4       Mutación de los clones.
PASO 2.5       Calcular la afinidad de los clones.
PASO 2.6       Seleccionar a los clones con mayor afinidad.
PASO 2.7       Insertar en la población de anticuerpos a los mejores clones.
PASO 2.8       Generar aleatoriamente un porcentaje de nuevos anticuerpos.
          fin;
PASO 3    Presentar al anticuerpo con mayor afinidad.
```

Los anticuerpos pueden verse como arreglos de bits para representación binaria o como arreglos de números flotantes si se desea trabajar con representación real. En dichos arreglos es donde se codifican las variables de decisión. Además, cada anticuerpo cuenta con un valor de afinidad. Para el caso de optimización global el antígeno es representado por la función objetivo. La afinidad se calcula mediante la evaluación del vector de variables de decisión de cada anticuerpo en la función objetivo o antígeno.

Cuando hablamos de clonación simplemente nos referimos a realizar copias de un anticuerpo, es decir, copiar la estructura completa que los define.

La mutación se realiza en representación binaria cambiando un bit elegido por su contrario, es decir, si es 1 se cambia por 0 y viceversa. En el caso de la representación real se utilizan fórmulas bien conocidas en computación evolutiva para llevar a cabo la operación de mutación. Es importante señalar que la mutación se aplica con una probabilidad inversamente proporcional a la afinidad, es decir, entre mayor afinidad menor porcentaje de mutación y viceversa.

En cada ciclo se seleccionan de entre los anticuerpos y los clones mutados a los que formarán parte de la población de anticuerpos para la siguiente iteración basados en su afinidad.

Al generar de manera aleatoria nuevos anticuerpos lo que se desea es mantener diversidad en la población. Normalmente se generan entre 5 % y 20 % de la población total.

El algoritmo es presentado como un motor de búsqueda. Como en el caso de la mayor parte de las implementaciones de algoritmos evolutivos la población inicial se genera de manera aleatoria. Al seleccionar a los individuos más afines lo que se busca es imitar el proceso inmunológico de la detección. En lugar de seleccionar a los más afines puede tomarse toda la población de anticuerpos o posibles soluciones si dicha población no es muy grande. Para definir el tamaño de la población a utilizarse hay que tomar en cuenta que hay que realizar clones de cada individuo seleccionado.

El paso más importante durante el proceso es la mutación. Es el principal componente del motor de búsqueda y en conjunto con la clonación nos permite explorar la zona más prometedora encontrada hasta el momento. Lo anterior se consigue al realizar pequeñas perturbaciones en la estructura de los anticuerpos más prometedores permitiéndonos realizar búsquedas locales en esos casos, que a su vez nos pueden proporcionar nuevos anticuerpos con una afinidad mayor. Pero además existen los anticuerpos que tienen menor afinidad. En éstos, la mutación trabaja de manera distinta. Al elevar los porcentajes de mutación la estructura de los anticuerpos cambia de manera más notoria. Ésto nos permite explorar el espacio de búsqueda tratando de

encontrar zonas prometedoras distintas a las que el proceso evolutivo ha encontrado.

Es interesante este comportamiento debido a que estamos realizando las dos operaciones básicas de un motor de búsqueda (la explotación, en los anticuerpos más afines y la exploración, en anticuerpos menos afines) con el mismo operador: la mutación. En algoritmos genéticos estas dos operaciones se llevan a cabo con la cruza y la mutación, respectivamente.

Tanto la clonación como la mutación son imprescindibles en este esquema. Si quitáramos del algoritmo la primera, mantendríamos nuestro motor de búsqueda pero su calidad disminuiría considerablemente, comportándose básicamente como un algoritmo genético sin cruza o recombinación. Si lo que retiramos del esquema es la mutación, perderíamos nuestro motor de búsqueda ya que se mantendría la misma población inicial durante todo el proceso evolutivo.

3.3. El sistema inmune artificial como optimizador

3.3.1. Introducción

Aunque se ha desarrollado un número importante de aplicaciones de los sistemas inmunes artificiales [13, 27, 7], existen muy pocos trabajos en los cuales se use este tipo de heurísticas como componente principal de un motor de búsqueda destinado a resolver problemas de optimización con y sin restricciones. Los trabajos más representativos del área se describen brevemente a continuación, indicándose en cada caso su contribución y su posible relación con la propuesta presentada en esta tesis.

3.3.2. Propuestas para optimización global sin restricciones

Propuesta de Bersini y Varela

Bersini y Varela en 1990 fueron los primeros en trabajar con un sistema inmune artificial para resolver problemas de optimización [3]. Los autores toman en cuenta dos aspectos: la dinámica y la meta dinámica del sistema. La primera consiste simplemente en controlar la cantidad de anticuerpos mediante ecuaciones diferenciales. La segunda trata de controlar qué anticuerpos entran al proceso y cuáles son eliminados. Basados en las dos características propuestas desarrollaron su propio modelo de red inmune. El algoritmo propuesto tiene ciertos aspectos interesantes, como la combinación de una medida de afinidad y una función de aptitud. La función de aptitud es básicamente la función objetivo que se encarga de medir la calidad de un individuo o solución, y la medida de afinidad sirve para medir qué tan similar es un individuo respecto a

otro. Ambos mecanismos son base para un esquema que pretenda utilizar un sistema inmune artificial. Los autores aplicaron su algoritmo en problemas de optimización sin restricciones, llevando a cabo además comparaciones entre estrategias evolutivas y su modelo usando representación real. También realizaron comparaciones entre un algoritmo genético y su mecanismo usando representación binaria. Al final proponen al sistema inmune artificial como un buscador local paralelo de rápida convergencia.

Propuesta de De Castro y Von Zuben

El segundo esquema utiliza el principio de selección clonal. En [12] se presenta un algoritmo el cual trabaja con problemas de optimización multimodal. El algoritmo primero selecciona una cantidad de anticuerpos con mayor valor de afinidad y a éstos los clona. Es decir, realiza copias de ellos para posteriormente mutarlos utilizando porcentajes altos. Finalmente calcula aptitudes de los nuevos individuos generados con las operaciones de clonación y mutación anteriores y selecciona a los anticuerpos más aptos que pasarán a la siguiente iteración, es decir, que serán mantenidos en memoria.

El algoritmo es utilizado para resolver problemas de optimización multimodal y utiliza compartición de aptitud, la cual se basa en la evaluación de la función objetivo de cada individuo y su distancia con respecto a otras soluciones para definir su afinidad.

Este algoritmo no trabaja con problemas de optimización con restricciones. Existen otros usos para este esquema fuera de optimización como el que se presenta por los mismos autores en [11]. Incluso en [12] proponen un algoritmo basado en el mismo principio para problemas de reconocimiento de patrones.

En el marco teórico se presentó como trabaja este algoritmo en su versión de optimización.

Propuesta de Kelsey y Timmis

Existe otro esquema, desarrollado por Kelsey y Timmis [28], para resolver problemas de optimización multimodal que utiliza el principio de selección clonal.

Es básicamente el mismo esquema de selección clonal utilizado por De Castro y Von Zuben con ligeras variaciones El punto más importante de su propuesta es la utilización de un operador de mutación al que nombran “hipermutación somática contigua” donde la mutación se aplica a un subconjunto contiguo de bits. Tanto el

inicio como el tamaño del subconjunto se determina de manera aleatoria. Su población es significativamente más pequeña que la utilizada por De Castro y Von Zuben. Los autores demuestran de manera experimental que su propuesta converge al óptimo en un menor número de evaluaciones que un algoritmo híbrido y el algoritmo propuesto por De Castro y Von Zuben.

3.3.3. Propuestas para optimización global con restricciones

Propuesta de Hajela y Lee

El primer esquema para optimización global con restricciones, basado en un sistema inmune artificial, fue desarrollado por Hajela y Lee [24, 7] quienes proponen una simulación del sistema inmune con algoritmos genéticos, utilizan una representación binaria tanto para los antígenos como a los anticuerpos y distancia de Hamming para calcular afinidad. La principal aportación del algoritmo es proponer a los antígenos como soluciones factibles y a los anticuerpos como no factibles. Por ende, lo que se desea es llevar la búsqueda hacia zonas factibles utilizando el esquema de selección negativa.

El algoritmo funciona como sigue: primero se toma una población de manera aleatoria, se separan las soluciones que sean factibles de las que no lo sean, y se ordenan con base en su aptitud. Se calcula en un subconjunto de soluciones factibles un valor promedio de aptitudes y las soluciones más cercanas al promedio calculado son tomadas como antígenos. Se simula el proceso del sistema inmune entre los antígenos seleccionados y anticuerpos que serán los elementos no factibles que se tienen en la población. Dicha simulación es básicamente el cálculo de afinidad entre antígenos y anticuerpos. Finalmente se utiliza un algoritmo genético tradicional cuya población está definida por los anticuerpos y por el resultado del proceso del sistema inmune realizado con anterioridad tomando ya sea las mejores soluciones o un conjunto elegido de manera aleatoria. Se utiliza la función objetivo para calcular la aptitud, es decir, ya no se toman en cuenta las restricciones violadas. Los pasos anteriores se repiten hasta que se cumpla un criterio de paro.

Esta propuesta fue probada en algunos problemas de optimización estructural, donde se intenta optimizar el diseño de piezas para construcción.

Propuesta de Coello y Cruz

El segundo esquema fue presentado por Coello y Cruz [5] y consiste en el mismo modelo anterior pero con mejoras significativas. Por ejemplo, en este caso se puede lidiar con problemas en los cuales no se cuenta desde un inicio con soluciones que no violen restricciones. Esta ventaja es muy importante porque así no se limita demasiado la aplicación del algoritmo. Además, se incluye en el trabajo una implementación paralela que maneja un esquema de islas, es decir, poblaciones separadas que evolucionan simultáneamente.

El algoritmo se presenta a continuación: Se determina el número de procesadores y se crea el mismo número de subpoblaciones (llamadas demes). Se determina el tamaño de cada deme dividendo la población total entre el número de demes.

Por cada deme se genera una población aleatoria y se divide la población en dos grupos, los factibles (antígenos) e infactibles (anticuerpos). Si ninguno de los individuos es factible se utiliza la cantidad de restricciones violadas por individuo. Aquellos que violen menor cantidad de restricciones son elegidos como antígenos y posteriormente se lleva a cabo la simulación del sistema inmune. A cada cierto número de iteraciones se observa si se ha alcanzado la zona factible. De ser así, se divide la población en factibles e infactibles como se explicó al principio. Ahora la aptitud de todos los anticuerpos se hace cero. Después, la aptitud del anticuerpo es calculada mediante la afinidad de un determinado antígeno siguiendo los siguientes pasos: un antígeno se selecciona aleatoriamente y se obtiene una muestra de anticuerpos. Se calcula la afinidad, por ejemplo mediante distancia de Hamming o distancia Euclidiana. Se obtiene el que tenga la máxima afinidad y se le aumenta ese valor a su aptitud. Los demás anticuerpos se mantienen iguales en cuanto a su aptitud. Se realiza tres veces este proceso a partir de la selección de antígenos al azar. Basado en lo anterior se evoluciona a la población de anticuerpos con un algoritmo genético tradicional.

En cuanto a la migración de anticuerpos entre demes tomamos en cada deme al mejor individuo junto con algunos anticuerpos elegidos al azar y los sustituimos al azar en los demes receptores. Así se complementa el esquema.

Propuesta de Balicki

Finalmente, tenemos el trabajo de Balicki [1] donde se propone un esquema para asignar tareas en un sistema distribuido. Su algoritmo está basado en búsqueda tabú [23], que es una técnica de optimización utilizada normalmente para problemas combinatorios, en combinación con una simulación del sistema inmune utilizando el

algoritmo de selección negativa. Este esquema es aplicado a un problema multiobjetivo pero utiliza al sistema inmune para manejar restricciones.

Capítulo 4

Algoritmo propuesto

Existen distintos esquemas basados en el sistema inmune. En esta tesis se propone un algoritmo que trabaja con un esquema basado en el principio de selección clonal que se vio a grandes rasgos en cuanto a sus bases inmunológicas en el capítulo 2. A continuación se presenta un descripción más completa de la selección clonal y su funcionamiento, una justificación sobre su uso en otros esquemas existentes, así como el algoritmo propuesto.

4.1. Principio de selección clonal

Se denomina selección clonal a una teoría utilizada para explicar cómo los componentes (como componente definimos tanto a los antígenos como a los anticuerpos) del sistema inmune se adaptan para combatir a un agente extraño que invade al organismo. Por lo tanto, pertenece a la respuesta inmune adaptativa del organismo.

Recordemos que tenemos dos componentes básicos en el sistema inmune: los antígenos y los anticuerpos, y que es a partir de la introducción de los primeros que los segundos comienzan sus funciones.

En términos generales el principio de selección clonal funciona de la siguiente manera: Cuando un anticuerpo reconoce al antígeno, es seleccionado para que prolifere y produzca anticuerpos con su misma estructura química en grandes volúmenes. La reproducción es asexual y se realiza a través de la mitosis. Por lo anterior **no existe una cruza entre anticuerpos**. A los "hijos idénticos" de cada anticuerpo seleccionado se les conoce como clones. Es en ellos donde se realiza la adaptación que consiste en someterlos a un proceso de mutación con altos porcentajes. Del resultado

de este proceso se obtienen nuevos anticuerpos y mediante un proceso de selección se mantienen aquellos que tengan cierto grado de afinidad con respecto a los antígenos reconocidos al principio del proceso. Aquellos anticuerpos que no fueron mantenidos se desechan y regresan al torrente sanguíneo para poder reutilizar las proteínas que los formaban en la creación de nuevos individuos. Aquellos que sí fueron seleccionados se guardan como células de memoria donde se mantienen por algún tiempo para ser utilizadas, de ser necesario, en un futuro. En la figura 4.1 se ilustra lo descrito.

Los procesos y la interacción entre la mutación y la selección son análogos a la selección natural de las especies.

Hay dos puntos importantes desde el punto de vista computacional en este esquema:

1. La proliferación de un anticuerpo es directamente proporcional a la afinidad de éste con respecto a un antígeno detectado. Entre mayor sea la afinidad entre componentes mayor será la cantidad de descendientes y viceversa.

2. La mutación de cada uno de los clones es inversamente proporcional a la afinidad entre el anticuerpo que los produjo y el antígeno detectado. Entre mayor sea la afinidad entre componentes menor será el porcentaje de mutación y viceversa.

De Castro y Timmis [13] desarrollaron un primer algoritmo basado en el principio de selección clonal para reconocimiento de patrones. En el mismo trabajo los autores proponen modificaciones en el algoritmo de manera que pueda lidiar con problemas de optimización multimodal, es decir, problemas de optimización de funciones con varios óptimos locales. Primero se debe tomar como antígeno a la función objetivo y los anticuerpos, serán un conjunto de soluciones potenciales al problema. Los individuos en este caso ya no proliferan con respecto a su afinidad sino que tienen todos la misma probabilidad de hacerlo. Además, para mantener diversidad se introduce el esquema de compartición de aptitud de amplio uso en algoritmo genéticos. Se desea con ello que la búsqueda no se quede atrapada en un óptimo local.

De Castro y Timmis no abordaron el problema de optimización con restricciones.

4.2. Codificación y decodificación

Todos los problemas de optimización que se pretenden resolver con el algoritmo propuesto están definidos en el espacio de los números reales, por lo que es necesa-

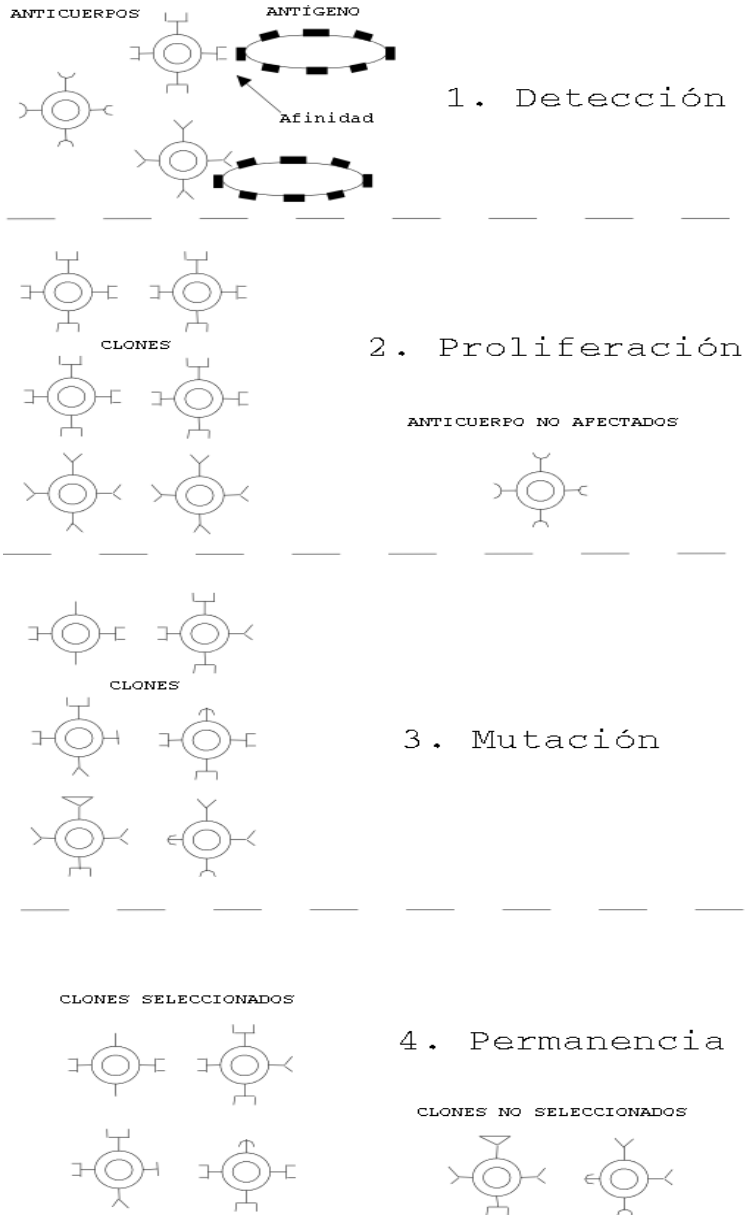

Figura 4.1: Principio de selección clonal

rio llevar a cabo algunas transformaciones para poder trabajar en el espacio de los números binarios.

Cada anticuerpo contiene un arreglo de bits que representa el vector de las variables de decisión, es decir todas las variables se colocan en un mismo arreglo.

El algoritmo propuesto genera aleatoriamente valores enteros entre 0 y $2^L - 1$, donde L es la longitud del arreglo de bits por variable. Estos valores son transformados a números reales dentro del intervalo del problema a través de la ecuación:

$$x = li + \frac{(i \times (ls - li))}{2^L - 1} \tag{4.1}$$

Donde li y ls son los límites inferior y superior de las variables de decisión del problema, respectivamente; i es un entero dado.

De esta manera, por ejemplo, si al generar nuestro número aleatorio entero recibimos un cero, al transformarlo a un número real usando la ecuación 4.1 obtendríamos el límite inferior. En el caso, de que el entero sea $2^L - 1$ nuestro número real sería el límite superior. Debe entonces resultar claro que con la ecuación 4.1 podemos obtener cualquier valor real presente en el intervalo de las variables del problema.

Es el número entero el que se codifica en un arreglo de bits para realizar las transformaciones y operaciones del algoritmo con los números binarios. Una vez llevados a cabo los procesos del algoritmo en el espacio binario, este valor se decodifica para obtener números enteros que serán transformados en números reales para poder ser evaluados y reportados. En la figura 4.2 se muestra este proceso.

Para decidir el valor adecuado de L se debe tomar en cuenta el intervalo de las variables de decisión y el número de decimales con los que se quiere representar dicha variable. De esta manera se podrá saber cual es el entero máximo que necesitamos generar de manera aleatoria. Debemos elegir L de tal manera que podamos representar dicho entero. Es claro que en varias ocasiones $2^L - 1$ nos producirá un número mayor al necesario. Es en esos casos cuando se presenta la redundancia de datos, es decir, dos enteros nos pueden mapear hacia el mismo real.

La principal razón para hacerlo así es por la facilidad de codificar en binario valores enteros en lugar de reales. El principal problema al utilizar éste esquema es la redundancia de datos.

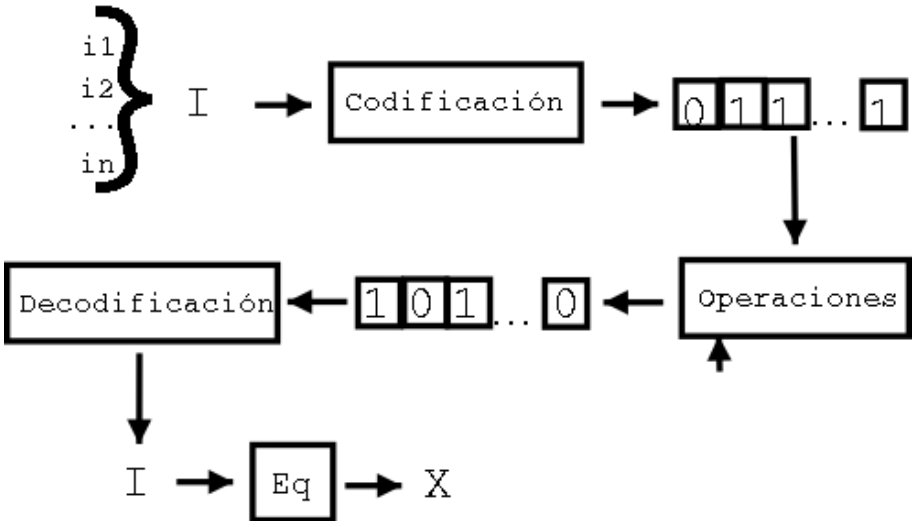

Figura 4.2: Proceso de codificación y decodificación. I es el vector de números enteros, n es el número de variables del problema, X es un vector de números reales obtenido a través de la ecuación 4.1.

4.3. Afinidad

Como se vio con anterioridad, la afinidad es un proceso básico dentro del esquema de selección clonal. Afinidad es una expresión que normalmente es adoptada para denotar qué tanto se asemeja una entidad a otra. El término puede ser visto de manera más precisa: qué tanto se parece un componente del sistema inmune con el entorno donde existe. Para nuestro caso, donde estamos resolviendo problemas de optimización, un anticuerpo corresponde a un punto en el espacio de búsqueda y su afinidad se especifica a través del valor de una función, que suele ser la función a optimizar. El concepto es equivalente al de aptitud utilizado en algoritmos evolutivos.

Se asume que cada anticuerpo interactúa con todos los antígenos que se encuentran en una pequeña región alrededor de él [13], caracterizada por un parámetro ϵ. El volumen V_ϵ resultante de la definición de ϵ es llamado región de reconocimiento. Un anticuerpo puede reconocer todos los antígenos que se encuentren dentro del volumen definido. La figura 4.3 ilustra esta idea. En optimización el volumen es representado por la zona de búsqueda definida por la función objetivo.

Existen varias maneras de medir la afinidad en los algoritmos basados en el sistema

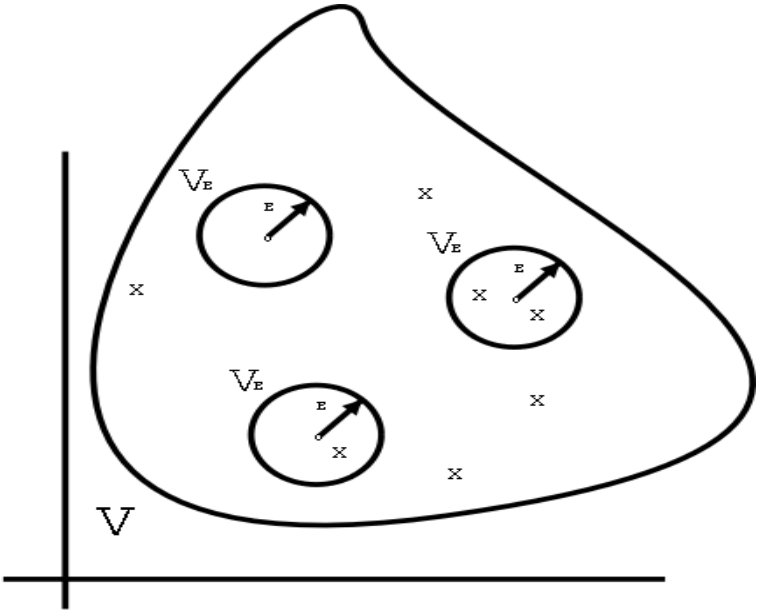

Figura 4.3: Volumen (V) en donde interactúan los anticuerpos (o) y antígenos (x) en el volumen de reconocimiento del anticuerpo V_E.

inmune artificial y depende directamente del espacio en que se encuentren las variables de decisión y del problema que se quiera resolver.

En el caso de reconocimiento de patrones: Para variables que están definidas en números enteros y reales se utiliza una distancia euclidiana y en el caso de representación binaria se trabaja con distancias de Hamming. En el caso de optimización se define, como se vio con anterioridad, a través de una función, conocida como función de afinidad, aunque también existen técnicas como [24], que utilizan distancias, como en el caso de reconocimiento de patrones, dependiendo de la representación. Normalmente se utiliza la función objetivo con ligeras modificaciones para permitir que todos los problemas sean de minimización.

En nuestra tesis no trabajamos con problemas de reconocimiento de patrones. Sin embargo, es importante definir su funcionamiento por dos razones principales: primera, para establecer las diferencias con respecto al esquema propuesto que lidia con problemas de optimización y segunda, para explicar por qué los esquemas basados en selección positiva y negativa no pueden ser utilizados como optimizadores en espacios restringidos a menos que sean híbridos.

Hemos mencionado que las distancias entre componentes nos permiten trabajar en problemas de reconocimiento de patrones. Es fácil saber de qué forma: Primero tomemos como entrada un patrón que requiere ser reconocido, ese será nuestro antígeno. Ahora nuestros anticuerpos serán una población finita de posibles patrones en el espacio de búsqueda. Con base en el parecido de los últimos con respecto al antígeno determinaremos su afinidad a través de una función que nos regresará la distancia entre componentes. En la figura 4.4 se muestra un ejemplo de este proceso.

Asumamos ahora el caso general en donde un anticuerpo es representado por un conjunto de coordenadas $\mathbf{Ab} =< Ab_1, Ab_2, ..., Ab_L >$ y a su vez un antígeno está dado por:$\mathbf{Ag} =< Ag_1, Ag_2, ..., Ag_L >$ donde las negritas corresponden a una cadena. Sin pérdida de generalidad se asume que los antígenos y anticuerpos tienen la misma longitud L. Tomando el punto de vista de reconocimiento de patrones, la interacción entre componentes, es evaluado mediante una medida de distancia, también denominada medida de afinidad, entre las cadenas que los definen.

Dada una cadena de atributos que define a un antígeno y un conjunto de anticuerpos, para cada cadena que define a un anticuerpo, se puede asociar un valor de afinidad con respecto al antígeno dado.

Recordando que se utiliza una medida de distancia para evaluar la afinidad entre

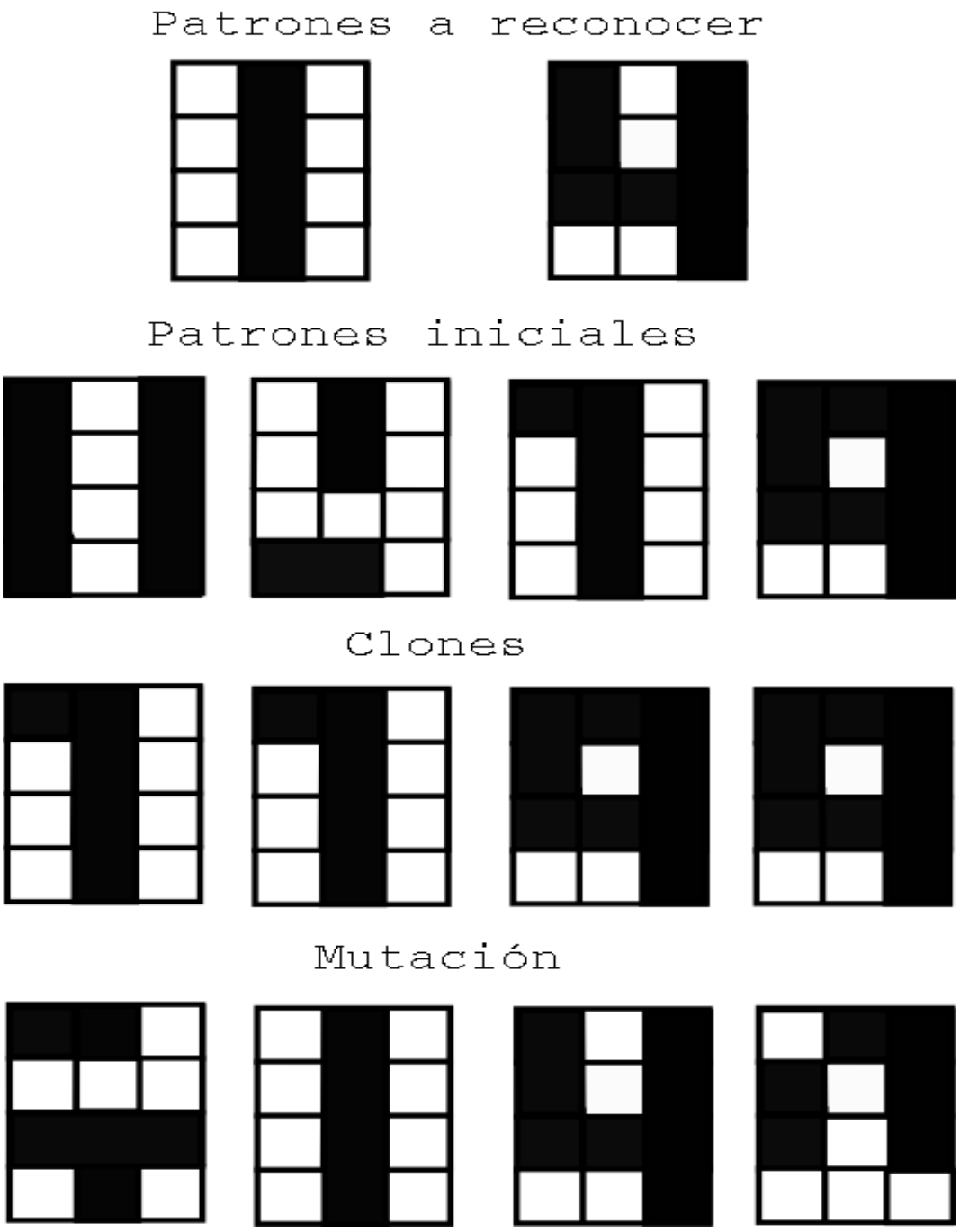

Figura 4.4: Ejemplo de selección clonal en reconocimiento de patrones.

componentes del sistema inmune artificial, en el espacio de los números reales puede ser calculado de muchas maneras, por ejemplo, las distancias de Euclides y de *Manhattan*.

Para el caso de la distancia euclidiana, la afinidad D entre un antígeno y un anticuerpo está dado por la ecuación 4.2. En la ecuación 4.3 se presenta el caso de la distancia de *Manhattan*.

$$D = \sqrt{\sum_{i=1}^{L} (Ab_i - Ag_i)^2} \tag{4.2}$$

$$D = \sum_{i=1}^{L} |Ab_i - Ag_i| \tag{4.3}$$

Si trabajamos en el espacio de los números binarios tanto antígenos como anticuerpos se representan a través de una secuencia de bits. En la ecuación 4.4 se muestra la forma de calcular la distancia de *Hamming* usada para evaluar la distancia entre cadenas de bits.

$$D = \sum_{i=1}^{L} \delta_i, \text{ donde } \delta_i = \begin{cases} 1 & \text{si } Ab_i \neq Ag_i \\ 0 & \text{en otro caso} \end{cases} \tag{4.4}$$

Si utilizamos una representación en números enteros se puede utilizar la distancia de *Hamming* al igual que con los números binarios como un caso particular.

Ahora que hemos presentado de forma general que las maneras de calcular las distancias, y por lo tanto las afinidades, entre antígenos y anticuerpos dependen de la representación, tomemos sólo el caso binario, el más utilizado en las implementaciones de sistemas inmune artificiales y el único en esta tesis.

Algunas de las principales razones de la amplia utilización de representación binaria son: La fácil manipulación y, en el caso de reconocimiento de patrones, la fácil interpretación gráfica de las cadenas binarias.

Para la implementación computacional de la distancia de Hamming puede simplemente ser utilizado el operador o-exclusivo (XOR) entre las cadenas que se desea medir. Esta operación es mostrada en la figura 4.5(a). La afinidad esperada entre dos cadenas es la mitad de su longitud siempre y cuando ésta sea la misma en ambas. También es posible, junto con la distancia de Hamming, medir afinidades mediante

el máximo número de bits consecutivos iguales entre cadenas el cual se muestra en la figura 4.5(b). Otra medida puede ser planteada basados en la siguiente ecuación:

$$D = D_H + \sum_i 2^{I_i} \tag{4.5}$$

donde D_H es la distancia total de Hamming calculada con la ecuación 4.4, y l_i es el tamaño de cada región complementaria i con 2 o más bits iguales consecutivos como se ilustra en la figura 4.5 (c).

Las anteriores son las más utilizadas, debido a la sencillez al ser implementadas y a su desempeño, aunque no son las únicas.

Por otro lado, si tomamos en cuenta que los esquemas de selección positiva y negativa utilizan como parte importante de su selección el parecido (afinidad) que pueda existir entre posibles soluciones con la mejor obtenida hasta el momento podemos darnos cuenta claramente que para problemas en espacios restringidos necesitamos una solución que sea factible para poder hacer funcionar dicho esquema. La dificultad estriba en que encontrar soluciones factibles en algunos problemas es ya de por si muy complicado. Por lo anterior, los esquemas hasta ahora propuestos que utiliza la selección negativa o positiva son híbridos [5, 24].

Sólo falta recalcar que en nuestro algoritmo la afinidad se calcula basados en la función objetivo, evaluando los anticuerpos (vectores de variables de decisión) en ella. Como se puede observar, entonces, es completamente distinto calcular las afinidades en problemas de reconocimiento de patrones a calcular en uno de optimización.

4.4. Selección

La selección de los anticuerpos para ser clonados en la selección clonal es proporcional a la afinidad con respecto a los antígenos. Es entonces muy importante establecer qué mecanismo de selección se debe implementar.

Podemos utilizar cualquier mecanismo de selección existente en algoritmos evolutivos como el de la ruleta, las jerarquías lineales, el torneo y la selección elitista entre otros [13]. Los dos primeros realizan la selección de manera probabilística. La selección por torneo puede ser determinista o probabilística. Finalmente la elitista realiza la selección de manera determinista. Se presenta a continuación una breve descripción de dichos mecanismos.

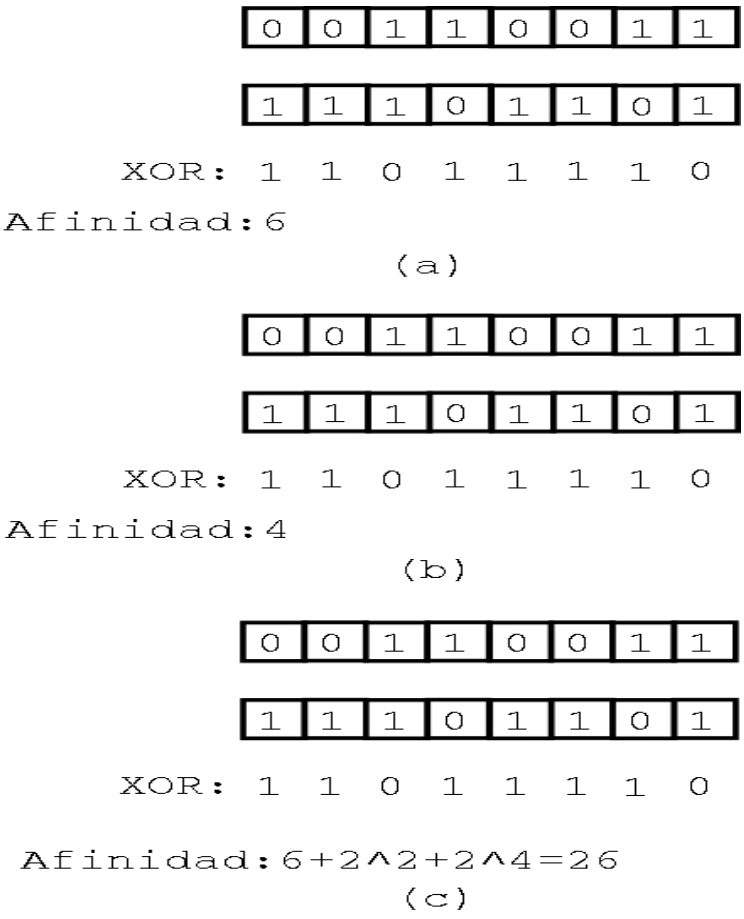

Figura 4.5: Diferentes medidas de afinidad basados en distancias de *Hamming*. (a) Distancia de *Hamming*: número total de bits complementarios. (b) Regla del número máximo de bits continuos. (c) Regla de múltiples bits continuos.

- **Selección por ruleta:** Es de las más utilizadas en algoritmos evolutivos. La probabilidad de elegir a un componente es directamente proporcional a su afinidad. Lo que se realiza básicamente es tomar un número aleatorio entre 0 y 1 el cual nos dice que componente será seleccionado basándonos en sus valores esperados, los cuales se calculan con la cantidad de componentes y sus afinidades.
- **Selección por jerarquías:** Es ampliamente utilizada para evitar convergencia prematura. Los individuos se clasifican con base en su afinidad pero se realizan algunas operaciones específicas para que sean seleccionados con base en una jerarquía (u orden especifico). De esta manera las probabilidades de ser seleccionados ya no dependen solamente de su afinidad. Es una técnica que se acopla, por ejemplo, a la ruleta.
- **Selección por torneo:** Es una técnica muy sencilla que introduce una presión de selección por lo general elevada. Funciona de la siguiente manera. Primero se revuelven todos los anticuerpos y se seleccionan comúnmente dos para que compitan entre ellos y gana el que mejor afinidad posea en el caso determinista; en el probabilístico el mejor individuo gana con cierta probabilidad, es decir, el peor gana ocasionalmente. Esta operación se repite hasta obtener la cantidad de anticuerpos ganadores deseados.
- **Selección elitista:** En este mecanismo de selección se ordenan los elementos con base a su afinidad y se seleccionan de manera totalmente determinista.

El algoritmo propuesto utiliza la selección elitista dado que en la selección clonal es muy importante tener una presión de selección alta por una razón: debido a que la mutación se maneja con altos porcentajes y como el algoritmo realiza búsquedas locales es deseable que en la mayoría de los casos se explore en la zona más prometedora que es donde se encuentran los mejores componentes que son lo que poseen la mayor afinidad.

4.5. Mutación

La mutación es la parte más importante del esquema propuesto y en general, de los sistemas inmunes artificiales basados en selección clonal. Tiene dos importantes funciones. Primero, es la responsable de promover y mantener diversidad. Segundo, asociada a un mecanismo de selección, la mutación incrementa la afinidad de la población [13].

Existen muchas maneras de llevar a cabo la mutación dependiendo de la representación utilizada. En nuestro caso, nos concentraremos específicamente en la representación binaria.

Para el caso de la representación binaria, una posición de la cadena de bits puede ser seleccionada de manera aleatoria, y su elemento cambiado por otro dentro del mismo alfabeto. En nuestro caso, el alfabeto sólo consta de dos elementos {0,1}. De esta manera si encontramos un 1 lo cambiamos por un 0 y viceversa. Si seleccionamos sólo una posición y cambiamos su contenido se le llama mutación de un solo punto. Si por el contrario seleccionamos más de una posición a eso se le conoce como mutación multi-puntos o de varios puntos [17]. En la figura 4.6 se ilustra este proceso. Por otro lado, existe otro tipo de mutación, utilizado en esta tesis, en el cual en lugar de seleccionar una posición lo que se hace es buscar un 0 con cierto porcentaje (pc) en cada una de las posiciones de la cadena de bits, de obtenerlo dejamos intacto el bit de esa posición y pasamos a la siguiente posición; si no, cambiamos el bit y pasamos a la siguiente posición. Este tipo de mutación se conoce como mutación uniforme. Hemos adoptado una pequeña variación de esta mutación, pues en el caso de los clones con mayor afinidad cambiamos sólo un bit, es decir, aplicamos la mutación uniforme, y cuando se cambie un bit dejamos de recorrer la cadena. En caso de que la cadena no haya mutado ningún bit volvemos a empezar el recorrido.

En nuestro algoritmo la mutación es inversamente proporcional a la afinidad de cada individuo. Es decir, se espera que los clones del individuo más afín tengan mutación de un solo punto, así como que se aumenten paulatinamente el número de puntos mutados para los siguientes clones de los anticuerpos menos afines. Es importante mencionar además que altos porcentajes de mutación destruyen la información contenida en los componentes por lo que se recomienda no utilizar porcentajes arriba de 35 %, es decir de aproximadamente un tercio del tamaño total de la cadena binaria. Este valor es tomado después de realizar un estudio empírico [13].

4.6. Manejo de restricciones

Los algoritmos hasta ahora propuestos (basados en el sistema inmune) por sí mismos no son capaces de resolver problema de optimización no lineal con restricciones. Es necesario entonces agregar al algoritmo un mecanismo que pueda lidiar con restricciones de todo tipo (lineales, no lineales, igualdad, desigualdad).

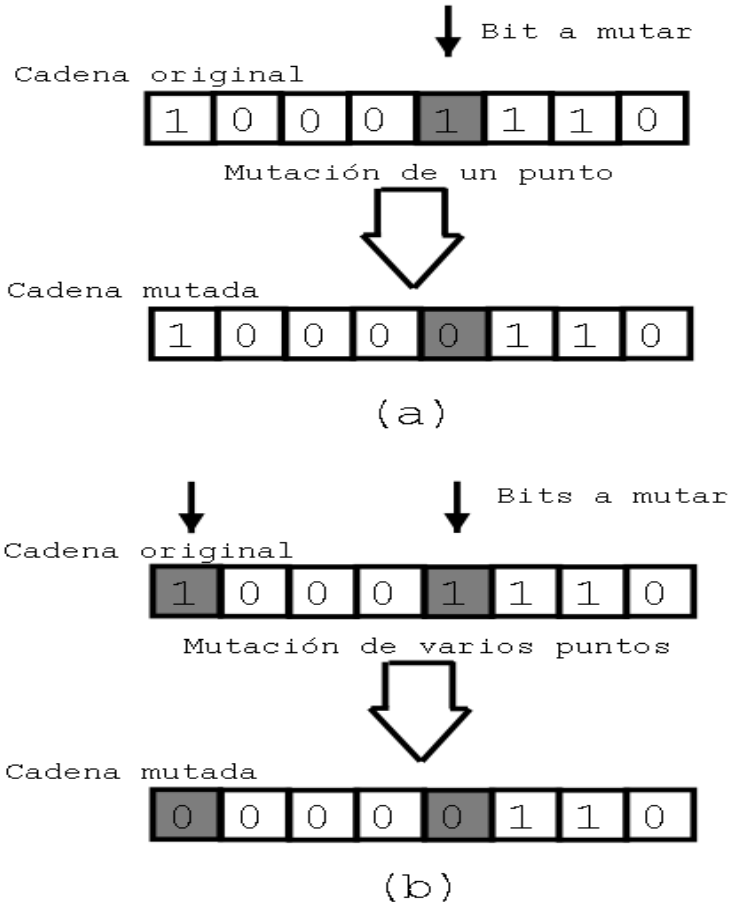

Figura 4.6: Dos tipos de mutación. (a) Mutación de un solo punto. (b) Mutación de varios puntos.

Existen muchos mecanismos para manejo de restricciones en la literatura [4]. Uno de los más sencillos, y que es el que adoptamos, es aquel en el que se siguen reglas para determinar qué individuo de la población es mejor que otro tomando en cuenta no sólo su aptitud (en nuestro caso afinidad) sino también la cantidad de restricciones violadas. A continuación se presenta el algoritmo que sigue estas reglas, utilizadas también en [14, 38]:

```
PASO 1     Para elegir a todos los anticuerpos haz
PASO 1.1       si flip(Porc)
                   Selecciona al mejor individuo (el factible sobre
                   el infactible; entre factibles, selecciona al que
                   tenga mayor afinidad y entre infactibles al que
                   cuente con menor número de restricciones violadas
                   y mayor afinidad).
PASO 1.2       de lo contrario
                   Selecciona al mejor individuo no factible entre
                   anticuerpos y clones.
```

La función flip se refiere a simular el lanzamiento de una moneda con un porcentaje Porc. En otras palabras, se espera que en un porcentaje de veces (dado) la función nos regrese un 1 y 0 en otro caso. Se puede observar además que se tiene un if-else. En el caso del if se siguen las reglas al seleccionar al individuo (el factible sobre el infactible; entre factibles, se selecciona al que tenga mayor afinidad y entre infactibles al que cuente con menor número de restricciones violadas y mayor afinidad). En otro caso se selecciona al mejor individuo infactible. Con ello se desea obtener diversidad en la población y mantener además las virtudes de las soluciones infactibles que se encuentran cerca de la frontera del espacio de búsqueda factible, dado que el óptimo en muchos problemas se encuentra en dicha frontera.

Al implementar este mecanismo se obtuvieron algunos resultados bastante buenos tomando en cuenta que es un mecanismo muy sencillo de manejo de restricciones. Dichos resultados se presentaran en los capítulos siguientes.

4.6.1. Jerarquías estocásticas

Otra técnica implementada fue la de las jerarquías estocásticas, la cual es una técnica de manejo de restricciones propuesta por Thomas P. Runarsson y Xin Yao [34]. En los últimos años ha sido la técnica a vencer ya que utiliza todas las funciones de prueba establecidas en [30] para problemas con restricciones y las resuelve de

manera satisfactoria.

Este mecanismo de manejo de restricciones utiliza una función definida de la manera siguiente:

$$\psi = f(X) + r_g \phi(g_i(X); i = 1, ..., m)$$

donde ψ se convierte en la nueva función a optimizar, $\phi \geq 0$ es la función de penalización y r_g son los factores de penalización. Como puede verse solamente toma en cuenta las restricciones de desigualdad g_j, pero esto es debido a que podemos transformar las funciones de igualdad en desigualdades como se vio anteriormente.

La función de penalización, a su vez, se define de la siguiente forma:

$$\phi(g_i(X); i = 1, ..., m) = \sum_{j=1}^{m} max\{0, g_i(X)\}^2$$

Al utilizar una función de penalización llegamos al problema de cómo definir los factores de penalización r_g. Los autores proponen un algoritmo basado en el ordenamiento por burbuja y un porcentaje P_f mediante el cual comparan las soluciones existentes en el problema en cada generación y las jerarquizan según dicha comparación.

El algoritmo de jerarquías estocásticas se presenta a continuación:

```
Entrada: Población de anticuerpos y clones.
Salida:  Población de anticuerpos y clones ordenada.

PASO 1       Hasta que una condición de terminación se cumpla realizar:
PASO 1.1       Para toda la población de entrada (N) realizar:
PASO 1.1.1       Generar un número aleatorio u mediante la función U(0,1)
PASO 1.1.2          Si la penalización entre el individuo y su ‘‘vecino
                    izquierdo’’ es igual ó u es menor a P_{f} realizar:
PASO 1.1.2.1           Si la afinidad del individuo es mayor a la
                       afinidad de su ‘‘vecino izquierdo’’ intercambia posición.
PASO 1.1.3          De lo contrario realizar:
PASO 1.1.3.1           Si la penalización del individuo es mayor a la penalización
                       de su ‘‘vecino izquierdo’’ intercambia posición.
               fin;
PASO 1.2       Si no se realiza ningún intercambio de posición termina el ciclo.
             fin;
```

donde $U(0,1)$ es un generador de números aleatorios. La condición de terminación es $\lambda - 1$. En nuestro caso definimos $\lambda = N$, es decir igual al tamaño de la población, que es el valor propuesto por los autores de tal manera que el mecanismo sólo dependa del porcentaje P_f. Si $P_f = 0$ se dice que la penalización es muy grande; en el caso de $P_f = 1$ la penalización es muy pequeña.

Se puede ver claramente que el algoritmo al realizar la jerarquización permite que soluciones que no son las mejores (infactibles o con afinidad baja) ocupen buenas posiciones. Esto es deseable para no perder las bondades de dichas soluciones que nos permiten explorar correctamente tanto los espacios no factibles como la frontera de la zona factible.

El mecanismo es implementado en una estrategia evolutiva obteniendo muy buenos resultados, como se mostrará al realizar la comparación. Además los autores hacen un estudio utilizando distintos valores para P_f y logran establecerlo en 0.45.

El esquema de manejo de restricciones de las jerarquías estocásticas se incorporó en nuestro algoritmo, produciendo mejores resultados que el primer esquema adoptado.

4.7. Estructura del algoritmo

En [13] se describen los elementos principales que componen a un sistema biológicamente inspirado. Tomando en cuenta que estamos trabajando en problemas de optimización con restricciones podemos ajustar estos elementos a lo siguiente:

- **Representación de los componentes:** Los antígenos son representados por la función objetivo $f(X)$ que se desea optimizar y las restricciones del problema. Los anticuerpos son representados por las variables del problema (x) que representan soluciones potenciales codificadas en arreglos de bits.
- **Mecanismo para evaluar la interacción entre componentes y su ambiente:** La afinidad corresponde a la evaluación de la función objetivo junto con las restricciones del problema mediante una función de penalización.
- **Procedimientos de adaptación:** El principio de selección clonal.

La implementación del sistema inmune artificial comienza con la definición de antígenos y anticuerpos. Los anticuerpos pueden verse como arreglos de bits para representación binaria, que es la que se utilizó en este algoritmo. En nuestro caso, la afinidad se calcula mediante la evaluación del vector de variables de decisión de

cada anticuerpo en la función objetivo con ligeras modificaciones para los casos de minimización que son necesarias para que el mejor componente del sistema inmune tenga la mayor afinidad. Claramente se ve que la afinidad es simplemente la aptitud adoptada en los algoritmos genéticos.

La mutación es uniforme, y se aplica utilizando porcentajes inversamente proporcionales a la afinidad de un individuo. De tal manera que los clones de los componentes más afines tienen pequeños porcentajes de mutación el cual aumenta entre menor sea la afinidad. El porcentaje de mutación comienza para los clones más afines con el inverso de la longitud del arreglo de bits donde se codifico el vector de variables de decisión y se duplica para cada anticuerpo clonado llegando hasta un máximo de 0.35 %. Lo anterior significa que para los clones del anticuerpo más afín se mutará un bit, para los clones del siguiente anticuerpo dos bits, etc. hasta llegar a máximo de un tercio del tamaño del arreglo.

En cada ciclo se seleccionan de entre los anticuerpos (en esta aproximación sólo se tienen anticuerpos en la población ya que la afinidad se calcula mediante la función objetivo y las restricciones) y los clones mutados que formarán parte de la población de anticuerpos para la siguiente iteración. Dicha selección se basa en la afinidad de los individuos: aquéllos que sean más afines son seleccionados.

En la implementación se generan de manera aleatoria nuevos componentes en cada iteración, buscando mantener diversidad en la población. En la implementación se genera el 20 % de la población total. Los elementos generados remplazan a los anticuerpos menos afines.

En el esquema propuesto la selección es elitista, dado que se seleccionan entre los clones y anticuerpos aquellos con afinidad más alta para que formen la nueva población. Para ser clonados, se selecciona a toda la población de anticuerpos pero se ordenan con base en su afinidad, de tal forma que los más afines producen más clones. Para este fin se utilizó la fórmula propuesta por de Castro [12] para generar clones:

$$N_c = \sum_{i=1}^{n} round\left(\frac{\beta \times N}{i}\right)$$

donde N_c es el número total de clones, N es el tamaño de la población de anticuerpos y n es la cantidad de anticuerpos seleccionados para ser clonados. En nuestro caso n es igual a N, dado que nosotros seleccionamos a toda la población de anticuerpos para ser clonada. La razón de seleccionar a todos los anticuerpos se debe a que al final lo que se desea es observar la capacidad del esquema propuesto como motor de búsqueda y el modificar el número de anticuerpos seleccionados sólo reduce el número

de evaluaciones. Además, si utilizamos tamaños de población pequeños, el número de evaluaciones no crece de manera importante. Se utilizó $\beta = 1$ porque se eligió una población pequeña y se generaron clones del mejor individuo, en cantidad, al menos igual al tamaño de la población. Cada elemento de la sumatoria anterior representa la cantidad de clones para el anticuerpo i. Finalmente, *round* se refiere a obtener el entero superior más próximo al valor real dado como argumento.

En general los cambios principales que se introdujeron al algoritmo de selección clonal para lidiar con problemas con restricciones tienen que ver directamente con el manejo de las mismas, es decir, ya no se calcula la afinidad sólo con la función objetivo sino que tenemos que tomar en cuenta las restricciones del problema, y hay que trabajar ahora con soluciones factibles e infactibles.

El algoritmo se presenta a continuación:

```
Entrada: Función de afinidad (función objetivo y restricciones),
         tamaño de población de anticuerpos.
Salida:  Anticuerpo con mayor afinidad.

PASO 1    Generar al azar la población de anticuerpos de tamaño N.
PASO 2    Calcular la afinidad de los anticuerpos.
PASO 3    Calcular el número de clones (Nc).
PASO 4    Hasta que una condición de terminación se cumpla realizar:
PASO 4.1       Ordenar a los anticuerpos con respecto a su afinidad.
PASO 4.2       Clonar los N anticuerpos de manera directamente proporcional a su
               afinidad.
PASO 4.3       Mutar los Nc clones de manera inversamente proporcional a su afinidad.
PASO 4.4       Calcular la afinidad de los Nc clones.
PASO 4.5       Encontrar la nueva población de anticuerpos seleccionando a los
               mejores  entre los Nc clones y los N anticuerpos.
PASO 4.6       Ordenar a los anticuerpos con respecto a su afinidad.
PASO 4.7       Generar aleatoriamente nuevos anticuerpos, calcular su afinidad y
               reemplazar con ellos a los peores en la población de anticuerpos.
          fin;
PASO 5    Presentar al anticuerpo con mayor afinidad.
```

La condición de terminación en nuestro caso es el número máximo de generaciones.

En el paso 4.5 es donde se utiliza un mecanismo de manejo de restricciones para seleccionar a la nueva población.

El número de evaluaciones realizadas por el algoritmo se puede determinar usando:

Num. de Evaluaciones = Num. de anticuerpos+(Num. de clones+(0.2×Num. de anticuerpos)) ×Num. de generaciones

El 0.2 multiplicado por los anticuerpos nos representa la cantidad de anticuerpos que se sustituyen de manera aleatoria en cada iteración para agregar diversidad, es decir, el 20 % de la población.

La fórmula se obtiene al contar los individuos que necesitan ser evaluados en cada iteración del algoritmo.

4.8. Parámetros utilizados por el algoritmo

Los parámetros que necesita el algoritmo son los siguientes:

1. **Tamaño de la población:** Es el número de anticuerpos.
2. **Número de generaciones:** El número máximo de iteraciones a realizarse.
3. **Precisión:** Resolución en bits con que se codificará cada variable de decisión.
4. **Características de la función:** El número y los límites tanto inferior como superior de las variables de decisión, la función objetivo y las restricciones de diseño.
5. **Porcentaje de anticuerpos a ser sustituidos:** Es el porcentaje de la población de anticuerpos que será sustituido por nuevos anticuerpos generados de manera aleatoria en cada iteración.

Se puede observar que el algoritmo propuesto no necesita parámetros adicionales a los requeridos por un algoritmo genético simple, a excepción del porcentaje de anticuerpos a ser sustituidos. Este parámetro se recomienda en [12] que se fije entre 5 % y 25 %.

4.9. La representación

En un principio el algoritmo se implementó en representación binaria tradicional pero presentaba un problema. Durante las iteraciones del algoritmo, al intentar realizar las búsquedas locales, es deseable que las mutaciones en los anticuerpos más

afines sean lo más pequeñas posibles, como vimos con anterioridad. Esto se consigue utilizando mutación uniforme y mutando solamente un bit en la cadena. Con la representación binaria tradicional el mutar un bit no necesariamente representa una pequeña perturbación en el espacio de las variables de decisión.

Pongamos el ejemplo que se presentó al intentar resolver el problema g01 (véase apéndice A), el primero con restricciones. El problema tiene 13 variables de decisión y el óptimo se encuentra en (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1). El mejor individuo encontrado tenía como vector de variables de decisión (1, 1, 1, 1, 1, 1, 1, 1, 1, 2.9781, 2.9781, 2.9781, 1). Al observar el número 2.9781 en representación binaria obtenemos:

1	1	1	1	0	0	1	1	1	1	1	1	1

Al decodificar esta cadena obtenemos el valor de 7807. Un valor para mejorar el resultado sería por ejemplo 7808 pero para llegar a ese código binario (hay que recordar que la mutación se lleva a cabo en la cadena binaria) es necesario modificar más de un bit (8 para ser exactos).

Ahora buscando el mejor valor posible que podemos alcanzar con nuestra precisión nos damos cuenta que es 2.9960 cuyo código binario se ve como sigue:

1	1	1	1	0	1	0	1	1	1	0	0	0

Al decodificar esta cadena obtenemos el valor de 7864 y nuevamente necesitamos modificar más de un bit si quisiéramos llegar a él desde 7807.

Si utilizamos el código de Gray sabemos que la distancia de Hamming entre dos cadenas binarias cuya decodificación corresponde a dos enteros consecutivos, en uno. Por ende, usando códigos de Gray, la mutación para los individuos más afines tiene el efecto deseado.

En las secciones siguientes se mostrarán las tablas que contienen los resultados obtenidos por el algoritmo en representación binaria tradicional y con códigos de Gray para que se verifique la calidad de las soluciones.

Capítulo 5

Optimización sin restricciones

Hasta el momento, no existen estudios que analicen al esquema de selección clonal del sistema inmune artificial como motor de búsqueda en problemas de optimización numérica. En la literatura se encuentran trabajos que utilizan esta heurística para resolver problemas específicos [12], pero no se realiza un estudio de sus capacidades para resolver problemas de optimización numérica genéricos (por ejemplo, usando conjuntos de funciones de prueba).

En [39] se recopilan algunas funciones de prueba con distintas características que permiten analizar el desempeño de una heurística cualquiera en problemas de optimización sin restricciones y determinar si es un buen motor de búsqueda. Este estudio es necesario dado que para resolver problemas con restricciones es importante contar, como primer paso, con un buen motor de búsqueda para problemas de optimización sin restricciones.

5.1. Funciones de prueba

En total son veintitrés funciones de prueba las que se recopilan en [39], y éstas fueron extraídas de [19], [2], [37] y [35]. Cabe mencionar que esta cantidad de funciones es mayor de la que normalmente se utiliza en estudios experimentales relacionados con algoritmos evolutivos.

Las características de las funciones presentadas en el apéndice A son las siguientes:

1. En todos los casos se busca minimizar la función objetivo.

2. Las primeras trece funciones tienen alta dimensionalidad, es decir manejan una gran cantidad de variables de decisión (30 cada una de ellas).

3. Las funciones a partir de f_1 y hasta f_5 son unimodales, es decir, tienen un solo óptimo local que resulta ser el global.

4. f_6 es una función de paso, tiene un solo mínimo y es discontinua.

5. f_7 es una función con ruido, especificado por una función $random[0, 1)$ con distribución uniforme.

6. Las funciones a partir de f_8 y hasta f_{13} son multimodales, donde el número de mínimos locales se incrementa de manera exponencial con base en el número de variables.

7. Las funciones a partir de f_{14} y hasta f_{23} son de baja dimensionalidad y con pocos mínimos locales, pero han mostrado ser difíciles de resolver usando heurísticas.

5.2. bootstrap

El bootstrap es un método estadístico utilizado para establecer si una muestra de tamaño n es representativa, es decir, que tanto cambia una muestra obtenida de otra. Este método calcula un intervalo de confianza mediante el cual podemos asegurar que si tomamos una muestra de nuestro universo, (en nuestro caso representado por un conjunto de soluciones proporcionadas por nuestro algoritmo) dicha muestra estará dentro del intervalo de confianza calculado. Entre más cercano este el intervalo de confianza al óptimo y entre más cerrado sea dicho intervalo más eficiente es nuestro algoritmo.

Para cada función realizamos treinta corridas del algoritmo y de esta manera obtenemos treinta soluciones para cada función. Pero si queremos saber qué tan representativa es la muestra obtenida, es decir si obtuviéramos otras treinta soluciones más qué tan distintas serían a las que poseemos en este momento, es necesario realizar la prueba de bootstrap [6]. Esto se hace extrayendo un gran número de muestras (mil) de tamaño n (treinta para nosotros) de la muestra original aleatoriamente y con reposición. Así, aunque cada remuestra (muestra de la muestra) tendrá el mismo número de elementos que la muestra original, mediante el remuestreo con reposición cada remuestra podría tener algunos de los datos originales representados en ella más de una vez, y algunos que no aparecieran. Por lo tanto, cada una de estas remuestras probablemente será levemente y aleatoriamente diferente de la muestra original.

Es muy importante, antes de realizar el bootstrap, verificar la distribución de nuestras soluciones, si es distinta a la normal podemos aplicar el método, de otra manera es necesario seguir otro procedimiento más sencillo que consiste simplemente en utilizar la función siguiente para calcular los intervalos de confianza:

$$x \pm t \times (\frac{s}{\sqrt{n}}) \tag{5.1}$$

donde x es la media de la población de muestras, t es el valor crítico de la distribución que puede ser encontrado en tablas y para nuestro caso para treinta muestras y con un intervalo de 95 % es 1.697, s es la desviación estándar y finalmente n es el tamaño de la muestra.

Entre las pruebas no paramétricas que comúnmente se utilizan para verificar si una distribución se ajusta o no a una distribución esperada (en particular a la distribución normal) se encuentran la prueba de Kolmogorov-Smirnov. La prueba de Kolmogorov-Smirnov es bastante potente con muestras grandes (de treinta para arriba). El nivel de medición de la variable y su distribución son elementos que intervienen en la selección de la prueba que se utilizará en el procesamiento posterior. De hecho, si la variable es continua con distribución normal, se podrán aplicar técnicas paramétricas. Si es una variable discreta o continua no normal, sólo son aplicables técnicas no paramétricas pues aplicar las primeras arrojaría resultados de dudosa validez.

La prueba de K-S de una muestra es una prueba de bondad de ajuste. Esto es, se interesa en el grado de acuerdo entre la distribución de un conjunto de valores de la muestra y alguna distribución teórica específica. Determina si puede pensarse razonablemente que las mediciones muestrales provienen de una población que tenga esa distribución teórica. En la prueba se compara la distribución de frecuencia acumulativa de la distribución teórica con la distribución de frecuencia acumulativa observada y se determina el punto en el que estas dos distribuciones muestran la mayor divergencia.

Esta prueba se realizó utilizando un programa llamado datalab versión 2.12 (H. Lohninger, 1999-2001), el cual nos arroja un valor conocido como “KS stadistic” y una tabla de valores críticos dependiente del número de muestras de la población (treinta en nuestro caso).

Por otro lado el bootstrap se realizó con el programa Resampling Stats versión 5.0.2 (Resampling Stats Inc. 1990-1999). Fue necesario escribir un código muy sencillo que se muestra a continuación:

```
READ file "c:/Documents and Settings/Propietario/Mis
```

```
documentos/tesis/pruebas/sin/tod1_1.txt" a
REPEAT 1000 SAMPLE 30 a b
MEAN b c SCORE c scrboard END

HISTOGRAM scrboard
PERCENTILE scrboard (2.5 97.5) interval
PRINT interval
WRITE file "c:/Documents and Settings/Propietario/Mis
documentos/tesis/pruebas/sin/bootstrap.txt" append interval
```

READ lee de un archivo ASCII y va introduciendo, en este caso, en el vector "a" los datos del archivo. REPEAT crea un ciclo que termina con la palabra END, en este caso se itera mil veces. SAMPLE toma treinta soluciones contenidas en "a" de manera aleatoria y las coloca en un nuevo vector "b" de donde posteriormente se obtiene la media la cual se resguarda en la variable "c" y más tarde en el vector "scrboard" se van almacenando los distintas medias obtenidas en cada una de las mil iteraciones. Con el comando PERCENTILE aplicado a el vector "scrboard" obtenemos los intervalos de confianza del bootstrap que nos permitirá saber si nuestra muestra es representativa. Entre paréntesis se coloca (2.5 97.5) que le indica a PERCENTILE que coloque los intervalos sin tomar en cuenta las peores ni los mejores 2.5 % medias contenidas en "scrboard", dándonos una confiabilidad de 95 %, que es la que se utiliza comúnmente [6]. PRINT imprime en pantalla el intervalo y con WRITE escribimos en un archivo el intervalo, utilizando append para que no borre el contenido anterior en el archivo si es que existe.

Tanto datalab como Resampling Stats trabajan sobre la plataforma Windows Xp.

5.3. Comparación de Resultados

En el artículo donde fueron utilizadas las funciones de prueba [39] se presenta una propuesta llamada "Fast evolutionary programming" (FEP), es un algoritmo evolutivo que se enfoca en mejorar la velocidad de convergencia. Los resultados obtenidos por FEP, para las funciones presentadas, en 30 corridas para cada una se muestran en la tabla 5.1.

Se observan 4 columnas en ella. La primera corresponde al nombre del problema o función, la siguiente al óptimo conocido de dicho problema, continuamos con la media de las treinta corridas por función y finalmente la desviación estándar de dicho número de corridas.

Problema	*Optimo*	*Media*	*Desv.Estandar*
$f1$	0.000000	5.7×10^{-4}	1.3×10^{-4}
$f2$	0	8.1×10^{-3}	7.7×10^{-4}
$f3$	0	1.6×10^{-2}	1.4×10^{-2}
$f4$	0	0.3	0.5
$f5$	0	5.06	5.87
$f6$	0	0	0
$f7$	0	7.6×10^{-3}	2.6×10^{-3}
$f8$	-12569.5	-12554.5	52.6
$f9$	0	4.6×10^{-2}	1.2×10^{-2}
$f10$	0	1.8×10^{-2}	2.1×10^{-3}
$f11$	0	1.6×10^{-2}	2.2×10^{-2}
$f12$	0	9.2×10^{-6}	3.6×10^{-6}
$f13$	0	1.6×10^{-4}	7.3×10^{-5}
$f14$	1	1.22	0.56
$f15$	0.0003075	5.0×10^{-4}	3.2×10^{-4}
$f16$	-1.0316285	-1.03	4.9×10^{-7}
$f17$	0.398	0.398	1.5×10^{-7}
$f18$	3	3.02	0.11
$f19$	-3.86	-3.86	1.4×10^{-5}
$f20$	-3.32	-3.27	5.9×10^{-2}
$f21$	-10	-5.52	1.59
$f22$	-10	-5.52	2.12
$f23$	-10	-6.57	3.14

Cuadro 5.1: Resultados de FEP para las veintitrés funciones propuestas.

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*	*Varianza*
$f1$	0	0	0	0	0	0	0
$f2$	0	0	0	0	0	0	0
$f3$	0	1367.333459	3332.792955	2983.979254	5963.842168	1169.413989	1367529.077034
$f4$	0	1.2×10^{-2}	4.8×10^{-2}	4.2×10^{-2}	9.9×10^{-2}	2.1×10^{-2}	4.4×10^{-4}
$f5$	0	27.651670	29.497907	28.690021	45.721806	3.747744	14.045584
$f6$	0	0	0	0	0	0	0
$f7$	0	9.4×10^{-3}	2.1×10^{-2}	2.08×10^{-2}	3.5×10^{-2}	7.8×10^{-3}	6.2×10^{-5}
$f8$	-12569.5	-12461.635709	-12002.845341	-12038.197850	-11580.446091	222.557557	49531.866110
$f9$	0	27.419357	38.231149	36.581194	55.514427	7.901376	62.431737
$f10$	0	0.000000	1.003750	1.485903	2.120073	9.1×10^{-1}	8.4×10^{-1}
$f11$	0	0	0	0	0	0	0
$f12$	0	4.837932	6.920707	6.951399	8.914832	1.099775	1.209505
$f13$	0	1.343344	1.702812	1.702141	1.973638	2.02×10^{-1}	4.1×10^{-2}
$f14$	1	9.98×10^{-1}	1.182497	1.202297	1.419513	1.28×10^{-1}	1.65×10^{-2}
$f15$	0.0003075	5.02×10^{-4}	7.75×10^{-4}	7.86×10^{-4}	9.95×10^{-4}	1.24×10^{-4}	0
$f16$	-1.0316285	-1.031628	-1.011343	-1.030898	-9.52×10^{-1}	2.46×10^{-2}	6.09×10^{-4}
$f17$	0.398	3.97×10^{-1}	4.02×10^{-1}	3.99×10^{-1}	4.21×10^{-1}	7.92×10^{3}	6.3×10^{-5}
$f18$	3	3	3	3	3	0	0
$f19$	-3.86	-3.862782	-3.849911	-3.862602	-3.813068	2.1×10^{-2}	4.5×10^{-4}
$f20$	-3.32	-3.321967	-3.246102	-3.297738	-3.029240	9.4×10^{-2}	8.88×10^{-3}
$f21$	-10	-10.150349	-5.011573	-4.714028	-2.071737	1.711871	2.930501
$f22$	-10	-10.376705	-5.182060	-4.743011	-3.839883	1.768181	3.126463
$f23$	-10	-10.508356	-5.340707	-5.128897	-4.493287	1.263930	1.597518

Cuadro 5.2: Resultados de la propuesta para las veintitrés funciones con codificación binaria tradicional.

Luego se presentan en la tabla 5.2 y en la tabla 5.3 los resultados obtenidos por el algoritmo propuesto en esta tesis con representación binaria tradicional y usando códigos de Gray, respectivamente, después de treinta corridas con cada una de las funciones.

En este caso, además de las columnas de la tabla 5.1, se presenta en la tercera columna la mejor solución obtenida por el algoritmo. En la siguiente columna se muestra la mediana de las treinta corridas. En la quinta columna se muestra la peor solución obtenida. Al final se muestra la varianza de las 30 corridas. Usamos una precisión de siete decimales para reportar los resultados.

En la tabla 5.4 se muestran, para ambas representaciones, el número de variables de cada problema, los bits utilizados para codificar cada variable y el número de evaluaciones realizadas en comparación con las utilizadas en [39]. Se buscó no realizar un número mayor de evaluaciones que las que utilizó FEP.

Existen 7 casos ($f_7, f_8, f_{15}, f_{20}, f_{21}, f_{22}, f_{23}$) en los cuales no realizamos el número máximo de evaluaciones (496825) en aras de permitir una comparación justa con FEP y además en estos problemas no logramos una desviación estándar igual a 0. Es entonces deseable verificar si al aumentar el número de evaluaciones en estos casos podemos mejorar las soluciones reduciendo nuestra desviación estándar. En la tabla 5.5 se muestran los resultados obtenidos por el algoritmo para esos problemas realizando 496825 evaluaciones.

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*	*Varianza*
$f1$	0	0	0	0	0	0	0
$f2$	0	0	0	0	0	0	0
$f3$	0	1.6×10^{-2}	1.7×10^{-1}	1.6×10^{-1}	3.3×10^{-2}	6.94×10^{-2}	4.82×10^{-3}
$f4$	0	3.5×10^{-5}	7.8×10^{-5}	7.2×10^{-5}	1.58×10^{-4}	3.1×10^{-5}	0
$f5$	0	4.04×10^{-3}	4.681896	1.896141	19.942745	5.684807	32.317029
$f6$	0	0	0	0	0	0	0
$f7$	0	1.24×10^{-2}	3×10^{-2}	2.13×10^{-2}	1.22×10^{-1}	2.39×10^{-2}	2.73×10^{-4}
$f8$	-12569.5	-12569.486618	-12451.048269	-12451.048284	-12214.171614	105.934467	11222.111287
$f9$	0	0	1.443355	9.94×10^{-1}	7.959662	1.899934	3.609750
$f10$	0	0	0	0	1×10^{-6}	0	0
$f11$	0	0	0	0	0	0	0
$f12$	0	0	0	0	0	0	0
$f13$	0	0	0	0	0	0	0
$f14$	1	9.98×10^{-1}	9.98×10^{-1}	9.98×10^{-1}	9.98×10^{-1}	0	0
$f15$	0.0003075	3.075×10^{-4}	3.31×10^{-4}	3.077×10^{-4}	7.5×10^{-4}	9.65×10^{-5}	0
$f16$	-1.0316285	-1.0316285	-1.0316285	-1.0316285	-1.0316285	0	0
$f17$	0.398	0.398	0.398	0.398	0.398	0	0
$f18$	3	3	3	3	3	0	0
$f19$	-3.86	-3.86	-3.86	-3.86	-3.86	0	0
$f20$	-3.32	-3.32	-3.26	-3.32	-3.20	5.94×10^{-2}	3.53×10^{-3}
$f21$	-10	-10.153200	-4.496644	-5.100772	-2.630472	1.734590	3.00880174
$f22$	-10	-10.402941	-7.193047	-5.128823	-1.837593	2.999228	8.99537157
$f23$	-10	-10.536410	-5.285619	-5.175647	-2.427335	2.819496	7.94955922

Cuadro 5.3: Resultados de la propuesta para las veintitrés funciones, con codificación de Gray.

Problema	bits × variable	Variables	Población	Clones	Gen. de la propuesta	Eval. de la propuesta	Eval. de FEP
$f1$	31	30	25	87	1200	110425	150000
$f2$	28	30	25	87	1200	110425	200000
$f3$	31	30	25	87	5400	496825	500000
$f4$	31	30	25	87	5400	496825	500000
$f5$	29	30	25	87	5400	496825	500000
$f6$	31	30	25	87	1200	110425	150000
$f7$	27	30	25	87	3000	276025	300000
$f8$	31	30	25	87	3000	276025	900000
$f9$	27	30	25	87	5400	496825	500000
$f10$	30	30	25	87	1200	110425	150000
$f11$	31	30	25	87	900	82825	200000
$f12$	30	30	25	87	1000	92025	150000
$f13$	30	30	25	87	1000	92025	150000
$f14$	30	2	25	87	500	46025	10000
$f15$	27	4	25	87	4200	386425	400000
$f16$	27	2	25	87	100	9225	10000
$f17$	28	2	25	87	100	9225	10000
$f18$	26	2	25	87	100	9225	10000
$f19$	24	3	25	87	100	9225	10000
$f20$	24	6	25	87	100	9225	20000
$f21$	27	4	25	87	100	9225	10000
$f22$	27	4	25	87	100	9225	10000
$f23$	27	4	25	87	100	9225	10000

Cuadro 5.4: Parámetros utilizados por el algoritmo propuesto

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*	*Varianza*
$f7$	0	3.03×10^{-3}	1.87×10^{-2}	1.28×10^{-2}	7.28×10^{-2}	1.59×10^{-2}	2.55×10^{-4}
$f8$	-12569.5	-12569.486618	-12569.486618	-12569.486618	-12569.486618	0	0
$f15$	0.0003075	3.075×10^{-4}	3.075×10^{-4}	3.075×10^{-4}	3.075×10^{-4}	0	0
$f20$	-3.32	-3.32	-3.26	-3.32	-3.20	5.94×10^{-2}	3.53×10^{-3}
$f21$	-10	-10.153200	-6.869122	-5.100772	-5.100772	2.409855	5.80739903
$f22$	-10	-10.402941	-6.447352	-5.128823	-5.128823	2.283760	5.21556013
$f23$	-10	-10.536410	-7.051914	-5.175647	-5.175647	2.556921	6.53784494

Cuadro 5.5: Resultados obtenidos para algunas funciones tras aumentar el número de evaluaciones.

Función	K-S Stadistic	Función	K-S Stadistic	Función	K-S Stadistic
$f1$	0.5000	$f9$	0.2967	$f17$	0.6224
$f2$	0.5000	$f10$	0.4610	$f18$	0.9664
$f3$	0.1061	$f11$	0.5000	$f19$	0.9999
$f4$	0.1760	$f12$	0.5000	$f20$	0.5056
$f5$	0.2096	$f13$	0.5000	$f21$	0.3414
$f6$	0.5000	$f14$	0.8086	$f22$	0.3400
$f7$	0.2363	$f15$	0.5000	$f23$	0.3062
$f8$	0.5000	$f16$	0.8489		

Cuadro 5.6: Resultados de la prueba Kolmogorov-Smirnov.

5.4. Análisis de resultados

El algoritmo muestra resultados muy distintos dependiendo de la representación binaria utilizada, comportándose de mucho mejor manera con la codificación de Gray.

Al realizar la prueba de Kolmogorov-Smirnov se obtuvo la tabla 5.6. Los valores críticos se muestran en la tabla 5.7. En aquellas muestras en las que su K-S stadistic sea mayor a las presentadas en la tabla 5.7 podemos decir que no tienen una distribución normal. Si en algún caso nuestra muestra nos arroja un K-S stadistic dentro o menor de los valores críticos decimos que tiene forma normal con cierto porcentaje de certeza, que se encuentra en la columna izquierda. En la mayoría de los casos se utiliza sólo el valor crítico que nos proporciona mayor certeza (en nuestro caso 0.1922) y entonces en todas las muestras que tengan una KS- stadistic menor a este valor crítico se dice que tienen una distribución normal.

Tenemos entonces dos casos donde la distribución de la muestra se adapta a la nor-

Valores críticos:

0.01	0.2928
0.05	0.2443
0.10	0.2191
0.15	0.2048
0.20	0.1922

Cuadro 5.7: Valores críticos de la prueba Kolmogorov-Smirnov para una muestra de treinta soluciones.

Función	Intervalo de confianza	Función	Intervalo de confianza	Función	Intervalo de confianza
$f1$	$0 - 0$	$f9$	$1.0285 - 2.70319$	$f17$	$0.39788736 - 0.39788736$
$f2$	$0 - 0$	$f10$	$1.33 \times 10^{-7} - 4.33 \times 10^{-7}$	$f18$	$3.0 - 3.0$
$f3$	$0.148498 - 0.191502$	$f11$	$0 - 0$	$f19$	$-3.86278215 - -3.86278215$
$f4$	$6.83 \times 10^{-5} - 8.76 \times 10^{-5}$	$f12$	$0 - 0$	$f20$	$-3.31803 - -3.28633$
$f5$	$3.77294 - 8.77138$	$f13$	$0 - 0$	$f21$	$-6.45991 - -4.70448$
$f6$	$0 - 0$	$f14$	$0.998004 - 0.998004$	$f22$	$-8.88406 - -6.698$
$f7$	$0.0191383 - 0.0338964$	$f15$	$0.000308 - 0.000308$	$f23$	$-7.05544 - -4.96727$
$f8$	$-12569.5 - -12569.5$	$f16$	$-1.031628 - -1.031628$		

Cuadro 5.8: Intervalos de confianza encontrados con bootstrap.

mal: $f3$ y $f4$. Sus histogramas se presentan en las figuras 5.1 y 5.2, respectivamente. En la figura 5.3 se muestra además una distribución normal.

Gráficamente se observa que la forma de las distribuciones para las dos funciones se asemeja bastante a la normal.

Como otro ejemplo veamos el caso de f5 y f23 donde su K-S stadistic se acerca bastante al valor crítico, pero al graficar sus histogramas se ve que no se acercan en ningún caso a una distribución normal. Véase las figuras 5.4 y 5.5.

En los casos donde todas las muestras de la población son iguales, es decir que se resuelve consistentemente la función, no es necesario realizar la prueba Kolmogorov-Smirnov (en todos lo casos la muestra no se asemeja a la normal, como ejemplo véase la figura 5.6).

Más aún no es necesario realizar en estos problemas el bootstrap dado que el intervalo de confianza se definirá con la solución del problema. Lo anterior se ve más claro en la tabla 5.8. En ella se muestran los intervalos de confianza obtenidos al realizar el bootstrap en esas funciones, además de los intervalos obtenidos para f3 y f4 con la ecuación 5.1.

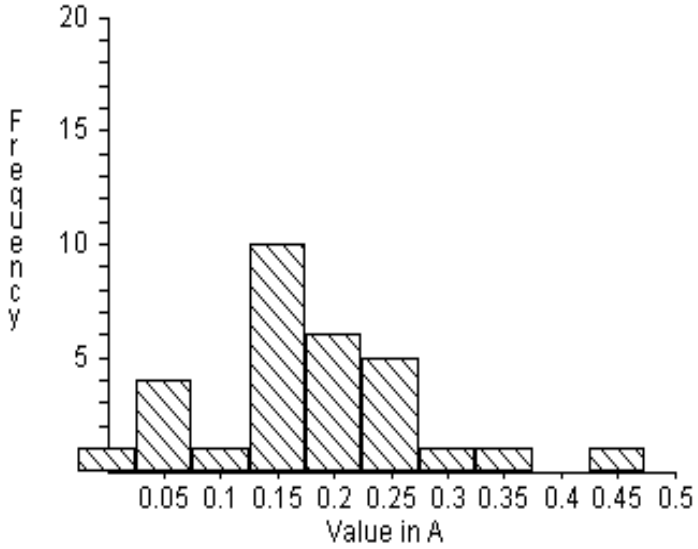

Figura 5.1: Histograma para f3

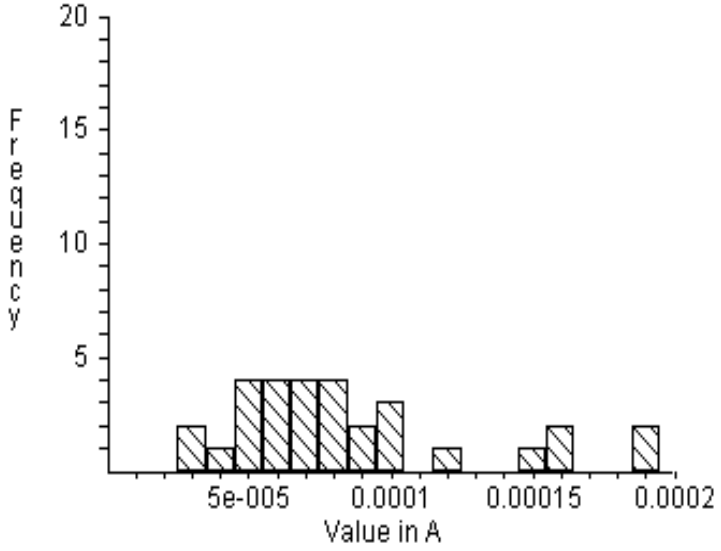

Figura 5.2: Histograma para f4

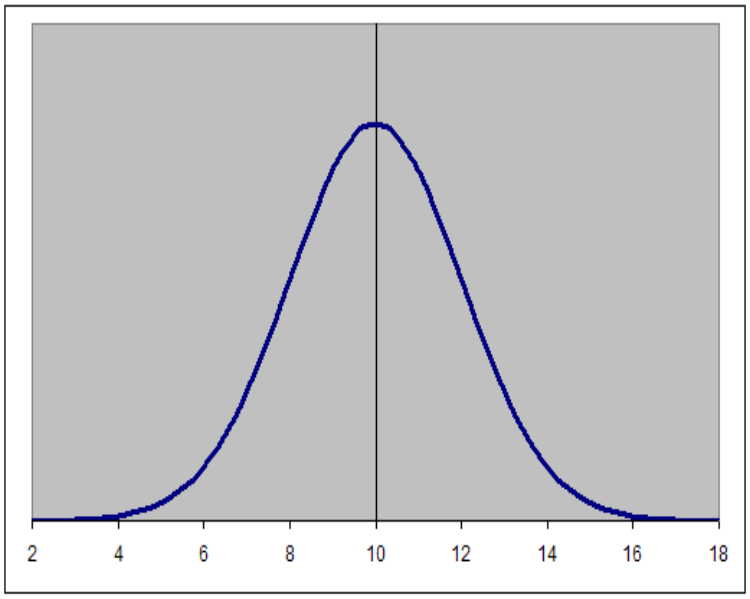

Figura 5.3: Distribución normal

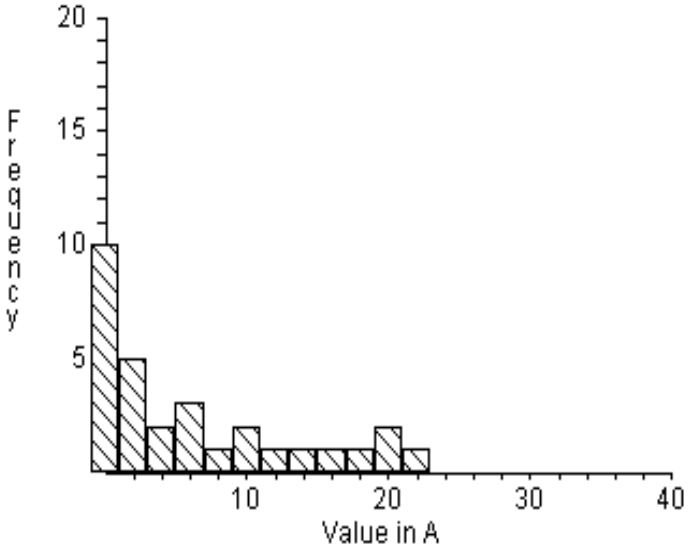

Figura 5.4: Histograma para f5

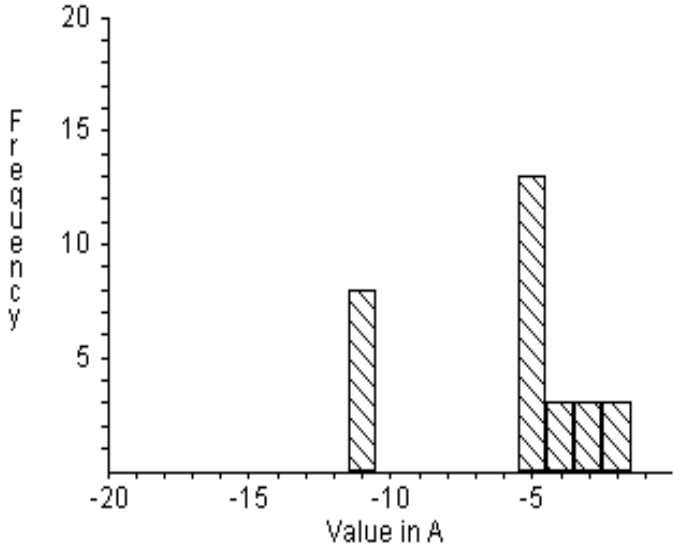

Figura 5.5: Histograma para f23

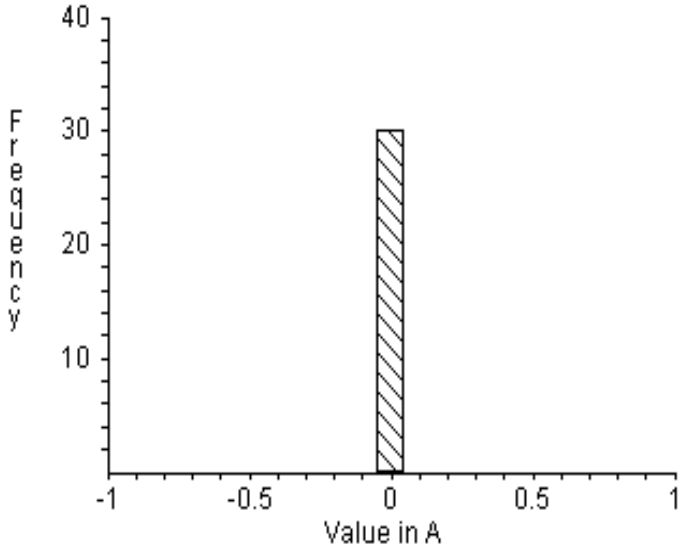

Figura 5.6: Histograma para f1

La tabla la podemos interpretar basados en que entre más cerrado sea el intervalo mayor será nuestra confianza sobre la representación de la muestra. Y en todos los casos el intervalo de confianza es lo suficientemente pequeño para tomar a la muestra como representativa.

Es claro que se encuentran buenos resultados para todas las funciones y en algunas se mejoran los resultados presentados en [39] utilizando en todos los casos menor cantidad de evaluaciones de la función objetivo.

En $f_1, f_2, f_6, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{16}, f_{17}, f_{18}$ y f_{19} se obtienen los óptimos con una desviación estándar de 0 por lo que podemos decir que se resuelven completamente 12 funciones, utilizando el algoritmo propuesto.

En f_3, f_4 y f_5 se obtienen valores competitivos si los comparamos con los de FEP.

Una cosa importante que podemos observar del desempeño del algoritmo es cuando analizamos los resultados para f_7, f_8 y f_{15}. La primera de ellas es unimodal y la segunda y tercera son multimodales, aunque la última tiene menor número de variables y óptimos locales. En las primeras corridas se igualan los resultados obtenidos en comparación a los presentados por FEP. La segunda muestra de corridas donde estamos aumentando el número de evaluaciones nos indica que el algoritmo sigue mejorando los resultados a la par que las evaluaciones aumentan resultando en una disminución drástica de la desviación estándar (f_8, f_{15}) eliminándose completamente en (f_7).

Ahora bien, un caso especial es f_{20}. Esta función presenta una característica especial para el algoritmo que le hace perder diversidad muy rápido. Es decir, los anticuerpos se parecen los unos a los otros y convergen prematuramente lo que provoca que el algoritmo se quede atrapado en un óptimo local con cierta frecuencia más allá del número de evaluaciones. Esto lo muestra la tercera tabla donde los resultados para f_{20} son los mismos aún aumentando considerablemente el número de evaluaciones.

Finalmente, en los casos de las funciones f_{21}, f_{22} y f_{23} la desviación estándar no se reduce notablemente. Esto es debido a que estas funciones tienen óptimos locales que provocan que los algoritmos se queden estancados en ellos, por lo que provocan resultados distintos en diferentes corridas. Lo interesante en este caso es que al aumentar el número de corridas el algoritmo obtiene mejores resultados, lo cual se puede observar en la media alcanzada para los tres casos mostrados en la tabla 3. A pesar de que el algoritmo cae en óptimos locales en varias ocasiones, suele ser el caso de que con una cantidad aceptable de iteraciones puede salir de ellos para mejorar el resultado.

Lo anterior es importante por que el algoritmo de manera “natural” puede salir

de óptimos locales aunque tiene cierta predisposición a caer en ellos.

El esquema propuesto resulta ser un motor de búsqueda capaz de resolver problemas de optimización con distintas características. Además resulta ser competitivo con respecto a FEP en cuanto a la velocidad de convergencia (número de evaluaciones necesarias para llegar al óptimo o a una buena aproximación), la que es una de las características más destacadas de FEP. Por lo anterior podemos enfocarnos, ahora sí, al estudio de la propuesta para problemas de optimización con restricciones de diseño, que es el tema principal de la tesis.

Capítulo 6

Optimización con restricciones

En el capítulo anterior pudimos constatar que el sistema inmune artificial tiene propiedades que lo convierten en un buen motor de búsqueda. Nuestra propuesta ahora debe demostrar que podemos resolver problemas de optimización con restricciones simplemente agregando a dicho motor un mecanismo capaz de manejarlas.

6.1. Funciones de prueba

Utilizamos un conjunto de funciones de prueba, propuestas en [30] las cuales pueden ser encontradas en el Apéndice A. Estas funciones son ampliamente utilizadas en el área de algoritmos evolutivos. Las características de dichas funciones se presentan en la tabla 6.1. En ella se muestra: el número de variables de decisión, el tipo de función objetivo, ρ que representa el porcentaje de individuos factibles de entre un millón de individuos generados de manera aleatoria, es decir, ρ nos representa de cierta manera qué tan grande es la zona factible del problema. Además de lo anterior se presenta el tipo de restricciones del problema en las 4 últimas columnas.

Podemos observar que hay funciones que tienen una zona factible muy pequeña como en el caso de g05, g07 y g13 en las que $\rho = 0.0000\,\%$, es decir, no se encuentra ningún individuo factible, por lo que llegar a la zona factible es muy complicado. Hay otros casos como en g02 donde la zona factible es muy grande pero llegar al óptimo es muy complicado ya que esta función tienen muchos óptimos locales.

g02, g03 y g12 son problemas de maximización, en los demás casos lo que se desea es minimizar la función.

Problema	n	**Tipo de función**	ρ	**LI**	**NI**	**LE**	**NE**
g01	13	cuadrática	0.0003 %	9	0	0	0
g02	20	no lineal	99.9973 %	1	1	0	0
g03	10	no lineal	0.0026 %	0	0	0	1
g04	5	cuadrática	27.0079 %	0	6	0	0
g05	4	no lineal	0.0000 %	2	0	0	3
g06	2	no lineal	0.0057 %	0	2	0	0
g07	10	cuadrática	0.0000 %	3	5	0	0
g08	2	no lineal	0.8581 %	0	2	0	0
g09	7	no lineal	0.5199 %	0	4	0	0
g10	8	lineal	0.0020 %	3	3	0	0
g11	2	cuadrática	0.0973 %	0	0	0	1
g12	3	cuadrática	4.7697 %	0	9^3	0	0
g13	5	no lineal	0.0000 %	0	0	1	2

Cuadro 6.1: Características de las trece funciones de prueba. ρ= porcentaje de individuos factibles encontrados entre un millón de individuos generados aleatoriamente. n=número de variables de decisión. LI=número de desigualdades lineales, NI=número de desigualdades no lineales, LE= número de igualdades lineales y NE=número de igualdades no lineales.

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*
$g01*$	-15.0	-14.868694	-14.660287	-14.799191	-12.789464	0.501459
$g02$	0.803619	0.775589	0.749575	0.752521	0.683894	0.025699
$g03*$	1.0	0.998915	0.970785	0.964906	0.928419	0.024912
$g04$	-30665.539	-30650.006917	-30460.854126	-30403.651278	-30366.985418	96.140797
$g05**$	5126.498	4093.774954	4968.677368	4978.866798	5613.728741	357.862789
$g06**$	-6961.814	741.274894	915.284233	771.120058	2832.827679	472.680087
$g07$	24.306	24.808704	30.866140	30.207166	35.445577	2.435335
$g08$	0.095825	0.095825	0.093398	0.093130	0.093130	0.000803
$g09$	680.630	684.128867	704.872630	694.119386	753.221034	17.584361
$g10**$	7049.25	4989.869373	7762.060646	7443.750040	10326.566021	1286.573548
$g11$	0.75	0.750295	0.865079	0.764238	1.567670	0.22056
$g12$	1.0	0.999996	0.907750	0.910354	0.725285	0.077762
$g13**$	0.053950	0.016984	0.197594	0.076935	0.796149	0.231488

Cuadro 6.2: Resultados de la propuesta con codificación binaria tradicional y reglas simples. * Menos de 25 % de las soluciones fueron no factibles. ** No se encontraron valores factibles.

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*
$g01$	-15.0	-15.0	-15.0	-15.0	-15.0	0.0
$g02$	0.803619	0.760753	0.700945	0.718346	0.562943	0.049646
$g03$	1.0	1.0	0.999688	0.999841	0.997992	0.000579
$g04$	-30665.539	-30665.504	-30662.678	-30663.057	-30658.778	1.946
$g05****$	5126.498	3852.789	4750.810633	4729.697	5542.256	345.874
$g06$	-6961.814	-6961.813	-6961.813	-6961.813	6961.810	0.000552
$g07$	24.306	24.945	27.017	26.668	30.007	1.709
$g08$	0.095825	0.095825	0.095825	0.095825	0.095825	0.0
$g09$	680.630	680.727	681.649	681.510	682.853	0.599
$g10**$	7049.25	7451.54	8344.18	8138.17	10509.34	962.58
$g11$	0.75	0.75	0.76	0.75	0.79	0.011
$g12$	1.0	1.0	1.0	1.0	1.0	0.0
$g13***$	0.053950	0.059553	0.059215	0.059553	0.062709	0.003000

Cuadro 6.3: Resultados de la propuesta con el mecanismo de reglas y representación basada en códigos de Gray. ** Más de 50 % de las soluciones fueron factibles. *** 15 % de soluciones factibles. **** No se encontraron valores factibles.

6.2. Comparación

Como se explicó en capítulos anteriores la primer representación utilizada al implementar la propuesta fue la binaria tradicional. Los resultados obtenidos en algunas funciones fueron buenas aproximaciones como puede verse en la tabla 6.2. Como se vio también con anterioridad obtuvimos mejoras significativas al cambiar la representación a una que adopte códigos de Gray. De esta manera, utilizando el mismo algoritmo obtuvimos los resultados mostrados en la tabla 6.3. En ambos casos se realizaron treinta corridas por cada una de las funciones. Además, los parámetros utilizados en cada caso se muestran en la tabla 6.4.

En el caso de representación basada en códigos de Gray, se resuelven todas las funciones aunque aún existen algunas en las que no se logra que todas las soluciones sean factibles. En el caso de la representación binaria tradicional no se obtiene para cuatro funciones ninguna solución factible (g5, g6, g10, y g13) y en otros dos casos

Problema	bits × variable	Variables	Población	Clones	Numero de Generaciones	Eval. de la propuesta
$g01$	31	13	50	207	1605	356360
$g02$	20	20	50	207	1605	356360
$g03$	31	10	50	207	1605	356360
$g04$	31	5	50	207	1605	356360
$g05$	29	4	50	207	1605	356360
$g06$	28	2	50	207	1605	356360
$g07$	31	20	50	207	1605	356360
$g08$	28	2	50	207	1605	356360
$g09$	27	7	50	207	1605	356360
$g10$	31	8	50	207	1605	356360
$g11$	20	2	50	207	1605	356360
$g12$	20	2	50	207	1605	356360
$g13$	31	5	50	207	1605	356360

Cuadro 6.4: Parámetros utilizados por el algoritmo propuesto

(g01 y g03) no todas las soluciones obtenidas son factibles.

Aunque se ve una mejora importante en las soluciones encontradas por el algoritmo basada en códigos de Gray, resultó preocupante que en tres funciones (que son las más difíciles de resolver) no se obtuvieron siempre soluciones factibles, sobre todo en g05 donde no logró obtenerse soluciones que lo sean.

Habíamos demostrado en el capítulo anterior que el motor de búsqueda es lo suficientemente bueno para pensar que con un buen mecanismo de manejo de restricciones podíamos obtener buenos resultados en problemas de optimización con restricciones. El uso de las reglas como mecanismo de manejo de restricciones es de los más simples posibles. Entonces cambiar el mecanismo a otro que haya probado tener mejores resultados fue la opción siguiente más plausible.

Jerarquías estocásticas [34] es un mecanismo de manejo de restricciones que ha tenido excelentes resultados en las funciones de prueba que se utilizan en esta tesis. Ha servido de referencia en varios trabajos sobre optimización global con restricciones. Pensamos entonces adaptar las jerarquías estocásticas a nuestro algoritmo y observar los resultados. De esta manera podemos realizar además la comparación entre motores de búsqueda, ya que las jerarquías estocásticas en su versión original trabajan con estrategias evolutivas. Los resultados obtenidos por la propuesta se presentan en la tabla 6.5. Por otra parte los resultados reportados por las jerarquías estocásticas se presentan en la tabla 6.6 [34]. Se reportan en ambas tablas los resultados de treinta corridas por cada función y con 356,360 y 350,000 evaluaciones de la función objetivo, respectivamente. En ambos casos se usó $P_f = 0.45$. Este valor resulta de un estudio empírico realizado por Runarsson y Yao, además de que en nuestro caso también se obtuvieron los mejores resultados con ese valor.

Otra forma de analizar el desempeño de nuestro algoritmo es compararlo contra

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*
$g01$	-15.0	-15.0	-15.0	-15.0	-15.0	0.0
$g02$	0.803619	0.802411	0.754560	0.756068	0.697751	0.028496
$g03$	1.0	1.0	0.999515	0.999728	0.997961	0.000600
$g04$	-30665.539	-30665.467826	-30663.307967	-30663.878098	-30660.690100	1.593303
$g05$	5126.498	5126.630079	5155.757166	5143.537515	5305.711893	35.025237
$g06$	-6961.814	-6961.813461	-6961.813308	-6961.813252	-6961.812834	0.000131
$g07$	24.306	25.100562	29.353506	29.417149	33.407723	2.314454
$g08$	0.095825	0.095825	0.095825	0.095825	0.095825	0.0
$g09$	680.630	680.734967	681.252938	681.107310	682.831150	0.470075
$g10$	7049.25	7082.837133	7799.129549	7688.885053	8913.864368	488.331819
$g11$	0.75	0.749900	0.751924	0.749971	0.775735	0.005215
$g12$	1.0	1.0	1.0	1.0	1.0	0.0
$g13$	0.053950	0.054814	0.286915	0.441018	0.545016	0.206451

Cuadro 6.5: Resultados de la propuesta con jerarquías estocásticas y usando códigos de Gray.

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*
$g01$	-15.0	-15.0	-15.0	-15.0	-15.0	0.0
$g02$	0.803619	0.803515	0.781975	0.785800	0.726288	0.02
$g03$	1.0	1.0	1.0	1.0	1.0	0.0
$g04$	-30665.539	-30665.539	-30665.539	-30665.539	-30665.539	0.0
$g05$	5126.498	5126.497	5128.881	5127.372	5142.472	3.5
$g06$	-6961.814	-6961.814	-6875.940	-6961.814	-6350.262	160.0
$g07$	24.306	24.307	24.374	24.357	24.642	0.66
$g08$	0.095825	0.095825	0.095825	0.095825	0.095825	0.0
$g09$	680.630	680.630	680.656	680.641	680.763	0.034
$g10$	7049.25	7054.316	7559.192	7372.613	8835.655	530.0
$g11$	0.75	0.75	0.75	0.75	0.75	0.0
$g12$	1.0	1.0	1.0	1.0	1.0	0.0
$g13$	0.053950	0.053957	0.067543	0.057006	0.216915	0.031

Cuadro 6.6: Resultados de las jerarquías estocásticas [34].

Problema	*Optimo*	*Mejor*	*Media*	*Mediana*	*Peor*	*Desv.Estandar*
$g01$	-15.0	-15.0	-15.0	-15.0	-15.0	0.0
$g02$	0.803619	0.765352	0.681926	0.676895	0.588115	0.052542
$g03$	1.0	0.997626	0.866954	0.891670	0.548497	0.109395
$g04$	-30665.539	-30665.528	-30634.787	-30648.761	-30532.113	36.100718
$g05$	5126.498	5126.484	5214.577	5146.832	5976.824	185.75843
$g06$	-6961.814	-6961.814	-6961.813	-6961.813	-6961.808	0.001009
$g07$	24.306	24.937	26.081	25.849	29.110	1.089143
$g08$	0.095825	0.095825	0.095825	0.095825	0.095825	0.0
$g09$	680.630	680.895	681.237	681.203	282.482	0.315588
$g10$	7049.25	7050.647	7193.793	7119.661	7873.170	208.04903
$g11$	0.75	0.75	0.75	0.75	0.75	0.0
$g12$	1.0	1.0	1.0	1.0	1.0	0.0
$g13$	0.053950	0.05394	0.11452	0.05422	0.45427	0.119805

Cuadro 6.7: Resultados de EAA.

otro que utilice algún esquema basado en un sistema inmune. En su tesis doctoral Ricardo Paramont [22] propone tres algoritmos basados en esquemas de sistemas inmunes artificiales. De los tres, el que tiene mejor comportamiento es el algoritmo evolutivo basado en la evolución de anticuerpos y autoanticuerpos (EAA). Con autoanticuerpos, Paramont se refiere a los anticuerpos que están fuera de la región factible. Usa representación binaria tradicional y básicamente lo que realiza es un algoritmo genético que simula con sus operaciones de mutación y cruza un sistema inmune artificial. Es importante mencionar que la cruza entre anticuerpos existe en su esquema. Además, utiliza una función de penalización para manejar las restricciones.

Los parámetros que utiliza además de los que se emplean en nuestra propuesta son: Número máximo y mínimo de descendientes de cada anticuerpo. Valor mínimo y máximo para el porcentaje de mutación. Valor mínimo y máximo para el porcentaje de cruza. Un valor de umbral para calcular un porcentaje que permite saber si a un individuo se le aplica la mutación y cruza.

Los resultados obtenidos por el EAA se presentan en la tabla 6.7. En ella se muestran treinta ejecuciones realizadas (cada una) con 500,000 evaluaciones de la función objetivo.

Función	K-S Stadistic	Función	K-S Stadistic
$g01$	1.0000	$g08$	0.5059
$g02$	0.0645	$g09$	0.2276
$g03$	0.1584	$g10$	0.1336
$g04$	0.2085	$g11$	0.3956
$g05$	0.2086	$g12$	0.8091
$g06$	0.3690	$g13$	0.2886
$g07$	0.1173		

Cuadro 6.8: Resultados de la prueba Kolmogorov-Smirnov.

Función	Intervalo de confianza	Función	Intervalo de confianza
$g01$	$-15.0 - -15.0$	$g08$	$0.095825 - 0.095825$
$g02$	$0.745731 - 0.763389$	$g09$	$681.107 - 681.429$
$g03$	$0.999405 - 0.999624$	$g10$	$7647.830486 - 7950.428612$
$g04$	$-30663.9 - -30662.8$	$g11$	$0.750393 - 0.753974$
$g05$	$5145.03 - 5169.24$	$g12$	$1.0 - 1.0$
$g06$	$-6961.81 - -6961.81$	$g13$	$0.21454 - 0.35804$
$g07$	$28.692191 - 30.014820$		

Cuadro 6.9: Intervalos de confianza encontrados con bootstrap.

6.3. Análisis de resultados

Nuevamente realizamos las pruebas de Kolmogorov-Smirnov y bootstrap (o en su caso utilizamos la ecuación ya definida) sus resultados se muestran en la tablas 6.8 y 6.9, respectivamente.

En este caso los valores de K-S Stadistic son bastante bajos, pero una vez más utilizando el mismo valor crítico que se presentó en el capítulo anterior (0.1922), tenemos que g02, g03, g07 y g10 muestran una distribución muy parecida a la normal, sobre todo g02. Y sus histogramas lo confirman, véase las figuras 6.1, 6.2, 6.3, 6.4. En estos casos nuevamente se utilizó la ecuación presentada en el capítulo anterior para calcular los intervalos de confianza.

Los intervalos son lo suficientemente pequeños en la mayoría de los casos para

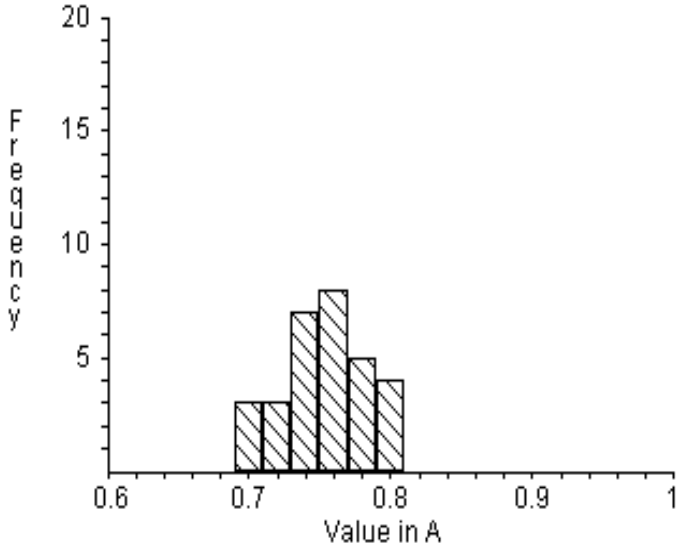

Figura 6.1: Histograma para g02

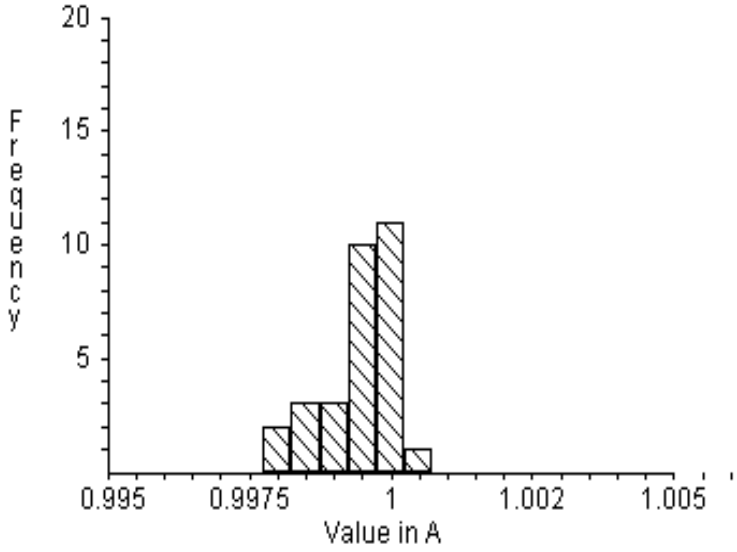

Figura 6.2: Histograma para g03

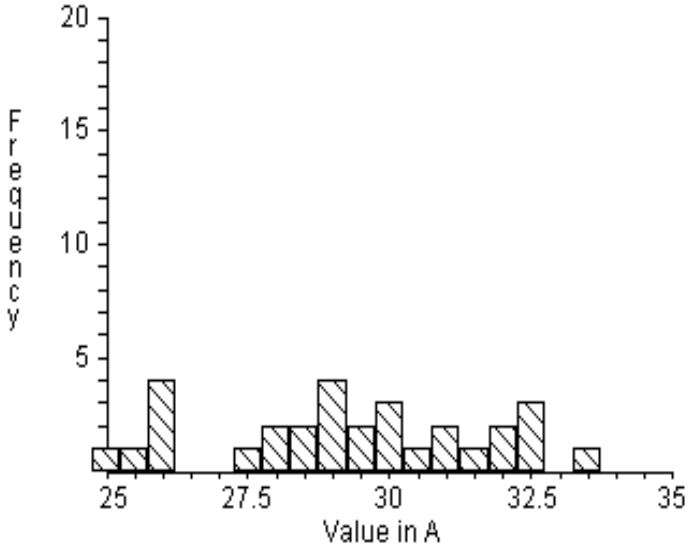

Figura 6.3: Histograma para g07

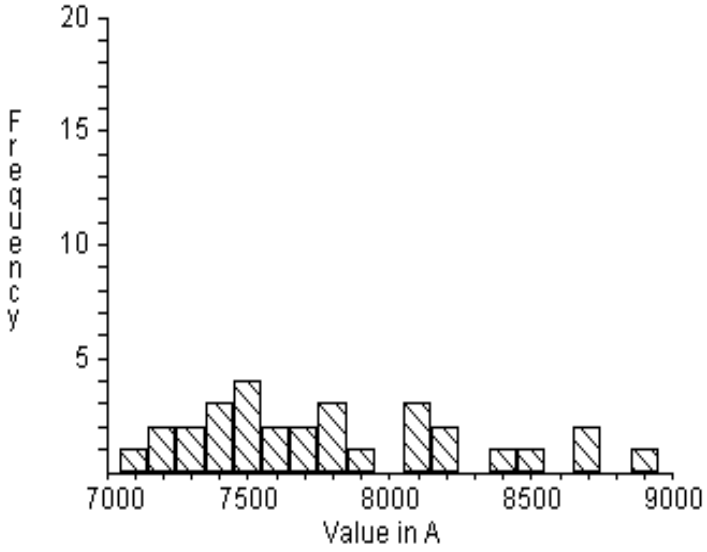

Figura 6.4: Histograma para g10

considerar a cada una de las muestras representativas, excepto en el caso de g10. Este problema representa un alta desviación estándar pero no es solo en nuestro caso sino en el de todos los algoritmo que se presentan en esta tesis y contra los que se compara.

Es muy importante mencionar que estamos obteniendo buenos resultados utilizando un número de evaluaciones muy parecido al que se reporta con jerarquías estocásticas y mucho menor al adoptado para el EAA. Obtenemos mejores resultados que los presentados por este último algoritmo en g02, g04 y g05, aunque somos superados en g10 y g13. En el caso de la comparación con jerarquías estocásticas podemos decir que los resultados de nuestra propuesta son competitivos aunque es claro que tenemos en algunos casos resultados inferiores. Esto resulta significativo si consideramos que las estrategias evolutivas suelen ser consideradas un mejor motor de búsqueda que el sistema inmune artificial en su esquema de selección clonal. Sin embargo, el hecho de usar representaciones distintas (las jerarquías estocásticas usan codificación real) dificulta la realización de una comparación justa. Por ello es necesario también hacer una versión con representación real de la selección clonal y ver si ésta sigue siendo competitiva. Se hace notar, sin embargo, que esa versión será susceptible al operador de mutación que se adopte.

Capítulo 7

Conclusiones y trabajo futuro

El objetivo principal de la tesis fue desarrollar un algoritmo basado en algún esquema del sistema inmune artificial que resultase competitivo con respecto a esquemas representativos del estado del arte, en problemas de optimización global con restricciones.

Como se observó durante el desarrollo de la tesis el objetivo se alcanzó. Para llevar a cabo dicho objetivo fue necesario, en primera instancia elegir el esquema basado en algún sistema inmune. Después, debimos probar las capacidades de dicho esquema en problemas sin restricciones para saber si se contaba con un buen motor de búsqueda.

Al principio no se tuvieron buenos resultados al implementar el esquema elegido. Al realizar un análisis del funcionamiento de la propuesta y de los resultados obtenidos, se llegó a la conclusión de que cambiando la representación binaria tradicional que adoptamos originalmente por una basada en códigos de Gray, nuestros resultados mejoraban significativamente.

Entonces, se realizó un estudio experimental que demostró que el algoritmo podía resolver un conjunto de funciones de prueba utilizando una menor cantidad de evaluaciones de la función objetivo que una propuesta cuya mayor virtud era precisamente esa. Esto resultó significativo porque pudimos entonces verificar que contábamos con un motor de búsqueda con buenas capacidades dado que las funciones sin restricciones eran de muy distintas características y nuestro algoritmo las resolvía con relativamente pocas evaluaciones de cada función.

Posteriormente, al saber que teníamos un buen motor de búsqueda el trabajo se enfocó a alcanzar el objetivo principal de la tesis. Nuevamente se resolvieron funciones de prueba, en este caso trece, pero ahora con restricciones. Los resultados obtenidos resultaron satisfactorios. Aún en los casos en que no se alcanzó el óptimo (o mejor valor conocido) se obtuvieron al menos buenas aproximaciones con un número de

evaluaciones igual al de las jerarquías estocásticas y mucho menor al de el EAA. Al obtener resultados al menos tan buenos como el EAA y en muchos casos mejores, con un número significativamente menor de evaluaciones de la función objetivo, claramente se puede notar que nuestra propuesta es mejor. Sin embargo, debe hacerse notar que nuestros resultados en (algunos casos) no son tan buenos como los obtenidos por las jerarquías estocásticas pero no dejan de ser competitivos. Por otro lado, hay que resaltar que se cuenta con una técnica muy sencilla de implementar que cuenta con la mutación como operador principal de búsqueda, que a su vez también es muy simple. Lo importante aquí fue entonces demostrar que una técnica sencilla puede resolver problemas complejos de optimización con restricciones.

Durante muchos años se han desarrollado diversos esquemas basados en algoritmos genéticos para resolver problemas con restricciones, aunque muchos de ellos han resultado útiles en problemas del mundo real, su desempeño en funciones de prueba estándar ha sido, tradicionalmente, malo. En nuestro caso, utilizando los mismos operadores que en un algoritmo genético (pero utilizando la mutación inversamente proporcional a la afinidad) pudimos obtener resultados bastante competitivos. Estos resultados son un claro indicativo de que en optimización global el poder de la mutación es de gran relevancia, a tal grado de llegar a prescindirse de la cruza que es un operador tan común en los algoritmos genéticos.

Lo dicho anteriormente nos puede dar pauta para establecer el trabajo futuro que permita mejorar el algoritmo. En primer lugar se deben probar otros tipos de mutación para representación binaria, aunque esto ofrece la limitante de que hay muy pocos métodos de mutación para dicha representación. El siguiente paso natural sería intentar con una representación distinta a la binaria. En la representación con números reales en algoritmos evolutivos se cuenta con una gran cantidad de opciones para llevar a cabo la mutación, por lo que un estudio en esta representación podría aportar ideas para mejorar el comportamiento del algoritmo.

Finalmente, se podría intentar resolver más funciones con restricciones, sobre todo las que tienen restricciones de igualdad o aquéllas con espacios de búsqueda muy pequeños que es donde más dificultades tiene el algoritmo. Con esto se podrían descubrir algunas limitaciones del algoritmo.

Apéndice A

Funciones de prueba

A.1. Funciones sin restricciones

Las veintitrés funciones de prueba adoptadas se recopilan en [39] y se reproducen a continuación:

A.1.1. Modelo de esfera

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

$$-100 \leq x_i \leq 100$$
$$min(f_1) = f_1(0, ..., 0) = 0$$

A.1.2. Problema de Schwefel 2.22

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

$$-10 \leq x_i \leq 10$$
$$min(f_2) = f_2(0, ..., 0) = 0$$

A.1.3. Problema de Schwefel 1.2

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

$$-100 \leq x_i \leq 100$$
$$min(f_3) = f_3(0, ..., 0) = 0$$

A.1.4. Problema de Schwefel 2.21

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

$$-100 \leq x_i \leq 100$$
$$min(f_4) = f_4(0, ..., 0) = 0$$

A.1.5. Función generalizada de Rosenbrock

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

$$-30 \leq x_i \leq 30$$
$$min(f_5) = f_5(1, ..., 1) = 0$$

A.1.6. Función de paso

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

$$-100 \leq x_i \leq 100$$
$$min(f_6) = f_6(0, ..., 0) = 0$$

A.1.7. Función cuadrática con ruido

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

$$-1.28 \leq x_i \leq 1.28$$
$$min(f_7) = f_7(0,...,0) = 0$$

A.1.8. Problema generalizado de Schwefel 2.26

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

$$-500 \leq x_i \leq 500$$
$$min(f_8) = f_8(420.9687,...,420.9687) = -12569.5$$

A.1.9. Función generalizada de Rastrigin

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

$$-5.12 \leq x_i \leq 5.12$$
$$min(f) = f(0,...,0) = 0$$

A.1.10. Función de Ackley

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

$$-32 \leq x_i \leq 32$$
$$min(f_{10}) = f_{10}(0,...,0) = 0$$

A.1.11. Función generalizada de Griewank

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

$$-600 \leq x_i \leq 600$$
$$min(f_{11}) = f_{11}(0, ..., 0) = 0$$

A.1.12. Funciones generalizadas y penalizadas

$$f_{12}(x) = \frac{\pi}{30}\left\{10sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2[1 + 10sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} +$$

$$\sum_{i=1}^{30} u(x_i, 10, 100, 4)$$

$$-50 \leq x_i \leq 50$$
$$min(f_{12}) = f_{12}(1, ..., 1) = 0$$

$$f_{13}(x) = 0.1\left\{sin^2(\pi 3x_1) + \sum_{i=1}^{29}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + sin^2(2\pi x_{30})]\right\} +$$

$$\sum_{i=1}^{30} u(x_i, 5, 100, 4)$$

$$-50 \leq x_i \leq 50$$
$$min(f_{13}) = f_{13}(1, ..., 1) = 0$$

donde:

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

A.1.13. Función de Schekel

$$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$$

$$-65.536 \leq x_i \leq 65.536$$
$$min(f_{14}) = f_{14}(-32, -32) \approx 1$$

donde:

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & ... & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & ... & 32 & 32 & 32 \end{pmatrix}$$

A.1.14. Función de Kowalik

$$f_{15}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$$

$$-5 \leq x_i \leq 5$$
$$min(f_{15}) \approx f_{15}(0.1928, 0.1908, 0.1231, 0.1358) \approx 0.0003075$$

Haciéndose uso de la tabla siguiente:

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

A.1.15. Función Espalda del Camello

$$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

$$-5 \leq x_i \leq 5$$
$$min(f_{16}) = f_{16}(0.08983, -0.7126) = -1.0316285 \text{ y}$$
$$min(f_{16}) = f_{16}(-0.08983, 0.7126) = -1.0316285$$

A.1.16. Función Branin

$$f_{17}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)cosx_1 + 10$$

$$-5 \leq x_1 \leq 10 \quad 0 \leq x_2 \leq 15$$
$$min(f_{17}) = f_{17}(-3.142, 12.275) = 0.398,$$
$$min(f_{17}) = f_{17}(3.142, 2.275) = 0.398 \text{ y}$$
$$min(f_{17}) = f_{17}(9.425, 2.425) = 0.398$$

A.1.17. Función Goldstein-Price

$$f_{18}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]\times$$

$$[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_1^2)]$$

$$-2 \leq x_i \leq 2$$
$$min(f_{18}) = f_{18}(0, -1) = 3$$

A.1.18. Familia de funciones de Hartman

$$f(x) = -\sum_{i=1}^{4} c_i exp\left[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right]$$

$$0 \leq x_j \leq 1$$

Con $n = 3$ y 6 para $f_{19}(x)$ y $f_{20}(x)$, respectivamente.

$$min(f_{19}) = f_{19}(0.114, 0.556, 0.856) = -3.86$$
$$min(f_{20}) = f_{20}(0.201, 0.150, 0.477, 0.275, 0.311, 0.657) = -3.32$$

Haciendo uso de las tablas siguientes:

i	$a_{ij}, j = 1, 2, 3$			c_i	$p_{ij}, j = 1, 2, 3$		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.03815	0.5743	0.8828

i	$a_{ij}, j = 1, ..., 6$						c_i	$p_{ij}, j = 1, ..., 6$					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

A.1.19. Familia de funciones de Shekel

$$f(x) = -\sum_{i=1}^{m}[(x - a_i)(x - a_i)^T + c_i]^{-1}$$

$$0 \leq x_j \leq 10$$

Con $m = 5, 7$ y 10 para $f_{21}(x), f_{22}(x)$ y $f_{23}(x)$, respectivamente.

$$min(f_{21}) = min(f_{22}) = min(f_{23}) = f(x_{local_opt}) \approx \frac{1}{c_i} \text{ para } 1 \leq i \leq m$$

donde:

$$x_{local_opt} \approx a_{ij}$$

Haciendo uso de la tabla siguiente:

i	$a_{ij}, j = 1, ..., 4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

A.2. Funciones con restricciones

Estas funciones son descritas en [30]:

A.2.1. g01

Minimizar:

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

sujeto a:

$$\begin{aligned} g(x)_1 &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\ g(x)_2 &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\ g(x)_3 &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\ g(x)_4 &= -8x_1 + x_{10} \leq 0 \\ g(x)_5 &= -8x_2 + x_{11} \leq 0 \\ g(x)_6 &= -8x_3 + x_{12} \leq 0 \\ g(x)_7 &= -2x_4 - x_5 + x_{10} \leq 0 \\ g(x)_8 &= -2x_6 - x_7 + x_{11} \leq 0 \\ g(x)_9 &= -2x_8 - x_9 + x_{12} \leq 0 \end{aligned}$$

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

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

A.2.2. g02

Maximizar:

$$f(x) = |\frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} ix_i^2}}|$$

sujeto a:

$$g(x)_1 = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$
$$g(x)_1 = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

$n = 20$, $0 \leq x_i \leq 10 (i = 1, ..., n)$

El óptimo es deconocido la mejor aproximación encontrada es:

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

A.2.3. g03

Maximizar:

$$f(x) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$$

sujeto a:

$$h(x)_1 = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

$n = 0$, $0 \leq x_i \leq 1$

$x^* = \frac{1}{\sqrt{n}} (i = 1, ..., n)$, $max(f) = f(x^*) = 1$

A.2.4. g04

Minimizar:

$$f(x) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

sujeto a:

$$\begin{gathered}
g(x)_1 = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92.0 \leq 0 \\
g(x)_2 = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0 \\
g(x)_3 = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110.0 \leq 0 \\
g(x)_4 = 80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90.0 \leq 0 \\
g(x)_5 = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25.0 \leq 0 \\
g(x)_6 = 9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20.0 \leq 0
\end{gathered}$$

$78 \leq x_1 \leq 122$, $33 \leq x_2 \leq 45$ y $27 \leq x_i \leq 45 (i = 3, 4, 5)$

$min(f) = f(78, 33, 29.995256025682, 45, 36.775812905788) = -30665.539$

A.2.5. g05

Minimizar:

$$f(x) = 3x_1 + 0.000001x_i^3 + 2x_2 + (0.000002/3)x_2^3$$

sujeto a:

$$\begin{gathered}
g(x)_1 = -x_4 + x_3 - 0.55 \leq 0 \\
g(x)_2 = -x_3 + x_4 - 0.55 \leq 0 \\
h(x)_3 = 1000sin(-x_3 - 0.25) + 1000sin(-x_4 - 0.25) + 894.8 - x_1 = 0 \\
h(x)_4 = 1000sin(x_3 - 0.25) + 1000sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0 \\
h(x)_5 = 1000sin(x_4 - 0.25) + 1000sin(x_4 - x_3 + 0.25) + 1294.8 = 0
\end{gathered}$$

$0 \leq x_i \leq 1200 (i = 1, 2)$ y $-0.55 \leq x_i \leq 0.55 (i = 3, 4)$

$min(f) = f(679.9453, 1026.067, 0.1188764, -0.3962336) = 5126.4981$

A.2.6. g06

Minimizar:

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$$

sujeto a:

$$g(x)_1 = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0$$
$$g(x)_2 = (x_1 - 6)^2 + (x_2 - 5)^2 - 86.81 \leq 0$$

$13 \leq x_1 \leq 100$, $0 \leq x_2 \leq 100$

$min(f) = f(14.095, 0.84296) = -6961.81388$

A.2.7. g07

Minimizar:

$$f(x) = x_1^2+x_2^2+x_1x_2-14x_1-16x_2+(x_3-10)^2+4(x_4-5)^2+(x_5-3)^2+2(x_6-1)^2+5x_7^2+7(x_8-11)^2$$
$$+2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

sujeto a:

$$g(x)_1 = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0$$
$$g(x)_2 = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0$$
$$g(x)_3 = -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0$$
$$g(x)_4 = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0$$
$$g(x)_5 = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0$$
$$g(x)_6 = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0$$
$$g(x)_7 = 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0$$
$$g(x)_8 = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0$$

$-10 \leq x_i \leq 10 (i = 1, ..., 10)$

$min(f) = f(2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644$
$, 9.828726, 8.280092, 8.375927) = 24.3062091$

A.2.8. g08

Minimizar:

$$f(x) = \frac{sin^3(2\pi x_1)sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

sujeto a:

$$g(x)_1 = x_1^2 - x_2 + 1 \leq 0$$
$$g(x)_2 = 1 - x_1 + (x_2 - 4)^2 \leq 0$$

$0 \leq x_i \leq 10(i = 1, 2)$

$min(f) = f(1.2279713, 4.2453733) = 0.095825$

A.2.9. g09

Minimizar:

$$f(x) = (x_1-10)^2+5(x_2-12)^2+x_3^2+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4-4x_4x_7-10x_6-8x_7$$

sujeto a:

$$g(x)_1 = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0$$
$$g(x)_2 = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4x_5 \leq 0$$
$$g(x)_3 = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0$$
$$g(x)_4 = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0$$

$-10 \leq x_i \leq 10(i = 1, ..., 7)$

$min(f) = f(2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$
$= 680.6300573$

A.2.10. g10

Minimizar:

$$f(x) = x_1 + x_2 + x_3$$

sujeto a:

$$g(x)_1 = -1 + 0.0025(x_4 + x_6) \leq 0$$
$$g(x)_2 = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0$$
$$g(x)_3 = -1 + 0.01(x_8 - x_5) \leq 0$$
$$g(x)_4 = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.33 \leq 0$$
$$g(x)_5 = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0$$
$$g(x)_6 = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0$$

$100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000 (i = 2, 3)$ y $10 \leq x_i \leq 1000 (i = 4, ..., 8)$

$min(f) = f(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979) = 7049.3307$

A.2.11. g11

Minimizar:

$$f(x) = x_1^2 + (x_2 - 1)^2$$

sujeto a:

$$h(x)_1 = x_2 - x_1^2 = 0$$

$-1 \leq x_1 \leq 1 (i = 1, 2)$

$min(f) = f(\pm\frac{1}{\sqrt{2}}, \frac{1}{2}) = 0.75$

A.2.12. g12

Minimizar:

$$f(x) = \frac{100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2}{100}$$

sujeto a:

$$g(x) = (x_1 - p)^2 + = (x_2 - q)^2 + = (x_3 - r)^2 - 0.0625 \leq 0$$

$0 \leq x_i \leq 10 (i = 1, 2, 3)$ y $p, q, r = 1, 2, ..., 9$

$min(f) = f(5, 5, 5) = 1$

A.2.13. g13

Minimizar:

$$f(x) = e^{x_1 x_2 x_3 x_4 x_5}$$

sujeto a:

$$h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0$$
$$h_2(x) = x_2 x_2 - 5 x_4 x_5 = 0$$
$$h_3(x) = x_1^3 + x_2^3 + 1 = 0$$

$-2.3 \leq x_i \leq 2.3 (i = 1, 2)$ y $-3.2 \leq x_i \leq 3.2 (i = 3, 4, 5)$

$min(f) = f(-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645) = 0.0539498$

Bibliografía

[1] Jerzy Balicki. Multi-criterion evolutionary algorithm with model of the immune system to handle constraints for task assignments. *In Leszek Rutkowski, Jörg H. Siekmann, Ryszard Tadeusiewicz, and Lotfi A. Zadeth, editors, Artificial Intelligence and Soft Computing - ICAISC 2004, 7th International Conference, Proceedings*, pages 394–399, 2004.

[2] T. Bäck and H. P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1:1–23, 1993.

[3] H Bersini and F. J. Varela. The immune recruitment mechanism: A selective evolutionary mechanism. *Proc. of the* 4^{th} *Int. Conf. on Genetic Algorithms*, pages 520–526, 1991.

[4] Carlos A. Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, 2002.

[5] Carlos Artemio Coello and Nareli Cruz-Cortés. Hybridizing a genetic algorithm with an artificial immune system for global optimization. *Engineering Optimization*, 36(5):607–634, 2004.

[6] Paul R. Cohen. *Empirical Methods for Artificial Intelligence*. The MIT Press, 1995.

[7] David Corne, Marco Dorigo, and Fred Glover. *New ideas in optimization.* McGraw Hill Publishing Company, 1999.

[8] Nareli Cruz-Cortés and Carlos Artemio Coello. Multiobjective optimization using the clonal selection principle. *in Erick Cantú-Paz et al. (editors), Genetic and Evolutionary Computation Conference–GECCO'2003. Proceedings, Part I, Lecture Notes in Computer Science, Chicago, Illinois, USA*, 2723:158–170, 2003.

[9] Nareli Cruz-Cortés and Carlos Artemio Coello. Using artificial immune systems to solve optimization problems. *en Alwyn Barry (editor) 2003 Genetic and Evolutionary Computation Conference. Workshop Program Chicago, Illinois, USA,*, pages 312–315, 2003.

[10] Dipankar Dasgupta and Zbigniew Michalewicz (editors). *Evolutionary Algorithms in Engineering Applications*. Springer, 1997.

[11] L. Nunes de Castro and F. J. Von Zuben. AInet: An artificial immune network for data analysis. *Book Chapter in Data Mining: A Heuristic Approach. H. A. Abbass, R. A. Sarker, and C. S. Newton (eds.), Idea Group Publishing, USA, Chapter XII*, pages 231–259, 2001.

[12] L. Nunes de Castro and F. J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6(3):239–251, 2001.

[13] Leandro Nunes de Castro and Jonathan Timmis. *Artificial Immune Systems: A new Computational Intelligence Approach*. Springer, 2002.

[14] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186:311–338, 2000.

[15] B.E. Dodd and P. J. Lincoln. *Inmunología de los grupos sanguineos*. El manual Moderno S. A., 1975.

[16] Dipankar Dasgupta (editor). *Artificial immune systems and their applications*. Springer, 1998.

[17] A. E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. Springer, 2003.

[18] D. J. Farmer, N. H. Packard, and A.S. Perelson. The immune system, adaptation and machine learning. *Physica D*, 22:187–204, 1986.

[19] D. B. Fogel. *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn press, 1991.

[20] S. Forrest, S.A. Hofmeyr, and A. Somayaji. Computer immunology. *Communications of the ACM*, 40(10):88–96, 1997.

[21] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 202–212, 1994.

[22] Ricardo Paramont Hernández García. *Uso de modelos del sistema inmune para manejo de restricciones con algoritmos genéticos.* Tesis para obtener el grado de Doctor en Ciencias de la Computación, Centro de Investigación en Computación, México D.F., 2004.

[23] Fred Glover and Manuel Laguna. *Tabu search.* Kluwer Academic Publishers, 1997.

[24] P. Hajela and J. Lee. Constrained genetic search via schema adaptation: An immune network solution. *Structural Optimization*, 12(1):11–15, 1996.

[25] G.W. Hoffmann, M.W. Benson, G. M. Bree, and P.E. Kinahan. A teachable neural network based on an unorthodox neuron. *Physica D*, 22:233–246, 1986.

[26] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, 1992.

[27] Z. Ji and D. Dasgupta. Artificial immune system (AIS) research in the last five years. *Proceedings of the Congress on Evolutionary Computation Conference (CEC)*, pages 8–12, 2003.

[28] J. Kelsey and J. Timmis. Immune Inspired Somatic Contiguous Hypermutation for Function Optimisation. In E. Cantú-Paz et al, editor, *Genetic and Evolutionary Computation Conference - GECCO 2003 of Lecture Notes in Computer Science*, volume 2723, pages 207–218, Chicago, USA., 2003. Springer-Verlag.

[29] J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection.* MIT Press, 1992.

[30] S. Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrainted parameter optimization. *Evolutionary Computation*, 7(1):19–14, 1999.

[31] Ph. Letonturier. *Inmunología general.* Masson, 1981.

[32] Zbigniew Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. *In J. R. McDonnell, R. G. Reynolds and D. B. Fogel (Eds.), Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155, 1995.

[33] Singuresu S. Rao. *Engineering optimization: Theory and Practice*. Wiley Inter-Science, 1996.

[34] Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[35] H. P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, 1995.

[36] H. Sieburg, H. McCutchan, O. Clay, and L. Caballero. Simulation of HIV-infection in artificial immune systems. *Physica D*, 45:208–228, 1990.

[37] A. Törn and A. Zilinskas. Global optimization. *Lecture Notes in Computer Science*, 350, 1989.

[38] F. Jiménez J. L. Verdegay. Evolutionary techniques for constrained optimization problems. *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany*, 1999.

[39] Xin Yao, Yong Liu, Ko-Hsin Liang, and Guangming Lin. Fast evolutionary algorithms. *Communications of the ACM*, 40(10):88–96, 1997.