VNIVERSIDAD NACIONAL
AVFNOMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN INGENIERÍA

Metodología para el Diseño de Aplicaciones con Sistemas Clasificadores Genéticos

T E S I S

QUE PARA OBTENER EL GRADO DE:

DOCTOR EN INGENIERIA

P R E S E N T A:

FELIPE PADILLA DIAZ

DIRECTOR DE LA TESIS:
DR. STANISLAW RACZYNSKI GAWN.

MÉXICO, D.F. **MAYO 2003.**

FACULTAD DE INGENIERIA

U N A M

FELIPE PADILLA DÍAZ

Doctorado en Ingeniería

DEPFI-UNAM

N° de cuenta: 98856646; N° de expediente: 11981458

Registro CONACYT: 57904

Director de tesis: **Dr. Stanislaw Raczynski Gawn**

Comité de asesores:
Dr. Francisco Cervantes Pérez.
Dr. Jesús Savage Carmona.
Dra. Katya Rodriguez Vázquez.
Dr. Marcelo Mejia Olvera.
Dr. Alfredo Weitzenfeld Ridel
Dr. Osvaldo Cairo Battistutti

Prefacio

Las primeras investigaciones con las que comencé eran en el campo de los Sistemas Expertos y Redes Neuronales en la Universidad Autónoma de Aguascalientes a partir de 1996 y fue entonces que participé en el congreso internacional de inteligencia artificial con una ponencia relacionada con Sistemas Expertos en el ICIMAF de la Habana, Cuba. En este congreso fue para mi gran sorpresa de que aproximadamente el 70% de los artículos presentados fueron acerca de la computación evolutiva ya que para esos años era un tema de grandes esperanzas en resolver problemas principalmente de optimización combinatoria, mi experiencia en este tema era que solamente los conocía por nombre mas no como y cuando podía usarlos y fue como comencé a investigar sobre estos y ver la posibilidad de aplicarlos a diferentes problemas que veníamos resolviendo con otras técnicas mas estudiadas como la de recocido simulado, tabú, escalando la colina, redes neuronales, etc. y lo primero que hice fue preguntarle a expertos que amablemente ponían su dirección electrónica en las páginas de internet para ver por donde empezar y fue así como me inicié el contacto con investigadores de renombre como son (Dr. Marcelo Mejía Olvera, Dr. Francisco Cervantes Pérez, Dr. David Goldberg, Dr. Carlos Coello Coello, Dr. Erick Cantú, Dr. Stanislaw. Raczynsky, etc.) y gracias al apoyo de todos ellos fue como incursioné en este terreno novedoso y atractivo para resolver problemas con comportamiento duro y complejo de la computación.
Lo primero que empecé a desarrollar fue una aplicación que el Dr. David Goldberg en su libro [56] maneja con gran éxito y está relacionado a problemas de optimización combinatoria y una vez estudiando la forma de trabajar de esta técnica quise emplearlo a un problema robusto y mi primer incógnita fue la de si existía un modelo o metodología para el desarrollo de este tipo de aplicaciones y después de hacer una investigación de campo con diferentes maestros, doctores e investigadores que tenían ya trabajos presentados en congresos en relación con este tópico me encontré con la respuesta generalizada de que no existía una forma o metodología para este tipo de desarrollo y aunque si lo veían viable la verdad era que estos mismos desarrollos se hacían con la metodología que propusiera el mismo desarrollador y fue así como me interese en crear una metodología que paso a paso nos fuera diciendo como llevar a cabo una aplicación con AGs, una vez que empecé a trabajar con esta y que según yo tenía terminada la misma se me presentó la oportunidad de ser parte del grupo de ILLIGAL (Illinois Genetic ALgorithms) liderado por el Dr. David Goldberg quién después de revisar mi avance de tesis me sugirió enfocar la metodología hacia la parte de los sistemas clasificadores por la sencilla razón de que al realizarlo con clasificadores estaría resolviendo a la vez la parte de genéticos y resolvía un problema aun mayor que era el de aprendizaje y predicción de nuevo conocimiento a través de los algoritmos genéticos.

Es importante recalcar que al ingreso al doctorado perseguía varios objetivos esenciales que se fueron cumpliendo cada uno de ellos de acuerdo con el avance del trabajo, los objetivos fueron:

1.- Participar en la elaboración de artículos para eventos internacionales de la inteligencia artificial. Esto se fue cumpliendo con la publicación de 18 artículos en diferentes congresos tanto nacionales como internacionales y aunque el área en que trabaje en la tesis es relativamente nueva y aun no se ha desarrollado completamente, esto lo hizo mas interesante y a la vez complicado de llevar a la práctica en estos cinco años del doctorado, pues mientras que las referencias principales de esta tesis son de solamente como máximo de diez años, en otros campos se tienen referencias hasta de 30 años.

2.- Participar en estancia en el extranjero con investigadores de gran prestigio internacional. Nunca me esperé realizar una estancia de por lo menos de un año con un grupo tan fuerte como lo es el grupo de ILLIGAL (ILLInois Genetic ALgorithm) de la Universidad de Illinois en Urbana-Champaign con el Dr. David Goldberg y esta se debió sin duda al soporte de mis asesores en México, quienes a través de sus contactos y consejos fue que conocí al Dr. Erick Cantú (exalumno del Dr. Francisco Cervantes y del Dr. Marcelo Mejía en el ITAM) que gracias a sus comentarios fui aceptado por el Dr. Goldberg.

Agradecimientos.

Sin duda son innumerables a las personas que hay que agradecerle por el apoyo que recibí en esta etapa de mi vida y empezaría haciéndolo con mi familia que estuvo al pendiente de cada avance o escalón que daba en estos cinco años que invertí en obtener este trabajo doctoral, ellos son:

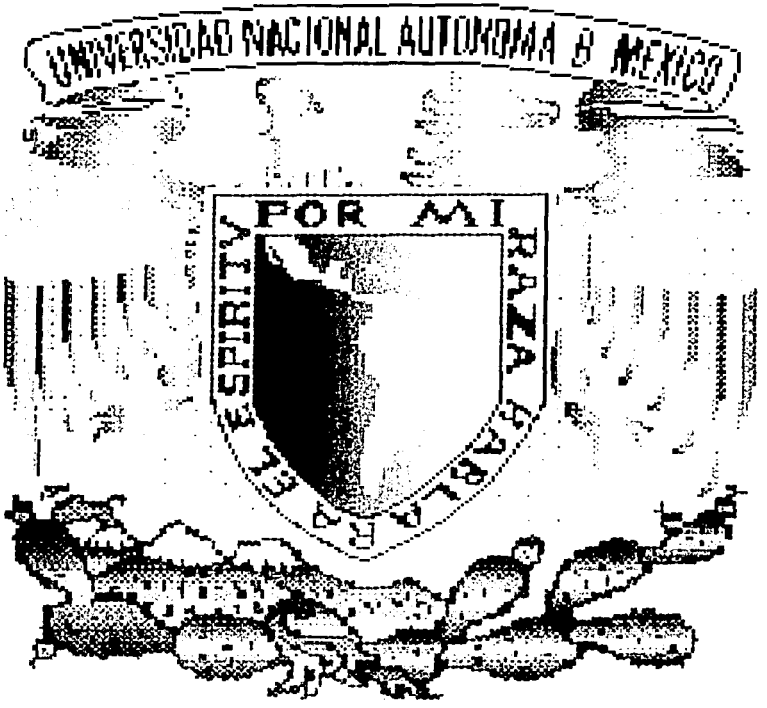
Mi esposa Lety ,
Mi hijo Ludwig y
Mi hija Marlet,
Mis papás Salvador y Aurora,
Mis hermanos y hermanas.

Agradecer de manera especial la paciencia y asesorías de mis asesores del doctorado:

Dr. Francisco Cervantes Pérez (ITAM).
Dr. Stanislaw Raczynski Gawn (UP).
Dr. Marcelo Mejia Olvera (ITAM).
Dr. Alfredo Weitzenfeld Ridel (ITAM).
Dr. Jesús Savage Carmona (UNAM).
Dra. Katya Rodríguez Vázquez (UNAM).
Dr. Osvaldo Cairo Battistutti (ITAM) y al
Dr. David Goldberg (IILIGAL, USA)

A mis compañeros de trabajo, a mi institución (UAA) que siempre me ha apoyado en el crecimiento profesional, al Conacyt y PROMEP por la confianza por contar con el apoyo económico en los primeros años de estudios y a mis compañeros de estos cinco años que estuvimos trabajando en equipo para facilitar el trabajo y me refiero a Francisco Alvarez Rodríguez y a Manuel Mora Tavares.

INDICE



CAPÍTULO I
INTRODUCCIÓN Y CONTENIDO DE LA TESIS.

1.1 Introducción.

La capacidad de aprender es una de las características centrales de la inteligencia, y esto hace que el aprendizaje haya formado parte de la investigación en Inteligencia Artificial desde prácticamente los orígenes de la misma.

Un componente incuestionable de la inteligencia humana es la habilidad de aprender de los errores, adaptarse a un ambiente continuamente cambiante, y generalizando este mismo con la experiencia en terrenos desconocidos. Una meta desafiante para la Inteligencia Artificial (IA) es entender y reproducir esta capacidad innata de adaptación para desarrollar los sistemas artificiales capaces de operar autónomamente en el complejo medio ambiente. Por otro lado, la adaptación y aprendizaje en los dominios complejos son procesos caros que requieren una cantidad grande de recursos computacionales. La disponibilidad de arquitecturas paralelas y los ambientes de la programación indica el paralelismo como un acercamiento viable para acelerar el aprendizaje y más aún si estos ambientes los dotamos de técnicas que aprovecharan estos mismos.

Las aplicaciones del aprendizaje son hoy en día muy numerosas. En concreto, es importante destacar la gran aportación que los algoritmos de aprendizaje tienen en el desarrollo práctico de algunos sistemas basados en sistemas evolutivos, ya que como es conocido, las técnicas enumerativas son difíciles de aplicar en muchos casos, el aprendizaje permite, en estos casos, una extracción del conocimiento mucho mas eficaz. Este conocimiento podría ser posteriormente validado por un experto.

La utilización de los Sistemas Clasificadores (SC), ha demostrado ser una técnica extraordinariamente útil en la descripción y modelación de muchos sistemas. La utilización de dicha técnica ha permitido la modelización cualitativa de sistemas. Esta misma es mucho más comprensible y simple que otras alternativas de la misma inteligencia artificial como son las redes neuronales y la lógica difusa. Sin embargo, el éxito comercial de muchos sistemas basados en este tipo de técnicas mediante un Sistema Clasificador hace necesario

el desarrollo de nuevos modelos de aprendizaje que permitan adquirir tal conocimiento y mas aun se hace de vital importancia de contar con una metodología para el desarrollo que nos garantice planear y diseñar la aplicación en forma y tiempo.

Esta tesis hace énfasis en el desarrollo de una metodología para aplicaciones donde intervienen sistemas clasificadores haciendo mención especial al tipo de aprendizaje que se desarrollará a través del componente descubridor de reglas por medio del Algoritmo Genético y del componente de Asignación de Crédito.

Los sistemas clasificadores, son sistemas de aprendizaje adaptables que se aprovechan de los algoritmos genéticos para aprender las estrategias del comportamiento. El modelo aprende en paralelo y coopera intercambiando la información, es decir material genético.

1.2 Sistemas clasificadores.

Una variedad de aplicaciones de diferentes tópicos, como los robots autónomos, teoría de juegos y aplicaciones robustas en tiempo real son muy difíciles de resolver a través de los métodos de inteligencia artificial clásicos. Por ejemplo, un vehículo autónomo que opera en el mundo real. Este tiene que persuadir múltiples obstáculos, posiblemente objetivos contradictorios, como moverse a un destino tan rápido como le sea posible, evitando los obstáculos y encontrar un lugar para recargar sus baterías cuando estas se encuentren bajas. El mundo real es típicamente un ambiente dinámico, por consiguiente el sensor de la entrada del sistema tiene que tratar continuamente con los nuevos escenarios. Además, el sistema puede percibir el ambiente en forma estocástica, debido a las interfaces ruidosas y/o a la presencia de otras entidades humano o artificial cuyo comportamiento es imprescindible. Operar autónomamente en tal ambiente, el sistema debe ser robusto y flexible con respecto a los cambios. Bajo tales condiciones, la codificación del comportamiento del sistema puede ser muy difícil, si no impracticable, porque este requiere un conocimiento detallado del problema y del algoritmo para resolverlo. Esta falta de conocimiento significa que es también difícil de aplicar métodos simbólicos de inteligencia artificial por razonamiento y deducción tales como los mismos Sistemas Expertos. Estos

métodos confían en una cantidad significante de tareas específicas de conocimiento, que tiene que ser adquirido y codificado en el sistema antes de que pueda explotarse para resolver la tarea. Además, el conocimiento requerido no puede ser fácilmente accesible o puede ser demasiado caro obtenerlo. Este problema es bien conocido en inteligencia artificial como el cuello de botella de la adquisición de conocimiento.

Cuando el conocimiento necesario no está disponible al diseñador, el propio sistema, tiene que adquirirlo del ambiente que opera en el. El sistema tiene que actuar directamente con el entorno en forma reciproca. Tiene que aprovecharse de su experiencia para construir un modelo interno de la tarea que sea capaz de generar predicciones y realizar las acciones apropiadas que iguale cuando el conocimiento de la tarea está incompleto o incorrecto. Asi mismo, el sistema tiene que ser capaz de afinar este modelo cuando vaya aumentando su información del problema en forma automática.

Los sistemas clasificadores son un paradigma de Aprendizaje de Máquina y fue inicialmente introducido por Holland a finales de los 70s, fueron diseñados explícitamente para satisfacer y resolver los hechos mencionados anteriormente, empleando para ello: aprendizaje por interacción, robustez y generalización del conocimiento.

El resto de esta sección da una apreciación global de los SCs y de su relación con otros paradigmas de aprendizaje.

1.3 Aprendizaje por Refuerzo.

El Aprendizaje por Refuerzo (AR) está aprendiendo por medio de premios o castigos, en la ausencia de instrucciones explícitas en cómo resolver el problema. El aprendizaje por refuerzo se relaciona con el aprendizaje que se usa en Psicología de Prueba y Error, dónde el comportamiento es escogido según sus consecuencias en producir el refuerzo. Un Sistema de Aprendizaje se define "como un agente activo que actúa recíprocamente con el ambiente". El agente puede percibir su ambiente y puede realizar acciones que modifican este ambiente. El aprendizaje por refuerzo asume que la única reacción del ambiente es un

signo escalar llamado el refuerzo, que evalúa la acción tomada por el agente. El objetivo del agente es realizar las acciones que aumenten al máximo el refuerzo en el transcurso del tiempo.

La reacción de la Evaluación es el hecho principal que distingue el aprendizaje por refuerzo del Aprendizaje Dirigido o Supervisado (AS), un paradigma alternativo que caracteriza una gran parte de investigaciones actuales en el Aprendizaje de la Máquina, del reconocimiento de patrones con redes neuronales. El AS asume la existencia de un maestro externo que conoce la "mejor" acción para realizarse en cualquier situación dada. Cualquier acción que el agente prueba, el maestro indica directamente lo que es o debe ser correcto, o, equivalentemente le proporciona asi mismo el error de la acción (Fig. 1.1). Este tipo de reacción o regeneración es instructiva, porque esta lleva información explícita de cómo modificar el comportamiento del agente. Desde un gradiente de error como una función de acciones, el agente puede minimizar este mismo a través del descenso del mismo gradiente. Esto es exactamente lo que pasa con el algoritmo de propagación hacia atrás (back-propagation usado comúnmente en Redes Neuronales), una técnica ampliamente usada en los algoritmos de AS y que fue ampliamente dada a conocer con el perceptrón multi-nivel de las redes neuronales. Mientras una sola instrucción puede usarse para mejorar la estrategia de selección de la acción, un solo refuerzo no significa nada por si mismo. Primero, este no dice nada en absoluto de cómo modificar las acciones, porque el gradiente del refuerzo es una función de acciones no conocida. Además, el agente no puede inferir de un solo refuerzo si este es el mejor alcanzable en la situación actual o si una acción diferente resultaría mejor. La única manera de obtener esta información es la exploración activa, o ensayo de prueba y error: el agente tiene que interactuar directamente con la tarea del entorno y observar los efectos de sus acciones. Comparando las evaluaciones obtenidas durante la exploración, el agente puede relacionar estos cambios a las acciones tomadas, y así estima la información del gradiente que necesitó para mejorar su comportamiento.

Un rasgo distintivo del aprendizaje por refuerzo es el refuerzo por retardo, porque una acción afecta no sólo el refuerzo inmediato sino también los eventos futuros y, por consiguiente, los refuerzos futuros. A veces, aumentando al máximo el refuerzo inmediato

no puede aumentar al máximo un criterio de la actuación a largo plazo. Además, en varias tareas, el refuerzo es esparcido, por ejemplo, este viene solamente al final de una larga secuencia de acciones. En un juego típico de mesa, el último resultado puede ser el único refuerzo, positivo en el caso de una victoria o negativo para una pérdida. Análogamente, un robot sólo puede conseguir un refuerzo cuando alcanza un objetivo o cuando esta ha completado una tarea dada.

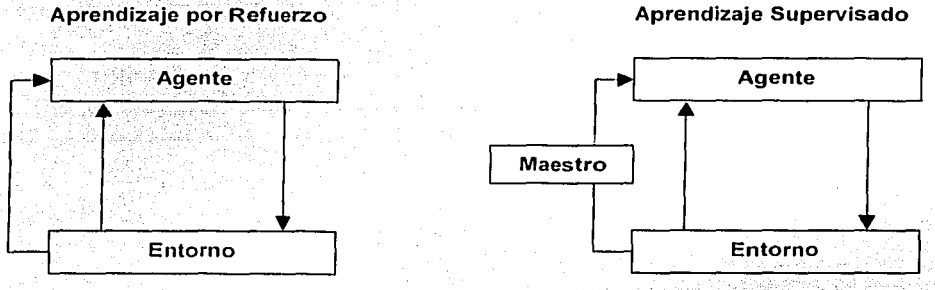


Figura 1.1: Aprendizaje por Refuerzo y Aprendizaje Supervisado.

Normalmente, el agente tiene que aumentar una función de refuerzos futuros, el retorno, cuya definición depende de la tarea. Si la interacción del refuerzo-agente se ve como una secuencia de tareas, el retorno simplemente podría ser una suma de refuerzos.

Si la interacción es continua, los refuerzos futuros normalmente son descontados por un parámetro que determina hasta qué punto deben tenerse en cuenta términos consecuentes para decidir el comportamiento actual.

Cualquier agente del aprendizaje por refuerzo tiene que tratar con un problema desafiante: encontrando un compromiso entre la tendencia a aprovecharse de lo que ya sabe y la necesidad de ganar continuamente la información del entorno. Para aumentar al máximo la recompensa, el agente debe escoger la acción que ha demostrado ser el más eficaz en el refuerzo, según la experiencia del pasado. Por otro lado, para descubrir qué acción

realmente es la mejor. el agente tiene que probar acciones que han sido previamente abandonadas. Este problema es conocido como el dilema EE (Exploración/Explotación), que es fundamental en el aprendizaje por refuerzo y, en general, en cualquier forma de ensayo de aprendizaje de prueba y error. Si el agente sólo explota o aprovecha el conocimiento, es decir este siempre escoge determinísticamente la mejor opción que ha experimentado, esta pierde la oportunidad de probar otras probablemente mejores opciones. Por otro lado, si el agente sólo explora, es decir este siempre escoge la acción en forma aleatoria, este acumulará muchísima información del entorno pero este nunca aprovechará las tareas de maximización de refuerzos (Fig. 1.2). Cualquier método de aprendizaje por refuerzo debe incluir una estrategia para equilibrar la exploración y explotación. Típicamente, la exploración debe ser más intensiva en el aprendizaje más temprano y cuando hay información pequeña sobre el ambiente. Cuanta más información se pone disponible, la exploración debe reducirse para obligarle al agente que aproveche lo que tiene el aprendizaje progresivamente.

Figura 1.2: Punto de equilibrio de la *Exploración y Explotación.*

Para obtener información confiable, la exploración debe ser más intensiva en los ambientes estocásticos que en los ambientes determinísticos. Si el ambiente es dinámico, el nivel de exploración siempre debe permanecer alto, o podría variarse según la actuación actual del agente.

Una variedad de técnicas se han desarrollado para dirigir problemas de Aprendizaje por Refuerzo. Los SCs [76][10] fueron diseñados para dirigir el refuerzo retardado y resolver problemas que requieren memoria. Ellos provinieron de la representación simbólica de la

ciencia cognoscitiva, y se aumentó con los paradigmas inspirados biológicamente para obtener la capacidad de adaptación.

Un nuevo campo de aprendizaje por refuerzo empezó en la década de los 80s, reuniendo ideas de la psicología, aprendizaje animal, teoría de control y estadística. El Crítico Heurístico adaptable [121], aprendizaje por retardo-TD(λ) [122], y Q-learning [129], son entre los algoritmos más representativos producidos por estas teorías, resumido por Sutton y Barto en su libro [123]. Aunque estos acercamientos vienen de campos diferentes (principalmente de la Programación Dinámica), ellos comparten metas y métodos y se han correlacionado a menudo integrando estas ideas de manera satisfactoria. La idea de aprender es comparando estimaciones sucesivas de la misma predicción temporal, formalizado por Sutton en [122] y estando debajo de la mayoría de los métodos de aprendizaje por refuerzo basado en la Programación Dinámica, como Q-learning, anteriormente en el juego de ajedrez propuesto por Samuel ya utilizaban estas ideas [114]. Dorigo [40] mostraba la conexión entre Q-learning y un modelo simplificado de SC. Q-learning también se ha estudiado como un método de asignación de crédito en SCs. Wilson propuso usar Q-learning en lugar del Bucket Brigade Algorithms (BBA) en su ZCS [141] y posteriormente en el modelo XCS, modelos simplificados que no usan la memoria del paso de mensajes. En el mismo periodo, otros investigadores en paralelo desarrollaron la asignación de Q-Credit independiente (QCA) [52][14], un método Q de asignación de crédito basado en aprendizaje para SCs con memoria interior.

1.4 Algoritmos genéticos en el aprendizaje de Máquina .

Debido a sus capacidades de adaptación tan poderosas, los AGs se han reconocido por mucho tiempo como algoritmos de optimización o de adaptación, pues la definición de la palabra optimizar es encontrar la mejor solución y muchas veces no sabremos si es, o no la mejor, simplemente de acuerdo a la experiencia sabremos que hemos encontrado una buena solución y probablemente no exista otra mejor.

En uno de sus últimos trabajos sobre AGs [73], Holland plantea el Aprendizaje de Máquina en forma Genética a través de la definición del lenguaje Broadcast, un Sistema de Producción que es equivalente a la Máquina de Turing y al mismo tiempo enfatiza la comunicación con el AG. Combinado con un estudio previo de procesadores de Esquemas [74], algunos años mas tarde el lenguaje Broadcast dio origen al primer SC llamado Sistema Cognoscitivo-1 (CS-1) [77].

CS-1 contenía todos los elementos de los SCs modernos:

- el concepto de interacción entre el entorno y el agente a través de los descubridores y efectores,
- un sistema de la actuación manejado sintácticamente por reglas simples,
- una asignación de crédito, y
- un AG que crea, recombina y reemplaza las reglas según su aptitud.

Los conceptos introducidos originalmente por Holland han tenido cada vez mayor auge y participación en el desarrollo de investigación básica de este campo, aunque aun son pocas las aplicaciones que compiten con otras técnicas similares.

Para mencionar algunas de las aplicaciones y trabajos relacionados con esta técnica se puede citar a los trabajos de Goldberg [66][55], Booker [9][10], Riolo [111][112], y Wilson [138][139][141][142][143].

Paralelamente Smith en la Universidad de Pittsburg dio un acercamiento algo diferente a los SCs. Esta aportación fue llamada LS-1 [118][119]. La diferencia principal entre LS-1 y CS-1 queda en la definición de los genotipos manipulada por el AG. En CS-1, un genotipo es una regla cuya aptitud refleja su utilidad. La población que evoluciona bajo el AG es la base de conocimiento de reglas que manejan la conducta del agente. Por otro lado, en LS-1 un genotipo codifica la base de conocimiento entera de reglas, es decir el comportamiento entero del problema. El modelo LS-1 no incluía ninguna asignación del crédito explícita a las reglas individuales que pertenecen al genotipo. Después, Grefenstette perfeccionó este modelo [68].

De acuerdo con De Jong [32] hay una gran variedad de aproximaciones a los sistemas de aprendizaje basadas en AGs. Las tres más importantes son las siguientes:

- Restringir los cambios estructurales a la modificación de parámetros que controlan la conducta del subsistema de tarea, y usar los AGs para desarrollar una estrategia que rápidamente localicen las combinaciones útiles de los valores de los parámetros.
- Usar AGs para cambiar las estructuras de datos complejas, como las agendas, que controlan la conducta del subsistema de tarea.
- Usar AGs para cambiar el subsistema de tarea, de modo que éste evolucione por si mismo.

Normalmente un subsistema de tarea se considera como un sistema de producción formado por un conjunto no ordenado de reglas, que representa la población de individuos que seria explotado y explorado por el subsistema de aprendizaje basado en AGs.

Hay dos modos de usar los sistemas de producción para representar al conjunto de la población:

- La Aproximación Pittsburg [118], donde cada miembro de la población representa un conjunto de reglas de producción, y por tanto, una población es un conjunto de conjuntos de reglas. Cada conjunto de reglas tiene asignado un fitness, calculado en base a una medida de rendimiento, el cual es usado por el AG para navegar en el espacio de posibles conjuntos de reglas.
- La Aproximación Michigan [77], donde cada miembro de la población representa una regla de producción individual, y por tanto, una población es un conjunto de reglas. Cada regla tiene asociado un fitness calculado en base a un algoritmo de asignación de crédito.

La aproximación Pitt conlleva un mayor gasto computacional, tanto en tiempo de proceso como en memoria, que la aproximación Michigan. Ahora, ¿cual de las dos aproximaciones

es mejor en el sentido de ser más efectiva al evolucionar el subsistema de tarea? No está clara la respuesta, depende del caso de aplicación. Se acepta que la aproximación Pitt es más útil en entornos de trabajo que no requieren soluciones en tiempo real en los cuales cambios radicales de conducta son permisibles, mientras la Michigan es más útil en entornos de trabajo en tiempo real en los cuales cambios radicales de conducta no pueden ser tolerados [32].

1.5 Características y Problemas Abiertos de los SCs.

Una de las diferencias principales de los SCs con respecto a los Sistemas Expertos y otros Sistemas basados en reglas de la inteligencia artificial es básicamente la forma en que son operadas las reglas.

La mayoría de los sistemas basados en reglas de la inteligencia artificial usan reglas complejas, especializadas de alto nivel, cuyas correcciones y coherencia sólo pueden hacerse por medio del ingeniero en conocimiento. Además, las reglas son difíciles de crear en una primera instancia. Por otro lado, el SC usa reglas sintácticamente simples donde la exploración de ellas puede descubrirse fácilmente y pueden manipularse por los algoritmos heurísticos como puede ser el mismo AG. No se requieren exactitud y coherencia: las reglas incoherentes y contradictorias se permiten al mismo tiempo y competir entre ellas.

De muchas maneras, son mayormente relacionados los SCs con las redes neuronales principalmente porque pueden compartir los mismos conceptos de aprendizaje. En las redes neuronales, la representación del conocimiento es distribuida entre unidades de cómputo simple que operan en paralelo. El cómputo global surge de la cooperación y las interacciones locales entre estas unidades simples. Como consecuencia, además los SCs comparten con las redes neuronales las propiedades importantes, como la robustez y capacidades de la generalización. Desde la primera regla que se aplica a la representación de conceptos diferentes, el sistema tiende a dar las respuestas similares cuando es presentado con entradas similares. Este rasgo aumenta la habilidad del sistema de tratar con las situaciones imprevistas, así como descubrir y representar las regularidades del entorno. Una ventaja de los SCs encima de las redes neuronales es la transparencia semántica.

Mientras es difícil deducir cómo una red neuronal resuelve una tarea de un juego de pesos de conexión, el conocimiento producido por un SC es más fácil de inspeccionar y manipular.

Una diferencia extensa que distingue al SC de la inteligencia artificial clásica es la independencia de la tarea. Los SCs no requieren una cantidad grande de conocimiento de la tarea específica, porque sus operadores y sus mecanismos de aprendizaje son dependientes de la representación en lugar de ser dependiente de la misma tarea. Para aplicar un SC a un problema diferente, sólo sus interfaces con el ambiente tienen que ser definidas. Este rasgo evita algunos de los problemas crónicos de la inteligencia artificial clásica, como el cuello de botella de la adquisición de conocimiento. De hecho, el conocimiento sobre la tarea, así como su representación, autónomamente surge de la interacción con la propia tarea.

Podría uno pensar que todos estos rasgos interesantes de SC representan la solución esperada por mucho tiempo para resolver todos los problemas encontrados por diseñadores de sistemas inteligentes. Durante las últimas dos décadas, se han investigado los SC a fondo y aplicado a una gran variedad de tareas de diferentes dominios, de la robótica, juegos, para controlar problemas de predicción. Sin embargo, la mayoría de las promesas entusiásticas de Holland aun no se han cumplido. Los mecanismos de aprendizaje de los SCs merecen una extensa investigación para aumentar su generalización y aprender mayores capacidades del problema. Un número considerable de problemas todavía tienen que ser resuelto antes de que los SCs puedan aplicarse a problemas reales y complejos que llevamos a cabo en forma rutinaria en las empresas y en el mundo en general. Entre estos problemas podemos mencionar, cómo generar y conservar largas cadenas de reglas en las tareas de refuerzo con retardo, cómo contrastar la tendencia a generar reglas que nos llevan a una convergencia prematura y desde luego a una solución subóptima. Otro gran problema en los SCs es la pérdida de buenas reglas al aplicar los AGs en el componente descubridor de nueva información, hasta hora inevitable y la última es que no existía una metodología para el diseño de aplicaciones donde intervienen los SCs. Para tratar de corregir estos problemas, investigadores han modificado y han integrado los mecanismos de aprendizaje básicos de muchas maneras y por mi parte he tratado de integrar todas estas teorías en la metodología para minimizar riesgos y tiempos innecesarios de diseño de la aplicación. En mi propuesta se aprovechan las contribuciones de los AGs para evitar caer en reglas innecesarias como la

incorporación de mecanismos de nichos [104] y por otro lado se restringió la unión, para promover la cooperación de la regla. Muchos investigadores se han enfocado en modelos de SC sin memoria, qué es más simple al estudio pero ellos no pueden tratarse de los ambientes parcialmente notables. Un cambio sustancial al modelo básico es el que expuso Wilson [141][142] y que nos sirvió de base para la propuesta de la metodología.
Mientras, en el modelo original de Holland, la aptitud de una regla refleja el refuerzo esperado por su uso, en el modelo XCS de Wilson propone usando la exactitud de la predicción del refuerzo como la aptitud. Este acercamiento ha producido los resultados interesantes resolviendo el problema de generalización, porque tiende a generar reglas generales y exactas. Una contribución de esta tesis para mejorar las capacidades de aprendizaje en los SCs es la combinación de varias teorías de la Programación Dinámica (Q-learning y aprendizaje VQQL) con los aprendizajes iniciales propuesto por Holland.

1.6 Objetivo de la tesis.

A medida que el tiempo avanza nos encontramos constantemente con conceptos que hace unos años eran desconocidos y mas aun, estos mismos eran subutilizados, tal es el caso de que existen conceptos relativamente novedosos no en cuanto a su aparición, sino como técnica de aplicación en problemas reales de una manera eficiente y oportuna, tal es el caso de los mismos sistemas clasificadores, técnicas heurísticas como grasp, data-warehouse, lógica difusa, comercio electrónico, técnicas consideradas ahora como metaheurísticas, etc.
No solo el concepto de SC es relativamente nuevo sino que existen subdivisiones o aplicaciones muy especificas que los convierten aun mas atractivos para los investigadores, tal es el caso de los sistemas clasificadores donde son combinados con otras técnicas de inteligencia artificial para su mejor desempeño, entre estas técnicas se encuentran las redes neuronales, la lógica difusa y los Sistemas Expertos que hacen aun mas poderosa la herramienta.

Por otro lado, existen metodologías para resolver un gran número y diversidad de problemas pero hasta la fecha no existe una para el desarrollo de aplicaciones con SC y las preguntas que tratamos de responder con esta tesis fueron:

- ¿Las metodologías actuales sirven para resolver las aplicaciones donde intervienen los SCs con éxito?
- ¿Por ser una técnica relativamente nueva no ha habido hasta la fecha propuestas de metodologías que nos lleve paso a paso a la resolución de este tipo de aplicaciones?

El propósito fundamental de esta tesis es el de mostrar una metodología que sea capaz de llevar al usuario o programador al diseño eficiente de aplicaciones donde intervienen los SCs y esto por la sencilla razón de que actualmente se diseña empleando metodologías que no son apropiadas para estas aplicaciones. El desarrollo de la metodología se deriva del trabajo y asesoría de mis asesores en el transcurso de estos cinco años del doctorado y así mismo se mencionan los últimos resultados encontrados para el desarrollo ideal de cada uno de los parámetros involucrados en la misma.

Desde luego sabemos que no hay una metodología correcta para todos los propósitos y ninguna metodología está completa, como toda la abstracción en algún sentido desde los detalles de la realidad. Aun así, el uso de una metodología apropiada puede ayudar de manera considerable en el control de un proyecto de software y es por ello que mi propuesta de tesis es **UNA METODOLOGÍA QUE SEA CAPAZ DE LLEVAR EN MANERA EFICIENTE AL DISEÑO DE APLICACIONES CON SISTEMAS CLASIFICADORES GENÉTICOS** y que irá dando respuesta a resolver las preguntas iniciales.

Ahora bien, es necesario contar con un modelo o metodología capaz de resolver cualquier problema que se tenga, y es preciso sin duda definir cada uno de los pasos que intervienen en estos o de lo contrario caemos en el gran riesgo de estar experimentando emplear algo que no sabemos si después de invertir recursos (tiempo, recursos humano, infraestructura, etc.) en esta tarea, lleguemos a una solución confiable. Y es por ello que la metodología que se propone servirá no tanto para toda aquella persona que ya conoce de cómo y dónde usar sistemas clasificadores genéticos (SCG), sino también le servirá a toda aquella que tiene interés de implementar una solución por esta vía, es decir la aplicación de los Algoritmos Genéticos para problemas de Aprendizaje de Máquinas y que para la tesis toma el nombre de sistema clasificador genético.

1.7 Estructura de la tesis.

El contenido de cada uno de los capítulos queda de la siguiente manera:

Capítulo 1: Introducción y contenido de la tesis

Capítulo 2: Fundamentos de Algoritmos Genéticos

Capítulo 3: Sistemas clasificadores de aprendizaje.

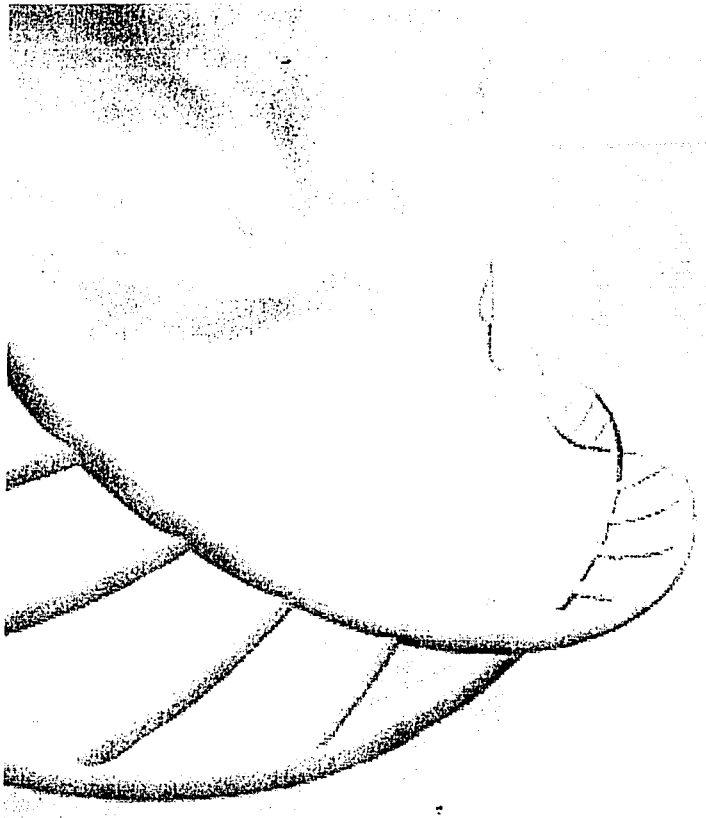
Capítulo 4: Metodología para el diseño de aplicaciones con sistemas clasificadores genéticos.

Capítulo 5: Conclusiones y trabajos futuros.

Anexo A: Pruebas y resultados diversos de la tesis.

Anexo B: Glosario.

Bibliografía.



CAPÍTULO 2
FUNDAMENTOS DE ALGORITMOS GENÉTICOS.

2.1 Introducción.

En este capítulo muestro los fundamentos básicos, terminología, desarrollos y resultados de los AGs en general, así como el teorema fundamental en que se desarrolla la teoría de estos mismos. La exposición incluye la descripción de los componentes de un AG, su funcionamiento, el teorema de esquemas, las modificaciones, extensiones y técnicas avanzadas, para tratar con más detalle los AG diseñados para problemas de secuenciación (PS). Se parte del análisis de la dificultad que presenta el AG simple para modelar este tipo de problemas y de cómo esto originó la aparición de diferentes representaciones del cromosoma. Se proponen diversas formas de agrupar las representaciones del cromosoma.

2.2 Algoritmos Genéticos.

Los AG surgen como:

- herramientas para la solución de complejos problemas de optimización producto del análisis de los sistemas adaptativos en la naturaleza, y como resultado de abstraer la esencia de su funcionamiento.

Los AG son procedimientos de propósito general, basados en los principios de la genética y la evolución. Ellos trabajan modificando repetidamente una población de estructuras artificiales a través de la aplicación de los siguientes operadores genéticos:

- Operador de selección.
- Operador de cruzamiento y el,
- Operador de mutación

La meta en optimización es encontrar la mejor solución posible o soluciones a un problema, con respecto a uno o más criterios. Para utilizar los AG es necesario encontrar una posible estructura para representar las soluciones. Pensando este asunto como el problema de

buscar en un espacio de estados, una instancia de esta estructura representa un punto o un estado en el espacio de búsqueda de todas las posibles soluciones. Así, una estructura de datos en el AG consistirá en un cromosoma. Un cromosoma es comúnmente una cadena de bits, sin embargo, otras representaciones del cromosoma pueden incluir vectores de números reales [30][43][60][61] y programas de computadoras de alto nivel [88]. Habitualmente la estructura cromosomática tiene longitud fija ℓ.

Cada cromosoma (cadena) es una concatenación de un número de subcomponentes llamados genes. La posición de un gen en el cromosoma se conoce como locus y sus valores como alelos. En la representación como cadena de bits, un gen es un bit, un locus es su posición en la cadena y un alelo es su valor 0 ó 1. El término biológico genotipo se refiere a la estructura genética total de un individuo. El término fenotipo se refiere a las características observables de un individuo y corresponde a su estructura decodificada en el AG.

Tomemos un ejemplo sencillo para comprender lo hasta aquí expuesto. Sea el problema de optimización genética la maximización de la siguiente función de dos variables:

$$F(x_1,x_2) = x_1 + x_2,$$

$$\text{donde } 0 \leq x_1 \leq 1 \text{ y } 0 \leq x_2 \leq 1.$$

Una forma de codificación para las variables reales es transformarlas en cadenas de enteros binarios de una longitud apropiada para lograr la precisión deseada. Para nuestro ejemplo la codificación en 8 bits es suficiente para cada variable x_1 y x_2. La decodificación de estas variables se hace dividiendo el entero binario correspondiente por 2^{7-1}. Por ejemplo, 00000000 representa 0/127 ó 0, 11111111 representa 127/127 ó 1. La estructura de datos a optimizar es una cadena de 16 bits que representa la concatenación de las codificaciones de x_1 y x_2. La variable x_1 se ubica en los primeros 8 bits y la variable x_2 se ubica en los 8 bits más a la izquierda. El genotipo de un individuo es una cadena de 16 bits, mientras que el fenotipo es una instancia del tuplo $<x_1,x_2>$. El genotipo es un punto en el espacio de Hamming de dimensión 16 donde el AG realiza su búsqueda. El fenotipo es un punto en el espacio dos dimensional de las variables decodificadas.

Al optimizar una estructura usando un AG se necesita una medida de la calidad de cada

estructura en el espacio de búsqueda. La función de adaptabilidad es la encargada de esta tarea. Para trabajar con los AGs primero es conveniente partir de una definición que nos lleve a clarificar que pretende abarcar esta técnica y aunque existen al igual que todos los conceptos quien los define de una u otra forma, lo haré de acuerdo a como lo realiza John Koza [88], donde nos muestra los elementos básicos que deben tener un AG:

"El Algoritmo Genético, es un algoritmo matemático altamente paralelo que transforma un conjunto (población) de objetos matemáticos individuales (típicamente cadenas de caracteres de longitud fija que se ajustan al modelo de las cadenas de cromosomas), cada uno de los cuales se asocia con una aptitud, en una población nueva (es decir, la siguiente generación) usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y sobrevivencia del más apto y tras haberse presentado de forma natural una serie de operaciones genéticas (notablemente la recombinación sexual)".

2.3 Funcionamiento de un AG.

A continuación menciono un diagrama básico donde se observa el ciclo de un AG básico (figura 2.1).

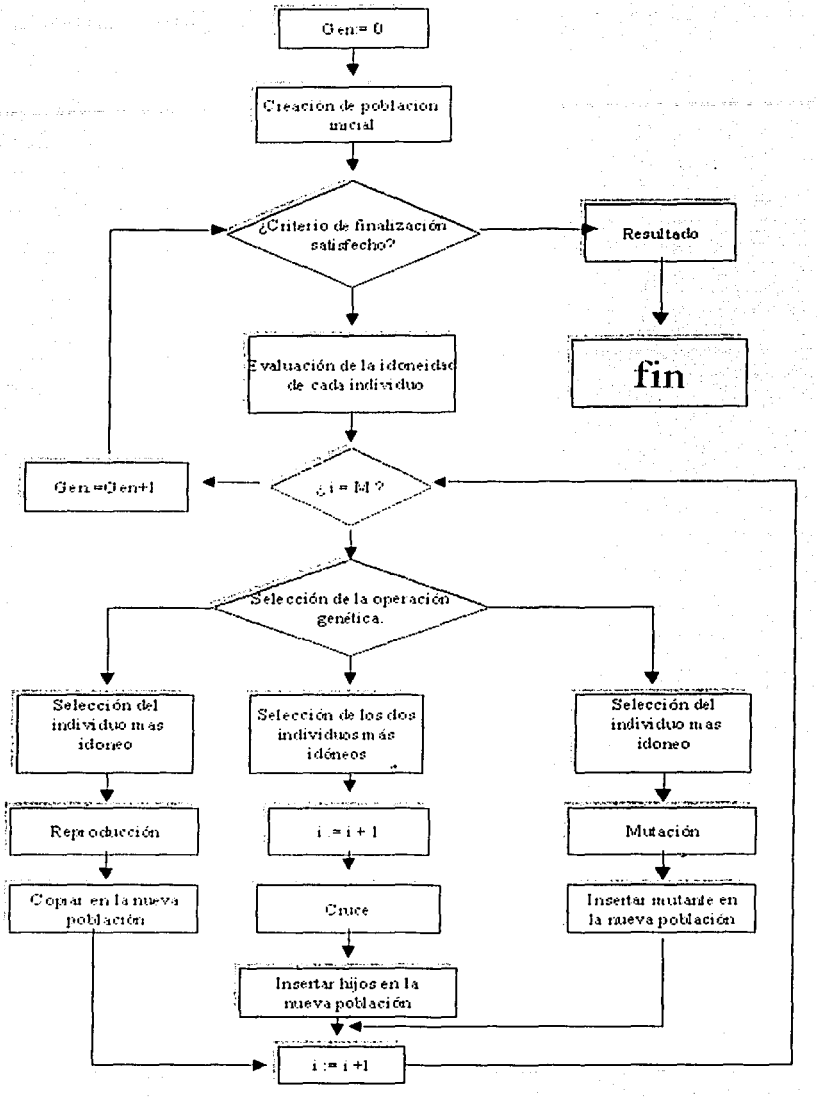


Figura 2.1: Diagrama básico de algoritmo genético secuencial.

El AG simple genera aleatoriamente una población de "n" estructuras (cadenas, cromosomas o individuos) y sobre esta actúan los operadores transformando la población. Una vez completada la acción de los tres operadores genéticos (selección, cruzamiento y mutación) se dice que ha transcurrido un ciclo generacional. El operador de selección realiza la selección de las cadenas de acuerdo a su adaptabilidad para el posterior apareamiento. El operador de cruzamiento realiza la recombinación del material genético de dos cadenas padres. El operador de mutación al estilo del operador natural realiza la mutación de un gen dentro de un cromosoma o cadena a sus diferentes formas alelomorfas. Para cada uno de estos operadores está asociado el uso de probabilidades y la generación de números aleatorios.

El AG ejecuta para un número fijo de generaciones o hasta que se satisface algún criterio de parada. Se asigna a cada individuo **i** de la población una probabilidad de selección $\mathbf{P_S(i)}$, de acuerdo a la proporción entre la adaptabilidad de **i** y la adaptabilidad de la población total,

$$P_S(i) = f(i) / \sum_{j=1}^{n} f(j) .$$

Entonces se seleccionan (con reemplazamiento) un total de "**n**" individuos para posteriores procesamientos genéticos de acuerdo a la distribución definida por $\mathbf{P_S(i)}$. La forma más simple de selección proporcional es la ruleta [56].

Intuitivamente, la función de adaptabilidad, puede verse como una medida de ganancia, utilidad o ajuste que queremos maximizar. Copiando cadenas de acuerdo a su adaptabilidad significa que las cadenas con valores más altos tienen mayores probabilidades de contribuir en uno o más hijos en la próxima generación. Cuando una cadena es seleccionada para reproducción, se hace una copia exacta de la misma y pasa a una bolsa de apareamiento (matching), de donde actuarán próximos operadores. Este operador, es una versión de la selección natural Darwiniana de la supervivencia. En las poblaciones naturales la adaptabilidad es una cierta habilidad de las criaturas a sobrevivir a los depredadores, la pestilencia y otros obstáculos hacia la madurez y consecuente reproducción. En el esquema artificial la función de adaptabilidad es el árbitro final de la vida o muerte de los individuos (cadenas o cromosomas).

Después de la selección, los **n** individuos que se encuentran en la bolsa de apareamiento

son acoplados aleatoriamente para producir "**n/2**" parejas. Existe una probabilidad de cruzamiento fija $\mathbf{P_c}$. Para cada pareja el cruzamiento puede ocurrir o no. Con probabilidad **1-** $\mathbf{P_c}$ el cruzamiento no ocurre y ambos individuos pasan intactos a la mutación. De lo contrario la pareja produce dos hijos por el cruzamiento y sólo ellos pasan a la mutación.

La obtención de individuos nuevos a partir de los ya existentes es una de las características más interesantes e importantes en el trabajo de los AGs. Después del cruzamiento comienza a efectuarse la mutación. Para todas las cadenas que llegan al estado de mutación, cada uno de sus bits son modificados con probabilidad $\mathbf{P_m}$. Los bits de cada cadena son mutados independientemente, es decir, la mutación de un bit no afecta la probabilidad de mutación de los otros bits. El AG simple trata la mutación sólo como un operador secundario con el rol de restaurar la pérdida de material genético. Por ejemplo, supongamos que todas las cadenas en una población tienen valor **0** en una posición determinada y la solución optimal tiene un **1** en esa posición. El cruzamiento no puede regenerar un **1** en esa posición mientras que la mutación sí.

La población resultante de la mutación reemplaza totalmente la población anterior completando una generación. Las generaciones siguientes realizan el mismo ciclo de selección, cruzamiento y mutación.

Desde el punto de vista de la comparación de los AGs con otros métodos de búsqueda se pueden enmarcar sus diferencias en cuatro aspectos.

1. Trabajan con una codificación de los parámetros y no con los parámetros mismos.
2. Buscan a partir de una población de puntos y no de un punto simple.
3. Usan directamente la función objetivo y no la derivada u otro conocimiento auxiliar.
4. Usan reglas de transición probabilísticas y no determinísticas.

2.4 Teorema de Esquemas.

Mientras un AG explícitamente procesa cadenas, implícitamente este procesa esquemas, los cuales representan similaridades entre las cadenas. El AG no puede proponerse visitar todos los puntos en el espacio de búsqueda, y sí muestrear un número suficientemente grande de

hiperplanos en regiones de alta adaptabilidad del espacio de búsqueda. Cada hiperplano corresponde a un conjunto de subcadenas similares de alta adaptabilidad.

Definición.- Un esquema es una cadena de longitud ℓ (la longitud de las cadenas de la población), tomada del alfabeto {0,1,*} donde * permite tomar un 0 ó 1 en una posición particular. El "*" es un metasímbolo, que es una herramienta notacional para describir todas las posibles similaridades entre las cadenas.

Cada esquema representa el conjunto de todas las cadenas de longitud ℓ, en cuyas posiciones con valores 0 y 1 son idénticas.

Ejemplo: El esquema 10**1, representa el conjunto {10001,10011,10101,10111}. Los elementos de este conjunto se conocen como instancias del esquema que ellos representan. Los esquemas son llamados también, subconjuntos de similaridad porque ellos representan subconjuntos de cadenas con similaridades en ciertas posiciones fijas. Las posiciones fijas de un esquema son las posiciones de la cadena con valores 0 y 1.

Sin embargo, no todos los esquemas son creados igual, algunos son más específicos que otros.

Ejemplo: El esquema 0 1 1 * 1 * * es una construcción más definida acerca de las similaridades importantes que el esquema 0 * * * * * *. Además hay esquemas que se expanden más que otros.

Ejemplo: El esquema 1 * * * * 1 * expande una porción más grande que el esquema 1 * 1 * * * * .

Dos propiedades importantes de los esquemas el orden y la longitud definida se relacionan con este hecho.

Definición.- El orden de un esquema H, denotado por o(H), es simplemente el número de posiciones fijas (es decir, de posiciones con valores en el alfabeto V = {0,1}) presentes en el patrón.

Ejemplo: O(0 1 1 * 1 * *) = 4 y O(0 * * * * * *) = 1

Definición.- La longitud definida de un esquema H, denotada por δ(H), es la distancia entre la primera y la última posición especificada de la cadena.

Ejemplo: δ(0 1 1 * 1 * *) = 5 - 1 = 4 y δ(0 * * * * * *) = 0

Dos conceptos se hacen necesarios para valorar la importancia de un esquema dentro de una población.

Definición.- La adaptabilidad promedio del esquema H, denotada por f(H), será definida como $f(H) = \sum_{k=i_1}^{i_m} f(k) / m$. donde $k = i_1, \ldots, i_m$ son las cadenas instancias del esquema H y m la cantidad de dichas cadenas.

Definición.- La adaptabilidad promedio de la población total, denotada por ftotal, será:

$$ftotal = \sum_{j=1}^{n} f(j) / n \text{ , } j = 1 \ldots, n, \text{ n el tamaño de la población.}$$

Con esta información es posible hablar del concepto de Bloques constructores.

Definición.- Se denominan bloques constructores a los esquemas que son de longitud definida pequeña, de bajo orden y de adaptabilidad promedio por encima de la población total.

Sobre estos Bloques constructores es que se define el incremento o decremento de los esquemas en una población y se analiza el efecto combinado de la selección, cruzamiento y mutación. Esto se conoce como teorema de esquema o teorema fundamental de los AG como lo califica Goldberg en su libro[56].

Teorema de esquemas (Probabilidad de cruza).- Sea P_{cruce}, la probabilidad de que un par de cromosomas sea entrecruzado y sea P_{mut} la probabilidad de que una mutación ocurra en un locus dado. Si m(H,t+1) es el número de instancias del esquema H en la población en el tiempo t, entonces

$$m(H,t+1) \geq \frac{m(H,t)F(H)}{f\text{total}} \left[1 - \frac{P_{cruce}\delta(H)}{\ell - 1} - o(H)\ P_{mutacion} \right]$$

Y gráficamente podemos observar el siguiente esquema, donde apreciamos el hiperplano del esquema de orden 1 y longitud 3 (figura 2.2).

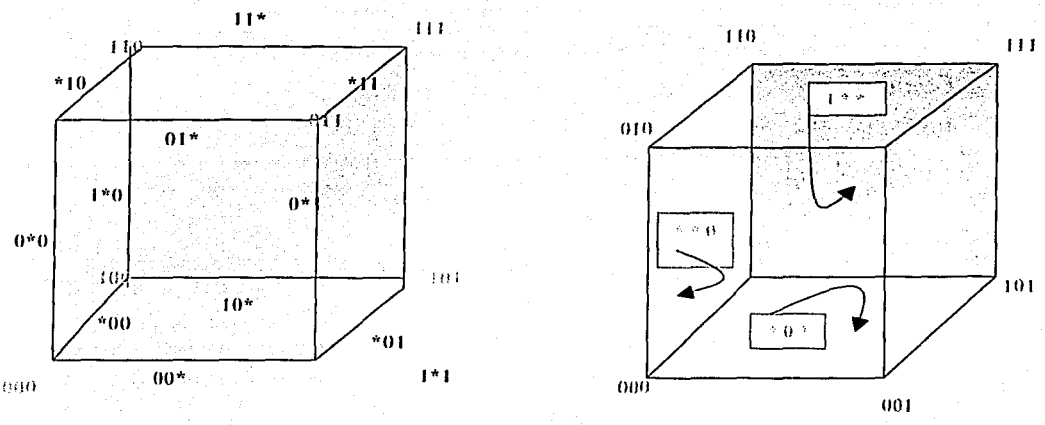


Figura 2.2: Representación de esquemas.

Demostración: Para demostrar el teorema de esquema, Holland analiza los efectos de la selección, cruzamiento y mutación como sigue: se supone que un paso de tiempo **t** dado, existen **m** instancias de un esquema particular **H** contenidos en la población **A(t)** donde escribimos **m = m (H,t)**. Durante la selección, una cadena es copiada de acuerdo a su adaptabilidad o más precisamente una cadena A_i se convierte en seleccionada con probabilidad

$$P_S(i) = f(i)/ \sum_{j=1}^{n} f(j) .$$

Después de seleccionada una población no superpuesta de tamaño **n** con reemplazamiento desde la población **A(t)**, se espera tener **m(H,t+1)** instancias del esquema **H** en la población en el tiempo **t+1** dada por la ecuación

$$m(H,t+1) = m(H,t)\, n\, f(H) / \sum_{j=1}^{n} f(j).$$

Si se reconoce que la adaptabilidad promedio de la población total puede ser escrita como: $f_{total} = \sum_{j=1}^{n} f(j)/n$, entonces la ecuación de crecimiento de la selección del esquema quedaría:

$$m(H,t+1) = m(H,t)\, f(H) / f_{total}$$

En otras palabras, el número de instancias de un esquema particular crece como la razón de la adaptabilidad promedio del esquema a la adaptabilidad promedio de la población. Este mecanismo es llevado a cabo con todo esquema **H** contenido en una población particular **A** en paralelo.

Si se asume que un esquema particular rebasa al promedio por encima una cantidad $\mathbf{c} \bullet f_{total}$, con **c** como una constante, se tiene:

$$m(H,t+1) = m(H,t)\,(f_{total} + c \bullet f_{total}) / f_{total} = (1 + c)\, m(H,t)$$

Si se comienza en el tiempo **t = 0** y se asume un valor estacionario de **c**, se obtiene la ecuación

$$m(H,t) = m(H,0)\,(1 + c)^{t}$$

Aquí se puede reconocer una progresión geométrica o la análoga discreta de una forma exponencial. La selección asigna un incremento exponencial de instancias de los esquemas

de promedio por encima y decremento exponencial de instancias de los esquemas de promedio por debajo.

Efecto del cruzamiento sobre el número esperado de instancias de un esquema en la población t+1: ¿Cuáles esquemas son afectados por cruzamiento y cuáles no?

Ejemplo: Sea $\ell = 7$, A = 0 1 1 1 0 0 0, H_1 = * 1 * * * * 0, H_2 = * * * 1 0 * *

Los dos esquemas H_1 y H_2 son representados en la cadena A.

Al utilizar un cruzamiento simple como el descrito en la sección anterior. Al determinar el punto de cruce, este produjo la ruptura de las cadenas entre las posiciones 3 y 4, como se observa en el siguiente ejemplo.

Ejemplo: A = 0 1 1 | 1 0 0 0, H_1 = * 1 * | * * * 0, H_2 = * * * | 1 0 * *

El esquema H_1 será destruido porque el 1 de la posición 2 y el 0 de la posición 7 serán ubicados en diferentes descendientes (ellos están en lados opuestos del separador que marca el punto de cruce). Con el mismo punto de cruce H_2 sobrevivirá porque el 1 de la posición 4 y el 0 de la posición 5 pasan intactos a un único descendiente. El esquema H_1 es menos probable que sobreviva a un cruzamiento que el esquema H_2.

Si el sitio del cruzamiento es seleccionado de forma aleatoriamente uniforme entre $\ell - 1 = 7-1 = 6$ sitios posibles, entonces el esquema H_1 es destruido con probabilidad

$$P_{destrucción} = \delta(H_1)/(\ell - 1) = 5/6,$$

este sobrevive con probabilidad, $P_{supervivencia} = 1 - P_{destrucción} = 1/6$.

Para el esquema H_2 la longitud es $\delta(H_2) = 1$ y este es destruido durante un evento en seis cuando el sitio de corte sea seleccionado entre las posiciones 4 y 5, así pues $P_{destrucción}$ =

1/6 o la probabilidad de supervivencia es $P_{supervivencia} = 1 - P_{destrucción} = 5/6$.

Más generalmente, un limite inferior sobre la probabilidad de supervivencia al cruzamiento $P_{supervivencia}$ puede ser calculada para cualquier esquema. Como, un esquema sobrevive, cuando el sitio de cruzamiento cae fuera de la longitud definida, la probabilidad de supervivencia bajo un simple cruzamiento es:

$$P_{supervivencia} = (1 - \delta(H)) / (\ell - 1)$$

ya que el esquema puede ser destruido si el sitio de cruzamiento cae entre las posiciones de los valores bien definidos del esquema que determina su longitud definida y hay $\delta(H)$ posibles formas de destruirlo. Luego, la posibilidad de destruirlo es $\delta(H)/(\ell - 1)$.

Si el cruzamiento es el mismo realizado por selección aleatoria digamos con probabilidad P_{cruce} en un pareo particular, la probabilidad de supervivencia puede darse por la expresión

$$P_{supervivencia} > 1 - P_{cruce} * \delta(H) / (\ell - 1)$$

la cual se reduce a la primera expresión cuando $P_{cruce} = 1.0$.

El efecto combinado de la selección y cruzamiento puede considerarse en la siguiente expresión:

$$m(H,t+1) \geq \frac{m(H,t)F(H)}{f\text{total}} \left[1 - \frac{P_{cruce}\delta(H)}{\ell - 1}\right]$$

El esquema H crece o muere dependiendo de un factor de multiplicación. Ahora, este factor depende de si el esquema está por encima o por debajo del promedio de la población y si el esquema tiene longitud definida relativamente pequeña o grande. Claramente, aquellos esquemas con promedio alto y longitud definida pequeña, serán muestreados con una razón de incremento exponencial.

Efecto de la mutación sobre el número de instancias de un esquema en la población t+1: Con el objetivo de que un esquema H sobreviva, todas las posiciones fijas necesitan ellas mismas sobrevivir. Por tanto desde que un simple alelo sobreviva con probabilidad $(1 - P_m)$ y desde que cada una de las mutaciones es estadísticamente independiente, un esquema particular sobrevivirá cuando cada una de las O(H) posiciones fijas en el esquema sobrevivan. Multiplicando la probabilidad de supervivencia $(1 - P_m)$, O(H) veces obtenemos la probabilidad de supervivencia de la mutación. $(1 - P_m)^{O(H)}$. Para valores pequeños de P_m ($P_m < 1$), la probabilidad de supervivencia del esquema puede aproximarse por la expresión $1 - O(H)\ P_m$.

Por tanto se llega a que un esquema particular H recibe un número esperado de instancias en la próxima generación bajo selección, cruzamiento y mutación, dada por la siguiente ecuación (ignorando pequeños términos del producto cruzado).

$$m(H,t+1) \geq \underbrace{\frac{m(H,t)F(H)}{f\text{total}}}_{\text{Selección}} \left[\underbrace{1 - \frac{P_{cruce}\,\delta(H)}{\ell - 1}}_{\text{Cruzamiento}} - \underbrace{o(H)\ P_{mutacion}}_{\text{Mutación}} \right]$$

2.5 Población.

Cada iteración de un AG simple crea una población totalmente nueva de una población existente. Este es llamado AG generacional. El AG que reemplaza sólo una fracción pequeña de cadenas a la vez, es llamado AG de estado fijo.

Tamaño de la población.- Muchos trabajos se han escrito relativos a la influencia del tamaño de la población en la convergencia del AG [104]. En principio, es lógico pensar que el trabajo con poblaciones pequeñas corren el riesgo de representar pobremente el espacio de soluciones. Por otro lado, las poblaciones de gran tamaño consumen mayor tiempo computacional. Sobre esta disyuntiva y como un trabajo teórico, Goldberg [57] obtuvo en

su investigación que el tamaño óptimo de una población de cadenas binarias, crece exponencialmente con la longitud de la cadena. Sin embargo, en diferentes resultados empíricos muchos autores sugieren tamaños de poblaciones tan pequeños como 30 individuos [28]. En [1] se obtienen tamaños de población entre ℓ y 2ℓ para un tipo de problemas. Goldberg posteriormente propone tamaños de población más pequeños que en su primer trabajo [65]. Investigaciones recientes se han acercado a un número cercano al óptimo para determinar el tamaño de la población y que garantice buenos resultados. Usualmente la población inicial es generada aleatoriamente. En los trabajos de [110][81] encontramos que el sembrado es bueno usarlo cuando el comportamiento de los genes o variables tienen diferentes tamaños y el cromosoma es demasiado grande (mayor a 500 bits). Mi opinión al respecto es que al usar sembrado si bien se esperaría ganar en que el resultado convergiera mas rápidamente en la realidad es que esta es mínima y se pierde mas tiempo en diseñar el sembrado que el tiempo ganado en procesar la información el mismo programa.

2.6 Selección.

Uno de los operadores mas importantes sin duda es la forma de cómo son seleccionados los individuos que nos servirán para codificar nuestra información y con ella poder encontrar una solución deseada a nuestro problema, existen una gran diversidad de métodos de selección y cruzamiento y en estos apartados mencionaré los que considero han tenido mayor aceptación a diferentes aplicaciones.

Selección Proporcional.- Este nombre describe un grupo de esquemas de selección originalmente propuestos por Holland [72], que eligen individuos de acuerdo a los valores de sus funciones objetivo. En este caso, el "valor esperado" de un individuo (es decir, el número esperado de veces que un individuo será seleccionado para reproducirse) es la aptitud del individuo dividida entre la aptitud promedio de la población. Existen algunas variantes de esta técnica, como es el de **la Ruleta** [32] que es el método más común de implementación y que Goldberg ha empleado en la mayoría de sus aplicaciones con mayor éxito. El de selección por **sobrante estocástico** [9][15]. **Elitismo** que fue introducido por De Jong en su tesis doctoral [32] y establece que el mejor individuo o los mejores de la población sobrevivan de generación en generación. La estrategia de elitismo básica copia el

mejor individuo de la población actual a la próxima, esta técnica es la que ha arrojado mejores resultados en los trabajos realizados por Goldberg y Dorigo [56][42]. **Universal Estocástico** [7] que trata de minimizar la mala distribución de los individuos en la población en función de sus valores esperados. usando esta técnica. los individuos más aptos se multiplicarán rápidamente impidiendo que el AG explore más porciones del espacio de búsqueda. A este fenómeno se le conoce como "convergencia prematura". **Escalamiento Sigma** [99]. para lidiar con los problemas de la selección proporcional, se ha experimentado con varias técnicas de escalamiento (métodos que mapean la aptitud original de una cadena con los valores esperados a fin de hacer al AG menos susceptible a la convergencia prematura), un ejemplo es el "escalamiento sigma", que mantiene la presión de selección (es decir. el grado en el cual se les permite a los individuos mas aptos tener descendientes) relativamente constante a través del paso de las generaciones. en vez de depender en las varianzas de aptitud en la población. Usando escalamiento sigma. el valor esperado de un individuo. es una función de su aptitud. la media de la población y la desviación estándar de la población. Al inicio de una corrida, cuando la desviación estándar de las aptitudes es típicamente alta. los individuos más aptos no recibirán el mayor segmento de la ruleta. hacia el final. el proceso se invertirá.

Inconvenientes de la selección proporcional: Sucede que en las generaciones iniciales de un AG los valores de adaptabilidad promedio son bajos. La presencia de algunas cadenas con un valor de adaptabilidad relativamente alto provoca que el mecanismo de selección proporcional asigne un número grande de copias a estas supercadenas provocando así la convergencia temprana. Por el contrario cuando el AG se encuentra en los últimos estados, es decir. cuando está convergiendo. la varianza entre los valores de adaptabilidad de las cadenas es pequeña y por tanto el esquema de selección proporcional asigna más o menos igual número de copias para todas las cadenas. perdiéndose el objetivo de promover las mejores cadenas.

En la selección por **jerarquías** propuesto por Baker [7] los individuos de la población se ordenan de mejor a peor (en base a su aptitud), y cada uno se selecciona tanta veces como corresponda de acuerdo a una función de asignación no incremental; posteriormente, se

efectúa selección proporcional de acuerdo a dicha asignación. No hay necesidad de escalar las aptitudes en este caso. Esto trae ventajas y desventajas, por ejemplo, podemos prevenir la convergencia prematura al evitar usar las aptitudes reales (ventaja), pero no sabremos qué tanto supera un individuo a otro en términos de aptitud (desventaja). Esta técnica evita que un pequeño grupo de individuos sea seleccionado demasiadas veces, reduciendo la presión de selección cuando la varianza de aptitud es baja, pues las diferencias entre las aptitudes absolutas no afecta a la jerarquización.

La mayor parte de los AGs son generacionales, es decir, a cada generación la nueva población consiste completamente de los hijos generados por los padres de la generación anterior (aunque algunos de estos hijos pueden ser idénticos a sus padres).
En algunos esquemas como el elitista, las generaciones sucesivas se traslapan de cierta forma (una porción de la generación anterior se retiene en la nueva población). La fracción de nuevos individuos de cada generación se ha denominado la "brecha generacional" [32]. En la selección de estado uniforme [131], sólo unos cuantos individuos son reemplazados en cada generación, usualmente un número reducido de los individuos menos aptos se reemplaza por hijos que resultan de la cruza y mutación de los individuos más aptos. Esta técnica suele usarse cuando se evolucionan sistemas basados en reglas (y el recordar lo que ya se ha aprendido) es importante y en la que los miembros de la población resuelven colectivamente (y no de manera individual) el problema en cuestión. Esta técnica ha sido analizada por Syswerda [125][126], Whitley [134] y De Jong [36]. Goldberg [61] demostró que la técnica de selección de estado uniforme incluida en el programa GENITOR [131] es O(n log n).

2.7 Cruzamiento

Producto de la importancia que tiene el operador de cruzamiento para los AGs diferentes técnicas de cruzamiento han sido propuestas y analizadas.

Estudios empíricos y teóricos han comparado los diferentes tipos de cruzamiento en cuanto a una medida para cuantificar la destrucción de esquemas, y a la potencialidad de

exploración del espacio de búsqueda [11][43][125][35]. Otro aspecto estudiado es la relación tamaño de la población y tipo de cruzamiento [34]. Entre los cruzamientos que mayormente se usan se encuentran el de uno y dos puntos de cruza y el uniforme propuestos por Goldberg [56] y su empleo es muy sencillo (figura 2.3).

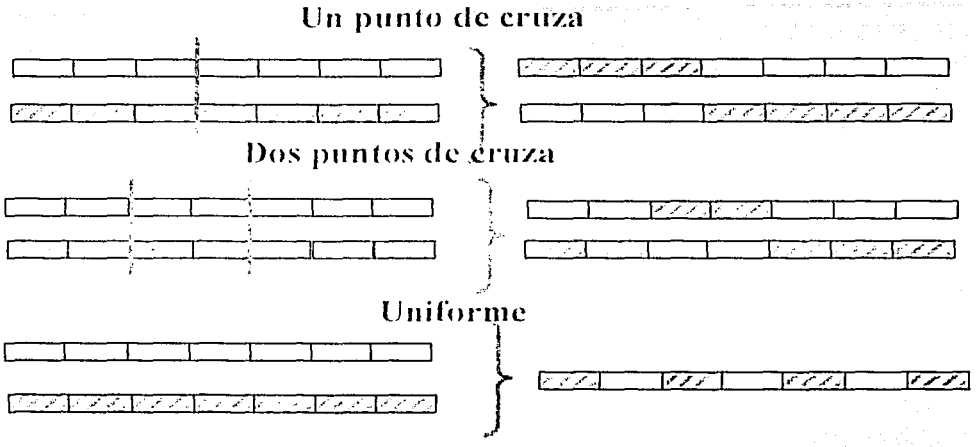


Figura 2.3: Uno y dos puntos de cruza y cruzamiento uniforme.

Variantes de la representación del cruzamiento en los algoritmos genéticos.- Se entienden como problemas de secuenciación u ordenamiento aquellos en los cuales las soluciones buscadas son secuencias ordenadas de objetos tal que la ordenación dada cumple con alguna medida de optimalidad. En ellos se involucran las permutaciones. Como ejemplos podemos citar, el Problema del Agente Viajero, el problema de coloreo de un grafo, el problema de programación de tareas y el problema de búsqueda de un ciclo hamiltoniano, entre otros. Todos ellos clasificados en la clase de problemas NP- completos [51]. La aplicación de los AG a problemas de secuenciación ha tropezado con la dificultad de que su representación no se ajusta fácilmente a los AG. El problema radica en el hecho de que la utilización del cruzamiento tradicional produce soluciones ilegales que no contienen a todos los elementos de la permutación y contienen elementos duplicados [56]. En esta clase de problemas un cromosoma es una secuencia ordenada de objetos, en este sentido lo tradicional es poner etiquetas numéricas a dichos objetos. Motivados por los principios en

que se basa el operador de cruzamiento tradicional de incorporar los mejores atributos de dos padres en un nuevo individuo, las primeras investigaciones se orientaron a crear operadores análogos que produjeran hijos legales. Esto condujo a la proliferación de diferentes formas de representar un cromosoma como es el de la representación por adyacencia [67], cruzamiento de arcos alternados [81] donde la principal desventaja de estos operadores es que tienden a destruir buenos recorridos.

El cruzamiento heurístico que emplea la experiencia de las conexiones de cada punto para formar una nueva mejorada [2].

La representación ordinal [67] que representa un recorrido como una lista de "n" ciudades.

La representación de caminos [64] que es la forma más común de indicar un recorrido. Muchos operadores aparecen reportados en la literatura sobre esta representación. Los operadores básicos son: cruzamiento de correspondencia parcial (PMX), el cruzamiento de orden (OX) y el cruzamiento cíclico (CX).

Ejemplo.

Un recorrido 5 ⇔ 1 ⇔ 7 ⇔ 8 ⇔ 9 ⇔ 4 ⇔ 6 ⇔ 2 ⇔ 3

es representado simplemente como (5 1 7 8 9 4 6 2 3)

El Cruzamiento de correspondencia parcial (PMX) propuesto por [64] construye un hijo seleccionando una subsecuencia de un recorrido desde un padre y preservando el orden y posición de tantas ciudades como sea posible del otro padre. Para esto, se generan dos puntos de cruce.

Ejemplo:

p_1 = (123|4567|89)

p_2 = (452|1876|93)

El procedimiento es el siguiente se genera:

h_1 = (XXX|1876|XX)

h_2 = (XXX|4567|XX)

Aquí se define una transformación:

$$1 \Leftrightarrow 4,\quad 8 \Leftrightarrow 5,\quad 7 \Leftrightarrow 6 \text{ y } 6 \Leftrightarrow 7.$$

Se llenan el resto de las ciudades donde no hay conflictos:

$$h_1 = (X23|1876|X9)$$
$$h_2 = (XX2|4567|93)$$

Entonces la primera X en h_1 debería ser 1, pero crea un conflicto, luego este es reemplazado por 4, en virtud de $1 \Leftrightarrow 4$, y así sucesivamente. De esta forma tenemos:

$$h_1 = (423|1876|59)$$
$$h_2 = (182|4567|93)$$

El cruzamiento PMX analiza importantes similaridades en el valor y ordenamiento simultáneamente, cuando es usado con un esquema de reproducción apropiado [64].

El cruzamiento de orden OX propuesto por [29] construye un hijo seleccionando una subsecuencia de un recorrido de un padre y preservando el orden relativo de las ciudades del otro padre.

Ejemplo: Con dos puntos de cruce marcados con |.

$$p_1 = (123|4567|89)$$
$$p_2 = (452|1876|93)$$

Los segmentos entre los puntos de cruce son copiados en los hijos:

$$h_1 = (XXX|4567|XX)$$
$$h_2 = (XXX|1876|XX)$$

Luego, comenzando desde el segundo punto de cruce de uno de los padres, las ciudades del otro padre son copiadas en el mismo orden, omitiendo los símbolos ya presentados. Cuando se alcanza el final de la cadena, se continúa desde la primera posición de la misma. Por ejemplo, la secuencia de ciudades en el segundo padre (desde el segundo punto de cruce) es:

9-3-4-5-2-1-8-7-6,

después de quitar las ciudades 4,5,6, y 7 las cuales están en el primer hijo tenemos,

9-3-2-1-8,

Esta secuencia es colocada en el primer hijo, comenzando por el segundo punto de cruce:

$O_1 = (218|4567|93)$

Similarmente se obtiene el otro hijo,

$O_2 = (345|1876|92)$.

El operador OX analiza la propiedad de que el orden de las ciudades es importante y no su posición.

El cruzamiento cíclico CX propuesto por [103] construye hijos de tal forma que cada ciudad (y su posición) se toman de uno de los padres.

Ejemplo:

$p_1 = (123456789)$

$p_2 = (412876935)$

Para producir el primer hijo se toma la primera ciudad del primer padre,

$$O_1 = (1XXXXXXXX)$$

La próxima selección es la ciudad 4 pues es justamente la ciudad desde el padre p_2 que está debajo de la ciudad 1 seleccionada. Luego las ciudades 8, 3 y 2. Cuando se forma un ciclo el resto de las ciudades se toman del otro padre. Finalmente, tenemos:

$$O_1 = (123476985),$$
$$O_1 = (412856739).$$

El CX preserva las posiciones absolutas de los elementos en las secuencias padres.

Otros operadores han sido definidos para la representación de caminos, por ejemplo [126] definió dos versiones modificadas del operador de cruzamiento de orden, estas son el cruzamiento basado en el orden (OX2) y el cruzamiento basado en la posición (PBX). Sin embargo, estos operadores resultaron ser en algún sentido equivalentes. Algunas comparaciones teóricas y empíricas de estos operadores básicos se reportan en [49][56][103][120][126].

Operadores de esta misma representación se reportan para problemas de sumergir un grafo completo en un hipercubo. Ellos son operadores al estilo del operador de orden, de transformación parcial y cíclico pero en este caso solo generan un punto de cruce y el resultado del cruzamiento es un sólo hijo.

Muchas de las aplicaciones de los AG a problemas de ordenamiento se han concentrado sobre el problema del agente viajero para el cual la información relevante no es la posición de cada elemento en la cadena sino cuales elementos son adyacentes. Un intento preliminar de utilizar la información de los enlaces lo hizo [68] con una clase de operador heurístico. Sin embargo, el operador transfería sólo el 60% de los arcos de sus padres, es decir, que un 40% de los arcos era seleccionado aleatoriamente. La representación por listas de adyacencias, introducida por primera vez por [131], sigue la motivación de aprovechar los

elementos adyacentes de la cadena. Una cadena padre es trasladada a una lista de ciudades con sus vecinos asociados.

Whitley y sus colaboradores desarrollaron el operador de recombinación de aristas (ERX) para usar con esta representación y fue usado en el problema del agente viajero y en problema de programación de tareas [135]. La idea que está detrás de este operador es que un hijo debe ser construido exclusivamente teniendo en cuenta los arcos que están presentes en ambos padres. Para esto se apoya en una lista de arcos. Este operador transfiere más del 95% de los arcos de ambos padres a un solo hijo. Dos años más tarde, el operador de recombinación de aristas fue mejorado [120]. La idea fue distinguir entre los arcos que se repiten desde ambos padres y los que no se repiten. Además, introducen una modificación para hacer mejores elecciones cuando los arcos deben ser tomados aleatoriamente.

Fox [49] representó un recorrido como una matriz binaria de precedencia. Esta representación cumple un conjunto de propiedades sobre las cuales fue posible definir dos operadores, el de intersección (IX) que preserva todas las relaciones de precedencia las cuales son comunes a ambos padres y el de unión (UX), que combina algunas relaciones de precedencia tomadas de cada padre. Fox reportó que los mismos tenían buenas ejecuciones cuando se comparaban con otros operadores de cruzamiento, sobre la base del número de evaluaciones requeridas de la función objetivo a lograr soluciones de la misma calidad. Sin embargo, el tiempo computacional requerido es más alto. En [109] se introduce un cruzamiento tipo unión (UX2), pero en este caso sobre una representación de caminos haciendo más compacta la representación y logrando una mayor eficiencia.

La representación basada en matrices de adyacencia fue considerada por Michalewicz [98], esta representación dio resultados razonables para diferentes casos de prueba que incluye los problemas de 30 a 512 ciudades. Otro trabajo [79] usa esta misma representación, pero define diferentes operadores de cruzamiento y un operador de inversión heurístico. Los operadores de cruzamiento de matrices (MX) intercambian las entradas de las dos matrices padres después que un cruzamiento de un punto o de dos puntos es realizado.

Representación de listas de posición. frecuentemente los problemas de ordenamiento y

programación están más relacionados con la posición actual de cada elemento de la cadena. En la representación de listas de posición se tiene en cuenta este principio [98].

Ejemplo: Para codificar la lista (bdceaf) se comienza con una lista de referencia (abcdef). Entonces cada carácter en la lista de posición corresponde a la posición de ese elemento en la permutación.

La ventaja de esta representación es que explícitamente almacena la posición donde cada elemento aparece en la lista ordenada. La dificultad es que el cruzamiento tradicional no se puede ejecutar. Las soluciones hijos válidas pueden ser logradas por romper las duplicaciones en alguna forma. Dos operadores fueron introducidos en [98], el cruzamiento de ruptura de duplicados # 1 (TBX1) y el cruzamiento de ruptura de duplicados # 2 (TBX2). El cruzamiento de ruptura de duplicados #1 (TBX1) incorpora un procedimiento matemático para aplicar un criterio de elección cuando existen dos elementos diferentes compitiendo por la misma posición. El cruzamiento de ruptura de duplicados # 2 (TBX2) es similar al anterior, pero en lugar de escoger dos subcadenas para ser intercambiadas solo se escogen dos elementos.

Para tener una idea más completa de los AG aplicados al TSP es conveniente darle especial mención a los siguientes trabajos que han contribuido a poder argumentar ciertas consideraciones sobre el TSP y los AG. Estos son [102][127] los cuales usan los AG para la optimización local del TSP. El algoritmo de Mühlenbein, que logra una evolución inteligente de los individuos, usa un algoritmo de optimización local, en el esquema de selección los individuos por encima de la adaptabilidad promedio tienen más hijos. Se aplican operadores de cruzamiento y mutación. El operador de cruzamiento es una versión del cruzamiento de orden (OX). En vez de empezar por el segundo punto de cruce, las ciudades desde el otro padre son copiadas en el mismo orden desde el principio de la cadena, omitiendo los símbolos ya escritos. Los resultados experimentales fueron alentadores encontrando un recorrido de 532 ciudades con una solución encontrada al 0.06% de la solución optima. De esto se deriva la siguiente experiencia: un buen programa evolutivo para el problema del agente viajero debe incorporar operadores de mejoramiento

local (al estilo de la mutación), basados sobre algoritmos para la optimización local, junto a un diseño cuidadoso de operadores binarios, es decir, operadores de cruzamiento, el cual incorporaría información heurística acerca del problema y por último podemos hablar de un operador que en ocasiones se puede prescindir de el o en otros casos el valor de este es muy pequeño solo para asegurar diferentes cromosomas dentro de la población cuando por medio de los otros operadores el valor de estos se vuelve idéntico, en el capítulo 4 y 5 se menciona los detalles de la aplicación de este operador genético.

CAPÍTULO 3
SISTEMAS CLASIFICADORES DE APRENDIZAJE.

3.1 Introducción.

En este capítulo muestro la teoría de los Sistemas Clasificadores de Aprendizaje (SCA) que servirá como introducción a la metodología que propongo en la presente tesis y la detallo en el siguiente capítulo.

Una de las críticas que se oyen más a menudo respecto a la Inteligencia Artificial, es que las máquinas no se pueden considerar "inteligentes" hasta que no sean capaces de aprender a hacer cosas nuevas y adaptarse a las nuevas situaciones, en lugar de limitarse a hacer aquellas actividades para las que sean programadas. Por ello, los investigadores de IA están destinando muchos esfuerzos en la investigación de sistemas de aprendizaje capaces de dotar a las máquinas de "inteligencia" y una forma es emplear el modelo del sistema clasificador de aprendizaje.

3.2 Sistema Clasificador de Aprendizaje.

Un sistema clasificador de aprendizaje es un programa que es capaz por un lado, de tomar decisiones en base a la experiencia acumulada en la resolución de problemas, y por otro, de perfeccionar sus habilidades. Un importante dilema a resolver en los sistemas de aprendizaje es cómo determinar que cambios son precisos y cómo hacerlos para perfeccionar sus habilidades. esto es conocido como el proceso de exploración y explotación del espacio de solución [143]. Un sistema clasificador de aprendizaje está formado por dos componentes como se expresa a continuación:

(i) un subsistema de tarea o componente de rendimiento. cuya conducta se modifica a lo largo del tiempo vía un proceso de aprendizaje y,

(ii) un subsistema de aprendizaje, responsable de observar al subsistema de tarea a lo largo del tiempo y efectuar los cambios oportunos en su conducta,

el dilema se puede expresar en cómo determinar el espacio de cambios estructurales posibles y las operaciones legales que permiten hacerlos, de forma que la conducta del sistema mejore. Resolver este dilema no es nada fácil, y sobre todo cuando afecta a tareas que requieren gran capacidad de adaptación robusta, como por ejemplo las desarrolladas en

entornos que varían en el tiempo, pobremente definidos y con continuos cambios. Algunas aproximaciones al dilema que están obteniendo gran éxito se han basado en modelos de organismos naturales, como por ejemplo sucede en el caso de las redes neuronales, la computación evolutiva, los algoritmos genéticos, etc.

Los Algoritmos Genéticos como se ha mencionado anteriormente son una familia de algoritmos de búsqueda adaptativos de propósito general que usan los principios inspirados en los modelos genéticos de las poblaciones naturales para evolucionar las soluciones a problemas. Su idea básica consiste en mantener una población de estructuras de conocimiento (cromosomas) que evolucionan a lo largo del tiempo a través de un proceso de competición y variación controlada. Cada estructura de la población representa una solución candidata al problema concreto, y tiene asociado una adecuación o probabilidad de supervivencia (fitness) que determina la valía de la estructura y que se usa en el proceso de competición para elegir las estructuras a partir de las cuales generar otras nuevas. Las nuevas son creadas usando operadores genéticos tales como el cruce y la mutación.

Los AGs han tenido gran éxito en problemas de optimización y búsqueda. La razón es su habilidad para explotar la información acumulada sobre el espacio de búsqueda, inicialmente desconocido, de cara a reconducir la búsqueda de soluciones sobre espacios de búsqueda más útiles, esto es, su robustez. Esta es su característica clave, y sobre todo en espacios de búsqueda complejos, grandes, y desconocidos, donde las técnicas clásicas de búsqueda, como las enumerativas, grasp, heurísticas, etc., son inapropiadas. Esto nos permite pensar en los AGs como una técnica apropiada para el diseño de sistemas de aprendizaje, en los cuales las acciones de control requeridas para controlar el subsistema de tarea varían rápidamente de forma que no pueden ser preestablecidas.

De acuerdo con De Jong [32] hay una gran variedad de aproximaciones a los sistemas de aprendizaje basadas en AGs. Las tres más importantes son las siguientes:

- Restringir los cambios estructurales a la modificación de parámetros que controlan la conducta del subsistema de tarea, y usar los AGs para desarrollar una estrategia que rápidamente localicen las combinaciones útiles de los valores de los parámetros.
- Usar AGs para cambiar las estructuras de datos complejas, como las agendas, que controlan la conducta del subsistema de tarea.
- Usar AGs para cambiar el subsistema de tarea, de modo que éste evolucione por si mismo.

Normalmente un subsistema de tarea se considera como un sistema de producción formado por un conjunto no ordenado de reglas, que representa la población de individuos que seria explotado y explorado por el subsistema de aprendizaje basado en AGs.

Históricamente hay dos modos de usar los sistemas de producción para representar al conjunto de la población:

- La Aproximación Pittsburg [118], donde cada miembro de la población representa un conjunto de reglas de producción, y por tanto, una población es un conjunto de conjuntos de reglas. Cada conjunto de reglas tiene asignado un fitness, calculado en base a una medida de rendimiento, el cual es usado por el AG para navegar en el espacio de posibles conjuntos de reglas.
- La Aproximación Michigan [75], donde cada miembro de la población representa una regla de producción individual, y por tanto, una población es un conjunto de reglas. Cada regla tiene asociado un fitness calculado en base a un algoritmo de asignación de crédito (algoritmo de bucket-brigade (BBA)) [75] que es usado en el modelo de clasificadores que expone Goldberg y Holland y la otra forma combinada de aprendizaje que ha dado buenos resultados en las investigaciones es el aprendizaje a través de Q-learning inicialmente propuesto por Watkings [130] y que está basado en la programación dinámica.

En la aproximación Pittsburg (Pitt) el AG evalúa implícitamente reglas individuales, mientras en la aproximación Michigan lo hace explícitamente de acuerdo a un algoritmo.

La aproximación Pitt conlleva un mayor gasto computacional, tanto en tiempo de proceso como en memoria, que la aproximación Michigan. Ahora, ¿cual de las dos aproximaciones es mejor en el sentido de ser más efectiva al evolucionar el subsistema de tarea? No está clara la respuesta, depende del caso de aplicación. Se acepta que la aproximación Pitt es más útil en entornos de trabajo off-line en los cuales cambios radicales de conducta son permisibles, mientras la Michigan es más útil en entornos de trabajo on-line en tiempo real en los cuales cambios radicales de conducta no pueden ser tolerados [32].

Dado que estamos interesados en tratar problemas en tiempo real en los cuales las condiciones de trabajo varían muy rápidamente, este estudio se centrará en los sistemas de aprendizaje basados en AGs inspirados en la aproximación Michigan, y más concretamente en una clase de sistemas de aprendizaje de propósito general conocidos con el nombre de Sistemas de Clasificadores (SCs) cuyos fundamentos fueron presentados por Holland en [72][73][75]. Ahora bien, existen versiones de SCs que usan la aproximación Pitt, como por ejemplo el propuesto por Carse y Fogarty [25] en entorno fuzzy, pero los más estudiados son los basados en la aproximación Michigan. No obstante, como argumenta Wilson y Goldberg [137] quizás en el futuro una síntesis de ambas aproximaciones sea necesaria de cara a alcanzar SCs más flexibles.
En base a todos estos conceptos es que se empieza a proponer la metodología la cual nos llevará al diseño de una aplicación con sistemas clasificadores y principalmente el aprendizaje en el componente descubridor de reglas que se hará por algoritmos genéticos y es de donde toma el nombre de Sistema Clasificador Genético (SCG).

3.3 Descripción del sistema clasificador.

Las ideas básicas de los SCs se presentaron por Holland en [72][73] y el primer SC se presentó por Holland y Reitman en [77]. Desde entonces diferentes tipos de SCs se han propuesto en la literatura [56][14][137][128][25][141].

Broker define a los SCs como [10] "Sistemas paralelos basados en reglas, de paso de mensajes, que son capaces de interactuar con el entorno o medio ambiente y de aprender reforzadamente mediante la asignación de recompensas y el descubrimiento de reglas (Figura 3.1)".

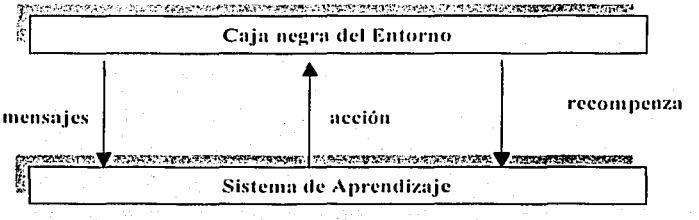


Figura 3.1: Diagrama básico del sistema clasificador de aprendizaje.

Típicamente operan en entornos tales como el mundo del robot, sistemas económicos, juegos como el ajedrez, etc., en los que se exhiben algunas de las siguientes propiedades:

- aparición constante de eventos nuevos, acompañados por grandes cantidades de ruido o datos irrelevantes;
- continuos, a menudo en tiempo real, requerimientos para actuar;
- objetivos definidos implícitamente o inexactamente;
- y, obtención de recompensas tras la realización de la secuencia de acciones.

Así mismo en la figura 3.2 podemos observar la estructura de un SCA donde se enfatizan los tres elementos básicos de un clasificador y que serán claves para la metodología propuesta: interacción con el medio ambiente, mecanismos de deducción de reglas a través

de reglas de producción y la incorporación de los Algoritmos Genéticos para encontrar nuevas reglas (Aprendizaje).

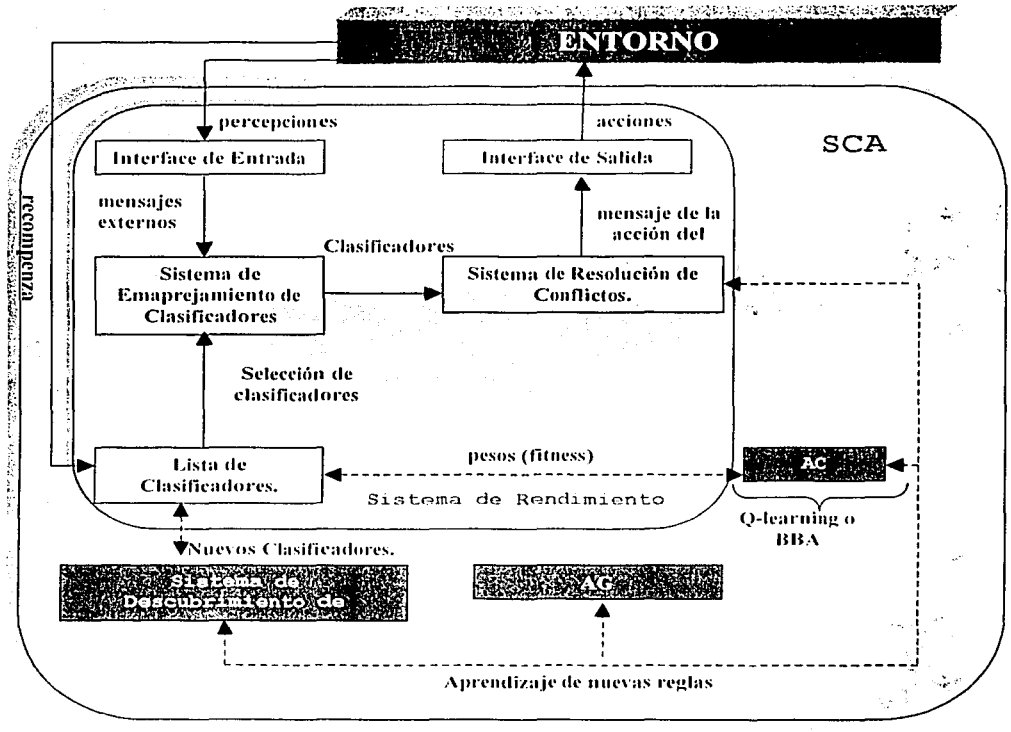


Figura 3.2: Estructura y componentes de interacción en los SCA.

Todos los componentes del modelo entran en funcionamiento en un ciclo de ejecución formado por los siguientes pasos:

1. Inicialmente, se crea un conjunto de clasificadores de manera aleatoria o mediante la necesidad de tener clasificadores nuevos de acuerdo al entorno tomando en cuenta la información del dominio del problema asignándoles el mismo peso a todos los clasificadores.
2. La interfase de entrada codifica las señales de entrada como mensajes externos.

3. Se agregan todos los mensajes externos a la lista de mensajes.
4. El sistema de emparejamiento o reconocimiento de patrones determina el conjunto de clasificadores que son cubiertos por los clasificadores que tiene información del entorno.
5. El sistema de resolución de conflictos (RC) resuelve los conflictos entre los clasificadores cubiertos y determina el conjunto de clasificadores activos.
6. Se borra la lista de mensajes.
7. Se colocan los mensajes disparados por los clasificadores activos en la lista de mensajes.
8. Se permite que los efectores que se activaron por algún mensaje efectúen sus acciones sobre el entorno. En caso de conflicto o inconsistencia se llama al sistema RC.
9. Si se detecta una señal de recompensa, se distribuye con el sistema de Asignación de Crédito (AC) aplicando el proceso de aprendizaje BBA ó Q-learning.
10. Si el sistema de AC está en un estado estable, se aplica el sistema de descubrimiento de clasificadores sobre el sistema de rendimiento.
11. Regresar al Paso 2.

Un SCA implementa el sistema de producción, trabajando en la memoria como una lista de mensajes, el cual son usualmente cadenas binarias. Algunos de estos mensajes son enviados por el sistema detector y ellos sugieren condiciones del medio ambiente. Otros mensajes son puestos por las reglas internamente a través de los mensajes internos por medio de los efectores. El sistema de producción es implementado en un SCA a través del ciclo condición-mensaje-acción. El ciclo condición-mensaje-acción tradicionalmente es el aprendizaje interno que es generado por un AG. En la mayoría de los SCA's las reglas (o clasificadores) tienen condiciones que son tomadas del alfabeto {1,0,#}. Si aparece # en una acción del clasificador, esta es interpretada como un carácter comodín, es decir que puede ser un 1 o 0.

El Sistema de Mensajes y Reglas (SMR) es la columna vertebral del componente computacional del SCA. La información fluye del entorno a través de los detectores (son los ojos y orejas del los sistemas clasificadores) donde este es decodificado a uno o mas mensajes de longitud finita. Estos mensajes del medio ambiente son puestos a una lista de mensajes de longitud fija, donde el mensaje entonces activa las reglas que son nombrados "clasificadores". Cuando es activado, un clasificador o regla pone el mensaje en la lista de mensajes. Estos mensajes pueden entonces llamar a otros clasificadores o ellos pueden causar una acción a ser tomados a través del accionador del sistema llamado "efectores" (salidas). En esta vía los clasificadores combinan las señales del medio ambiente hacia el interior para determinar que el sistema debería hacer y pensar a continuación. O sea este coordina el flujo de información donde estas son puestas (detectores) y procesadas (lista de mensajes y almacenamiento de clasificadores) y hacia donde se llama a ser activado (efectores).

Un mensaje dentro de un sistema clasificador es implementado como una cadena de caracteres del alfabeto longitud fija. Si limitamos al alfabeto binario, obtendríamos:

<mensaje> ::= {0,1}'

Un clasificador es una regla de producción y quedaría de la siguiente manera:

<clasificador>::= <condición> : <mensaje>

Y la condición es un reconocimiento simple de patrones, donde el carácter # es agregado al alfabeto:

<condición> ::= {0,1,#}'

En la tabla 3.1 tenemos algunos ejemplo de clasificadores con sus mensajes:

Indice	Clasificador (mensaje)
1	01## : 0000
2	00#0 : 1100
3	11## : 1000
4	##00 : 0001

Tabla 3.1: Ejemplo de clasificadores.

A su vez se han desarrollado diferentes modelos del manejo de los clasificadores y estos acrónimos dependen de su aplicación y sus variantes de cómo es modelado el aprendizaje. A continuación se presenta algunos de los modelos mas estudiados:

ACS	Anticipatory CS	Wolfgang Stolzmann, 1996.
CCS	Corporate CS	Andy Tomlinson & Larry Bull, 1999.
CSM	CS with Memory	Hayong Harry Zhou, 1985
ECS	Holme's EpiCS	Holmes, 1996.
FCS	Fuzzy CS	Takeshi Furuhashi, Ken Nakaoka & Yoshiki, 1994.
HCS	Hierarchical CS	Lingyan Shu and Jonathan Schaeffer, 1989.
LCS	Learning CS	Término genérico.
OCS	Organizational CS	Jason Wilcox,1995, Takadama, 1999.
PCS	Predictive CS	Piet Spiessens, 1990.
SCS	Simple CS	David Goldberg, 1989.
VCS	Variable CS	Lingyan Shu & Jonathan Schaeffer, 1989.
XCS	Special CS	Stewart W. Wilson, 1995.
ZCS	Zeroth-level CS	Stewart W. Wilson, 1994

Cada parte del SCA cumple una serie de funciones y, por tanto, los SCs están activos en tres niveles funcionales:

- El nivel de rendimiento, desarrolla actividades como la interacción medioambiental y el procesamiento de mensajes.
- El nivel de Asignación de Crédito, desarrolla la actividad de aprendizaje mediante la modificación de pesos [56][73] o por la precisión de cada regla [142].
- El nivel de descubrimiento de reglas, desarrolla la actividad de aprendizaje mediante el descubrimiento de nuevas reglas por medio de un AG.

a) **Sistema de Rendimiento.-** Es el encargado de interactuar con el exterior. Está compuesto de seis elementos básicos:

1. Interfase de Entrada. Está formada por detectores (al menos uno) que se encargan de captar el estado actual del entorno y codificarlo en mensajes estándar (mensajes de entrada). Normalmente, todos los mensajes suelen tener la misma longitud y se representan usando el alfabeto binario {0,1} es decir, los mensajes se representan mediante cadenas binarias de longitud fija.
2. Interfase de Salida. Está formada por efectores (al menos uno) que se encargan de convertir los mensajes de acción o salida en acciones que actúan modificando el estado del entorno.
3. Lista de Clasificadores. Es el conjunto de reglas o clasificadores que representa el conocimiento que posee el sistema de aprendizaje. Cada clasificador se representa como una cadena de símbolos definidos sobre el alfabeto {0,1,#} con el formato Condición / Acción, de modo que la parte condición especifica los mensajes que satisfacen o activan el clasificador (mensajes externos) y la parte acción especifica los mensajes (mensajes internos) que son enviados cuando el clasificador es satisfecho. Normalmente en cada ciclo de acción del SC un número indeterminado de clasificadores se disparan en paralelo [56][12].
4. Sistema de recombinación de patrones. Es el encargado de recombinar los mensajes con las condiciones de los clasificadores e identificar qué clasificadores se satisfacen o cubren.
5. Lista de Mensajes. Contiene todos los mensajes que existen en cada momento circulando en el SC, tanto los generados en la interfase de entrada como los generados por los clasificadores que disparan (en la presente tesis se evita esta parte y los mensajes pasan directamente a cada ciclo que le corresponda de acuerdo al modelo que propone Wilson [142]).
6. Sistema Resolutor de Conflictos (RC). Es el encargado de resolver los conflictos que puedan surgir entre los clasificadores en un ciclo de funcionamiento del SC. Se pueden presentar dos situaciones conflictivas:

- Cuando el número de clasificadores satisfechos es mayor que el número que puede albergar las lista de mensajes.
- Cuando las acciones propuestas a los efectores por los mensajes de acción colocados por los clasificadores son inconsistentes, por ejemplo "alinear a la derecha" y a la vez exigir "alinear a la izquierda".

En ambos casos el sistema RC decide qué clasificadores considera tomando como clave alguna medida de utilidad asociada a los mismos, como puede ser, por ejemplo, la adecuación a la situación y el peso asociado o utilidad pasada.

El sistema RC, estableciendo una analogía económica, funciona como un mercado de oferta y demanda entre los clasificadores. Los clasificadores recombinados ofertan una porción de sus medidas de utilidad y, los conflictos son resueltos usando una distribución de probabilidad sobre esas ofertas. Los clasificadores que ofertan una mayor porción tienen más posibilidades de disparar, y por tanto, de poner sus mensajes en la lista de mensajes. Cuando un clasificador dispara ve decrementado su peso asociado en un valor equivalente al ofertado. Finalmente, cuando se obtiene una recompensa del entorno los clasificadores que han disparado se la reparten de acuerdo a su participación en la acción desencadenada usando algún algoritmo de AC. Es por ello que se dice que los sistemas de RC y AC constituyen el motor de inferencia de los SCs y por tanto, sus actividades están interrelacionadas.

Un esquema descriptivo del sistema de rendimiento o de ejecución es presentado en la figura 3.3.

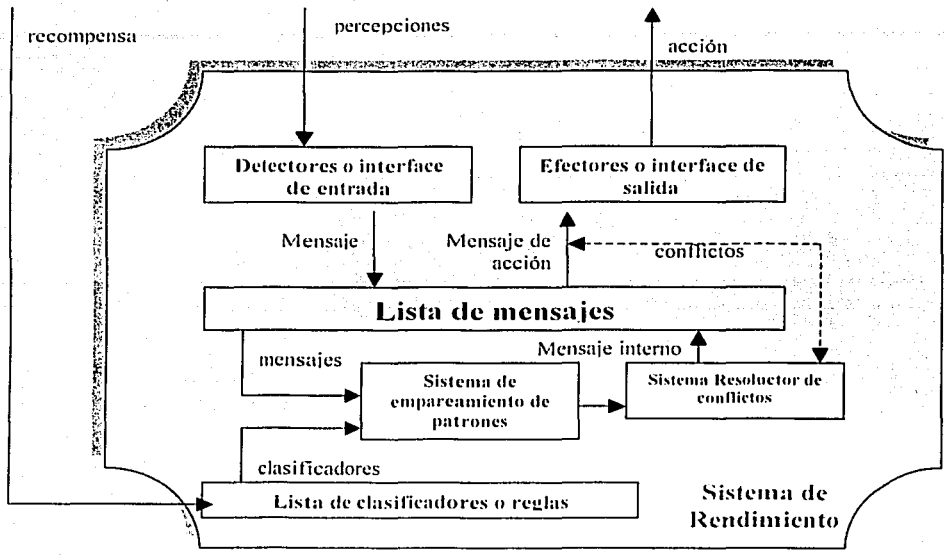


Figura 3.3: Sistema de rendimiento del sistema clasificador de aprendizaje.

b) Sistema de Asignación de Crédito (AC).- El aprendizaje de un SC está compuesto por dos procesos:

El aprendizaje de un SC está compuesto por procesos:

- El proceso de aprendizaje desarrollado en el sistema de AC a través del BBA o Q-learning.
- El proceso de aprendizaje desarrollado en el sistema descubrimiento de clasificadores a través del algoritmo genético.

En el primero, se aprende el uso del conjunto de clasificadores por medio de la recompensa (feedback) procedente del entorno y, en el segundo, el aprendizaje consiste en crear nuevos,

En el primero, se aprende el uso del conjunto de clasificadores por medio de la recompensa (feedback) procedente del entorno y, en el segundo, el aprendizaje consiste en crear nuevos, y posiblemente más útiles clasificadores, usando la experiencia pasada, es decir, los antiguos clasificadores. Por tanto, la principal tarea del sistema de AC incluye la actividad del aprendizaje por modificación y ajuste de los pesos de los clasificadores.

Tradicionalmente se vienen usando dos diferentes esquemas para controlar la acción del sistema de AC: *Bucket Brigade Algorithms (BBA) o Q-learning.*
El Algoritmo de BBA es un esquema de aprendizaje local que precisa pocos requerimientos computacionales a diferencia del Q-learning, tanto de memoria como de tiempo de CPU.
En un sistema clasificador con este esquema, el BBA se enlaza con el sistema de RC, formando el mecanismo que conduce la competición entre los clasificadores del sistema clasificador, conocido con el nombre del sistema AC/RC. En cada proceso de competición, cada clasificador C_j hace su oferta $Oferta_j$.

$$Ofertaj = b \bullet Peso_j \bullet Especificidad_j \quad \text{"sin soporte"}$$

ó

$$Ofertaj = b \bullet Peso_j \bullet Especificidad_j \bullet Soporte_j \text{"con soporte"}$$

Donde b es una constante pequeña llamada factor de riesgo (por lo general 1/10), $Peso_j$ es el peso del clasificador C_j (inicializado con el mismo valor para todos los clasificadores), $especificidad_j$ es la especificidad de C_j, y

$$\text{Especificidadj} = \frac{\text{Longitud del clasificador} - \text{número de bits con NO \#}}{\text{Longitud del clasificador}}$$

$$Soportej = \sum_{m:\ \exists T \varepsilon T_j (m \varepsilon T)} Intensidad\ (mensaje)$$

Es el soporte de C_j, donde T_j es el conjunto de todos los mensajes de la lista de mensajes que satisfacen a C_j, Intensidad(mensaje) es la intensidad del mensaje m, esto es, un valor igual a la oferta del clasificador que envió el mensaje m.

La probabilidad de que un clasificador gane la competición viene dada por la expresión siguiente:

$$\frac{\text{Oferta}_j}{\sum_{C_k \varepsilon S} \text{Oferta}_k}$$

donde S es el conjunto de todos los clasificadores satisfechos. Un clasificador ganador Cj, reduce su peso en el valor de su oferta, es decir,

$$\text{Peso}_j = \text{Peso}_j - \text{Oferta}_j,$$

y esa cantidad Oferta_j se reparte entre los predecesores, es decir, entre los clasificadores cuya actividad o actividades anteriores posibilitan que C_j sea activo, de acuerdo a la siguiente expresión:

$$\text{Peso}_j = \text{Peso}_j + \frac{\text{Oferta}_j}{|P_j|} \quad \forall\ C_j \varepsilon P_j,$$

donde Pj es el conjunto de predecesores de Cj, esto es,

$$P_j = \{C_k : \exists T \varepsilon T_j \ \exists m \varepsilon T(C_k \text{ envió } m)\}$$

Lógicamente si un clasificador no oferta lo suficiente para ganar, él no paga nada. Cuando se recibe una recompensa externa, ésta es repartida igualmente entre los clasificadores que enviaron los mensajes "acción" que activaron los efectores.

Para clarificar el uso del BBA muestro a continuación un ejemplo típico que es presentado por Goldberg en su libro [56]:

		Tiempo = 1				Tiempo = 2				Tiempo = 3			
Indice	Clasificador	Peso	Mensaje	Match	Oferta	Peso	Mensaje	Match	Oferta	Peso	Mensaje	Match	Oferta
1	01##:0000	200		E	20	180	0000			220			
2	00#0:1100	200				200		1	20	180	1100		
3	11##:1000	200				200				200		2	20
4	##00:0001	200				200		1	20	180	0001	2	18
	(Entorno)	0	0111			20				20			

		Tiempo = 4				Tiempo = 5				Payoff
1	01##:0000	220				220				220	
2	00#0:1100	218				218				218	
3	11##:1000	180	1000		18	196				196	
4	##00:0001	162	0001	3	16	156	0001			206	50
	(Entorno)	20				20				20	

c) El Sistema de Descubrimiento de Clasificadores.- El proceso de descubrimiento de clasificadores en un SCs se realiza por AGs. es decir, el sistema desarrolla su proceso de aprendizaje generando nuevos clasificadores aplicando un AG sobre el conjunto de clasificadores, en este proceso en los sistemas clasificadores es de suma importancia el como es implementado el AG, pues se corre el gran riesgo de perder reglas (al emplear los distintos operadores genéticos) buenas y desviarnos de la solución, mientras que la ganancia al emplear el AG es sin duda que nos permitirá obtener un mayor abanico de soluciones posibles.

Básicamente el funcionamiento del AG en los SCA es como sigue:

- un AG selecciona los clasificadores de mayor adecuación o fitness como padres para a partir de ellos, formar nuevos clasificadores usando los operadores genéticos y fragmentos de su composición.

- El fitness de un clasificador es determinado por su peso asociado, asignado por el sistema de AC, en vez de mediante una función de adecuación. En los clásicos SCs la población del AG suele ser una porción del conjunto de clasificadores (aquellos con fitness más alto) y,
- con el fin de preservar el sistema de rendimiento, el AG sólo puede reemplazar un subconjunto del conjunto total de clasificadores, es decir, un subconjunto de los "m" peores clasificadores se reemplaza y de acuerdo a estudios realizados que hicimos en ILLIGAL recomendamos que m tome el valor entre el 10% y 20% de los que tengan un valor menor de fitness de la población seleccionada para evitar la pérdida de buenos clasificadores y por otro lado la de conservar la evolución de la población.

1. El sistema de descubrimiento de clasificadores entra en funcionamiento cuando el sistema AC alcanza una situación de equilibro o uniformidad, es decir, cuando el peso de cada clasificador refleja realmente su utilidad. Esto suele suceder normalmente entre 1000 y 10000 ciclos de aplicación del sistema AC, y por tanto con una frecuencia muy baja.

3.4 Codificación en los sistemas clasificadores.

Muchas críticas se han hecho a los sistemas clasificadores en cuanto a la selección de una codificación para las soluciones del problema de optimización. Tradicionalmente, las cadenas binarias han sido utilizadas por la simplicidad de su implementación y porque esta representación maximiza el número de esquemas producidos representadas por medio de las reglas de producción. Cuando el alfabeto usado para codificar las soluciones es superior al binario, es necesario redefinir los operadores de cruzamiento y mutación adecuadamente.

Un número grande de problemas de optimización tiene variables continuas que asumen valores reales. Una técnica común para codificar variables continuas en un alfabeto binario usa una codificación entera de punto fijo, cada variable es codificada usando un número fijo de bits binarios. Los códigos binarios de todas las variables del problema son

concatenados para obtener la cadena representante de una solución del espacio de búsqueda. En este punto algunos problemas se presentan y es que la codificación binaria de una variable provoca diferencias substanciales entre la distancia de las codificaciones a nivel de bits (genotipo) y las distancias de las variables propiamente (fenotipo). Este problema se enfrentó con la utilización de los códigos de Gray (de cambio mínimo). Otros tipos de codificaciones son reportadas en la literatura como es el caso de la codificación dinámica que permite variar el número e interpretación de los bits asignados a un parámetro particular a través de la corrida, las representaciones de longitud variable, la codificación delta [136] cuyos bits en las cadenas expresan una distancia con respecto a alguna solución parcial previa.

CAPÍTULO 4

METODOLOGÍA PARA EL DISEÑO DE APLICACIONES CON SISTEMAS CLASIFICADORES GENÉTICOS.

4.1 Introducción.

Los métodos generales desarrollados para la resolución de problemas y técnicas de búsqueda al inicio de la era de la inteligencia artificial demostraron no ser suficientes para resolver los problemas orientados a las aplicaciones, ni fueron capaces de satisfacer los difíciles requerimientos de la investigación. A este conjunto de métodos, procedimientos y técnicas, se le conoce como Inteligencia Artificial Débil.

La principal conclusión que se deriva de este trabajo fue que los problemas difíciles sólo podrían ser resueltos con la ayuda del conocimiento específico acerca del dominio del problema. La aplicación de estas ideas dio lugar al desarrollo de aplicaciones con técnicas heurísticas como son los algoritmos genéticos, sistemas clasificadores, sistemas basados en conocimiento y al aparecimiento de la ingeniería cognoscitiva, como ramas de la inteligencia artificial. La definición de estos sistemas puede ser dada por: "sistemas computarizados capaces de resolver problemas donde el dominio del conocimiento es limitado o bien no existe el mismo". La solución es esencialmente nueva y de hecho en muchas de las ocasiones no existe manera de probar si la solución obtenida es la óptima o que tan cercana a ella se encuentra. El simple concepto dado, puede causar confusión ya que muchos sistemas basados en programas convencionales podrían ser incorrectamente categorizados como sistemas evolutivos y en muchas de las ocasiones se puede confundir el problema con los sistemas basados en conocimiento. Esta inconsistencia puede ser aclarada, sobre la base de tres conceptos fundamentales que distinguen a los SCGs de los programas algorítmicos convencionales y de los programas generales basados en búsqueda (inteligencia artificial débil).

1. La separación que existe entre el conocimiento, y la forma cómo éste es utilizado.
2. El uso de conocimiento específico de un determinado dominio.
3. La naturaleza heurística, antes que algorítmica del conocimiento utilizado.

Dicho de una manera simple, los programas convencionales utilizan algoritmos para resolver problemas, mientras que los sistemas clasificadores como parte de los Sistemas Evolutivos (algoritmos genéticos y sistemas clasificadores) resuelven problemas donde las

soluciones algorítmicas no existen o son muy costosas para ser implementadas. A continuación entraremos al detalle de la metodología que propongo y con ello cumplir el principal objetivo de la tesis.

4.2 Metodología Propuesta.

El Proceso de Desarrollo de la metodología queda de la siguiente manera (figura 4.1a y 4.1b):

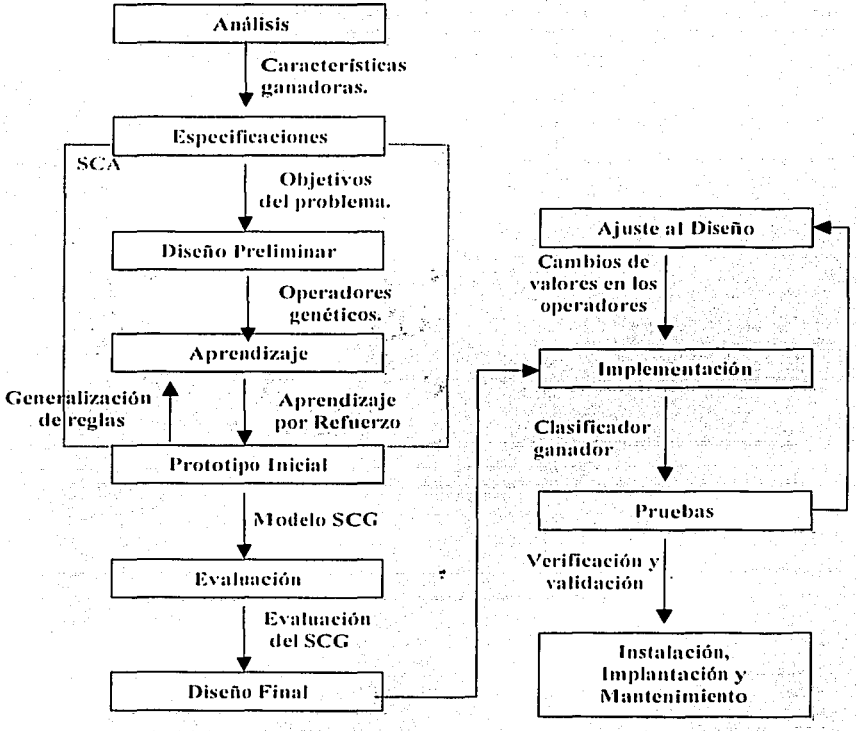


Figura 4.1a: Proceso de Desarrollo de la Metodología SCG.

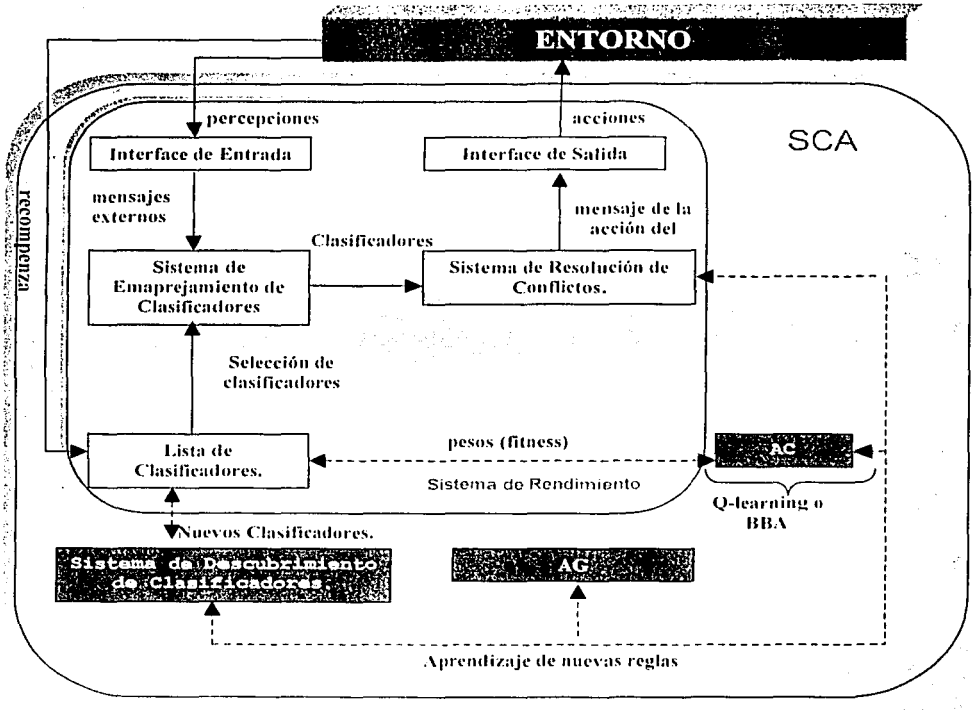


Figura 4.1 b: Incorporación del proceso del SCA a la metodología.

4.2.1 Análisis del Problema.

Como primer paso es necesario evaluar el problema y los recursos disponibles para determinar la aplicabilidad de una solución. Se debe realizar un análisis costo-beneficio del SCG propuesto para saber si su desarrollo puede ser garantizado. Puede también requerirse de una investigación de mercado o un examen profundo del propósito de la aplicación solicitada, para determinar la efectividad del costo del sistema. Aquí mismo se analiza si es conveniente o posible resolver el problema por alguna otra técnica. Así mismo los SCG trabajan donde otras técnicas no lo pueden hacer, es decir son buenos al igual que los Algoritmos Genéticos en terrenos de comportamiento duro, es decir, del tipo NP-Completo. Es necesario recalcar que típicamente los Sistemas Clasificadores Genéticos operan en entornos tales como el mundo de robots, sistemas económicos y juegos, como el ajedrez, en los que se exhiben algunas de las siguientes propiedades:

1. aparición constante de eventos nuevos, acompañados por grandes cantidades de ruido o datos irrelevantes;
2. requerimientos continuos para actuar, a menudo en tiempo real;
3. objetivos definidos implícitamente o inexactamente; y,
4. obtención de recompensas tras la realización de la secuencia de acciones.

En la siguiente tabla se inicia el análisis del problema y donde se deja de manifiesto contra que otras técnicas se está compitiendo, reflejando sus diferencias entre ellas.

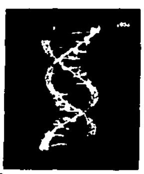
Tabla 4.1: Hoja de Análisis del problema

		Fecha:	
Descripción del Problema:	**Técnicas consideradas para el análisis del problema.**	**Puntos más importantes para considerar en un problema en los SCGs.**	**Observaciones:**
	**Técnicas Enumerativas:** Lineal ( ) Exhaustiva. ( ) Otra:_____ ( ) _____ **Técnicas :** SCA (AGs) ( ) Tabú ( ) Hill-Climbing ( ) Simulated Annealing ( ) Grasp ( ) Otra:_____ ( ) _____	1. Facilita trabajar con una codificación de las soluciones en vez de trabajar con la solución en si. SI ( ) NO ( ) Inseguro ( ) 2. Facilita trabajar con poblaciones en paralelo. SI ( ) NO ( ) Inseguro ( ) 3.- Se emplea una función objetivo en vez de utilizar gradientes u otro mecanismo de maximización. SI ( ) NO ( ) Inseguro ( ) 4. Emplea reglas probabilísticas de transición en vez de reglas determinísticas. SI ( ) NO ( ) Inseguro ( ) 5. El problema puede variar de tamaño de incógnitas en: O(n) ( ) O(n!) ( ) O(nlog n) ( ) O(a^n) ( ) O(n^a) ( ) O(n^2) ( ) a= constante; n=variación.	_____ _____ _____ _____ _____ _____ _____ _____ _____ _____
laboró:_____		**Revisó:**_____	

El primer planteamiento a resolver para el análisis será la de determinar si el problema a tratar puede ser resuelto a través de tantas técnicas tradicionales ya probadas (métodos analíticos, simplex, transporte, etc.) o de lo contrario determinar si puede ser tratado por alguna técnica de optimización combinatoria heurística como los Sistemas Clasificadores de Aprendizaje.

Es necesario recalcar que un mismo problema dependiendo de lo que se desea resolver puede ser o no resuelto con una misma técnica, por ejemplo si se desea localizar cual sería

Tabla 4.1: Hoja de Análisis del problema

		Fecha:	
Descripción del Problema:	**Técnicas consideradas para el análisis del problema.**	**Puntos más importantes para considerar en un problema en los SCGs.**	**Observaciones:**
	**Técnicas Enumerativas:** Lineal ( ) Exhaustiva. ( ) Otra:________ ( ) **Técnicas :** SCA (AGs) ( ) Tabú ( ) Hill-Climbing ( ) Simulated Annealing ( ) Grasp ( ) Otra:________ ( )	1. Facilita trabajar con una codificación de las soluciones en vez de trabajar con la solución en si. SI ( ) NO ( ) Inseguro ( ) 2. Facilita trabajar con poblaciones en paralelo. SI ( ) NO ( ) Inseguro ( ) 3.- Se emplea una función objetivo en vez de utilizar gradientes u otro mecanismo de maximización. SI ( ) NO ( ) Inseguro ( ) 4. Emplea reglas probabilísticas de transición en vez de reglas determinísticas. SI ( ) NO ( ) Inseguro ( ) 5. El problema puede variar de tamaño de incógnitas en: $O(n)$ ( ) $O(n!)$ ( ) $O(n \log n)$ ( ) $O(a^n)$ ( ) $O(n^a)$ ( ) $O(n^2)$ ( ) * a= constante: n=variación.	
Elaboró:		**Revisó:**	

El primer planteamiento a resolver para el análisis será la de determinar si el problema a tratar puede ser resuelto a través de tantas técnicas tradicionales ya probadas (métodos analíticos. simplex, transporte, etc.) o de lo contrario determinar si puede ser tratado por alguna técnica de optimización combinatoria heurística como los Sistemas Clasificadores de Aprendizaje.

Es necesario recalcar que un mismo problema dependiendo de lo que se desea resolver puede ser o no resuelto con una misma técnica. por ejemplo si se desea localizar cual sería

la mejor manera de invertir cierta cantidad de dinero en la bolsa de valores y nos encontramos en que la decisión a tomar es en un lapso de tiempo pequeño, entonces puede ser resuelto los sistemas Clasificadores, pero si el número de variables se reduce y la decisión se puede tomar en un lapso mayor entonces la decisión de usar otra técnica tradicional gana terreno.

A continuación se explica a que objetivo se desea llegar con el hecho de emplear un algoritmo genético a nuestro problema y una vez que se justifique el empleo del AG se incorporarán los conceptos de los SCAs para darle "inteligencia a nuestro problema" logrando así un Sistema Clasificador Genético (SCG).

Justificación de emplear estrategias evolutivas.- Uno de los problemas computacionales a resolver en optimización combinatoria es la explosión combinatoria. La explosión combinatoria se encuentra en situaciones donde las elecciones están compuestas secuencialmente, es decir, dado un conjunto de elementos se pueden obtener diferentes arreglos ordenados de estos, permitiendo una vasta cantidad de posibilidades. Situaciones de este tipo ocurren en problemas de inversión financiera, manejo de inventarios, diseño de circuitos integrados, manejo de recursos hidráulicos, mantenimiento de sistemas, etc.

Una característica recurrente en los problemas de optimización combinatoria es el hecho de que son muy "fáciles" de entender y de enunciar, pero generalmente son "difíciles" de resolver. Podría pensarse que la solución de un problema de optimización combinatoria se restringe únicamente a buscar de manera exhaustiva el valor máximo o mínimo en un conjunto finito de posibilidades y que usando una computadora veloz, el problema carecería de interés matemático, sin pensar por un momento, en el tamaño de este conjunto.

Los intentos por tratar con el problema de la explosión combinatoria han encontrado muchos obstáculos. Por ejemplo, no es suficiente contar con un "conocimiento experto" para manejarlos de manera efectiva, de igual manera no es suficiente confiar en el poder computacional de alta velocidad de las súper computadoras. Algunos problemas clásicos donde la explosión combinatoria prevalece, muestran que un intento por generar todas las alternativas relevantes por computadora no es una tarea factible. Así, por ejemplo, en el problema del agente viajero, en el cual se tiene que salir de una ciudad y regresar a la misma después de haber visitado (con costo mínimo de viaje) todas las demás ciudades, si

se tienen n ciudades en total que recorrer entonces existen (n-1)! soluciones factibles, y si una computadora que pudiera ser programada para examinar soluciones a razón de un billón de soluciones por segundo: la computadora terminaría su tarea, para n = 150 ciudades (que es un problema pequeño para muchos casos prácticos) varios años.

Al respecto Glover [53] expone: "la clave para tratar con tales problemas es ir un paso más allá de la aplicación directa de la destreza y del conocimiento del experto, y generar recursos para un procedimiento especial (o marco) el cual monitorea y dirige el uso de esta habilidad y conocimiento. Careciendo de tal procedimiento, las reglas expertas se pueden empantanar, permitiendo un punto donde ninguna mejora puede percibirse, a menos de que existan alternativas superiores."

En la actualidad, la investigación se ha dirigido hacia el diseño de buenas heurísticas, es decir, algoritmos eficientes con respecto al tiempo de cómputo y al espacio de memoria, y con cierta verosimilitud de entregar una solución "buena" esto es, relativamente cercana a la óptima mediante el examen de sólo un pequeño subconjunto de soluciones del número total, esto de acuerdo a como lo define Colin R. Reeves [110] "Una heurística es una técnica la cual busca soluciones cercanas al óptimo a un costo computacional razonable sin poder garantizar su optimalidad y en muchos casos sin poder establecer que tan cercana del óptimo está una solución factible en particular".

El principal problema de algunos algoritmos heurísticos es la dificultad de escapar de la optimalidad local como pasa con la técnica Hill-climbing. Lo anterior ha propiciado que el enfoque de la inteligencia artificial haya revivido como solución de problemas que requieren de la búsqueda heurística. Recientemente varias aproximaciones han surgido del manejo de problemas de decisión complejos, como son: algoritmos genéticos, redes neuronales, recocido simulado, búsqueda tabú, análisis de objetivos y búsqueda dispersa, entre otros.

Existen varios métodos para determinar si es bueno resolver el problema con determinada técnica o metodología y muchas de las veces son usadas dichas técnicas por la sencilla razón de que: no se analizaron si la técnica que se está usando es una de las mejores (no necesariamente la mejor opción, pues es difícil determinarlo) y la otra es que se tiene experiencia en resolver problemas similares con la técnica que se conoce y se evalúa en

base a costo-beneficio en el sentido de que probablemente exista no solo una técnica mejor, sino dos o tres, pero esto llevaría mucho tiempo en conocerlas y a lo mejor es demasiado tarde para el problema en función.

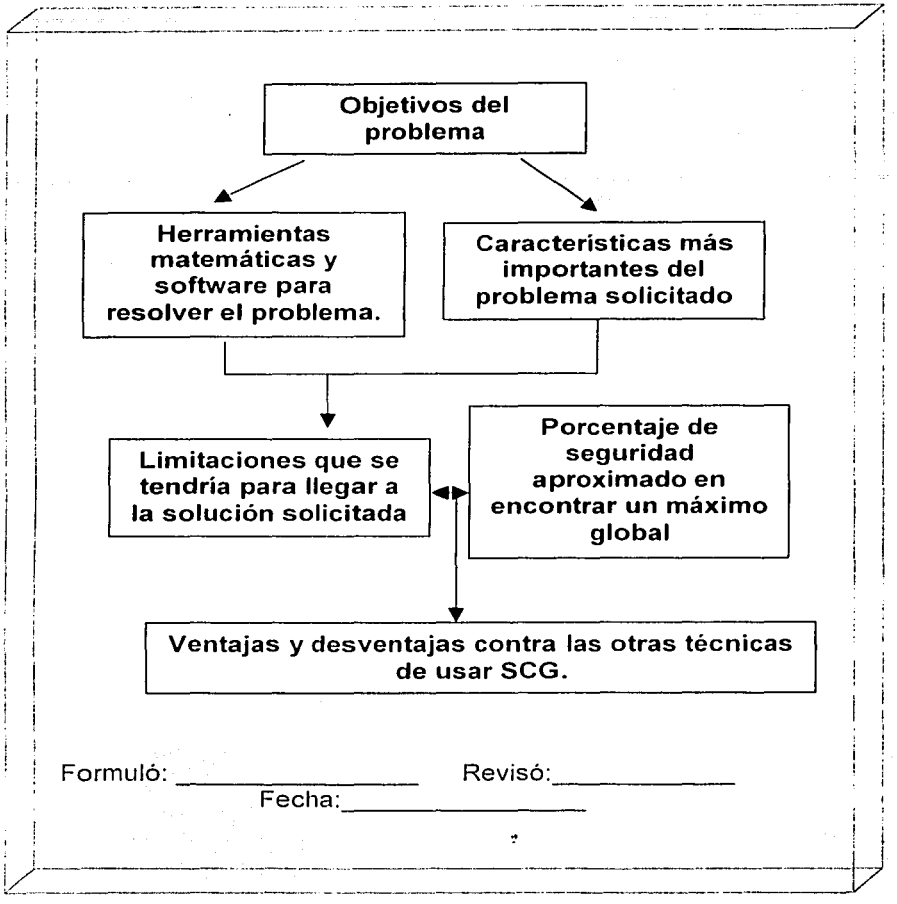
El SCA, al igual que otras formas de software, tiene como objetivo crear soluciones computacionales a problemas diversos. A pesar de que los SCA se basan principalmente en procesos heurísticos antes que algorítmicos, el desarrollo de un SCA tiene un Proceso de Desarrollo similar al de un sistema de software convencional. Sin embargo, existen varias diferencias significativas en el desarrollo de una aplicación donde intervienen los SCAs (como parte de la ingeniería cognoscitiva y la ingeniería de software).

- Una de las mayores diferencias es el tipo de conocimiento que se representa. La ingeniería de software involucra la representación de procedimientos algorítmicos bien definidos y típicamente bien conocidos por muchas personas; mientras que en los SCAs se involucra la representación de conocimiento heurístico amplio, impreciso, mal definido, que está almacenado en la mente de pocos expertos y hasta en ocasiones no se sabe mucho del problema. Debido a que el conocimiento heurístico no es ampliamente conocido ni entendido, tienen que utilizarse ciertas técnicas para lograr transferirlo desde las mentes de los individuos que lo poseen, hasta una representación computarizada, como sucede también en los sistemas basados en conocimiento.
- Otra de las diferencias significativas está relacionada con la naturaleza y la cantidad de conocimiento. Mientras la naturaleza y la cantidad del conocimiento requerido para resolver un problema algorítmico tradicional pueden ser razonablemente bien estimados, este no es el caso para los SCAs. Típicamente, la naturaleza y la cantidad de conocimiento requeridos dentro de un SCA para resolver un problema no es bien conocido, aún por los propios expertos y es necesario basarse en procesos estocásticos. Esto dificulta la predicción del esfuerzo total requerido para desarrollar un SCA y es por ello que es importante combinar la predicción del conocimiento que usan los modelos clasificadores con los avances del cálculo de nuevas reglas con AGs dando como resultado un Sistema Clasificador Genético (SCG). Además, puede ser difícil llegar a un diseño

adecuado desde las etapas iniciales del proyecto, dando lugar al problema denominado dilema del cambio y a la definición de los problemas NP-Completos [36].

4.2.2 Especificación de Requerimientos.

Formalizar y poner por escrito lo que fue adquirido durante la fase de análisis. Esto permite determinar los objetivos del proyecto, ojalá, de una manera inequívoca y establece los medios para obtener dichos objetivos. La experiencia acumulada en el desarrollo de SCG demuestra que sin tener especificaciones no es posible diseñar un SCG de real utilidad. El documento de especificaciones debe plantear claramente y discutir los objetivos y las características del sistema, el entorno del usuario, y las limitaciones, pues es necesario que el usuario conozca claramente las debilidades de emplear esta técnica heurística y a su vez el acercamiento a la solución que puede arrojar y mas por tratarse de una técnica que es considerada como heurística y se debe de hacer notar las ventajas y las desventajas que ofrece esta misma llenando la hoja de especificaciones que se muestra a continuación para un mejor control del diseño.



a) Objetivos del problema.

b) Herramientas matemáticas y software para resolver el problema.- Si es requerido alguna característica especial de software es el momento de considerarlo para que sea parte de la planeación del proyecto y del mismo lenguaje a considerar, hay que hacer notar que actualmente los desarrollos que se han hecho han sido primordialmente en lenguaje C y Pascal (por lo general lenguajes procedurales) y aun no ha habido desarrollos en lenguajes declarativos como el Prolog (trabajo de investigación a desarrollar).

c) Características más importantes del problema solicitado.- Detallar cada una de las partes que intervienen en el desarrollo para ver alcances y necesidades que se requieran y con ello poder elaborar un plan de trabajo que este lo mas cercano a la realidad.

d) Limitaciones que se tendría para llegar a la solución solicitada.- Debemos de tener un gran dominio y certeza de los problemas que pueden surgir en el desarrollo de la aplicación por medio de sistemas clasificadores con la idea de decidir si es conveniente o no continuar con el proceso de desarrollo o definitivamente no se considera como buena opción tratarlo por esta técnica, entre las principales limitaciones que se tienen es que el comportamiento de las variables no estén bien definidas esto nos ocasionará problemas desde diseñar la forma del clasificador y obtener la función de aptitud.

e) Porcentaje de seguridad aproximado en encontrar un máximo global.- Una vez trabajado con los puntos anteriores tendremos la confianza de que esta alternativa nos llevará a una buena solución y debemos comprometernos a que la solución arrojada por el sistema esté dentro de un intervalo de confianza que nos aseguré la inversión y la decisión de continuar.

f) Ventajas y Desventajas de usar SCGs.- Dejar por escrito cada una de las ventajas y desventajas que tendremos con este desarrollo para clarificar el compromiso y por otra parte que el riesgo este compartido.

g) Formuló:_____ Revisó:_____

Autorizó:_____

4.2.3 Diseño Preliminar.

Esta etapa considera las decisiones de alto nivel necesarias para preparar y desarrollar rápidamente el prototipo inicial, así como el proceso que se emplea en la etapa del aprendizaje de los clasificadores. Se hace un bosquejo del tipo de codificación, tamaño y forma que tendrán nuestras variables (cromosoma o clasificador) y los valores que pueden ir tomando cada una de ellas. Específicamente, esta etapa determina el paradigma de representación del conocimiento (clasificador), sacando provecho de emplear codificación binaria (o Gray), y por otro lado considera la herramienta escogida para construir el prototipo. En esta etapa puede ser necesaria una considerable recopilación de conocimiento tanto de los expertos, como de fuentes impresas, para poder tomar decisiones sólidas acerca del paradigma de representación del conocimiento que será impuesto para cada uno de los clasificadores (decisiones como si será necesario emplear diferentes tamaños de clasificadores o solamente uno) y la herramienta necesaria. En la actualidad se tienen muchas dudas sobre el tamaño ideal de cada uno de los operadores genéticos y cada año surgen nuevos enfoques para atacar este problema. De acuerdo a diferentes experimentos realizados nuestra propuesta para el tamaño de la población es:

En el caso de codificación es Binaria,

SI	longitud del clasificador ≤ a 25 bits
Entonces	tamaño de la población máximo será de 50 individuos
ELSE	tamaño de la población = 2 * tamaño del clasificador;

Si la codificación es Alfanumérica el tamaño de la población siempre será el doble del tamaño del clasificador.

Las variables de decisión asociadas con el modelo de optimización a ser resueltos se codifican para formar un cromosoma o clasificador que a su vez tendrán la forma de una regla de producción (condición-acción). El fitness o la función de ganancia de cada miembro de la población se determina por una función objetivo basada en penalizar a los

clasificadores más lejanos a la solución. La parte donde intervienen los AGs es encontrar la solución óptima de cada clasificador usando tres operadores básicos:

1. Selección.
2. Cruzamiento o reproducción, y
3. Mutación.

Que son los operadores que nos garantizan la construcción exponencial de nuestra solución como lo expresa el teorema de esquemas con los Bloques constructores. Inicialmente, los cromosomas o clasificadores son seleccionados aleatoriamente, los que poseen un fitness relativamente alto son asignados a entrar a la población de reproducción. El aprendizaje del SCG estará dado por Q-learning cuando el desarrollo requiera resultados de predicción y que no sea de tiempo real, en otro caso emplear la asignación de crédito que BBA (Bucket Brigade Algorithms) que se explicó en el capitulo anterior y por el AG. Se recomienda emplear en cualquier aplicación valores ya probados [62][142][20] (ver anexo A).

4.2.4 Aprendizaje en los SCGs.

El refinamiento de habilidades se ha basado en el aprendizaje automático, y se centra en el uso del aprendizaje por refuerzo. En esta sección se comentan algunos de los sistemas de aprendizaje más utilizados, detallándose en más profundidad el aprendizaje por refuerzo, y más concretamente en el algoritmo Q-Learning.

El aprendizaje es un fenómeno que cubre muchas facetas (adquisición de nuevo conocimiento, desarrollo de habilidades, organización del conocimiento adquirido, etc.) y que pueden organizarse alrededor de dos métodos de aprendizaje [99].

- La Adquisición de Conocimiento, o el aprendizaje de información simbólica organizada mediante leyes, modelos, teorías, reglas, etc. junto con la capacidad de aplicar esa información de una manera efectiva.
- El Refinamiento de Habilidades, o perfeccionamiento de habilidades motoras o cognitivas a través de la práctica.

Un posible método para clasificar los sistemas de aprendizaje viene dado por la estrategia de aprendizaje subyacente, o visto de otro modo, por la cantidad de conocimiento que el sistema infiere. En un extremo estarían aquellos sistemas en los que todo el conocimiento está programado directamente, y por tanto el sistema no tiene que inferir nada. En el otro extremo, estarían los sistemas que parten de una base de conocimiento más limitada pero que infieren nuevo conocimiento mediante observaciones y experimentos (el cual es nuestro caso). A continuación se describe una posible clasificación de los sistemas de aprendizaje siguiendo este criterio debida a [99].

1. Aprendizaje Memorístico e Implantación Directa de Conocimiento. No se requiere ningún mecanismo de inferencia o transformación del conocimiento por parte del sistema que está aprendiendo.

2. Aprendizaje desde Instrucciones: Adquisición de conocimiento desde un profesor u otro tipo de fuente de información, de forma que se transforma el conocimiento que se recibe en un lenguaje de entrada a una representación interna más fácil de utilizar. La fuente de información puede ser un profesor, el entorno externo, o el propio sistema.

3. Aprendizaje por Analogía. Adquisición de nuevo conocimiento o habilidades aumentando el conocimiento existente y que tiene grandes similitudes con los nuevos hechos o habilidades aprendidos.

4. Aprendizaje desde Ejemplos. Dado un conjunto de ejemplos y contraejemplos de un concepto, se induce una descripción general del concepto que envuelve todos los posibles ejemplos, pero ningún contraejemplo.

5. Aprendizaje por Observación y Descubrimiento (aprendizaje no supervisado). Incluye sistemas de descubrimiento, creación de criterios de clasificación para generar jerarquías taxonómicas, y tareas similares que no requieren un profesor externo que supervisa el aprendizaje.

Aspectos clave en el Aprendizaje por Refuerzo.- Con el objetivo de definir cualquier problema a resolver con aprendizaje por refuerzo, deben considerarse algunos puntos clave [100].

a) Recompensa Retardada

Otros sistemas de aprendizaje requieren conocer para cada episodio o instancia del aprendizaje (estado percibido y acción ejecutada) la clase a la que pertenecen, es decir, si han llegado al estado objetivo o no. Sin embargo, muchas veces el agente no sabe inmediatamente si ha alcanzado su objetivo tras ejecutar una acción, sino que la consecución del objetivo viene dado por la ejecución de varias acciones de forma consecutiva. Esto genera el problema denominado asignación de crédito temporal que consiste en un marcado "a posteriori" de las acciones que han tenido un impacto positivo o negativo en la consecución del objetivo. Estas marcas permiten una asignación de refuerzos apropiada en cada instancia del aprendizaje.

b) Exploración versus Explotación

En muchas tareas de aprendizaje, existe una relación entre la utilización de conocimiento previamente adquirido cuando se genera un plan (explotación), o intentar buscar nuevas alternativas con las que se obtenga el objetivo más eficazmente (exploración). Es por tanto necesario buscar una estrategia de exploración, ya que ésta influye en la velocidad con la que se obtienen las estrategias, y la calidad de dichas estrategias.

c) Estados Observables Parcialmente

En muchas situaciones, los sensores de los agentes proporcionan sólo información parcial del entorno. Este problema implica que el agente pudiera recibir del entorno estados con diferentes niveles de detalle, que pudieran hacer que el agente no diferencie unos estados de otros, o incluso que estados iguales le parecieran distintos.

d) Indeterminismo en acciones y refuerzos

En muchos entornos, la ejecución de una acción desde un estado determinado, puede que lleve al agente a estados finales distintos. Esto puede ser debido tanto al ruido que suelen tener asociados los sensores, como al que suelen tener las acciones

que ejecuta. Esto implica que la recompensa que recibe el agente al ejecutar esa acción, puede ser distinta dependiendo de si el estado final que el agente observa es un estado objetivo o no.

e) Dominios Continuos o de gran tamaño

La mayoría de los dominios del mundo real son continuos, y por tanto, los estados distintos que se reciben del entorno son infinitos. Además, en la mayoría de los casos, también son infinitas las acciones que se pueden llevar a cabo en cada momento. Por tanto, es necesario un mecanismo que permita discretizar ese mundo real y así minimizar el número de estados y acciones posibles en el entorno. A este problema se le suele denominar generalización de estados y acciones. Como se explicará en posteriores capítulos, en este trabajo se ha utilizado la cuantificación vectorial como mecanismo de discretización del conjunto de estados posibles en el entorno.

f) Integración de varias habilidades adquiridas

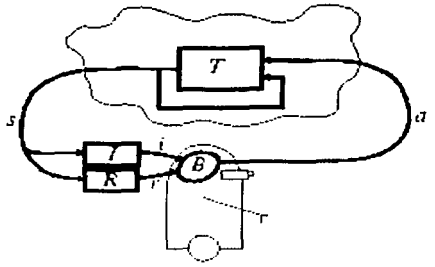
En la mayoría de dominios, los agentes deben adquirir diversas habilidades. Dichas habilidades deben poder ser adquiridas individual o colectivamente, de forma que se permita la adquisición de otras nuevas, o que unas no interfieran negativamente sobre las otras.

El aprendizaje por refuerzo trata de resolver el problema de cómo un agente autónomo que recibe información sensorial y actúa en un entorno puede aprender a elegir acciones óptimas para alcanzar sus objetivos. Este problema genérico cubre tareas tales como aprender a controlar un robot móvil, aprender a optimizar operaciones en fábricas, o aprender a jugar a juegos de mesa.

En cada momento el agente ejecuta una acción en su entorno y un entrenador proporciona un refuerzo o un castigo para indicar lo deseable del estado al que se pasa. La tarea del agente es aprender de los premios o castigos que recibe a elegir secuencias de acciones que generen un mayor refuerzo positivo, y de manera indirecta, a alcanzar de manera eficiente sus objetivos [101].

Se trata, por tanto, de un sistema de aprendizaje supervisado que recibe información del entorno externo y de un "profesor" que produce refuerzos positivos o castigos según la calidad de los resultados a los que llega el agente y para ello seguimos el modelo básico que se muestro en la figura 4.2.

Figura 4.2: Modelo de Aprendizaje por Refuerzo.



Un agente es conectado a su entorno vía percepción y acción. En cada paso de la iteración el agente recibe una entrada sensorial, i, como indicación del estado actual, s, del entorno; entonces el agente decide tomar una acción a que genera como salida; esta salida cambia el estado del entorno, y el valor de esta transición de estado es comunicada al agente a través de un señal de refuerzo r, tal y como se muestra en la figura 4.2.

El comportamiento, B, del agente elige acciones que incrementen la suma de todas las señales de refuerzo recibidas a lo largo del tiempo. Formalmente, el modelo consiste de

1. Un conjunto discreto de estados, S;
2. Un conjunto discreto de acciones del agente, A;
3. Un conjunto de señales de refuerzo escalares, R, típicamente R = {0,1}.

La figura anterior también incluye una función de entrada, I, que determina cómo percibe el agente el estado del entorno; otra que realiza las transiciones de estado, T; y una última R que calcula el refuerzo que debe adquirir el agente en cada momento. Con estas funciones, se termina la descripción de los denominados procesos de decisión de Markov, que es el modelo más utilizado para definir los problemas de refuerzo retardado, y que define T y R como:

- R : SxA (R, que para cada par estado acción proporciona su refuerzo.

- T : SxA (((S), donde un miembro de ((S)es una distribución de probabilidad sobre el conjunto S(es decir, transforma estados en probabilidades). Decimos que T(s,a.s') es la probabilidad de que se realice una transición desde s hasta s' ejecutando la acción a.

El trabajo del agente es encontrar una política, (, que maximice alguna medida de refuerzo a largo plazo.

Un inconveniente del aprendizaje por refuerzo es que se asume que el entorno debe estar discretizado, y que el total de dicho entorno puede ser enumerado (implica un orden entre los estados) y almacenado (implica un conjunto de estados finito). Estos son algunos de los principales problemas que se encuentran al aplicar el aprendizaje por refuerzo en diferentes problemas. Por ir adelantando algunas cuestiones, cabe indicar que dicho entorno es un entorno de infinitos estados con valores continuos. Esto obliga al uso de alguna técnica de discretización, que además limite el número de estados del entorno a un número viable, desde el punto de vista del almacenamiento de memoria y tamaño del conjunto de casos de prueba necesarios para hacer el aprendizaje. A esto último se le denomina generalización de los pares estado-acción.

Uno de los enfoques más usados dentro de aprendizaje es el aprendizaje supervisado a partir de ejemplos, para después predecir la salida de nuevas entradas. Cualquier sistema de predicción puede verse dentro de este paradigma, sin embargo, ignora la estructura secuencial del mismo. En algunos ambientes, muchas veces se puede obtener sólo cierta retroalimentación o recompensa o refuerzo (ejemplo, gana o pierde). El refuerzo puede darse en un estado terminal y/o en estados intermedios. Los refuerzos pueden ser componentes de la utilidad actual a maximizar o sugerencias de la utilidad actual (ejemplo, buena movida)

En aprendizaje por refuerzo el objetivo es aprender como mapear situaciones a acciones para maximizar una cierta señal de recompensa.

En general, al sistema no se le dice que acción debe tomar y el debe de descubrir qué acciones dan el máximo beneficio.

Aspectos importantes del aprendizaje por refuerzo: (i) se sigue un proceso de prueba y error, y (ii) la recompensa puede estar diferida

Otro aspecto importante es el balance entre exploración y explotación

Para obtener buena ganancia uno prefiere seguir ciertas acciones, pero para saber cuales, se tiene que hacer cierta exploración.

Todos los sistemas de aprendizaje por refuerzo tienen metas explícitas, obtienen información del ambiente y pueden escoger entre varias acciones para influenciar el ambiente

Elementos:

- Política (π): define cómo se comporta el sistema en cierto tiempo. Es un mapeo (a veces estocástico) de los estados a las acciones.
- Función de recompensa (R): define la meta. Mapea cada estado-acción a un número, la recompensa, indicando lo deseable del estado.
- Función de valor (V): indica lo que es bueno a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en ese estado (predicciones de recompensas). Se buscan hacer acciones que den los valores más altos, no la recompensa mayor.

  Las recompensas están dadas por el ambiente, pero los valores se deben de estimar (aprender) en base a las observaciones.

  Algoritmos genéticos, recocido simulado, y otros métodos de optimización buscan en el espacio de políticas sin ver las funciones de valor.

  Aprendizaje por refuerzo aprende mientras interactúa con el ambiente

- Modelo del ambiente (opcional): imita el comportamiento del ambiente. Se puede usar para hacer planeación al considerar posibles situaciones futuras basadas en el modelo.

Ejemplo: aprender a jugar gato.

Para cada posible estado del juego, se asigna un valor (estimación de la probabilidad de ganar a partir de ese estado).

A estados con tres X's les ponemos 1, con tres O's les ponemos 0, y al resto 0.5.

Nos ponemos a jugar y para seleccionar que hacer hacemos lo que estimamos nos va a dar mejores resultados (tiene mejor valor en la tabla o función de valor). Ocasionalmente, seleccionamos movimientos aleatorios (exploratorios).

Al movernos ajustamos los valores de los estados retroalimentando los valores a los estados antes del movimiento. Si s' denota el estado siguiente, y $V(s)$ el valor de un estado s, lo podemos hacer como sigue:

$$V(s) \leftarrow V(s) + \alpha[V(s')-V(s)]$$

donde α es un número pequeño llamado parámetro del tamaño del paso.

Si α se reduce con el tiempo, el método converge. Si no se reduce a cero, también funciona con oponentes que cambian se estrategia de juego gradualmente.

Cuando tenemos espacios de búsqueda muy grandes es imposible visitarlos todos o simplemente guardar una tabla de valores para cada uno de ellos. Aquí es donde es posible usar aprendizaje supervisado para generalizar (aprender esas funciones de costo) a partir de unos cuantos estados.

Más formalmente, dado un estado $s_t \in S$ y una acción $a_t \in A(s_t)$, el agente recibe una recompensa r_{t+1} y se mueve a un nuevo estado s_{t+1}

El mapeo de estados a probabilidades de seleccionar una acción particular es su política (π_t). Aprendizaje por refuerzo especifica como cambiar la política como resultado de su experiencia.

No trata de maximizar la recompensa inmediata, sino la recompensa a largo plazo (acumulada).

La recompensa debe de mostrar lo que queremos obtener y se calcula por el ambiente.

Si las recompensas recibidas después de un tiempo t se denotan como: r_{t+1}, r_{t+2}, r_{t+3}, ... , lo que queremos es maximizar lo que esperamos recibir de recompensa (R_t) que en el caso más simple es:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

Si se tiene un punto terminal se llaman tareas *episódicas*, si no se tiene se llaman tareas *continuas*. En este último caso, la fórmula de arriba presenta problemas, ya que no podemos hacer el cálculo cuando T no tiene límite.

Podemos usar una forma alternativa en donde se van haciendo cada vez más pequeñas las contribuciones de las recompensas más lejanas:

$$Rt = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum \gamma^k r_{t+k+1}$$

Donde γ se conoce como la razón de descuento y está entre: $0 \leq \gamma \leq 1$.

Si $\gamma = 0$ se trata sólo de maximizar tomando en cuenta las recompensas inmediatas.

En aprendizaje por refuerzo se asume que se cumple con la propiedad Markoviana y:

$$P^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

el valor de recompensa esperado es:

$$R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s\}$$

π es un mapeo de cada estado $s \in S$ y acción $a \in A(s)$ a la probabilidad $\pi(s,a)$ de tomar la acción a estando en estado s. El valor de un estado s bajo la política π, denotado como $V^{\pi}(s)$, es el refuerzo esperado estando en estado s y siguiendo la política π.

Este valor esperado se puede expresar como:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{\sum \gamma^k r_{t+k+1} \mid s_t = s\}$$

y el valor esperado tomando una acción a en estado s bajo la política π ($Q^{\pi}(s,a)$):

$$Q^{\pi}(s,a) = E_{\pi}\{Rt \mid s_t = s, a_t = a\} = E_{\pi}\{\sum \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}$$

Para estimar V^{π} y Q^{π} podemos tomar estadísticas haciendo un promedio de las recompensas obtenidas (método tipo Monte Carlo).

Una forma sencilla de hacerlo sería:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

donde R_t es la recompensa actual (por lo que tendría que esperar hasta que se acabe el episodio).

Métodos tipo TD sólo tienen que esperar el siguiente paso.

TD (temporal difference) usan el error o diferencia entre predicciones sucesivas (en lugar del error entre la predicción y la salida final) aprendiendo al existir cambios entre predicciones sucesivas.

Ventajas:

- incrementales y por lo tanto fáciles de computar
- convergen más rápido con mejores predicciones

El más simple TD(0) es:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

Algoritmo TD(0)

Inicializa V(s) arbitrariamente y π a la política a evaluar.
Repite (para cada episodio):
Inicializa s
Repite (para cada paso del episodio):
a ← acción dada por π para s.
Realiza acción a; observa la recompensa, r, y el siguiente estado, s'
V(s) ← V(s) + α[r + γV(s') – V(s)]
s ← s'.
hasta que s sea terminal

La actualización de valores tomando en cuenta la acción sería:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$$

y el algoritmo es prácticamente el mismo:

Inicializa Q(s,a) arbitrariamente.
Repite (para cada episodio):
Inicializa s;
Selecciona una a a partir de s usando la política dada por Q
(ejemplo ε-greedy)
Repite (para cada paso del episodio):
Realiza accion a, observa r, s'
Escoge a' de s' usando la política derivada de Q
Q(s,a) ← Q(s,a) + α[r + γQ(s',a') – Q(s,a)]
s ← s'
a ← a';
hasta que s sea terminal

La otra parte del aprendizaje es dado a través del algoritmo genético y como lo explique en el capítulo anterior este debe ser usado hasta que el componente de asignación de crédito realizado ya sea por el algoritmo bucket brigada o Q-learning esté estacionario es decir que los clasificadores tengan conflicto para encontrar una buena solución en la que solamente lo podremos obtener empleando las ventajas de los AGs que lo usaremos seleccionando una

porción de los peores clasificadores de la población y de esta seleccionaremos al 50% para efectuar el cruzamiento y mutación solo con esta pequeña población. A continuación muestro los diagramas que nos servirán para esta incorporación a los sistemas clasificadores por medio de las figuras 4.3. 4.4 y 4.5.

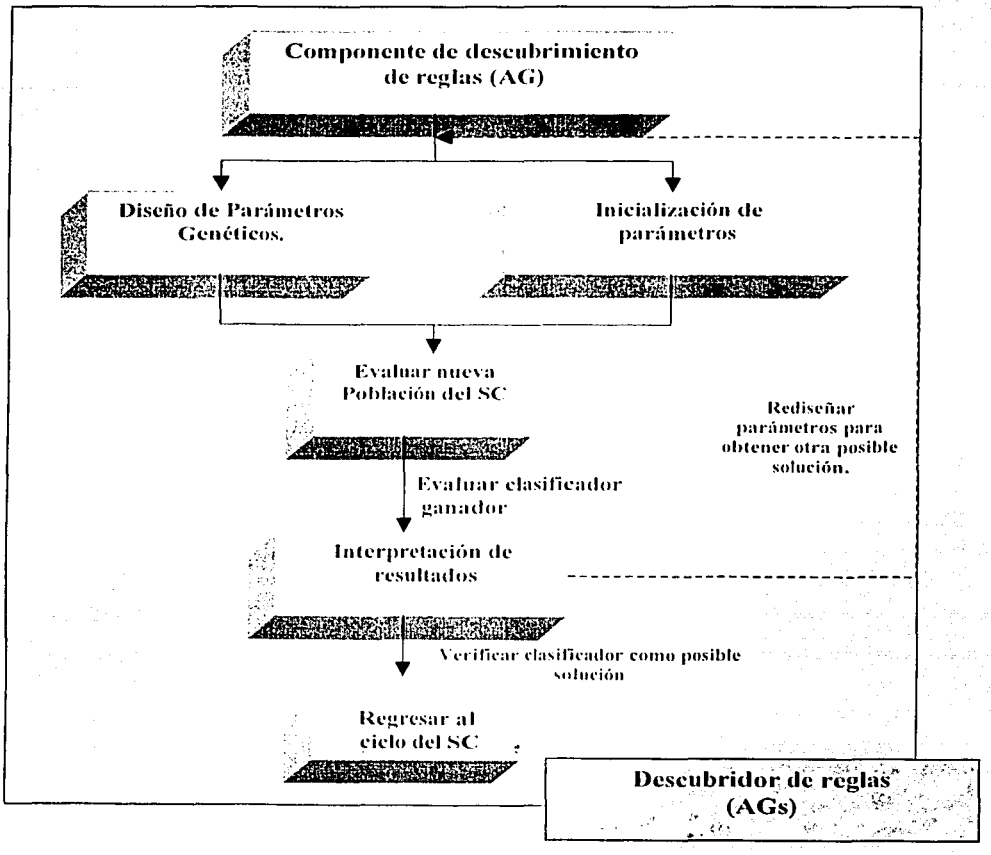


Figura 4.3: Ciclo Principal de enlace del SCG con el AG.

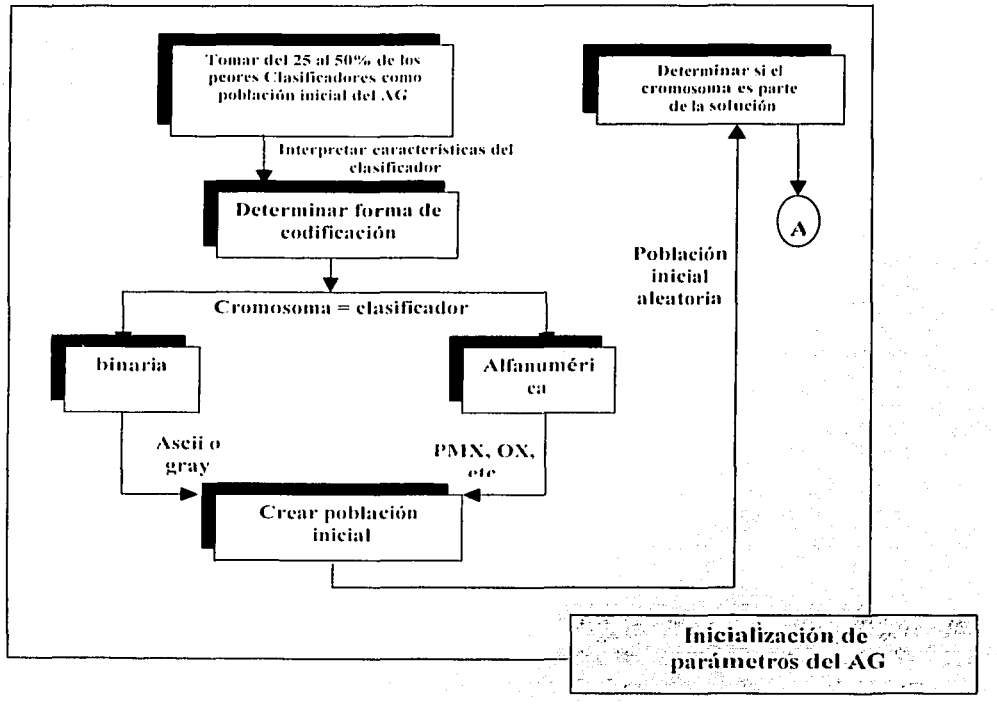


Figura 4.4: Inicialización de parámetros de la aplicación.

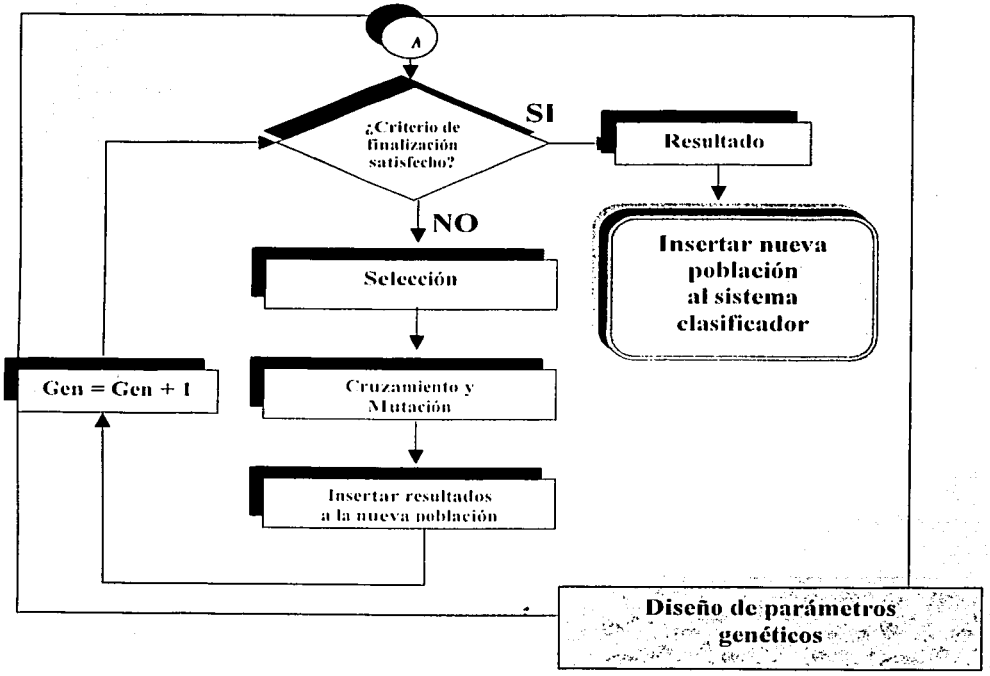


Figura 4.5: Diseño de Parámetros Genéticos

4.2.5 Prototipo (rápido) inicial y evaluación.

Esta es una etapa clave debido a que todas las decisiones tomadas en el diseño preliminar deben ser ya sea confirmada, rectificadas o desechadas, sobre la base del conocimiento recopilado de los expertos en el proceso hasta completar esta etapa. El prototipo inicial debe verse como el sistema completo, excepto que estará limitado en su cobertura. Debe incluirse una interfaz con el usuario relativamente bien definida y un subconjunto robusto de conocimiento de tal forma que los usuarios puedan juzgar su aceptabilidad. Esto no significa que el prototipo debe ser altamente robusto, simplemente debe reflejar la forma que tendría el sistema final que será construido. En general se recomienda que el prototipo inicial sea desechado una vez que se haya completado su evaluación. En esta etapa se completa todo lo referente a la incorporación de la teoría de los SCA y operadores genéticos y se prepara el prototipo que servirá de definición entre el usuario y el diseñador. La clave en la etapa del prototipo es que se debe extraer tanto conocimiento y opiniones de expertos y usuarios como sea posible para poder validar satisfactoriamente las decisiones de diseño. Es el momento de detectar y corregir cualquier inconveniencia con los operadores y la forma del clasificador. Como parte de nuestra propuesta experimentamos la adquisición del conocimiento en forma simbólica para obtener dos ventajas:

- primero, la facilidad de transferir el conocimiento aprendido a operadores similares a los empleados cotidianamente por los humanos y,
- segundo, la facilidad de combinar algoritmos genéticos con métodos de aprendizaje analíticos que explican el éxito de reglas derivadas en forma empírica.

4.2.6 Diseño Final.

El diseño final comprende la selección de las herramientas y de los recursos necesarios para desarrollar la aplicación a ser entregada.

En esta etapa también se incluye el seudo código para facilitar el desarrollo de las aplicaciones de los SCGs y principalmente nos muestra la secuencia y procedimientos necesarios en cualquier desarrollo en el anexo A se detalla el seudocógico completo de un sistema clasificador (ver anexo A).

4.2.7 Implementación.

Esta es la etapa que puede consumir la mayor parte del tiempo del Proceso de Desarrollo de un SCG, aun cuando exista un excelente diseño. La implementación comprende el proceso completo del clasificador (interpretación de los resultados) para todos los módulos y será necesario que el diseñador explique claramente como se puede interpretar cada resultado para que el usuario en una forma rápida y sencilla implemente los resultados en forma automática, pues por lo regular la codificación de los resultados estará dado por la evolución del clasificador en forma binaria o alfanumérica y es 7preciso que se deje claramente la interpretación que representaría cada gen del clasificador a notación coloquial.

4.2.8 Pruebas.

El asegurar la calidad de un SCG es una tarea muy importante que debe ser cuidadosamente planificada, especialmente a medida que el SCG es más grande y complejo o de aplicación crítica. El plan de pruebas de un SCG es bastante similar al que se prepara para un sistema de software convencional. Esto es, debe incluir procesos de verificación y validación. Estos dos elementos son diferentes en naturaleza, pero ambos son muy necesarios.

La verificación puede ser definida como una ayuda para que el sistema sea construido correctamente y consiste de dos procesos básicos: El chequeo de la concordancia que tiene

la aplicación con las especificaciones dadas y el chequeo de la consistencia y la solidez del cromosoma (errores semánticos y sintácticos).

La validación se define como el proceso que nos indica si hemos construido la aplicación correcta para las necesidades planteadas. Para ello se debe incluir y discutir aspectos importantes como: Qué es lo que se está validando, La metodología de validación, los criterios de validación y cómo y cuándo aplicar la validación.

Sobre la base de los resultados de las pruebas realizadas a la aplicación, el grupo responsable del desarrollo y el usuario deberán determinar finalmente si la aplicación está lista para ser aceptada. De ser este el caso, el usuario deberá notificar que el SCG está listo para ser empacado como una aplicación funcional. Además, deberá suscribir un documento de aceptación preliminar y consignar cualquier ajuste final que debe ser realizado, en caso de existir alguno.

4.2.9 Ajustes al Diseño.

A medida que el trabajo avanza y el diseñador tiene a la vista los problemas detectados, se deben realizar los ajustes necesarios al inicio de cada iteración. Si estos ajustes cada vez son relativamente más pequeños y no son retroactivos, se tiene una buena medida de que se está progresando. Pero si ocurre lo contrario, puede representar un serio retardo al proyecto y posiblemente requerir un cambio de paradigma; aquí es donde se recurre al ciclo de recalcular cada uno de los parámetros que intervienen en la aplicación y hasta en ocasiones se decide cambiar o ajustar el tamaño del cromosoma cuando el número de variables (genes) son demasiados (cuando el tamaño del cromosoma es mayor a 150 bits, se decide dividirlo en diferentes sub-aplicaciones). Lo que si no estará permitido ya en esta etapa es decidir cambiar de técnica, pues se consideraría un error de análisis y las pérdidas serían mas significativas (es importante considerar que con un buen análisis del problema, rara vez se presentará esta incógnita). Por otro lado, en esta etapa se decide re-considerar la forma de detener la aplicación, ya sea por número de corridas o por nivel de convergencia como se explicará mas adelante, así como si el tamaño de los operadores genéticos debe ser cambiado o la técnica empleada en la selección y el cruzamiento puede ser diferentes (rara vez es necesario modificar la mutación, se sugiere tomar siempre los valores predefinidos

de acuerdo a la fórmula dada que depende del tamaño de la población y del tamaño del cromosoma). Esta etapa en algunos casos es evitada, si se obtiene un "buen resultado" al momento de ejecutar el prototipo o bien el problema es considerado de tiempo real, es decir que no hay tiempo suficiente para realizar pruebas o modificaciones como es el caso en aplicaciones de la compra de acciones o en el caso de simulación de juegos.

4.2.10 Instalación, implantación y mantenimiento.

En la etapa final del Proceso de Desarrollo de un SCG se traslada la aplicación desarrollada, como un producto operativo, hacia el entorno de los usuarios. Para ello, se deben realizar varias actividades de instalación, implantación y mantenimiento similares a las de un sistema de software convencional.

a) Empacado del producto final.

Durante la etapa de empacado del producto final, se debe completar todo el trabajo que falte. Por ejemplo:

- Preparar los programas para instalación del SCG.
- Completar la documentación de la aplicación, especialmente el Manual del Usuario.
- Identificar los cambios procedimentales que pueden ser requisitos necesarios para que el AG se incorpore fácilmente al flujo de trabajo de las áreas operacionales de los usuarios.
- Si es del caso, preparar la interfaces necesarias para que el AG se comunique con otras aplicaciones.

b) Entrenamiento a los usuarios del producto final

El plan de entrenamiento para los usuarios debe incluir los siguientes aspectos:

- Presentación del producto final, exponiendo los aspectos más importantes de la aplicación desarrollada.
- Entrega del Manual de Usuario.
- Explicación de los procedimientos organizacionales requeridos.

- Entrenamiento en el producto.

c) Pruebas de aceptación de los usuarios finales

Luego que los usuarios están debidamente capacitados, se pueden realizar pruebas para determinar si la aplicación está operando satisfactoriamente dentro de su ambiente normal de trabajo. Cualquier potencial problema que sea identificado por el usuario, debe ser resuelto previo a la aceptación final.

d) Constancia de aceptación de los usuarios finales

Una vez que los usuarios han completado su entrenamiento y las pruebas realizadas los ha convencido que la aplicación está lista para operación normal, se debe suscribir un documento de aceptación final.

e) Implantación y Mantenimiento

Finalmente, el diseñador del AG debe ejecutar las acciones requeridas para que la aplicación sea aceptada. Asesoramiento y consultoría especializada pueden ser necesarios para planificar y ejecutar la implantación y el mantenimiento de la misma. A medida que se gane en experiencia y conocimiento acerca del problema que ayuda a resolver con el AG, debe irse incorporando en el mismo, a través de las actividades previstas en el plan de mantenimiento, es importante esta etapa desde el punto de vista de que para alargar la vida de la aplicación es necesario ir actualizando las últimas modificaciones que estén probadas por eventos o congresos reconocidos y antes de simplemente hacerle cambios a la aplicación actual deberán ser probadas cada una de estas modificaciones para analizar si tendrían efectos positivos o simplemente es algo peor de lo que ya se tiene.

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS FUTUROS.

5.1 Introducción.

Hablar de conclusiones puedo mencionar los trabajos relacionados con estos cinco años de investigación que fueron cada vez más fructíferos en la obtención de nuevo conocimiento en esta área tan cambiante y novedosa. En este tiempo tuve la fortuna de participar en 12 congresos internacionales, llevando a cada uno de ellos un trabajo relacionado con la tesis y que gracias al apoyo de mis asesores dichos trabajos tuvieron la aceptación que es requerida en este tipo de evento, además de participar en la estancia de un año en uno de los grupos mas importantes de esta área que es el grupo de ILLIGAL (Illinois Genetic Algorithms) de la Universidad de Illinois con el Dr. David Goldberg.

En las siguientes secciones menciono las principales conclusiones que arrojaron la presente tesis. asi como sus etapas.

5.2 Conclusiones de la tesis.

Puedo resumir el funcionamiento de un AG como una combinación balanceada de la exploración de nuevas regiones en el espacio de búsqueda y la explotación de regiones ya muestreadas. Este balance que controla la ejecución del AG está determinado por la selección correcta de los parámetros de control: probabilidad de cruzamiento, probabilidad de mutación y el tamaño de la población. Algunos estudios utilizan el peso de la selección como un parámetro adicional de control [63]. Un número de estudios se dirigen a seleccionar el conjunto optimal de parámetros de control para una colección de funciones de prueba [3][67][115].
Como resultados de estas aportaciones que se han probado puedo concluir lo siguiente:

1. Incrementando la probabilidad de cruzamiento se incrementa la recombinación de Bloques constructores pero también se incrementa la destrucción de buenas cadenas.

2. Incrementando la probabilidad de mutación se tiende a transformar la búsqueda

genética en una búsqueda aleatoria, pero esto también permite reintroducir pérdida de material genético.

3. Incrementando el tamaño de la población se incrementa la diversidad y reduce la probabilidad de que el AG haga una convergencia prematura, pero esto también incrementa el tiempo requerido para que la población converja a las regiones optimales en el espacio de búsqueda.

Entonces como sugerencia como lo he estado mencionando en los capítulos anteriores sugiero emplear valores predeterminados para cualquier aplicación donde intervengan los AGs, esto para facilitar lo que se ha estado probando en un gran número de artículos con buenos resultados y que a la vez se menciona en el problema que se presenta en la presente tesis con el programa GADEMO, dichos parámetros son:

1. Selección por ELITISMO es decir siempre escoger los cromosomas o clasificadores que tengan mayor éxito.
2. Cruzamiento con dos puntos de cruza con valor de 97 al 100% de entrecruzamiento,
3. Mutación que dependa del valor de la población y cruzamiento en base a la siguiente fórmula: mutación = (1 / P * $\sqrt{L}$) donde P = tamaño de la población y L = longitud del cromosoma o en su defecto emplear un valor muy pequeño entre 0 a 3%,
4. Codificación binaria.
5. Población que estará en función del cromosoma o clasificador en base a si el tamaño de este es mayor a 25, entonces la población será el doble del tamaño del cromosoma o en su defecto la población será de 50 individuos y siempre con población inicial aleatoria.
6. Detener el proceso del algoritmo genético por nivel de confiabilidad cuando la diferencia de la desviación estándar de las últimas cinco generaciones sea menor al solicitado por el usuario (que puede ser del 1 al 5% de confiabilidad).

Otro de los problemas que se resolvieron con la presente tesis es la de los tamaños ideales u

"óptimos" para los aprendizajes de BBA y q-learning. pues en la mayoría de las veces solo se menciona la fórmula y el desarrollo de la misma es muy abierto, asi es después del diseño de experimentos con las pruebas que desarrollamos en la UIUC llego a las siguientes conclusiones:

1. Emplear un valor de $\gamma = 0.9$ para cuando empleemos el aprendizaje de Q-learning.
2. Para el aprendizaje o asignación de crédito a través del algoritmos de bucket brigade propuesto inicialmente por Holland recomiendo usar el valor para la valor para la constante de descuento $\alpha = 0.1$ ya que esto nos dará el 10% de descuento en la oferta de los clasificadores seleccionados sin alterar demasiado el valor de aptitud que trae anteriormente.

Para seleccionar los parámetros de control es necesario tener en cuenta la interacción entre los diferentes operadores genéticos. El propio problema de seleccionar el juego de parámetros óptimos para el AG puede convertirse en un problema de optimización no lineal complejo y es por ello que la recomendación que hago anteriormente ha demostrado resolver esta gran incógnita.

Muchas críticas se han hecho a los AG en cuento a la selección de una codificación para las soluciones del problema de optimización. Tradicionalmente, las cadenas binarias han sido utilizadas por la simplicidad de su implementación y porque esta representación maximiza el número de esquemas producidos [56]. Cuando el alfabeto usado para codificar las soluciones es superior al binario, es necesario redefinir los operadores de cruzamiento y mutación adecuadamente.

Un número grande de problemas de optimización tiene variables continuas que asumen valores reales. Una técnica común para codificar variables continuas en un alfabeto binario usa una codificación entera de punto fijo, cada variable es codificada usando un número fijo de bits binarios. Los códigos binarios de todas las variables del problema son concatenados para obtener la cadena representante de una solución del espacio de búsqueda. En este punto algunos problemas se presentan y es que la codificación binaria de una variable provoca diferencias substanciales entre la distancia de las codificaciones a nivel de bits (genotipo) y las distancias de las variables propiamente (fenotipo). Este

problema se enfrentó con la utilización de los códigos de Gray (de cambio mínimo) [24]. En la práctica las representaciones usando el código de Gray son más útiles para aplicaciones de optimización de funciones multiparamétricas. Otros tipos de codificaciones son reportadas en la literatura como es el caso de la codificación dinámica que permite variar el número e interpretación de los bits asignados a un parámetro particular a través de la corrida [116], las representaciones de longitud variable [56], la codificación delta [133] cuyos bits en las cadenas expresan una distancia con respecto a alguna solución parcial previa.

Existen sin duda una gran variedad de modelos de sistemas clasificadores y por muchos años, los SCA fueron estudiados solamente de acuerdo al modelo básico de Holland. Sin embargo, recientemente muchos investigadores se han enfocado ha resolver diferentes tipos de aplicaciones con la estructura básica de un SCA y para ello han tenido que hacer cambios al modelo de Holland, dando como resultado diferentes tipos como es el modelo de Wilson XCS [142] (principal fuente de inicio para la metodología propuesta). Cada uno de estos modelos presenta ventajas en el contexto en el que fueron desarrollados y respeta la estructura básica de los SCA que es la de crear aprendizaje a través de la adaptación de cada una de las reglas o clasificadores existentes.

Con el empleo de procesos definidos nos ayudará a diseñar de una mejor manera nuestro modelo que deseemos de una mejor manera y con estos pasos que se expusieron en esta tesis se estima que las ganancias para lograr éstas son significativas [105].

La decisión de usar cierta codificación o lenguaje tiene un impacto directo en la herramienta que será seleccionada. En muchos casos puede ser aplicable y muy útil realizar una descripción gráfica de los diferentes módulos del sistema, empleando las herramientas CASE propias de los sistemas de software convencionales. Para cada uno de los módulos, el diseño debe incluir las especificaciones de las entradas típicas y las salidas o conclusiones esperadas. A diferencia de otras aplicaciones robustas, aquí no existe un lenguaje específico a ser usado, sin embargo como propuesta propia sugiero emplear un lenguaje declarativo como el PROLOG para el desarrollo de aplicaciones con sistemas clasificadores por su motor de inferencia para la generalización del conocimiento.

Históricamente, las primeras aplicaciones con sistemas clasificadores y algoritmos genéticos fueron desarrollados utilizando lenguajes de programación como PASCAL y C, a medida que el desarrollo de ellos iba aumentado en cantidad y complejidad, la comunidad científica comenzó a buscar formas diferentes de desarrollos y generaron herramientas como el toolbox GAOT de MATLAB y herramientas parecidas a los Shells de los Sistemas Expertos como es el caso del GENEHUNTER, donde cada desarrollo se facilita desarrollando la aplicación con un ambiente gráfico como es el Visual Basic, JAVA o DELPHI.

Tradicionalmente LISP y PROLOG han sido los lenguajes que se han utilizado para la programación de Sistemas Expertos. Estos lenguajes ofrecen características especialmente diseñadas para manejar problemas generalmente encontrados en inteligencia artificial. Por este motivo se los conoce como lenguajes de inteligencia artificial. Una de las principales características que comparten los lenguajes LISP y PROLOG, como consecuencia de su respectiva estructura, es que pueden ser utilizados para escribir programas capaces de examinar a otros programas, incluyendo a ellos mismos. Esta capacidad se requiere, por ejemplo, para hacer que el programa explique sus conclusiones. Esto sólo puede hacerse si el programa tiene la capacidad de examinar su propio modo de operación.

5.3 Resultados de la metodología propuesta.

La prueba de la metodología ha sido utilizada para aplicaciones experimentales en la Universidad de Illinois en diferentes problemas como: movimiento de un brazo mecánico, teoría de juegos (en el juego de damas españolas), resolución de problemas de multiplexor y recorridos de caminos con predicción de conocimiento, obteniendo principalmente ventajas en cuanto a:

1. Reducción de errores de diseño [104],
2. Reducción en el tiempo de diseñar la aplicación y
3. Facilidad de desarrollo en problemas complejos.

Al emplear la metodología en un SCG se obtuvieron ahorros de tiempo en el diseño de la aplicación con SCG en un 40% en el tiempo total del diseño de la aplicación y por consiguiente reduciendo de costos y algo todavía mas importante fue que en el 50% de 30 aplicaciones donde inicialmente se pensaba que la mejor opción era utilizar un AG, SCA o SCG en comparación de utilizar cualquier técnica conocida parecida (Tabú, Simulated Annealing, GRASP, Hill climbing, técnicas de investigación de operaciones) en la etapa del análisis se comprobaba que no era el mejor camino para resolver el problema que se tenía y se decidió emplear algún otro método y de ese 50% rechazado, anteriormente ya se habían usado para resolverlo mecanismos evolutivos como AGs o SCG, comprobando que en esas ocasiones no se hacía un análisis del problema en forma detallada. Estas pruebas se hicieron con el apoyo del Dr. Kurian de la Universidad de Illinois en Urbana-Champaign en cuatro grupos (con diferentes materias de aplicación con técnicas evolutivas) y en total se trabajo con 22 aplicaciones autónomas que fueron dirigidas por cada uno de los profesores que impartían la materia.

5.3 Trabajos Futuros.

Como se puede apreciar en la presente tesis, existen muchas incógnitas por resolver para que estas técnicas sean competitivas contra otras ya probadas como lo son las redes neuronales, hill-climbing y técnicas de investigación de operaciones, pero lo que se ha podido apreciar es que se prometen buenos resultados para problemas de comportamiento NP-Completo y nuestra tarea a partir del año 2003 en colaboración con varios investigadores reconocidos del ITAM (Dr. Marcelo Mejía), con la Universidad de Illinois (Dr. Martín Butz y Dr. Kurian), de la Universidad Panamericana (Dr. Stanisalw Raczynski) es:

1. Proponer un modelo de generalización para la predicción del conocimiento en los SCGs basado en el aprendizaje tradicional que puede ser el propuesto por la corriente de Piaget.
2. Combinar los sistemas clasificadores con otras técnicas (Redes Neuronales, Sistemas Expertos, GRASP, etc.) para aplicar las ventajas de cada uno de ellas.

3. Demostrar que usando el lenguaje PROLOG en el desarrollo de aplicaciones con sistemas clasificadores ahorra tiempo de desarrollo y facilita el aprendizaje de nuevas reglas a través del AG.
4. Desarrollar aplicaciones reales aplicando las ventajas de la metodología a diversos problemas.
5. Investigar la forma de mejorar la predicción del aprendizaje con el modelo ACS en combinación con el modelo que se uso en la tesis, el XCS [142].
6. Crear un tutorial de SCG que esté disponible en internet, y
7. Crear un grupo de trabajo en la UAA para colaborar en el desarrollo de los puntos anteriores con instituciones con experiencia en este tipo de trabajos como son las UNAM, ITAM, UP y Universidades extranjeras como la UIUC, Universidad de Paris, etc.

5.4 Otras fuentes de actualización de sistemas clasificadores.

Como mencioné anteriormente, el tema de los sistemas clasificadores ha cobrado gran importancia en los últimos diez años y creo que se debe en gran parte que es una técnica que combinada con los aportes de investigación del aprendizaje podremos tener aplicaciones "inteligentes" es decir que el propio sistema este programado para autoajustarse y poder tomar decisiones por si solo en terrenos de comportamiento NP-completo. A continuación muestro algunas direcciones de internet donde podemos encontrar los últimos avances de los mismos y asi mismo podremos interactuar con los investigadores que están desarrollando este conocimiento, por comodidad de la literatura empleo el acrónimo de sistema clasificador de aprendizaje en inglés (LCS) para facilitar la búsqueda de esta información.

a) La WEB de LCS .- Alwyn Barry's LCS Web prove una gran riqueza de información, incluyendo nuevas publicaciones de LCS, una lista de investigadores con sus correos electrónicos y su página, una colección de implementaciones de LCS y un glosario de LCS. Esta página la recomiendo como una primer entrada al

conocimiento de los LCSs, http://www.cs.bath.ac.uk/~amb/LCSWEB/

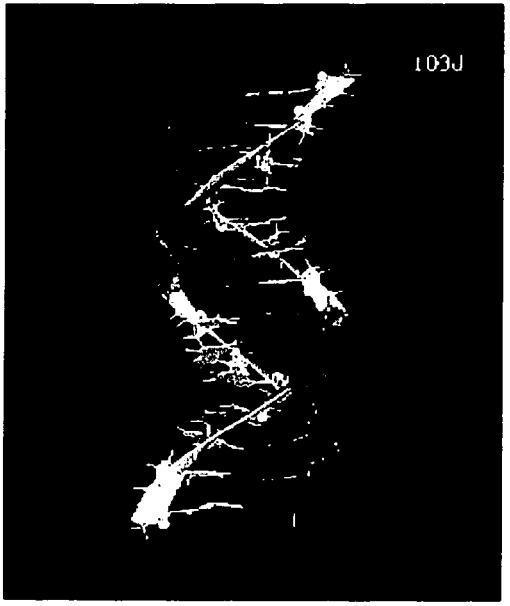
b) Lista de correos de LCS .- página personal de John Holme's donde se pueden encontrar los correos electrónicos de muchos investigadores de LCS y artículos interesantes y aplicaciones de LCSs, enviar correo a John Holmes a la dirección: jholmes@ceeb.med.upenn.edu to suscribe.

c) Bibliografía de LCS .- Este crecimiento de bibliografía está creciendo rapidamente y aquí se puede observar alrededor de 600 referencias, muchos con resumen del artículo, http://www.cs.bris.ac.uk/~kovaes/lcs/search.html.

d) El archivo de LCS .- Donde encontraremos un apéndice de bibliografías de LCS que contiene artículos completos y estadísticas que muestran la efectividad del artículo y sus contribuciones.
http://www.cs.bris.ac.uk/~kovaes/lcs/lcs.archive.html.

e) Índice de investigaciones .- Esta dirección ofrece referencias y textos completos de un gran número de artículos de LCS. Se espera que esta dirección vaya en aumento y este software busca automáticamente referencias cruzadas de artículos de internet, http://citeseer.nj.nec.com.

f) Lista Digest acerca de algoritmos genéticos (GA-List).- GA-List es una larga lista de artículos acerca de algoritmos genéticos en la cual algunos de ellos se correlacionan con los de LCS, para subscribirse a la lista escribir: ga-list-request@aic.nrl.navy.mil con "suscribe" en la parte del cuerpo de mensaje al enviar el correo electrónico.



ANEXO A
PRUEBAS Y RESULTADOS.

Introducción.

En este anexo se mencionarán algunas de las aplicaciones que nos sirvieron en la presente tesis para efectuar el análisis que intervienen en la metodología propuesta, algunas de estas aplicaciones fueron desarrolladas de acuerdo a las necesidades propias de los requerimientos que empleaba y otras fueron tomadas de internet con la recomendación de algunos de los autores para que fueran consideradas en el presente trabajo.

Aplicaciones desarrolladas para la tesis.

a) GADEMO.- Es un programa desarrollado en Delphi 5 para monitorear el desarrollo de las aplicaciones de los operadores para el problema propuesto por el Dr. Goldberg en su libro de optimización y a la vez mejorado por el Dr. Carlos Coello Coello (todo esto en Turbo Pascal) y posteriormente hicimos los cambios pertinentes para el diseño de experimentos de esta tesis y concluir lo que en diferentes literaturas se afirmaba en cuanto el tamaño de cada uno de los operadores genéticos (figura AA.1, AA.2), en este programa podemos probar distintas estrategias y combinaciones de los operadores básicos que intervienen en los algoritmos genéticos como puede ser el de detener el ciclo de operación después de "n" corridas o por un nivel de convergencia, el poder cambiar el tamaño básico del cruzamiento y mutación o bien tomar valores predefinidos de acuerdo con el tamaño del cromosoma y por último podemos crear tablas dinámicas (figura AA.3). Y para ello se programó esta aplicación para darle respuesta al problema que se menciona a continuación, donde observamos que dicho problema es típico de optimización combinatoria y gracias a dicho programa pudimos observar los cambios que nos favorecen al cambiar cada uno de los operadores genéticos en este ejemplo. El problema a resolver fue: "Un grupo de financieros mexicanos ha resuelto invertir 10 millones de pesos en la nueva marca de vino "Carta Nueva". Así pues, en 4 ciudades de las principales de México se decide iniciar una vigorosa campaña comercial: México en el centro, Monterrey en el noroeste, Guadalajara en el occidente y Veracruz en el oriente. A esas 4 ciudades van a corresponder las zonas comerciales I, II, III y IV. Un estudio de mercado ha sido realizado en cada una de las zonas citadas y han sido establecidas curvas de ganancias medias, en millones de pesos, en

función de las inversiones totales (almacenes, tiendas de venta, representantes, publicidad, etc.). Para simplificar los cálculos, se supone que las asignaciones de créditos o de inversiones deben hacerse por unidades de 1 millón de pesos. La pregunta es: ¿ en dónde se deben de asignar los 10 millones de pesos de los que se dispone para que la ganancia total sea máxima ?.

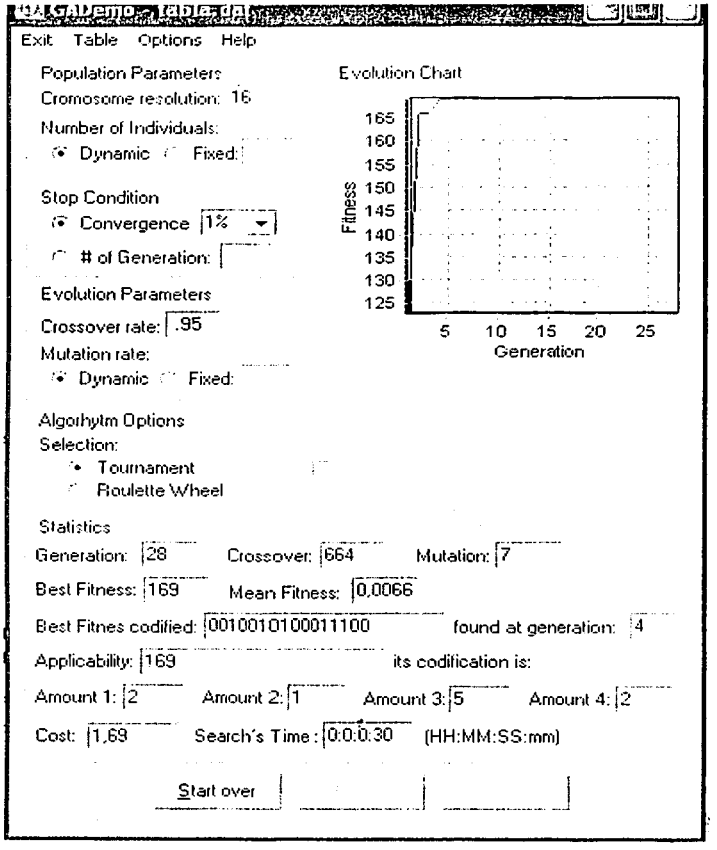


Figura AA.1: Pantalla de entrada de parámetros en GADEMO.

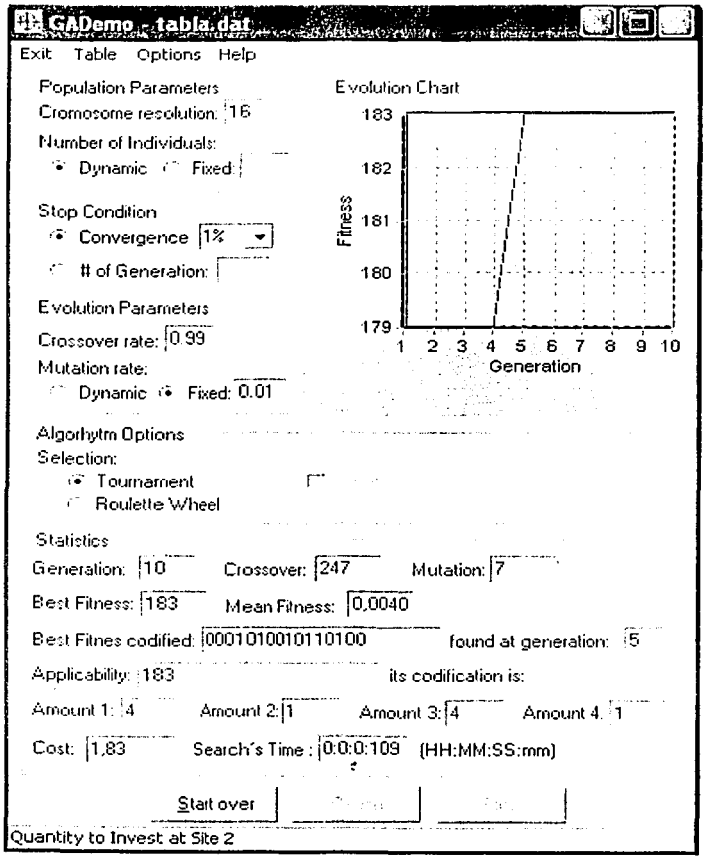


Figura AA.2: Pantalla de entrada de parámetros en GADEMO.

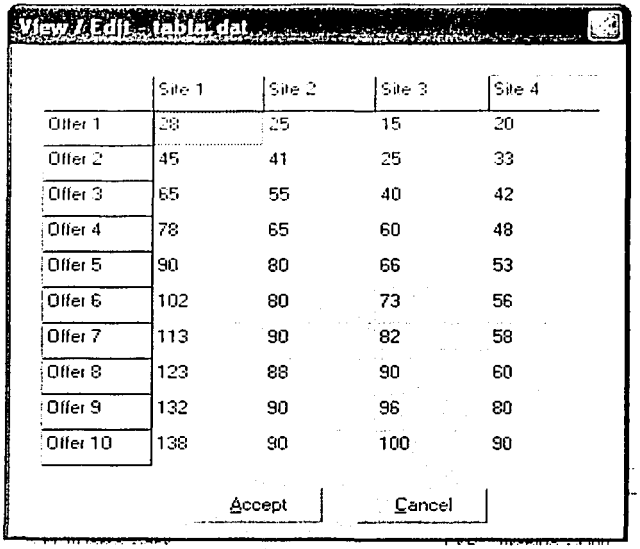


Figura AA.3: Tabla dinámica de GADEMO.

b) GAOT.- Se empleó el toolbox de GAOT para MATLAB para comprobar la utilidad de los algoritmos genéticos en problemas continuos.

c) Y por último se empleó el GENEHUNTER para efectuar distintos pruebas de aprendizaje en los sistemas clasificadores.

Seudocódigo del desarrollo del sistema clasificador.

- La condición $C \in \{0,1,\#\}^L$ que especifica los estados de entrada en el cual el clasificador puede ser aplicado. (matches).
- La acción $A \in \{a_1,.....a_n\}$ que especifica la acción (posiblemente una clasificación) que el clasificador propone.
- La predicción p estima (guarda un promedio de) el pago esperado si el clasificador es seleccionado y esta acción es tomada por el sistema.

Más sin embargo, cada clasificador guarda ciertos parámetros adicionales:

- El error de predicción "ϵ" el cual estima el error hecho en la predicción.
- El fitness "F" que nos da el valor de aptitud de los clasificadores.
- La experiencia "exp" que cuenta el número de veces desde que esta creación que el clasificador ha pertenecido a una acción.
- El tiempo gastado "ts" (time-step) que denota el tiempo de la última ocurrencia de un Algoritmo Genético en una acción en el cual perteneció a un clasificador.
- El conjunto del tamaño de la acción "as" (average size) estima el tamaño del promedio de la acción.
- La numerosidad "num" refleja el número de micro-clasificadores (clasificador ordinario).

Para referirnos a uno de los atributos de un clasificador "cl", usaremos x_{cl} donde x puede ser cualquier atributo mencionado anteriormente (por ejemplo $x \in \{C,A,p,\epsilon,f,exp,ts,as,num\}$).

SCG:

1. inicializar el entorno *env*
2. inicializar el programa de refuerzo *rp*
3. inicializar SCG
4. CORRER EL EXPERIMENTO

CORRER EL EXPERIMENTO().

```
1       ρ-1 ← 0
2       do {
```

```
3       σ ← env: get situation
4       GENERATE MATCH SET [M] out of [P] using σ
5       GENERATE PREDICTION ARRAY PA out of [M]
6       act ← SELECT ACTION according to PA
7       GENERATE ACTION SET [A] out of [M] according to act
8       env: execute action act
9       ρ ← rp: get reward
10      if ([A] – 1 is not empty)
11              P ← ρ - 1 + γ * max (PA)
12              UPDATE SET [A] using P possibly deleting in [P]
13              RUN GA in [A] – 1 considering σ - 1 inserting and
                        possibly deleting in [P]
14      if (rp: eop)
15              P ← ρ
16              UPDATE SET [A] using P possibly deleting in [P]
17              RUN GA in [A] considering σ inserting and
                        possibly deleting in [P]
18              empty [A] – 1
19      else
20              [A] – 1 ← [A]
21              ρ - 1 ← ρ
22              σ - 1 ← σ
23 } while (termination criteria are not met)
```

GENERATE MATCH SET ([P], σ).

```
1       initialize empty set [M]
2       while ([M]) is empty)
3               for each classifier cl in [P]
4                       if (DOES MATCH classifier cl in situation σ)
5                               add classifier cl to set [M]
6               if (the number of different actions in [M] < θ_mna)
7                       GENERATE COVERING CLASSIFIER cl_c considering [M] and σ
8                       add classifier cl_c to set [P]
9                       DELETE FROM POPULATION [P]
10                      empty [M]
11      return [M]
```

DOES MATCH (cl,σ).

```
1       for each attribute x in C_cl
2               if ( x ≠ # and x ≠ the corresponding attribute in σ)
3                       return false
4       return true
```


GENERATE COVERING CLASSIFIER ([M], σ).

```
1       initialize classifier cl
2       initialize condition C_cl with the length of σ
3       for each attribute x in C_cl
4               if (RandomNumber[0,1] < P_#)
5                       x ← #
6               else
7                       x ← the corresponding attribute in σ
8       A_cl ← random action not present in [M]
9       p_cl ← p_I
10      ∈_cl ← ∈_I
```

```
11      f_cl ← f_l
12      exp_cl ← 0
13      ts_cl ← actual time t
14      as_cl ← 1
15      num_cl ← 1
16                                              return cl
```

GENERATE PREDICTION ARRAY ([M])

```
1       initialize prediction array PA to all null
2       initialize fitness sum array FSA to all 0.0
3       for each classifier cl in [M]
4               if (PA[A_cl] = null)
5                       PA[A_cl] ← p_cl * f_cl
6                                                                       else
7                       PA[A_cl] ← PA[A_cl] + p_cl * f_cl
8               FSA[A_cl] ← FSA[A_cl] + f_cl
9       for each possible action A
10              if (FSA[A] is not zero)
11                      PA[A] ← PA[A] / FSA[A]
12      return PA
```

SELECTION ACTION (PA).

```
1       if (RandomNumber[0,1] < p_explr)
2               // Do pure exploration here
3               return a randomly chosen action from those not null in PA
4       else
5               // Do pure exploitation here
6               return the best action in PA
```

GENERATION ACTION SET ([M], act).

```
1       initialize empty set [A]
2       for each classifier cl in [M]
3               if (A_cl =act)
4               add classifier cl to set [A]
```

UPDATE SET ([A], P, [P]).

```
1       for each classifier cl in [A]
2               exp_cl++
3               // update prediction p_cl
4               if (exp_cl < 1/β)
5                       p_cl ← p_cl + (P – p_cl) / exp_cl
6               else
7                       p_cl ← p_cl + β * (P – p_cl)
8               // update prediction error ∈_cl
9               if (exp_cl < 1/β)
10                      ∈_cl ← ∈_cl + ( |P – p_cl | - ∈_cl) / exp_cl
11              else
12                      ∈_cl ← ∈_cl +β * ( |P – p_cl | - ∈_cl
13              // update action set size estimate as_cl
14              if (exp_cl < 1 / β)
15                      as_cl ← as_cl + (∑_{c∈[A]} num_c – as_cl) / exp_cl
16              else
17                      as_cl ← as_cl + β * ( ∑_{c∈[A]} num_c – as_cl)
18      UPDATE FITNESS in set [A]
19      if (doActionSetSubsumption)
```

```
20              DO ACTION SET SUBSUMPTION in [A] updating [P]

UPDATE FITNESS ([A]).
1       accuracySum ← 0
2       initialize accuracy vector k
3       for each classifier cl in [A]
4               if (∈cl < ∈0)
5                       k(cl) ← 1
6               else
7                       k(cl) ← α * ( ∈cl / ∈0 ) ^-ν
8               accuracySum ← accuracySum + k(cl) * numcl
9       for each classifier cl in [A]
10              fcl ← fcl + β * (k(cl) * numcl / accuracySum – fcl)

RUN GA([A],σ,[P]).
1       if (actual time t - Σcl∈[A] tscl * numcl / Σcl∈[A] numcl > θGA)
2               for each classifier cl in [A]
3                       tscl ← actual time t
4               parent1 ← SELECT OFFSPRING in [A]
5               parent2 ← SELECT OFFSPRING in [A]
6               child1 ← copy classifier parent1
7               child2 ← copy classifier parent2
8               numchild1 = numchild2 ← 1
9               expchild1 = expchild2 ← 0
10              if (RandomNumber[0,1] < X)
11                      APPLY CROSSOVER on child1 and child2
12                      pchild1 ← (pparent1 + pparent2) / 2
13                      ∈child1 ← (∈parent1 + ∈parent2) / 2
14                      fchild1 ← (fparent1 + fparent2) / 2
15                      pchild2 ← pchild1
16                      ∈child2 ← ∈child1
17                      fchild2 ← fchild1
18              fchild1 ← fchild1 * 0.1
19              fchild2 ← fchild2 * 0.1
20              for both children child
21                      APPLY MUTATION on child according to σ
22              if ( doGASubsumption)
23                      if (DOES SUBSUME parent1,child)
24                              numparent1++
25                      else    if (DOSE SUBSUME parent2,child)
26                              numparent2++
27                              else
28                              INSERT child IN POPULATION [P]
29              else
30                      INSERT child IN POPULATION [P]
31      DELETE FROM POPULATION [P]

SELECTION OFFSPRING ([A]).
1       fitnessSum ← 0
2       for each classifier cl in [A]
3               fitnessSum ← fitnessSum + fcl
4       choicePoint ← RandomNumber[0,1] * fitnessSum
5       fitnessSum ← 0
6       for each classifier cl in [A]
7               fitnessSum ← fitnessSum + fcl
```

```
8                   if (fitnessSum > choicePoint)
9                   return cl
```

APPLY CROSSOVER (cl1,cl2).

```
1       x ← RandomNumber[0,1] * (length of C_cl + 1)
2       y ← RandomNumber[0,1] * (length of C_cl + 1)
3       if (x > y)
4                   switch x and y
5       i ← 0
6                                                   do {
7                   if ( x ≤ i and i < y )
8                           switch C_cl1[i] and C_cl2[i]
9                   i++
10      } while (i < y)
```

APPLY MUTATION (cl, σ).

```
1       i ← 0
2       do {
3                   if (RandomNumber [0,1] < μ )
4                           if (C_cl[i] = # )
5                                   C_cl[i] ← σ[i]
6                           else
7                                   C_cl[i] ← #
8                   i++
9       } while ( i < length of C_cl )
10      if (RandomNumber[0,1] < μ)
11                  A_cl ← a randomly chosen other possible action
```

INSERT IN POPULATION (cl,[P]).

```
1       for all c in [P]
2                   if (c is equal to cl in condition and action)
3                           num_c++
4                           return
5       add cl to set [P]
```

DELETE FROM POPULATION ([P]).

```
1       if (Σ_c∈[P] num_c ≤ N)
2                   return
3       avFitnessInPopulation ← Σ_c∈[P] f_c / (Σ_c∈[P] num_c
4       voteSum ← 0
5       for each classifier c in [P]
6                   voteSum ← voteSum + DELETION VOTE of c
                                        with avFitnessInPopulation
7       choicePoint ← RandomNumber[0,1] * voteSum
8       voteSum ← 0
9       for each classifier c in [P]
10                  voteSum ← voteSum + DELETION VOTE of c
                                        with avFitnessInPopulation
11                  if (voteSum > choicePoint)
12                          if (num_c > 1)
13                                  num_c- -
14
                                                    else
```

```
15                                  remove classifier c from set [P]
16                          return

DELETION VOTE (cl, avFitnessInPopulation).
1       vote ← as_cl * num_cl
2       if (exp_cl > θ_del and f_cl / num_cl < δ * avFitnessInPopulation)
3               vote ←vote * avFitnessInPopulation / (f_cl / num_cl)
4       return vote

DO ACTION SET SUBSUMPTION ([A],[P]).
1       initialize cl
2       for each classifier c in [A]
3               if (c COULD SUBSUME)
4                       if (cl empty or number of # in C_c > number of # in C_cl
                           or (number of # in C_c = number of # in C_cl and
                           Random_Number[0,1] < 0.5))
5                       cl ← c
6       if (cl is not empty)
7               for each classifier c in [A]
8                       if (cl IS MORE GENERAL than c)
9                               num_cl ← num_cl + num_c
10                              remove classifier c from set [A]
11                              remove classifier c from set [P]

COULD SUBSUME (cl).
1       if (exp_cl > θ_sub)
2               if (ε_cl < ε_0)
3                       return true
4       return false


IS MORE GENERAL (cl_gen, cl_spec).
1       if (the number of # in cl_gen.C ≤ the number of # in cl_spec.C)
2               return false
3       i ← 0
4       do {
5               if (C_clgen.[i] ≠ # and C_clgen.[i] ≠ C_clspec.[i])
6                       return false
7       i++
8       } while (i < length of C_clgen)
9       return false

DOES SUBSUME (cl_sub, cl_los).
1       if (A_clsub = A_cllos)
2               if (cl_sub COULD SUBSUME)
3                       if (cl_sub IS MORE GENERAL than cl_los)
4                               return true
5                                       return false
```


Pruebas realizadas con los modelos XCS, ZCS y ACS.

El proceso de la creación de la metodología tuvo que ser apoyada con un modelo que ha demostrado tener buenos resultados en diferentes ambientes y el propósito de esta sección es mostrar dichos resultados con los tres modelos que ahora en día han tenido mayor auge y por lo tanto son los usados en estas corrientes por los investigadores que estudian los casos de los sistemas clasificadores y me refiero a los modelos XCS y ZCS propuestos por Wilson [141][142] y al modelo ACS propuesto por Stolzmann. De acuerdo a varias pruebas, encontramos los siguientes resultados concluyendo que en diferentes escenarios o ambientes el modelo XCS y ACS resultan ganadores: Rendimiento de la prueba "Animat" de los modelos ZCS, XCS y ACS .

La tarea del agente es encontrar el objetivo que se mostrará en cada de las pruebas que se mostrarán en las figuras, por ejemplo una comida, haciéndolo en lo posible en los menos números de pasos (el máximo 50 pasos). El agente puede entrar en las 8 posibles direcciones, sin embargo, sólo puede mover a través de los espacios vacíos. La posición inicial de Animat es escogida estocásticamente. Los sistemas clasificadores de aprendizaje (SCA) usados en los experimentos no tienen memoria o procedimientos de chequeo de las acciones. Así en algunos ambientes, el sistema clasificador en problemas No-Markovianos no puede encontrar el óptimo.

Las siguientes gráficas son el resultado de 10 experimentos con 200 pruebas. El modelo ZCS emplea un tamaño de población constante, mientras que en los modelos XCS y ACS cambia su tamaño. La comparación la hacemos solamente con estos tres modelos, pues en estos se extraen las características principales de los modelos anteriores y actualmente son los tres modelos que se emplean para desarrollar aplicaciones en el mundo real, en el caso de la metodología que se propone en esta tesis se emplea el modelo propuesto por Wilson (XCS) por ser el mas propicio para aplicaciones robustas y en tiempo real, además de considerar al fitness del SC de acuerdo con su precisión en lugar de la evaluación de pesos de los modelos alternos. A continuación se mencionan las principales pruebas que se desarrollaron con estos modelos:

- Entorno Markoviano: ***Woods1, Woods2, Woods14, MazeF1, MazeF2, MazeF3, Maze4, Maze5, Maze6, MazeMA1***

- Entorno No-Markoviano: ***Woods7, Woods100, Woods101, Woods101 ½, Woods102, Mazef4, Maze7, Maze10***

- **ZCS:** A Zeroth Level Classifier System [141],
- **XCS:** Classifier System based on Accuracy [142] y,
- **ACS:** Anticipatory Classifier System.

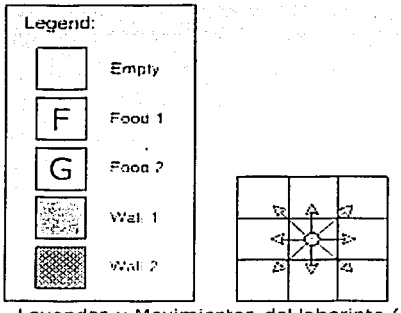


Fig. A.1. Leyendas y Movimientos del laberinto (Maze).

A.1 Woods1

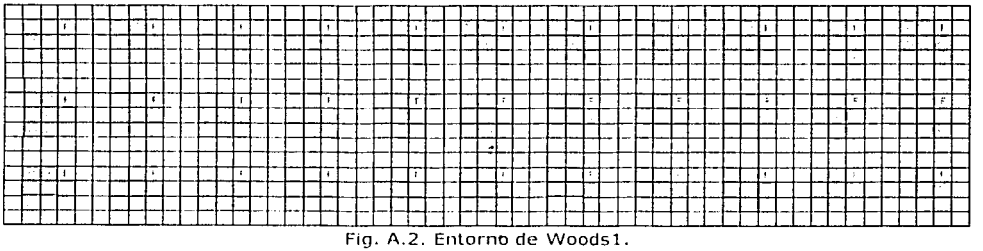


Fig. A.2. Entorno de Woods1.

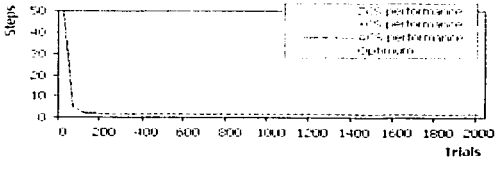


Fig. A.3. Rendimiento del SCA sobre Woods1

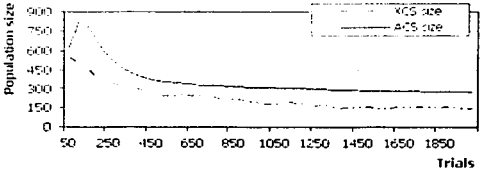


Fig. A.4. Tamaño de la población del SCA

A.2 Woods2

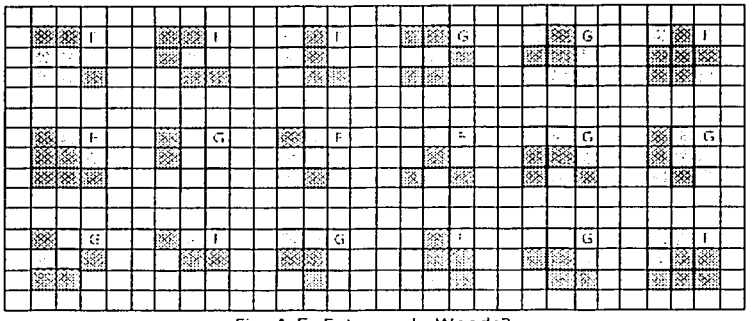


Fig. A.5. Entorno de Woods2.

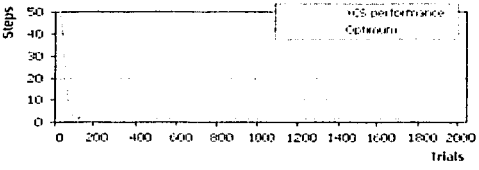


Fig. A.6. Rendimiento del SCA sobre Woods2

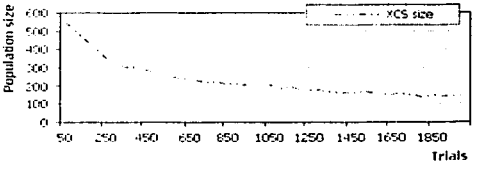


Fig. A.7. Tamaño de la población del SCA

A.3 Woods7

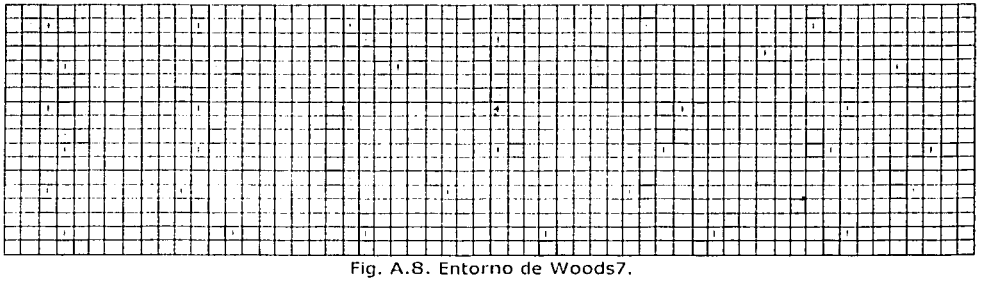


Fig. A.8. Entorno de Woods7.

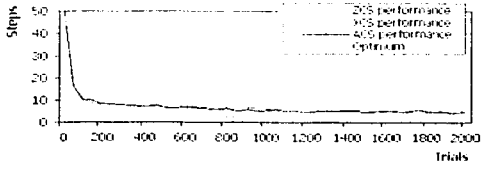


Fig. A.9. Rendimiento del SCA sobre Woods7

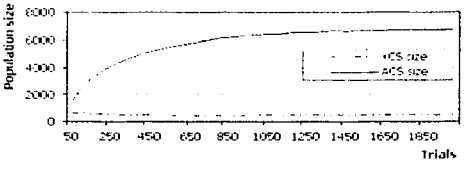


Fig. A.10. Tamaño de la población del SCA

A.4 Woods14

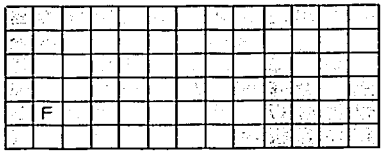


Fig. 6.11. Entorno de Woods14.

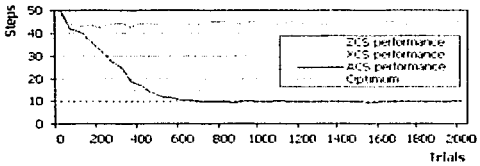


Fig. A.12. Rendimiento del SCA sobre Woods14

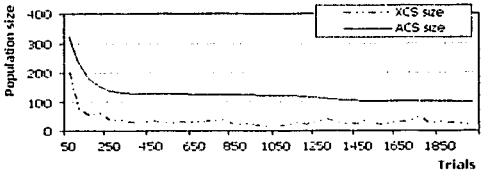


Fig. A.13. Tamaño de la población del SCA

A.5 Woods100

Fig. 6.14. Entorno de Woods100.

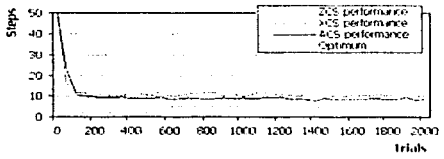


Fig. A.15. Rendimiento del SCA sobre Woods100

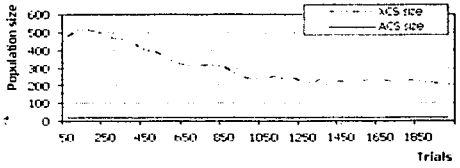


Fig. A.16. Tamaño de la población del SCA

A.6 Woods101

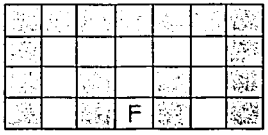


Fig. A.17. Entorno de Woods101.

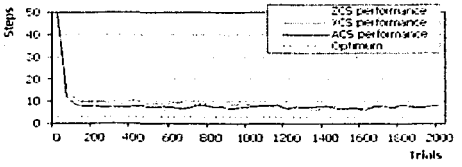


Fig. A.18. Rendimiento del SCA sobre Woods101

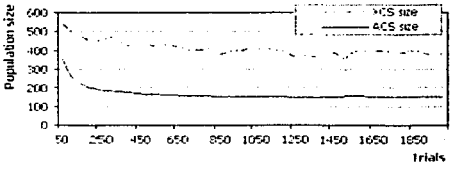


Fig. A.19. Tamaño de la población del SCA

A.7 Woods101 1/2

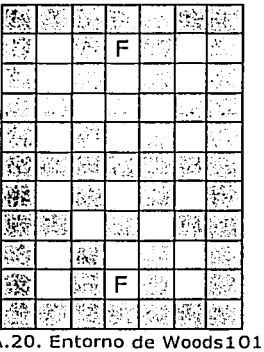


Fig. A.20. Entorno de Woods101 ½.

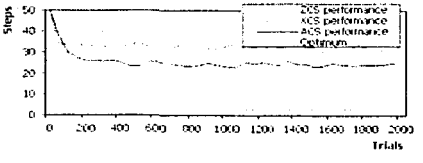


Fig. A.21. Rendimiento del SCA sobre Woods101 1/2

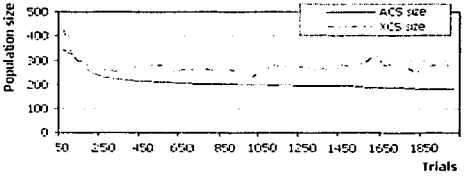


Fig. A.22. Tamaño de la población del SCA

A.8 Woods102

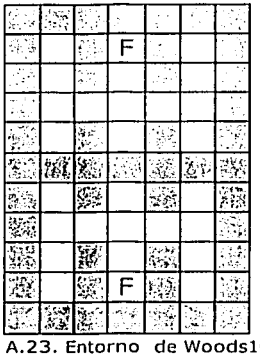


Fig. A.23. Entorno de Woods102.

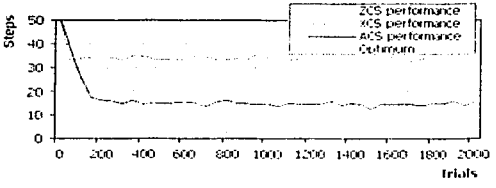


Fig. A.24. Rendimiento del SCA sobre Woods102

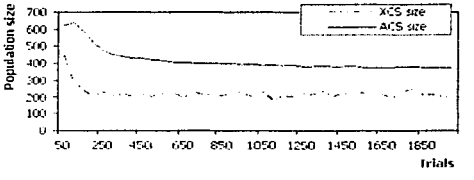


Fig. A.25. Tamaño de la población del SCA

A.9 MazeF1

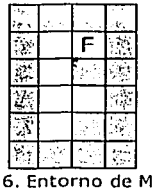


Fig. A.26. Entorno de MazeF1.

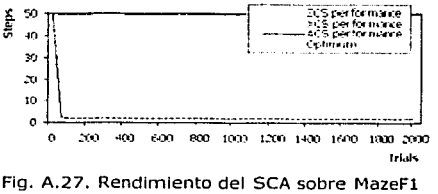


Fig. A.27. Rendimiento del SCA sobre MazeF1

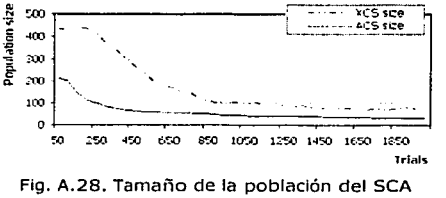


Fig. A.28. Tamaño de la población del SCA

A.10 MazeF2

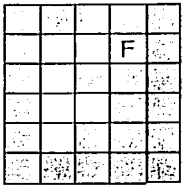


Fig. A.29. Entorno de MazeF2.

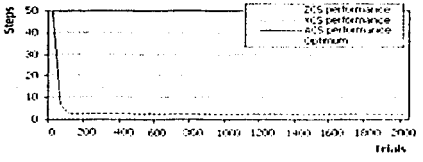


Fig. A.30. Rendimiento del SCA sobre MazeF2

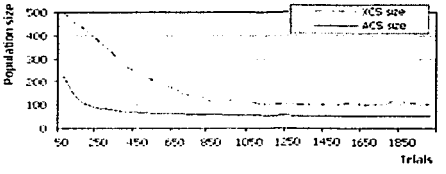


Fig. A.31. Tamaño de la población del SCA

A.11 MazeF3

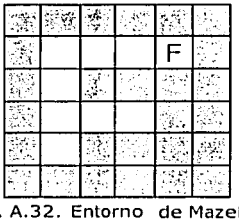


Fig. A.32. Entorno de MazeF3.

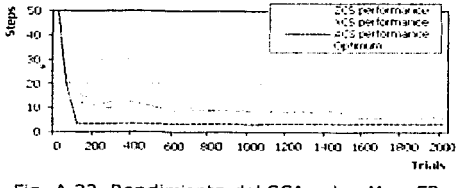


Fig. A.33. Rendimiento del SCA sobre MazeF3

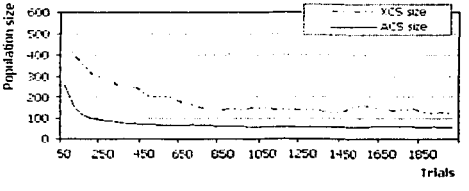


Fig. A.34. Tamaño de la población del SCA

A.12 MazeF4

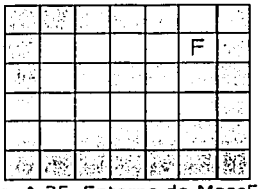


Fig. A.35. Entorno de MazeF4.

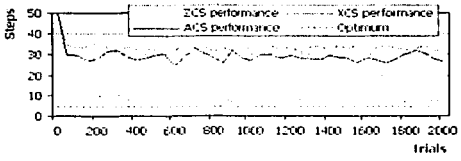


Fig. A.36. Rendimiento del SCA sobre MazeF4

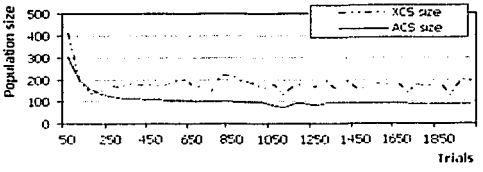


Fig. A.37. Tamaño de la población del SCA

A.13 Maze4

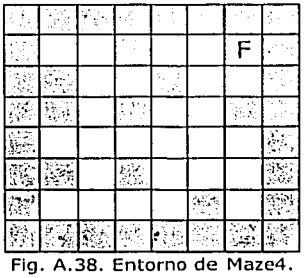


Fig. A.38. Entorno de Maze4.

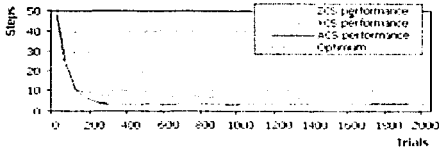


Fig. A.39. Rendimiento del SCA sobre Maze4

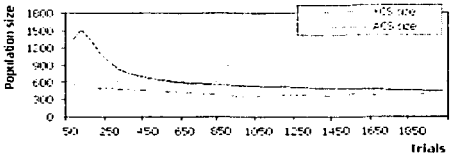


Fig. A.40. Tamaño de la población del SCA

A.14 Maze5

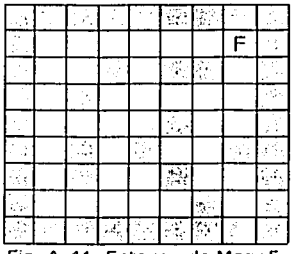


Fig. A.41. Entorno de Maze5.

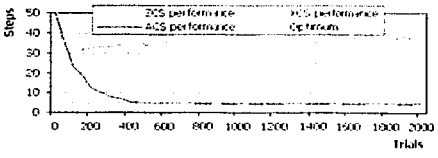


Fig. A.42. Rendimiento del SCA sobre Maze5

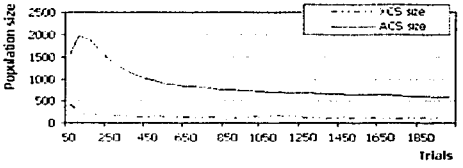


Fig. A.43. Tamaño de la población del SCA

6.15 Maze6

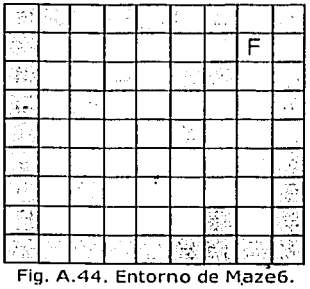


Fig. A.44. Entorno de Maze6.

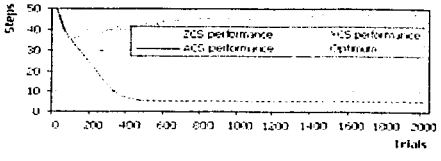


Fig. A.45. Rendimiento del SCA sobre Maze6

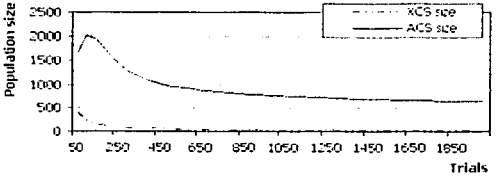


Fig. A.46. Tamaño de la población del SCA

A.16 Maze7

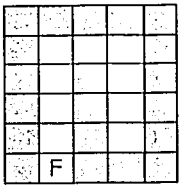


Fig. 6.47. Entorno de Maze7.

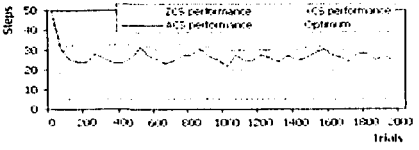


Fig. A.48. Rendimiento del SCA sobre Maze7

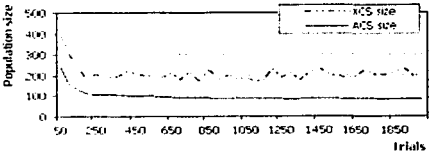


Fig. A.49. Tamaño de la población del SCA

A.17 Maze10

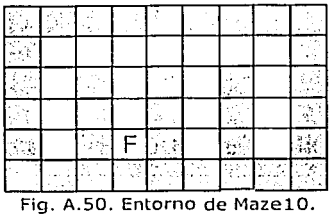


Fig. A.50. Entorno de Maze10.

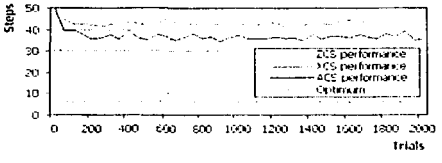


Fig. A.51. Rendimiento del SCA sobre Maze10

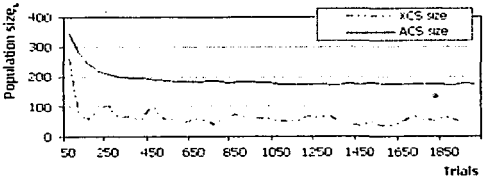


Fig. A.52. Tamaño de la población del SCA

A.18 MazeMA1

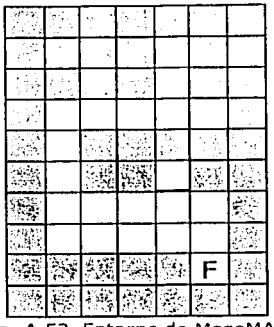


Fig. A.53. Entorno de MazeMA1.

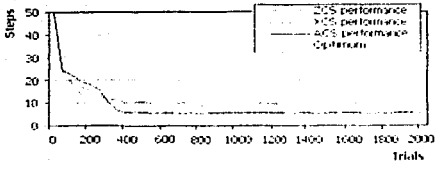


Fig. A.54. Rendimiento del SCA sobre MazeMA1

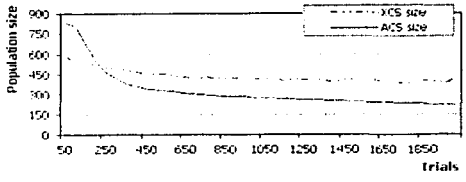


Fig. A.55. Tamaño de la población del SCA

Parámetros:

- ZCS from the book "The Computational Beauty of Nature", 1997
- ACS v2.0 (23 Feb 2001) University of Illinois at Urbana/Champaign.
- XCSJava v1.0 (06 Jun 2000) University of Illinois at Urbana/Champaign.

Parámetros usados en los experimentos:

ZCS parameters	XCS parameters	ACS parameters
Population size = 800	Max population size = 800	Beta = 0.05
Initial classifier strength = 20	Fall of rate in the fitness evaluation = 0.1	Gamma = 0.95
BB learning rate = 0.2	Learning rate for updating, prediction, pred. error, and action set size estimate = 0.2	Theta_i = 0.1
BB discount rate = 0.71	Discount rate = 0.95	Theta_r = 0.9
Tax rate for strength reduce = 0.1	Deletion delta = 0.1	R_ini = 0.5
Ga constants:	Exponent in the power function for the fitness evaluation = 5	Ir_ini = 0
GA crossover rate = 0.5	Threshold for the GA application in an action set = 25	Q_ini = 0.5
GA mutation rate = 0.002	Error threshold under which the accuracy of a classifier is set to one = 10	Avt_ini = 0
GA invocation rate = 0.25	Specified the threshold over which the fitness of a classifier may be considered in its deletion probability = 20	Q_alp_min = 0.5
Covering factor = 0.5	GA crossover rate = 0.8	Q_ga_decrease = 0.5
Probability of # in cover = 0.33	GA mutation rate = 0.04	U max = 100000
	Probability of # in cover = 0.33	Do pees = false
	Reduction of the prediction error when generating an offspring classifier = 0.25	Do mental acting steps = false
	Reduction of the fitness when generating an offspring classifier = 0.1	Do lookahead winner = false
	Experience of a classifier required to be a subsumer = 20	Probability of exploration = 1.0
	Do GA subsumption = true	The probability of applying an exploration biased action-selection = 0.5
	Do action set subsumption = true	Exploration bias method = 50/50
	Initial pred. value when generating a new classifier (e.g in covering) = 10.0	Do action planning = false
	Initial pred. error value when generating a new classifier (e.g in covering) = 0.0	Ga constants:
	Initial pred. value when generating a new classifier (e.g in covering) = 0.01	Theta_ga = 100
		Mu = 0.30
		X_type = two-point crossover
		Chi = 0.8
		Theta_as = 20
		Theta_exp = 20
		Do subsumption = true

ANEXO B
GLOSARIO

Alelo:

Alelo: Un alelo es un valor para un gen. Utilizando la definición de gen que hace referencia a una porción de cadena genética determinada, los alelos de ese gen serían todos los posibles valores que puede tomar ese segmento de cadena genética.

Genes alelos: a veces se habla de genes alelos, aún cuando sería más apropiado decir *valores de genes alelos*, o simplemente, *alelos*. Se dice que "dos genes son alelos uno respecto del otro cuando son rivales en cuanto al mismo lugar del cromosoma, como por ejemplo, los genes que determinan los ojos castaños y los que determinan los ojos azules". Más propiamente, esto sería lo mismo que decir que dos alelos (valores de gen) lo son del mismo gen si son rivales en el mismo lugar del cromosoma. Por ejemplo, los alelos que determinan los ojos castaños y los alelos que determinan los ojos azules son alelos del mismo gen: el gen que determina el color de los ojos.

Alelos Equivalentes: dos alelos son equivalentes si se comportan de forma idéntica en cuanto a la solución que representan. Por ejemplo, en un problema de optimización de ciertos parámetros, los genotipos podrían definirse como un conjunto de 5 valores numéricos con un decimal. Por ejemplo, dos genotipos podrían ser:

```
1.4; 6.3; 7.4; 4.1; 4.5
3.7; 8.4; 7.2; 2.1; 1.9
```

Sin embargo, es posible que el tercer parámetro represente una magnitud entera, y a la hora de evaluar la solución propuesta, sea indiferente el dígito decimal del tercer parámetro. En ese caso, 7.4 y 7.2 serían alelos equivalentes del tercer gen.

Dos alelos son equivalentes si uno de ellos puede ser sustituido por el otro obteniéndose como resultado un genotipo de exactamente el mismo peso (utilidad), y además, esto ocurre siempre, independientemente de los valores del resto de genes.

Superalelo: conjunto de alelos

Superalelos equivalentes: dos superalelos (conjuntos de alelos) son equivalentes si uno de ellos puede ser sustituido por el otro obteniéndose como resultado un genotipo válido.

Superalelos de peso equivalente: dos superalelos (conjuntos de alelos) son de peso equivalente si se comportan de forma idéntica en cuanto a la solución al problema que representan.

Dos superalelos (conjuntos de alelos) son de peso equivalente si uno de ellos puede ser sustituido por el otro obteniéndose como resultado un genotipo de exactamente el mismo peso, y esto ocurre siempre, independientemente de los valores del resto de genes.

Algoritmos Genéticos-Genetic Algorithms (AG-GA).

Los pasos que realiza un AG son:

1.- Se genera un conjunto de 1-N soluciones válidas al problema. Normalmente, esta(s) entidades se generan al azar. Valores típicos son 20-200.

2.- Se evalúan las soluciones existentes, de manera que se eliminan unas y se mantienen otras (otra posibilidad sería limitar el tiempo de ejecución)

3.- Se permite la reproducción o recombinación de genes (normalmente por parejas) de las entidades existentes. Por ejemplo, se realizan cruzamientos de patrones a partir de cierto punto elegido al azar, de forma que los nuevos patrones posean un segmento de cada uno de los progenitores.

4.- Se efectúan mutaciones (cambios al azar en los genes) de los nuevos patrones, según una tasa determinada.

5.- Se continúa en el paso 2 hasta que se encuentre una entidad que se considere con suficientemente peso o *fitness*

Algoritmos Miméticos:

Son la combinación de un algoritmo genético, que se encarga de realizar la búsqueda aplicando el énfasis en la exploración (exploración más a fondo del espacio de búsqueda), con otro algoritmo especializado en la explotación (convergencia más rápida hacia una solución).

Aprendizaje:

Supone un proceso inteligente, repetitivo, eficaz y eficiente.

Es interesante porque permite:

- resolver problemas cambiantes
- detectar y corregir conocimiento que se ha introducido inicialmente y es incorrecto
- resolver problemas en entornos inaccesibles
- Resolver problemas desconocidos: por ejemplo, reconocer patrones, aún cuando no sabemos cuáles estamos buscando.

En cuanto al tema que nos interesa, el aprendizaje se puede obtener de varias maneras:

- Imitando el cerebro: (Aprendizaje a nivel de individuo)

Imitando la evolución: (Aprendizaje a nivel de especie)

Autómatas Celulares-Cellular Automata (AC-CA).

Los autómatas celulares son redes de autómatas simples conectados localmente que producen una salida a partir de una entrada, modificando en el proceso su estado según una función de transición.

Un autómata celular es una máquina de estados finitos que consiste en una cuadrícula de células en la cual la evolución de cada célula depende de su estado actual y de los estados de sus vecinos inmediatos. En un autómata celular todos los autómatas simples o células pasan a la siguiente generación al mismo tiempo y según un mismo algoritmo de cambio que puede hacer variar su estado dentro de un conjunto limitado de estados.

Azar, Aleatorio:

Aleatorio: sin causa, y por tanto, teóricamente impredecible, excepto en forma de probabilidad

Proceso aleatorio: proceso cuyo resultado es impredecible, excepto en forma de probabilidad

Cadena aleatoria: cadena para la que no existe ninguna forma de comprimir su descripción

Se ha de notar que un proceso aleatorio no siempre genera cadenas aleatorias, y puede generar cadenas deterministas. También ocurre que un proceso pseudo aleatorio, como una función de azar de ordenador, puede generar cadenas deterministas.

No existen generadores de azar puros (perfectos), ni en software ni en la naturaleza. Es decir, no existe el azar, todo lo que hemos visto tiene una causa. Azar es la etiqueta que ponemos a aquello que aparentemente varia sin ninguna pauta o norma, es decir, su pauta es no parecer tener pauta; en realidad si hay una pauta, que es la ausencia de pauta. Se discuten dos excepciones a esta afirmación: el Big Bang y los procesos cuánticos. Podemos decir que la dirección del viento se origina al azar cuando no es así exactamente. Por eso llamamos azar a una pauta complicada (y a veces se obliga que sea homogéneamente distribuida, o distribuida según cierta forma) que nos sirve para un propósito, como las funciones random (RND) en programación.

Los humanos poseemos un razonamiento "inconsciente" o "impulsivo" que nos obliga a establecer cierto causa-efecto aun cuando un razonamiento más consciente o "de nivel superior" nos dicta lo contrario, esto ocurre, por ejemplo, cuando compramos un numero de lotería y nos dan el 00001. Nos parece "impulsivamente" (al menos a mi y a muchas otras personas consultadas) que ese numero tiene muchas menos probabilidades de salir

que el 34.788, cuando "conscientemente" (a alto nivel) sabemos perfectamente que tienen las mismas posibilidades...

...pues bien, no tienen las mismas posibilidades, aunque esto poco tiene que ver con el aspecto de los números. El que salga uno u otro número depende de la forma de las bolas y del bombo, la configuración inicial de bolas y la fuerza aplicada al bombo, etc. La capacidad de asignación de causa-efecto es muy ventajosa para la supervivencia de un individuo y por eso la naturaleza lo ha seleccionado, a ella debemos poder predecir los inviernos (después del otoño viene el invierno) y así evitar él frió y el hambre buscando una cueva y almacenando nueces. Probablemente la evolución hizo aparecer los razonamientos causa-efecto inconsciente antes que los conscientes.

En el ejemplo, probablemente el proceso mental que lleve a estas asociaciones sea algo parecido al que a continuación se describe. La persona observa los números que son premiados en los distintos sorteos, e inconscientemente trata de detectar regularidades en esos números. Los números por lo general no son fáciles de recordar, por lo que números como el "72.719" se olvidan fácilmente. Sin embargo, hay algunos muy llamativos, pero escasos, como el "33.333" o el "12.345." difíciles de olvidar. La persona detecta entonces que la mayoría de los números premiados son del tipo de los "difíciles de recordar" y por tanto desconfía de los "fáciles de recordar".

Decidir que ciertos sucesos se producen al azar puede detener ciertas investigaciones que podrían ser fructíferas. Aunque algo parezca aleatorio, puede que no sea una pérdida de tiempo el buscar causas. Esto es lo que ha sucedido en el estudio del caos: ciertos procesos de apariencia aleatoria pueden ser producidos por un conjunto muy pequeño de reglas simples y deterministas. Por tanto, es posible el camino inverso: dado un proceso de apariencia aleatoria, tal vez haya sido producido por un conjunto muy pequeño de reglas simples y deterministas. Plantear la hipótesis de que las bolas del bombo son "inteligentes" o tienen memoria, buscar una causa (por ejemplo, una inteligencia que se dedica a forzar una distribución homogénea de números) y otros razonamientos de este tipo, aunque no son correctos en si mismos (en el caso de las bolas), marchan en la dirección correcta hacia nuevos descubrimientos y en general son interesantes, lógicos y demuestran inteligencia.

Por supuesto, todavía mayor inteligencia demuestra el razonamiento lógico consciente que asigna la misma probabilidad a todas las combinaciones de números de la lotería, sean o no fáciles de recordar.

Se puede recurrir al azar para explicar la aparición de vida en la Tierra, en relación con las preguntas sobre la vida inteligente en otros planetas. ¿Cuál es la probabilidad de encontrarse una cajetilla de cigarrillos en el desierto? Baja, ciertamente; pero una vez que la hemos encontrado, una vez que, de la forma más sorprendente, en el lugar más inhóspito y solitario que podríamos imaginar, tenemos un suceso extraño, hemos visto la caja de cigarrillos y la tenemos en la mano, la probabilidad de que en las proximidades encontremos un billete de metro, aumentan enormemente. De poco sirve decir que la cajetilla de cigarrillos está ahí completamente al azar, que "nos ha tocado la lotería de encontrar la cajetilla". Tal vez sea cierto, pero detiene la investigación.

Cuando sucede algo improbable, podemos, o bien seguir pensando que es improbable, o bien empezar a plantearnos si eso que creíamos improbable tiene una causa, una razón, y que puede repetirse: que realmente no es improbable. Decidir que la causa de un suceso es el azar no conduce a nada nuevo. El azar, como el infinito, es algo de lo que podemos hablar pero con lo que nadie se ha topado aún. Lo que llamamos azar es una etiqueta que colocamos a un conjunto de causas que se nos presenta demasiado complejo como para poder entenderlo o identificarlo. Cualquiera puede programar una función de "azar" que genere números aparentemente aleatorios. Pero para el programador no existirá azar, sino un algoritmo bien conocido.

Caos:

Definición vulgar: Caos es la falta de orden por la mezcla desordenada de todos sus elementos. Lo opuesto es el orden ("Colocación de las cosas en el lugar que les corresponde"). Concierto, buena disposición de las cosas ente sí. Regla o modo que se observa para hacer las cosas. Serie o sucesión de las cosas."

Definición 2: Caos es la situación de desorden en que, según ciertas teorías o creencias, se encontraba el universo antes de adquirir su ordenación actual.

Definición 3: Sistema caótico es aquel tan sensible a las condiciones iniciales que pequeños cambios al principio se traducen en grandes cambios al final

Definición 4: Caos se define contradictoriamente como aquel comportamiento estocástico que ocurre en un sistema determinista. Estocástico significa aleatorio, y determinista hace referencia a que está gobernado por leyes exactas siendo su comportamiento totalmente predecible.

Es decir, es el comportamiento aleatorio que se da en un sistema no aleatorio: o para entendernos: es el comportamiento *aparentemente* aleatorio que se da en un sistema que realmente no es aleatorio.

Con "Caos" se designa a aquello a lo que los humanos aún no hemos podido clasificar en un orden. El verdadero caos no existe, es un producto de la mente humana. El verdadero caos es un término filosófico que implica la ausencia de capacidad de predicción.

Por ejemplo, se dice que las siguientes ecuaciones de Lorenz

```
x' = 3(y - x)
y' = -xz + 26.5x - y
z' = xy - z
```

poseen un comportamiento caótico.

Clasificadores Genéticos-Genetic Clasifiers (CG-GC) :

Los Clasificadores Genéticos son algoritmos evolutivos cuya finalidad es obtener un sistema clasificador. Normalmente los clasificadores genéticos hacen evolucionar un conjunto de reglas que serán las encargadas de realizar la clasificación, y en ese caso se pueden considerar como un tipo de Programación Evolutiva.

Computación Evolutiva (o Algoritmos Evolutivos).

Con el término de Computación Evolutiva o Algoritmos Evolutivos se engloba un conjunto de técnicas que basándose en la imitación de varios procesos naturales que intervienen en la evolución de las especies (selección natural, mutaciones, sobrecruzamientos) tratan de resolver complejos problemas de búsqueda, optimización, aprendizaje, predicción o clasificación.

La Computación Evolutiva trata de obtener aprendizaje imitando la evolución (Aprendizaje a nivel de especie). Son los métodos que basan su funcionamiento en una metáfora de la evolución biológica.

Dentro de la Computación Evolutiva-Evolutionary Computation(CE-EC) también llamado Algoritmos Evolutivos-Evolutionary Algorithms(AE-EA) se encuentran:

- Simulated Annealing.
- Algoritmos Genéticos.
- Programación Evolutiva.
- Clasificadores.
- Algoritmos Miméticos.
- Estrategias Evolutivas

Partiendo del conocimiento del funcionamiento de los Algoritmos Genéticos (probablemente la técnica más conocida de todas ellas), el resto de los algoritmos se pueden interpretar como variaciones o mejoras de los Algoritmos Genéticos (excepto Simulated Annealing que se consideraría una simplificación), añadiendo complejidad o modificando ciertas características del algoritmo.

En cuanto a los métodos empleados, podríamos considerar la Vida Artificial como un superconjunto de estas técnicas, ya que la Vida Artificial no restringe las metáforas empleadas a la imitación de los procesos evolutivos. Sin embargo, en cuanto a los objetivos perseguidos se trata de campos bien distintos, ya que en la práctica la Computación Evolutiva se utiliza para resolver problemas, y cuando se finaliza una simulación de Computación Evolutiva se pretende confiar en los resultados. En cambio la Vida Artificial se utiliza fundamentalmente para la investigación acerca del origen de la vida, sus fundamentos, sus requisitos y sus formas de desarrollo.

Paradigmas de Computación Evolutiva (Algoritmos Evolutivos)	Identificado con	Alfabeto	Población	Tipo de Mutaciones	Tasa de mutación	Individuos inmortales
Simulated Annealing (solidificación simulada, agregación simulada).	-	Cualquiera	1	Gaussianas	Variable (varianza)	No
Algoritmos Genéticos-Genetic Algorithms (AG-GA)	John H. Holland	Binario	1-N	Inversión	Constante	No
Programas de Evolución	-	Cualquiera	1-N	Variadas	Variable	Si
Estrategias Evolutivas-Evolutionary Strategies (EE-ES)	Michalewicz	Continuo	1 (simples) o N (múltiples)	Gaussianas	Variable (es parte de la cadena)	Si
Programación Genetica-Genetic Programming (PG-GP)	Koza	Instrucciones	1-N	Variadas	Variable	Si
Programación Evolutiva-Evolutionary Programming (PE-EP)	David Fogel	Reglas, autómatas	1-N	Variadas	Variable	Si
Clasificadores Genéticos-Genetic Clasifiers (CG-GC)	-	Reglas	1-N	Variadas	Variable	Si
Algoritmos Miméticos	-	Cualquiera	1-N	Variadas	Variable	Si

Cooperación:

Fuerza equilibradora, grado de penetración óptimo, compromiso, óptimo en calidad / precio.

Cromosoma:

Un cromosoma está formado por un conjunto de genes. En computación evolutiva es raro dividir la información genética en cromosomas y éstos, a su vez, en genes. Lo habitual es que cada individuo esté formado por un único cromosoma. Al existir ambos términos (cromosoma e individuo) disponibles, la comunidad científica que aplica computación evolutiva los ha utilizado para distinguir distintos matices de un mismo concepto. Así, cuando se habla de cromosoma, se suele hacer referencia a la cadena de bits o de símbolos que forman la entidad (genotipo). En cambio, cuando se usa el término individuo, se hace referencia a la interpretación de estos símbolos, a su significado como solución al problema.

Darwinismo Universal:

Argumenta (Dawkins) que toda vida, en cualquier lugar del Universo, habría evolucionado por medios darwinianos. Supone que la simple aparición de entidades autorreplicadoras ligeramente inexactas de cualquier tipo (genes, memes u otros) tiene el poder suficiente para que se produzca la evolución, y bajo condiciones adecuadas, se produce la unión de los replicadores para la creación de sistemas grandes (máquinas de supervivencia) que los dispersen favoreciendo su continua replicación.

Data Mining:

Los algoritmos genéticos y otras técnicas de Computación Evolutiva se pueden utilizar para reconocimiento de patrones en grandes bases de datos (Problema de clasificación). La típica aplicación es la de los bancos que desean prever la morosidad o no de un nuevo cliente a la hora de darle o no el crédito: el algoritmo se usa no solo para clasificar los clientes en probables morosos o no, sino en casos "difíciles" y "fáciles" de forma que la maquina sea capaz de tomar la decisión de conceder o no el préstamo en un caso sencillo siendo el hombre quien toma la decisión en los casos difíciles. Para ello, se crean reglas al azar que relacionan las variables y se les hace evolucionar en función de su habilidad para predecir con exactitud los casos que tenemos del pasado. Es el mismo método que se usa para predecir en bolsa o el tiempo atmosférico con computación evolutiva

Demostración:

El concepto de demostración es sumamente resbaladizo. Alguien puede definir que criterios son necesarios para "demostrar" algo, pero ¿cómo "demostrará" ese alguien que esos criterios son ciertos?

Existen varios significados de la palabra "demostrar":

- Demostrar como conseguir la seguridad absoluta en la certeza de una afirmación
- Demostrar como asegurar dentro de lo razonable la certeza de una afirmación
- Demostrar como la utilización correcta de unos axiomas

Demostrar algo según la primera definición del término no es posible, aunque esta afirmación tampoco es demostrable, según esta misma definición. Es decir, tampoco podemos tener la seguridad absoluta de que nunca podremos tener la seguridad absoluta en la certeza de una afirmación.

La segunda definición del término es la que habitualmente se emplea en las ciencias en general, y es subjetiva.

La tercera es propia de la lógica y de las matemáticas. En matemáticas, se considera "demostración" a la utilización coherente (correcta, sin errores) de unos axiomas (leyes que se consideran ciertas), sobre unas premisas (datos del problema actual). La conclusión obtenida no tiene por qué ser cierta, simplemente es coherente con los

axiomas y las premisas. Si las premisas no fueran ciertas o si los axiomas no fueran ciertos, no solucionaremos el problema, pero la demostración será correcta. Si el problema real es otro, y no el representado por las premisas, no solucionaremos el problema real, sino otro. En cuanto a la certeza de los axiomas, las apariencias indican que los axiomas utilizados en la lógica común y en las matemáticas son ciertos, pero no hay forma de estar seguros.

Desde este punto de vista, demostrar es como traducir, o reescribir. Es como computar, como ejecutar un programa de ordenador, como obtener unas salidas ante unas entradas. Hago hincapié en esto porque bajo este punto de vista, las entradas y salidas están formalizadas, son discretas. En cambio, en las dos primeras definiciones, o bien los conceptos que se manejan no están formalizados, son subjetivos; o bien no son discretos: son infinitos.

Determinista:

Con una causa, y por tanto, teóricamente predecible, si se conociera la causa

Proceso determinista: proceso cuyo resultado es predecible

Cadena determinista: cadena para la que si existe una forma de comprimir su descripción

Emergente:

Comportamiento emergente: comportamiento complejo producido a partir de una serie de comportamientos simples (por ejemplo, reglas simples). El comportamiento emergente se da como consecuencia de usar construcción "de abajo a arriba".

Propiedades emergentes: se trata de propiedades del sistema global que surgen de la interacción no sencilla de sus partes. En muchas ocasiones se obtienen resultados totalmente insospechados, difícilmente deducibles del conocimiento de las partes componentes y sus interacciones locales.

Epítasis:

El fenómeno de la epítasis consiste en la existencia de fuertes vinculaciones (también llamadas "interacciones") entre los genes, de forma que el efecto de unos genes inhibe o potencia otros, o incluso el resultado es tan complejo que no tiene sentido considerar los efectos de los genes de forma individual sino conjuntamente.

"El problema de la epítasis" o "el problema de la ausencia de bloques constructores" consisten en la existencia de estas interacciones entre los genes. Los problemas con poca epítasis son triviales, y se pueden solucionar con un simple método de escalada. Los

problemas con mucha epítasis son difíciles de resolver, incluso para los Algoritmos Genéticos. Un problema donde todos los genes tuvieran fuerte vinculación entre sí, es decir, cuyos efectos dependan del valor de todos los demás genes, no sería un problema irresoluble, pero sí el peor que nos podríamos encontrar, no sólo para las técnicas evolutivas, sino para cualquier otro método. Sería algo así como buscar un objeto con los ojos cerrados, y la mejor forma de tratarlo sería, lógicamente, una búsqueda secuencial.

Estrategias Evolutivas-Evolutionary Strategies (EE-ES) :

Las estrategias evolutivas surgieron inicialmente para resolver problemas de optimización paramétrica. Con el paso del tiempo fueron incorporando procedimientos propios de la computación evolutiva, con lo que han llegado a convertirse en una disciplina más.

Las *Estrategias Evolutivas Simples* son el modelo más sencillo de su paradigma, y aunque no son métodos evolutivos propiamente dichos contienen en su forma más pura gran parte de las ideas que incorporan gran parte de las *Estrategias Evolutivas Múltiples*.

En las estrategias evolutivas simples, la población esta compuesta por un solo individuo y sólo se usa el operador de mutación. Así se procede a la mutación del individuo y sustituirá al anterior si es más apto que su progenitor, en caso contrario permanece el individuo anterior.

Aunque los orígenes de ambas disciplinas partieron de enfoques distintos, los sucesivos desarrollos de ambos se han ido acercando hasta el punto de que, actualmente, una estrategia evolutiva se puede considerar como un tipo de algoritmo genético y viceversa.

Un aspecto característico de las estrategias evolutivas es que las tasas de mutación de cada gen se codifican como parte del genoma, por lo que también se encuentran sujetas a evolución.

Para describir estrategias evolutivas se usa la siguiente nomenclatura

(a+b). Reemplazo determinista por inclusión

(a,b). Reemplazo por inserción

a: tamaño de la población

a: tamaño de la descendencia

Exploración:

Fuerza innovadora, altruismo, exploración más a fondo del espacio de búsqueda.

Explotación: Fuerza conservadora, egoísmo, explotación, convergencia más rápida hacia una solución

Exponencial:

Un proceso de crecimiento es exponencial si su tasa de crecimiento es proporcional al tamaño ya alcanzado. Un proceso exponencial típico es una epidemia: cada individuo transmite un virus a varios, cada uno de los cuales lo transmite a su vez, por lo que el número de individuos afectados aumenta a un ritmo cada vez mayor.

Falsar:

Tratar de demostrar la falsedad de una hipótesis. Lo contrario de confirmar o verificar una teoría. Es interesante trabajar en ambos sentidos: intentando verificar una teoría y a la vez, intentando demostrar su falsedad. Sobre todo si la teoría tiene un origen empírico, es muy interesante este método, ya que se afianza la robustez de la teoría cada vez que ésta supera pruebas que intentan demostrar su negación.

Fenotipo:

Un fenotipo es un ser vivo, incluyendo el ser vivo como ente físico y su comportamiento. Un fenotipo es la consecuencia del desarrollo de la información genética que define un ser vivo (genotipo).

Feromona:

Hormona capaz de salir del cuerpo, de circular por el aire y de penetrar en otro cuerpo. Cuando una hormiga experimenta una sensación, la emite por todo su cuerpo y todas las hormigas de los alrededores la perciben al mismo tiempo que ella. Una hormiga estresada comunica al instante su pena al entorno, de suerte que éste solo tiene una preocupación: que cese el penoso mensaje encontrando un método para ayudar al individuo.

Filogenia:

Procedencia histórica de un grupo de seres vivos

Gaia:

Gaia es el concepto de planeta como ser vivo; significa Tierra (http://personal.redestb.es/hansburghardt/que.htm).

"Gaia es el ser vivo producto de la teoría del bioquímico inglés James Lovelock. En esta teoría se considera que el conjunto Tierra-vida es por sí mismo un ser vivo, que al igual que los seres que lo constituyen, realiza funciones impensables en la materia inerte. Este conjunto Tierra-vida recibe el nombre de Gaia, nombre con el que los griegos se referían a la diosa de la tierra".

Copérnico ya concebía al mundo como un animal, comparándolo con un organismo en el que el movimiento circular coexiste con el rectilíneo, del mismo modo que el organismo coexiste con su enfermedad. Según Bruno, la Tierra es un organismo cuyas partes están obligadas a moverse con el todo.

Gen:

Existen varias definiciones de gen. Cada definición hace referencia a distintas ideas, o lo que es lo mismo, se utiliza la palabra gen en diferentes sentidos. Por lo general, no coincide el objeto referenciado según más de una definición. Es evidente que sería mejor disponer de varios términos en vez de uno sólo, que lleva a confusión.

Los genes tienen básicamente dos funciones: reproducirse a sí mismos, e indicar el modo de construcción y comportamiento de un nuevo ser vivo completo. Los cromosomas (formados por genes) comprimen la información necesaria para realizar la construcción de un ser vivo completo. Además, los cromosomas están formados de tal manera que son capaces de crear copias de sí mismos.

Gen como unidad básica de herencia o como unidad básica de recombinación: según esta definición, un gen es la unidad mínima que se puede heredar, es decir, es la unidad mínima que puede ser tomada de uno de los progenitores para formar el nuevo individuo. Si las características heredables (como el aspecto de la cara) las descomponemos en otras subcaracterísticas (color de los ojos, color del pelo, tez), cuando ya no podamos dividir más esas características de forma que sigan siendo heredables, diremos que esa característica es debida a un gen.

Gen como unidad básica de mutación: según esta definición, un gen es la unidad mínima que se puede mutar. Este gen no tiene por qué coincidir con el gen como unidad básica de herencia.

Gen como unidad funcional: según esta definición, un gen es una secuencia orientada de nucleótidos, generalmente de ADN, que actúan como la unidad mínima de información

relativa a cuál es la constitución de otra secuencia orientada de nucleótidos de otro ácido nucleico, o de aminoácidos de una proteína.

Gen egoísta, o gen según la definición de Dawkins: esta definición es de una naturaleza distinta a las otras, ya que un gen según esta definición está formado por lo que serían multitud de genes según las otras definiciones, y además, permitiendo que estos genes componentes estén ubicados en distintas células e incluso en distintos individuos. En todas las células del cuerpo de un ser humano determinado existe la misma información genética: una larga cadena de ADN idéntica en el interior de cada célula. Para Dawkins, "*el gen egoísta no es sólo una porción física de ADN: es todas las réplicas de una porción de ADN, distribuidas por el mundo*". Es decir, el mismo alelo o valor en las mismas posiciones dentro de la cadena de ADN, es el mismo gen egoísta, ya se encuentre en uno o en distintos individuos. Un gen egoísta de los que habla Dawkins no sólo está simultáneamente en todas las células de nuestro cuerpo. Está simultáneamente en varios individuos. Esta abstracción de Dawkins sirve para expresar la siguiente idea: los genes tienen el objetivo de reproducirse a sí mismos, a costa de lo que sea, pero no a costa de otro segmento de ADN idéntico, ya que ambos son el mismo gen. Dawkins llama a los genes *egoístas* porque para ellos es inevitable intentar reproducirse a toda costa, a costa de cualquier otra cosa que encuentren a su paso, pero no a costa de otro segmento de ADN idéntico, ya que ambos son el mismo gen. Este concepto de egoísmo no tiene mucho que ver con lo que se entiende coloquialmente por egoísmo. Es más, con la teoría de Dawkins, la cooperación e incluso el altruismo (reales) entre individuos pueden ser explicados por el "egoísmo" (metafórico) de los genes.

Desde cierto punto de vista, la reproducción es equiparable al crecimiento. Podemos decir que las células de nuestro cuerpo cooperan, e incluso que se comportan de forma altruista, ya que no intentan reproducirse a costa de sus vecinas, sino que producen un crecimiento ordenado, formando un cuerpo, tal como conviene a los genes egoístas. A nivel de individuo, cuando un individuo coopera con otro y ambos comparten genes (y dos miembros de la misma especie suelen compartir más del 90% de ellos), podemos decir que en realidad lo que ocurre es que los genes egoístas se están ayudando a sí mismos. Desde este punto de vista, cuando dos individuos de la misma especie compiten, lo hacen por propagar su 10% diferencial. Es lógico que si dos individuos comparten el mismo nicho (por ejemplo, por ser de la misma especie), existirá una mayor competencia entre ellos. Pero en general, eliminando otros factores, un individuo tendrá una mayor tendencia a cooperar con otro (pudiendo elegir entre varios individuos), que será proporcional al número de genes en que coincidan. Para ello no hace falta un laboratorio, basta con observar parecidos físicos.

Genotipo:

Información genética que define un ser vivo. Información genética contenida en cada una de las células de un ser vivo. El genotipo contenido en las células sexuales puede desarrollarse, formando un ser vivo o fenotipo.

Genotipo válido: genotipo que representa una solución válida al problema, por muy mala que esta sea. Genotipo que representa una solución que se considera no prohibida.

Genotipo cuyo peso es mayor que el mínimo peso reconocido (normalmente, es un genotipo de peso mayor que cero). También puede definirse como el genotipo cuyo error es menor que el máximo reconocido (habitualmente, menor que infinito)

Golem:

Muñeco hecho de barro al que se infundió vida colocándole un corazón humano.

Hombre o mujer hecho de letras, o mejor dicho, construido por signos.

En el libro "Artificial Life explorer's kit" de Ellen Thro, publicado por SAMS Publishing, EE. UU., 1993, hay un pequeño apartado en el que habla sobre lo que es un Golem:

Golem es una palabra hebrea que significa sin sustancia, embriónico o incompleto. La referencia más vieja sobre Golems está en la Biblia, salmo 139, aunque en la versión del rey Jaime, por ejemplo, la palabra se traduce como "sustancia incompleta" (verso 16). La literatura talmúdica, y la compilación tradicional de las leyes y comentarios judíos, son otras fuentes de informacion sobre el Golem.

En la Europa medieval, se ubicaron en varias ciudades historias sobre rabies que creaban seres parecidos a los humanos. La más famosa data del siglo 16. Se dice que el rabi Judah Low de Praga, practicante de la Kabbala (un tipo de misticismo judío), Habia creado un autómata para servirle. En esta versión, la figura de barro fue animada cuando el rabí se coloco el nombre de Dios en la frente.

En los tiempos en que los judios eran perseguidos, el Golem era considerado como un protector, no como una amenaza. La ambivalencia en el personaje del Golem ha continuado hasta nuestro siglo. En 1915, una novela escrita por el alemán Gustav Meyrink, llamado El Golem, llevo la leyenda a una mayor audiencia (el libro ha sido traducido al ingles). En 1920 fue llevada a la pantalla, también en Alemania. Enfatizando las caracteristicas monstruosas del Golem, el filme fue un precedente de la pelicula original estadounidense Frankenstein en los 30's.

Heurística:

Heurístico: que incorpora conocimiento. Se utiliza normalmente en el sentido de conocimiento humano, en un problema o área particular. En muchos casos se trata de un conocimiento complejo, difícil de representar, y probablemente difícil de explicar por el propio experto humano.

Heurística: conocimiento acerca de un problema particular que difícilmente puede ser generalizado a otros problemas.

Algoritmo Heurístico: algoritmo que contiene conocimiento heurístico. Algoritmo fuertemente dependiente del problema que resuelve. Procedimiento que es posible que produzca una buena solución al problema, incluso la solución óptima, si somos afortunados; pero que puede no producir ninguna solución, o una solución pésima en otros casos.

Algoritmo Aproximado: algoritmo que siempre proporciona alguna solución, aún cuando ésta no es la óptima.

Holismo:

El todo es más que la suma de sus partes

Inteligencia Artificial-Artificial Intelligence (IA-AI).

Trata de construir máquinas con comportamiento aparentemente inteligente. El hombre es un ser inteligente. Los animales también (discutido).

En la Inteligencia Artificial se pueden observar, a grandes rasgos, dos enfoques diferentes en función del objetivo:

- La concepción de inteligencia artificial como el intento de desarrollar una tecnología capaz de suministrar al ordenador capacidades de razonamiento o discernimiento similares, o aparentemente similares a las de la inteligencia humana.
- La concepción de inteligencia artificial como investigación relativa a los mecanismos de inteligencia humana (por extensión, investigación relativa a la vida y al universo), que emplea el ordenador como herramienta de simulación para la validación de teorías.

El primer enfoque es por lo general el más práctico, se centra en los resultados obtenidos, en la utilidad, y no tanto en el método. En este enfoque se encuadran, por ejemplo, los Sistemas Expertos. Son temas claves en esta dirección la representación y la gestión del conocimiento. Algunos autores representativos de este enfoque podrían ser McCarthy y Minsky, del MIT.

El segundo enfoque está orientado a la creación de un sistema artificial que sea capaz de realizar los procesos cognitivos humanos. Desde este punto de vista no es tan importante la utilidad del sistema creado (qué hace), como lo es método empleado (cómo lo hace). Como aspectos fundamentales de este enfoque se pueden señalar el aprendizaje y la adaptabilidad. Ambos presentan gran dificultad para ser incluidos en un sistema cognitivo artificial. Esta orientación es propia de Newell y Simon, de la Carnegie Mellon University.

Es obligado indicar que frecuentemente ambas posturas no se pueden distinguir, ni siquiera en muchos trabajos de los autores mencionados como significativos en cada una de ellas.

En cuanto a los medios utilizados, se identifican también dos enfoques:

- Enfoque Top-Down, o Enfoque simbólico.

Se simulan directamente las características inteligentes del Ser Humano.

- Enfoque Bottom-Up, o Enfoque subsimbólico.

Se simulan los elementos de mas bajo nivel que componen o intervienen en los procesos inteligentes con la esperanza en que de su combinación emerja de forma espontánea el comportamiento inteligente.

Inferencia:

Mecanismo que permite obtener nuevo conocimiento a partir del existente.

Lineal:

Una función es lineal si:

$$\lambda \cdot f(x) = f(\lambda \cdot x)$$
$$f(x+y) = f(x) + f(y) \qquad \text{Para} \quad \lambda \in \Re$$

En matemáticas existe un tipo de relaciones que se expresan en forma de ecuaciones llamadas funciones. Las funciones expresan las relaciones entre cosas, o lo que es lo mismo, cómo varían unas cosas (llamadas por eso *variables*) al variar otras. En las funciones se puede considerar arbitrariamente una de las variables como variable independiente, que generalmente se denota con la letra x. Si tenemos dos variables, se tratará por ejemplo de analizar como varía (depende) una de ellas "y" respecto de la otra "x", que se considera independiente, ya que partimos de que conocemos su valor y = f(x).

Cuando en una ecuación la x esta elevada a la potencia 1, la ecuación es lineal y su grafica será una línea recta, como

$$y = 3x + 11.$$

Cuando en una ecuación la x esta elevada a una potencia diferente de 1, o la x es argumento de alguna función trascendente (como seno, coseno, logaritmo, etc.), entonces la ecuación es no lineal y su grafica no será una línea recta, como

$$y = x^2$$

$$y = \text{sen}(x)$$

La idea de linealidad implica que las cosas cambian en una proporción constante.

Supongamos que queremos una ecuación que modele el crecimiento de una población de cierto tipo, por ejemplo, de bacterias. Un modelo podría ser:

$$f(x) = 2x$$

Siendo:
x: número de bacterias actual.
f(x). número de bacterias que prevemos en la siguiente observación.

Esto significa que si en un momento dado hay una bacteria, en el siguiente periodo de tiempo habrá 2, si hay 2, habrá 4, si hay 4 habrá 8, etc.

La relación entre el número de bacterias en una observación x[t] y el número de bacterias en la siguiente observación x[t+1] es lineal.

$$x[t+1] = 2 * x[t]$$

Sin embargo, la relación entre el número de bacterias "x" y el instante de observación "t" no es lineal.

$$x[t] = 2^t$$

t	x[t]	x[t+1]
0	1	2
1	2	4
2	4	8
3	8	16
4	16	32
5	32	64
6	64	128

Podemos obtener una ecuación aplicando la otra sucesivas veces. El número de bacterias en una observación t es igual a 2 veces el número de bacterias en la observación anterior t-1, es decir:

```
x[t]=2*x[t-1]
x[t]=2*(2*x[t-2])
x[t]=2*(2*(2*x[t-3]))
x[t]=2*(2*(2*(2*x[t-4])))
...
x[t]=2^t
```

Es decir, el número de bacterias observadas, en función del instante de observación, es no-lineal. En la vida real son muy pocos los sistemas lineales, y la mayoría de las cosas funciona con relaciones no lineales que son las que hacen surgir fenómenos complejos. Como se ve, en la ecuación anterior la población crece hasta el infinito y no contemplamos el caso en el que a las bacterias se les acabe el espacio o el alimento, situación que complicaría la función.

El comportamiento de los sistemas no lineales, que son mayoría en el mundo real, es mucho más interesante que el de los lineales. Son los sistemas no lineales los que generan fenómenos complejos, caóticos, impredecibles. La no linealidad permite representar comportamientos en los que cuando el numero de bacterias es muy grande, se produce una tendencia a disminuir, y cuando sea pequeño a crecer.

Dependiendo del sistema y de la ecuación utilizada para modelarlo, podemos encontrar "atractores". Es decir, podría ser que llegue un momento en donde el numero de bacterias permanezca constante siendo esto un "punto" atractor. Por ejemplo, podemos tomar cualquier calculadora, introducir un número cualquiera y sacarle raíz cuadrada, una vez hecho esto, sacar la raíz cuadrada del resultado, y así repetidas veces. Observaremos que a la larga siempre terminaremos en el valor 1 y de ahí no se moverá. Eso es un atractor.

Podríamos encontrar que el atractor es un ciclo en donde después de un tiempo el número de bacterias oscila, por ejemplo, de la siguiente manera: 1500, 1525, 1513, 1500, 1525, 1513, 1500, 1525, 1513, etc. Sin embargo, podríamos tener un "atractor extraño", donde los números parezcan aleatorios a pesar de estar gobernados por reglas completamente deterministas.

Las características de una función lineal son las siguientes:

```
[1] f(ax) = a f(x) (donde a es constante).
[2] f(x + y) = f(x) + f(y).
```


Esto quiere decir que no linealidad sería aquello que no cumple con esto y aquí es donde radica el problema para definirla. Stanislaw M. Ulam decía que llamar a una ciencia "no lineal" seria como llamar a la zoología "el estudio de los animales no humanos".

Los fractales dibujados en el plano complejo son figuras complicadas formadas por una ecuación no lineal.

Los sistemas complejos se conciben como compuestos por elementos sencillos que interactuan entre si mediante relaciones no lineales.

En la ecuación f(x)=2x sabremos como se comportará el sistema por mas que hagamos crecer x. Pero si el sistema fuera no lineal, a la larga podría sorprendernos con un abrupto cambio de comportamiento. El movimiento de los planetas se rige por ecuaciones diferenciales no lineales. Al igual que en una ecuación algebraica, las ecuaciones diferenciales tienen una incógnita, pero mientras la solución de una ecuación algebraica es un número, la solución en una ecuación diferencial es una función. Estas funciones solución dependen del tiempo (el tiempo t es la variable independiente). Si encontramos la solución de una ecuación diferencial habremos dilucidado el comportamiento del sistema para cualquier periodo pasado o futuro, pues bastaría con dar un valor a t y sabríamos el valor de f(t).

En muchos textos se describen comportamientos lineales y exponenciales de forma que parece que los conceptos de lineal y exponencial son opuestos. No es exactamente así. Exponencial es un tipo de comportamiento no-lineal.

Locura:

"Todos nos volvemos cada dia un poco más locos, y cada uno de una locura diferente. Ésa es la razón por la que nos comprendemos tan mal los unos a los otros. Yo mismo me siento atrapado por la paranoia y la esquizofrenia. Además, soy hipersensible, cosa que deforma mi visión de la realidad. Lo sé. Ahora intento, en vez de sufrirla, utilizar esa locura como motor para todo lo que emprendo. Pero cuanto más triunfo, más loco me vuelvo. Y cuanto más loco me vuelvo, mejor alcanzo los objetivos que me fijo. La locura es un león furioso escondido en cada cráneo. Sobre todo, no hay que matarlo. Basta con identificarlo y domarlo. Vuestro león domesticado os conducirá entonces mucho más allá que cualquier maestro, que cualquier escuela, droga o religión. Pero, como ocurre con toda fuente de poder, hay un riesgo si uno juega demasiado con su propia locura: a veces el león, sobreexcitado, se vuelve contra quien quería domarle."

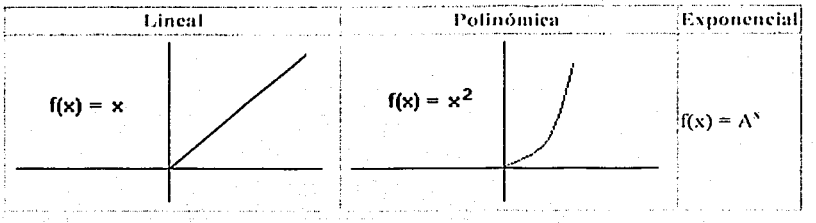
NP-Completo:

En un problema NP-Completo el espacio de estados del problema crece de forma exponencial (o asimilable a exponencial, como por ejemplo, factorial) ante incrementos lineales del número de elementos que intervienen en el problema.

El problema NP-Completo mas representativo es el "Problema del Viajante" que debe recorrer varias ciudades hasta volver al punto de partida incurriendo en el mínimo coste.

Cuando se habla de la complejidad de los problemas (o sistemas) no lineales, muchas veces ocurre que no se está hablando del problema en sí (un ejemplar de problema) sino que cuando se habla de que un problema es no lineal, a veces se está diciendo que se trata de un sistema de comportamiento no lineal, lo que en muchos casos significa una

dificultad para la predicción y la extrapolación, o lo que es lo mismo, dificultad para aprendizaje a partir de la experiencia. Y otras veces se está hablando de la forma en la que el espacio de estados de ese problema aumenta al aumentar linealmente el número de elementos que intervienen; precisamente, que incrementos lineales del numero de estados provocan incrementos exponenciales (no lineales) del numero de posibles soluciones a evaluar.



Para resolver un problema que ante incrementos lineales del numero de variables que maneja (ciudades) produce un incremento exponencial del espacio de estados (recorridos), y por tanto de las necesidades de cálculo, es necesario reducir también de forma exponencial el espacio de estados a analizar, por ejemplo utilizando algoritmos genéticos (AG). Los AG consiguen esto gracias a que cada cadena explora simultáneamente varias zonas del espacio de estados: el peso (fitness) de una cadena, en la práctica, no solo habla de esa cadena en concreto, sino también de las que pueden surgir a partir de ella, mediante combinación o mutaciones.

Ontogenia:

Curso vital de un individuo, desde su concepción hasta la madurez

Optimización:

Dada una función

```
Y = f(X1,X2,...Xn)
```

que representa el comportamiento del sistema o problema que tratamos de resolver, se trata de encontrar el conjunto de valores de las variables de entrada a la función (X1,X2,...Xn) tales que a la salida de la función (Y) se obtenga el valor deseado, teniendo en cuenta que el conjunto de valores de entrada debe cumplir ciertas restricciones.

Si el valor deseado (llamémosle k) de Y es único, podemos definir una nueva función llamada Error (E) como la diferencia entre el valor real obtenido por la función (Y) y el valor deseado (k), es decir:

```
E = Y - k
E = f(X1,X2,...Xn) - k
```

y en este caso, optimizar consistirá en minimizar la función E de error (obtener su mínimo). Otras veces no se tratará de obtener un valor determinado, sino el máximo valor posible (k = infinito) o el mínimo posible (k = -infinito)

Cualquier problema de optimización se puede interpretar como encontrar el máximo(s) y/o el mínimo(s) de una función. Desde este planteamiento, la optimización puede realizarse por el método (simbólico) de:

- Derivar la función a optimizar
- Igualar a cero la función derivada
- Extraer máximos, mínimos y puntos de inflexión
- Realizar la segunda derivada de la función
- Determinar que puntos son máximo, mínimo o punto de inflexión
- Aplicar las restricciones a las entradas para seleccionar sólo soluciones válidas

Sin embargo este método no puede ser empleado en muchísimas ocasiones, por diversas razones: por ejemplo, la función tiene que ser derivable dos veces, y se requiere la resolución de una ecuación. Esta es la razón de la existencia de multitud de métodos de optimización, cada uno adecuado a un cierto tipo de problemas.

Los métodos de optimización se pueden clasificar:

- Según la naturaleza de las soluciones:
  - numéricos: se tienen en cuenta los valores de los parámetros pero no su orden
  - combinatorios: se tienen en cuenta los valores de los parámetros y también su orden
  - de permutaciones: no se tienen en cuenta los valores de los parámetros pero sí su orden
- Según el grado de aleatoriedad en el proceso de búsqueda:
  - deterministicos: no intervienen procesos aleatorios ni pseudoaleatorios. Siempre se obtienen los mismos resultados ante las mismas entradas.
  - aleatorios o estocásticos: intervienen procesos aleatorios y/o pseudoaleatorios. Es posible obtener distintos resultados ante las mismas entradas.
- Según el grado de penetración (razón entre exploración y explotación).
  - En profundidad, explotadores, egoístas, elitistas, conservadores. Producen una convergencia muy rápida hacia una solución.
  - En anchura, exploradores, altruistas, innovadores. Producen una exploración muy a fondo del espacio de búsqueda.

  - Equilibrada. cooperación, compromiso. Es la combinación óptima de los dos anteriores, de forma que se obtiene una solución suficientemente buena (característica de una gran exploración) en un tiempo suficientemente corto (característico de una gran explotación).
- Según el número de candidatos existentes durante la búsqueda:
  - Simples: en todo momento existe una única solución que es refinada sucesivamente.
  - Múltiples: en todo momento existen varias soluciones.
- Según el tipo de búsqueda:
  - Búsqueda a ciegas: se parte de un absoluto desconocimiento de la utilidad de los distintos puntos del espacio del problema. Tan solo se conocen los límites del dominio del problema, es decir, el número de dimensiones y el número de valores o rangos de valores por cada dimensión. En algunos casos podría desconocerse la naturaleza (tipo de dato) y/o desconocerse los límites de alguna o todas las dimensiones, o incluso desconocerse la lista de las dimensiones relevantes (ya que podría haber algunas indiferentes) de todo el conjunto de dimensiones posibles.

Búsqueda heurística: existe algún tipo de conocimiento previo que permite clasificar algunas zonas del espacio de búsqueda como más prometedoras.

Paradigma:

Ejemplo, método, técnica, estrategia.

Pizarra:

Estructura global donde se almacenan datos manejados por un conjunto de agentes, de forma que se facilita la cooperación entre agentes a pesar de su distinta naturaleza.

Problema:

Un problema es un deseo de cambio. Un problema requiere de:

- una entidad capaz de sentir desequilibrios, capaz de desear lo que no es realidad.
- una realidad capaz de ser diferente a lo que la entidad desea.

Un problema es un desequilibrio entre la realidad y lo deseado. Cuando lo deseado no sólo es algo implícito al estado de la entidad (por ejemplo, la entidad tiene hambre, o la entidad necesita más luz del sol), sino explícito, es decir, está representado en algún lugar, está almacenado (por ejemplo, en un papel, en un ordenador, en el cerebro de la entidad que tiene hambre), tenemos otra "realidad" donde aplicar nuevamente el concepto de problema.

El problema se soluciona, o el desequilibrio desaparece cuando:

- la entidad consigue modificar la realidad según su deseo
- la entidad deja de desear ese cambio de realidad
- la entidad desaparece, muere (es un caso particular del anterior)

Los seres vivos pueden verse como "solucionadores de problemas" o "máquinas que desean cosas", aunque entonces sería difícil diferenciar cualquier interacción con el entorno con la acción de solucionar un problema, y entraría en la clasificación de ser vivo, por ejemplo, el fuego.

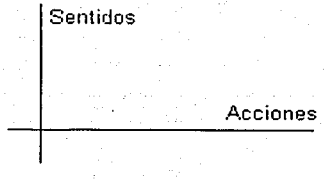
Tipos de Problemas:

- Cambiante: requiere aprendizaje
- Estático: permite planificación

Cualquier problema se puede enfocar como un problema de "búsqueda" de "optimización" o de "predicción". Un problema es predecir la salida ante la entrada, o buscar la entrada que produce cierta salida, o la entrada que produce el máximo valor de la salida.

Vamos a ver los casos más simples de resolución de problemas.

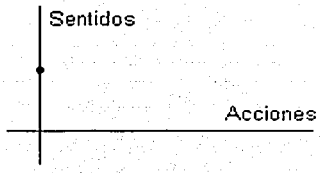
Desde el punto de vista del individuo que "tiene el problema" las entradas son sus acciones y los sentidos sus salidas. El ser se siente "bien" o "mal", tiene frío o no, y por otra parte puede realizar acciones. Expresado en una gráfica:



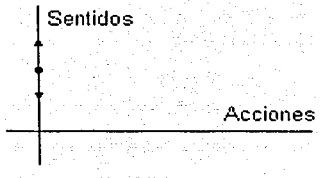
Esta gráfica no tiene en realidad dos ejes, sino muchísimos mas, pero como eso no es muy fácil de dibujar, vamos tener que imaginarlos viendo solo hay una entrada y una salida.

Casi siempre se considera el tiempo como una variable no modificable, que se incrementa constantemente. Vamos a pensar que es una mas de las acciones posibles, es decir, que podemos trasladarnos de un punto a otro del tiempo cuando lo deseemos, y sin embargo, optamos por incrementar su valor constantemente.

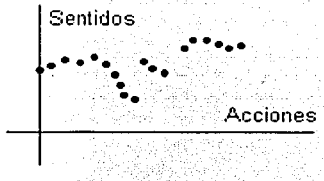
En un momento dado el ser siente algo, se encuentra bien o mal.



Las sensaciones del ser pueden modificarse, aunque no haga nada (por ejemplo, cada vez hace mas frío)

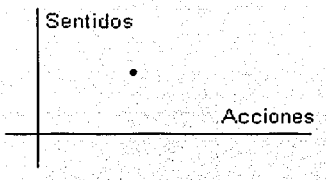


Suponiendo que el eje x es ahora el tiempo, la sensación se va modificando en el tiempo. Recordemos que hemos supuesto que el tiempo es modificable pero en principio lo mantenemos con incrementos constantes, y por tanto el tiempo es una más de las acciones posibles y por eso se representa en el eje de acciones.

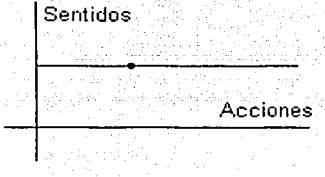


La salida puede variar aunque no realicemos acciones: esto representa un problema de predicción. Saber cual será el próximo punto de la salida.

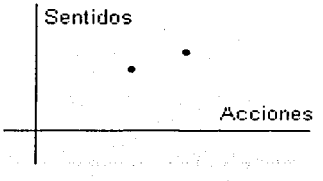
Hasta ahora el individuo no ha hecho nada. Ahora, por fin, realiza una acción (ignoramos el tiempo); después de realizarla, observa sus efectos; ante ella obtendrá una salida: tenemos un punto.



Ahora el razonamiento consiste en convertir ese punto en una recta horizontal: si una vez hice algo y lo vuelvo a hacer, es más probable que vuelva a suceder lo mismo, por un principio de inercia.



Lo hacemos pero no, no ha ocurrido exactamente lo mismo, sí algo parecido. Ahora tenemos otro punto. Entonces el razonamiento los une con una recta. Continuando este proceso, tendremos la nube de puntos que define la realidad.



Programación Evolutiva-Evolutionary Programming (PE-EP).

Este paradigma fue creado por David Fogel. En la Programación Evolutiva los genes codifican un autómata finito o máquina de estados. Cada gen puede consistir en una terna del tipo "estado origen + cadena reconocida + estado destino". En definitiva, cada gen es una regla que especifica las condiciones (cadena reconocida) que se deben cumplir para llegar a un estado (estado destino) suponiendo que la máquina se encuentra en un cierto estado (estado origen).

```
ESTADO + SUCESO => NUEVO ESTADO
```

Esta regla tiene una forma imperativa. Podríamos resumirla en la forma *"Si estoy realizando tal acción, y veo tal información, debo obligatoriamente cambiar a realizar tal acción"*. Si el autómata se encuentra en determinado estado, y ocurre cierto suceso, el autómata pasa a un nuevo estado. La "cadena reconocida" corresponde con los sentidos o sensores del autómata (el suceso), y los estados corresponden con las acciones o actuadores. Este planteamiento tiene grandes similitudes con una red neuronal, y es el más sencillo de programar, pero hace difícil encontrar la explicación a porqué el autómata realiza cierta acción.

Podemos interpretar la misma regla como declarativa, modificando el significado de cada elemento de la regla.

```
ESTADO + ACCIÓN => NUEVO ESTADO
```

Esta regla podríamos resumirla en la forma *"Si veo tal información, y realizo tal acción, estimo que como consecuencia de mi acción, la próxima información será tal"*. En este caso, los estados corresponden con la cadena reconocida y estamos suponiendo que las acciones del autómata pueden modificar los futuros valores de la cadena reconocida. Este planteamiento tiene grandes similitudes con un sistema experto. Requiere fijar un objetivo al autómata que consistirá en el estado ideal que el autómata debe observar, y permite explicar porqué el autómata realiza una acción.

Programación Genética-Genetic Programming (PG-GP).

Así como en los Algoritmos Genéticos los elementos de la cadena (genes) son un tipo de datos que puede ser bits, strings o integer, en Programación Genética los elementos de la cadena (genes) son instrucciones en un lenguaje de programación. La inducción de programas mediante programación genética está muy identificada con los trabajos de John Koza, que hacía evolucionar expresiones-S en un programa LISP. Koza fué discípulo de Holland, al igual que Goldberg, y patentó varios de sus algoritmos. Entre otras cosas, los últimos trabajos de Koza ofrecen pruebas de que la programación genética puede implementarse en lenguajes procedurales como C y C++, rompiendo así el mito de que sólo un lenguaje de las características propias de LISP eran capaces.

Programas de Evolución:

Los Programas de Evolución son un refinamiento de los Algoritmos Genéticos, propuesto por Michalewicz en 1994. En un programa de evolución, la codificación binaria de ceros y unos empleada por los Algoritmos Genéticos se amplía hasta admitir cualquier tipo de alfabeto. El hecho de utilizar ceros y unos evitaba ciertos problemas que los Programas de Evolución intentan resolver explícitamente, como por ejemplo, cómo se efectúa una mutación. De esta forma se consigue una codificación más cercana al problema a resolver.

Por lo general, los Programas de Evolución se han englobado dentro de los Algoritmos Genéticos (como una variante), y al revés (los Algoritmos Genéticos como un caso particular). En definitiva, actualmente se utilizan los términos de Algoritmo Genético y Programa de Evolución indistintamente para designar algoritmos evolutivos con cualquier tipo de alfabeto.

Cuando el alfabeto utilizado en un Programa de Evolución no es discreto sino continuo, es decir, cuando los genes son números reales con la máxima precisión (decimales) posible, los Programas de Evolución se confunden con las Estrategias Evolutivas.

Hay que puntualizar que en realidad toda la información que maneja un computador digital es discreta, pero en casos como éste, el nivel de detalle es tan grande que se puede considerar continuo. En las Estrategias Evolutivas el número de valores posibles de un gen es extremadamente grande, existiendo por lo general muy pequeñas diferencias entre los efectos de dos valores de gen consecutivos.

Pseudoaleatorio:

con causa, pero aparentemente impredecible, o impredecible en la práctica debido a la propagación de errores iniciales propia de un sistema caótico, pero predecible en forma de probabilidad

Proceso pseudoaleatorio: proceso cuyo resultado es aparentemente impredecible, excepto en forma de probabilidad

Cadena pseudoaleatoria: cadena para la que aparentemente no existe ninguna forma de comprimir su descripción

Redes Neuronales Artificiales-Artificial Neural Networks (RNA-ANN).

Son sistemas que intentan simular el cerebro a nivel de neurona.

Básicamente una RNA es una red de elementos muy sencillos que trabajan en paralelo, existiendo una interconexión masiva entre ellos y diferentes grados de "fuerza" de conexión entre unos y otros. La salida de la red se puede calcular según una fórmula conocida, determinada por el método elegido y extremadamente larga, pero relativamente fácil de implementar por programa.

Las Redes Neuronales Artificiales (ANN: Artificial Neural Networks) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, formados por un conjunto de unidades llamadas "neuronas" o "nodos" conectadas unas con otras.

Estas conexiones tienen una gran semejanza con las dendritas y los axones en los sistemas nerviosos biológicos.

El Primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de "actividad nerviosa". (Nota: Un artículo de Investigación y Ciencia -Jun 99- señala a Turing como el precursor). El modelo de McCulloch-Pitts es un modelo binario, y cada neurona tiene un escalón o umbral prefijado. Este primer modelo sirvió de ejemplo para los modelos posteriores de Jhon von Neumann, Marvin Minsky, Frank Rosenblatt, y muchos otros.

Una primera clasificación de los modelos de RNA podría ser, atendiendo a su similitud con la realidad biológica:

- Los modelos de tipo biológico. Este comprende las redes que tratan de simular los sistemas neuronales biológicos así como las funciones auditivas o algunas funciones básicas de la visión.
- El modelo dirigido a aplicación. Estos modelos no tienen porque guardar similitud con los sistemas biológicos. Sus arquitecturas están fuertemente ligadas a las necesidades de las aplicaciones para las que son diseñados.

Reduccionismo:

Un todo puede ser comprendido completamente si se entienden sus partes, y la naturaleza de su suma

Robot

Estas definiciones han sido tomadas de Isaac Asimov en :

(http://www.ciudadfutura.com/asimov/).

Un autómata es un ser totalmente inorgánico, metal y plástico, que podía realizar unas actividades muy limitadas, adivinar el provenir, o simplemente dar la mano a los

visitantes, o ajustar la pieza de un coche, realiza actividades repetitivas, y carece de inteligencia.

Un robot es una máquina móvil, mediante rodamientos o extremidades flexibles, controlada por una inteligencia artificial, puede pensar, decidir y aprende de su experiencia como por ejemplo R2-D2 de la película *La Guerra de las Galaxias*. Los robots positrónicos de la U.S. Robots, son robots o androides, mas habitualmente, con la diferencia de disponer de cerebros positrónicos y llevar incorporadas las tres leyes de la robótica.

Un androide es un robot construido para parecer humano, dispone de movilidad e inteligencia artificial, por ejemplo Sean Young en *Blade Runner*.

Un ciborg es un humano parcialmente artificial o un robot parcialmente humano por ejemplo *Robocop*. Pese a ello no tiene por que tener forma humana se podría considerar un ciborg a una nave espacial que tuviera un cerebro humano implantado y que la gobernara.

Un clon es una mutación genética, donde se manipula el desarrollo del embrión y el feto fuera del cuerpo de la madre, para hacer mutaciones que mejoren la raza humana o enriquecer el bolsillo de algunas multinacionales.

Un organismo biomecánico con apariencia humana o no se consigue mediante el empleo de la ingeniería genética y la nanotecnologia para crear híbridos de maquinas con funciones biológicas, esto puede ser por ejemplo, un vehículo exploratorio para hacer estudios de "terraforming" en un planeta deshabitado, armas de ataque inteligentes, naves estelares biomecánicas.

Un mech, es una máquina que en su interior tiene a un humano que lo gobierna y dirige, un exoesqueleto que protege e incrementa la fuerza del ocupante. Puede realizar diversos trabajos, como defensa territorial, almacenaje de contenedores, o exploración.

Las tres leyes de la robótica:

1.- Un robot no puede dañar a un ser humano, o por inacción, permitir que un ser humano resulte dañado.

2.- Un robot debe obedecer las ordenes dadas por los seres humanos, excepto cuando tales ordenes entren en conflicto con la primera ley.

3.- Un robot debe proteger su propia existencia, hasta donde esta protección no entre en conflicto con la primera o la segunda ley.

(Enfoque) Simbólico de la IA o IA Clásica:

Los simbólicos simulan directamente las características inteligentes que se pretenden conseguir. Como modelo de mecanismo inteligente a imitar, lo mejor que tenemos y más a mano es el Hombre. Desde este punto de vista, poco interesa simular los razonamientos que puedan efectuar los animales. El boom de los Sistemas Expertos, ahora de capa caída, fue producido por este planteamiento. Para los constructores de sistemas expertos, es fundamental la representación del conocimiento humano y debemos a ellos los grandes avances en este campo. Esto es debido a que, realizando una gran simplificación, se deben incluir en un sistema experto dos tipos de conocimiento: *"conocimiento acerca del problema particular"* y *"conocimiento acerca de cómo obtener más conocimiento"*. Para el primero existen técnicas como los Frames (marcos) que fueron los padres de lo que hoy conocemos como *Programación Orientada a Objetos*. El segundo es llamado también mecanismo de inferencia y requiere además de un método de búsqueda que permita seleccionar la regla a aplicar del conjunto total de posibles reglas. Esto puede parecer lo más sencillo, pero habitualmente es lo más difícil: se trata de no demorarse varios millones de años en elegir.

Como ejemplo representativo de la rama simbólica llevada al extremo tenemos el proyecto Cyc de Douglas B. Lenat, con un sistema que posee en su memoria millones de hechos interconectados. Según Lenat, la inteligencia depende del número de reglas que posee el sistema, y *"casi toda la potencia de las arquitecturas inteligentes integradas provendrá del contenido, no de la arquitectura"*. Para él, los investigadores que esperan poder resolver con una única y elegante teoría todos los problemas de inferencia y representación de conocimientos, padecen celos de la física: ansían una teoría que sea pequeña, elegante, potente y correcta.

Otro de los argumentos en la defensa de esta línea de trabajo se resume en la frase: *"El experto no piensa, el experto no aprende. El experto sabe."*

Otra de las áreas de interés es el Procesamiento del Lenguaje natural.

Simulación:

Simular es experimentar mediante un modelo de un sistema real. Así se evita construirlo, perturbarlo o destruirlo.

La principal ventaja de la simulación es que ofrece un análisis dinámico en paralelo de todos los elementos que intervienen en el sistema, cosa de la que en principio no es capaz el razonamiento consciente secuencial del ser humano.

Aplicaciones de la simulación:

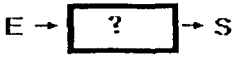
- Diseño de plantas e instalaciones
- Manufactura flexible
- Análisis de la capacidad
- Análisis de cuellos de botella
- Planificación de uso de recursos
- Optimización de flujo de materiales

- Control de inventarios
- Planificación de órdenes de trabajo
- Optimización de sistemas de mantenimiento

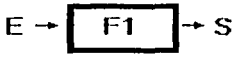
Las principales ventajas de la simulación son:

- El diseño del modelo ofrece una visión simplificada de los procesos productivos
- Permite la experimentación
- Ayuda a la toma de decisiones

Un tipo de problema informático clásico es crear un traductor (autómata o compilador), es decir, un programa capaz de transformar unas entradas en unas salidas.

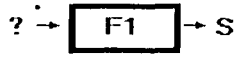


Una vez que hayamos terminado el trabajo, tendremos un programa o función F1 capaz de convertir unas entradas en unas salidas.



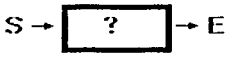
Si hablamos de hacer simulaciones, esta función F1 representará un sistema del mundo real. Las entradas serán las posibles acciones sobre ese sistema, y las salidas representarán la reacción del sistema ante esas entradas. Con el programa F1 podríamos predecir el comportamiento del sistema ante cada posible entrada.

Llegados a este punto, puede ser muy interesante conocer cuál debe ser la entrada necesaria para obtener cierta salida.

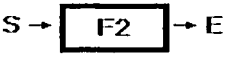


Esto se puede ver como un problema de optimización (optimizar el valor de la salida) o de búsqueda (encontrar la mejor entrada)

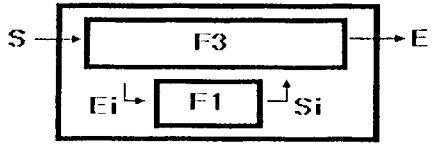
En definitiva, lo que queremos encontrar es el programa o función F2 capaz de decirnos cuál es la entrada que produce cierta salida deseada.



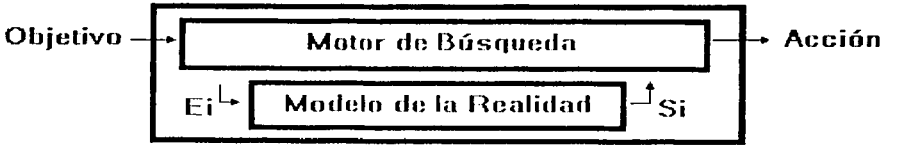
Cuando obtengamos F2, podremos fijar un objetivo S y calcular la acción E necesaria para conseguirlo.



Ahora bien, F2 deberá tener en cuenta a F1. F2 deberá disponer de un mecanismo interno (F3) capaz de realizar experimentos con F1 hasta encontrar una solución aceptable.



En definitiva, tenemos un sistema capaz de aconsejar la mejor acción para conseguir un objetivo, basado en un subsistema de búsqueda (u optimización) aplicado a un modelo de la realidad.



Algunos lenguajes de simulación son:

Discretos			Continuos
Redes	**Eventos**	**Actividades**	
SimnetII	Witness	SIMSCRIPT	Stella
GPSS	SIMAN		Dynamo
SLAMII	IDSS		CSHP
Q-GERTS	See Why		SLAM
GEMS	Auto-Simulations		Vensim
	Manman		I Think
	AIM		Simula

Simulated Annealing (solidificación simulada, agregación simulada, recocido simulado).

Su origen está en la Física estadística. Los procedimientos físicos de solidificación controlada consisten en calentar un sólido hasta que se funde, y seguidamente, ir enfriándolo de forma que cristalice en una estructura perfecta, sin malformaciones locales.

En la agregación simulada se parte de una solución válida a la que se le provocan pequeñas mutaciones aleatorias. A diferencia del método de Montecarlo, se puede aceptar una solución peor que la anterior. A medida que avanza el algoritmo, éste cada vez será más exigente en cuanto a las soluciones aceptadas. En al analogía con la solidificación física, se trata de tener un cuerpo (problema) a alta temperatura (admitiendo soluciones muy diversas a pesar de ofrecer malos resultados) e ir disminuyendo la temperatura (aumentando la exigencia del algoritmo) de forma que termine por solidificarse según la forma deseada (ofreciendo el mejor resultado).

Sistema: Conjunto de elementos y las interacciones entre los mismos.

"Un sistema es un conjunto de elementos relacionados entre sí, actuando en un determinado entorno (e interaccionando con él), con el fin de alcanzar un objetivo común y con la capacidad de autocontrol."

"Un sistema es un conjunto dado de elementos, sus comportamientos permanentes, y un conjunto de acoplamientos entre los elementos y entre los elementos y el entorno."

Sistema Experto: Sistema Basado en el Conocimiento (SBC) artificial que posee un comportamiento que se puede considerar equivalente al de un Experto en un área del conocimiento determinada.

Solipsismo:

Planteamiento filosófico que admite la existencia de uno mismo, sus sentimientos y creencias, pero reconoce la falta de seguridad en la existencia de todo lo demás: el resto de individuos, objetos y en definitiva, de todo el universo excepto uno mismo.

La argumentación es la siguiente: ya que toda la información que uno posee proviene de los sentidos y los sentidos son propensos a errores y engaños, cualquier información proporcionada por los sentidos no es fiable...

En cambio, el simple hecho de recibir información por los sentidos implica la seguridad en la certeza de la existencia de un receptor, que es uno mismo, y de algún tipo de sensores (que definen al receptor), pero no implica la existencia de un emisor distinto a uno mismo.

Curiosamente, la información objetiva relativa a uno mismo (soy alto, mi cuerpo está frío, mi cuerpo está cansado, mi estómago está dañado) también se recibe por los sentidos y también es propensa a errores, por lo que la información objetiva que representan los sentidos tampoco es fiable, aunque sí lo es la información desde el punto de vista subjetivo (creo que soy alto, siento frío en el cuerpo, estoy cansado, me duele el estómago).

Es decir, el hecho de recibir información por los sentidos, además de implicar la seguridad en la certeza de la existencia de uno mismo como receptor, también implica la certeza del significado subjetivo de los estímulos. No puedo estar seguro de ser alto, o de tener frío en el cuerpo, pero lo que es seguro es que yo creo que soy alto y que yo siento que tengo frío. Este último aspecto se hace patente en los sueños, donde se reciben estímulos provocados por uno mismo. En un sueño es posible creer que uno es alto cuando no lo es, o sentir frío cuando el cuerpo no posee una temperatura baja.

Sin duda, la vida puede ser un sueño, ya que no hay forma de distinguir entre sueño y realidad.

La falsedad del solipsismo es indemostrable, como es indemostrable la falsedad de cualquier axioma.

Sin embargo, el solipsismo es improbable (es decir, la falsedad del solipsismo es probable). El sentido común basado en el principio de la analogía entre los sucesos, nos dice que es más probable que las cosas sean lo que parecen, frente a que no sean lo que parecen, o ni siquiera existan.

(Enfoque) Subsimbólico de la inteligencia artificial:

Los esfuerzos de los subsimbólicos, se orientan a simular los elementos de más bajo nivel que componen o intervienen en los procesos inteligentes, con la esperanza en que de su combinación emerja de forma espontánea el comportamiento inteligente. Los ejemplos más significativos probablemente sean las Redes Neuronales Artificiales (RNA) y la Computación Evolutiva. Ambos, aunque parezcan fenómenos recientes, no son más jóvenes que los Sistemas Expertos (SE) de la inteligencia artificial clásica. Una posible causa de su menor publicidad y financiación es la necesidad de mayores cantidades de recursos informáticos en comparación con los SE. El Perceptrón de Rosenblat, antecedente de las RNA, apareció en 1959, y los algoritmos genéticos fueron desarrollados por Holland a finales de los años 60 bajo el nombre de "planes reproductivos".

Las grandes ventajas de los sistemas subsimbólicos son la autonomía, el aprendizaje y la adaptación, conceptos todos ellos relacionados.

Vida:

El "ser" puede ser, además, "ser vivo" (*ser un ser vivo*). El calificativo de *vivo* siempre se aplica (cuando no es metafóricamente, como en "el rock está vivo") a un ser material (una piedra no está viva, una rana sí). Una posible aproximación a la definición es basarla en una serie de procesos: nacen, crecen, intercambian energía, se mueven, tal vez se reproducen, suponemos que siempre mueren. Pero esos comportamientos los tienen otras muchas entidades que no consideramos vivas. ¿O deberíamos considerarlas?

Hacia abajo: ¿Están vivas las células de la piel de la rana?

Hacia arriba: ¿Está vivo nuestro equipo de fútbol?

¿Está vivo un idioma? Los idiomas, y algunos programas de ordenador nacen, crecen, se reproducen y mueren. Sin embargo, hay algo que indica al criterio popular que la rana sí está realmente viva y el programa de ordenador no ¿qué será? (Al final, se ofrece una posible respuesta.)

¿Un ser vivo debe ser consciente, al menos en un pequeño grado? Tal vez el concepto de ser vivo sea algo borroso, gradual equivalente a la conciencia. De igual forma que una nota "emerge" de una serie de fluctuaciones idénticas, y una melodía "emerge" de una serie de notas, cada una con su propia frecuencia, tal vez la conciencia emerja de los procesos vitales. Y tal vez no.

"La vida es algo que no puede expresarse con el color y la forma; su concepto implica abstracción unitaria de múltiples armonías, medidas, proporciones, movimientos, ritmos y luces, que se dan en el complejo psíquico y moral, orgánico y sensitivo, elemental y

múltiple, representativo y difuso, de condición microcósmica y egocéntrica, manifiestamente sometido a constante inquietud, evolución, transmutaciones y transubstanciaciones.".

"La vida consiste básicamente en trabajar y dormir. ¿Sólo eso? Bueno, de vez en cuando te puedes tomar una cerveza con los amigos" (Olaf el vikingo).

"La vida en sí no es nada, es sólo la oportunidad para algo."

"La vida es una enfermedad mortal de trasmisión sexual" (?)

Ser vivo sensible: *"Aquella entidad que posee un mecanismo de asignación de recompensa y/o castigo tal que le produce sensaciones de placer y/o dolor".*

Vida Artificial:

Intento por parte del Hombre, de crear vida, o algo parecido a la vida, mediante la combinación de símbolos (datos) y procesos de símbolos (programas) independientemente del soporte físico de estos símbolos y procesos.

BIBLIOGRAFÍA

[1] **Alander, J. T. (1992) :** On optimal population size of genetic algorithms. *Proc. CompEuro 92*, IEEE Computer Society Press. 65-70.

[2] **Antonoisse, J. (1989).** A new interpretation of schema notation that overturns the binary encoding constraint. *Proc. of 3er Intl. Conference on Genetic algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos, 70-79.

[3] **Back, T. (1992).** The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. R. Manner y B. Manderick (Eds.), *Parallel problem solving from nature, 2*. Amsterdan: Elsevier, 85-94

[4] **Back, T. (1993).** Optimal mutation rates in genetic search. *Proc. 5th Intl. Conf. on Genetic Algorithms and their Applications*. Morgan Kaufmann, San Mateo, 2-8.

[5] **Back, T. (1995).** Generalized convergence models for tournament - and (μ,λ) - selection. *Proc. 6th Intl. Conf. on Genetic Algorithms and their Applications*, 2-8.

[6] **Back, T., Hoffmeister, F., & Schwefel H. (1991).** A survey of evolution strategies. *Proc. 4th Intl. Conf. on Genetic Algorithms and their Applications*. Morgan Kaufmann. La Jolla, CA, 2-9.

[7] **Baker, J. E. (1985).** Adaptive selection methods for genetic algorithms. *Proc. of an International Conference on Genetic Algorithms and their applications*. J.J. Grefenstette (Ed). Lawrence Erlbaum Associates, Hillsdale, NJ.

[8] **Bernadó, E., Llorá, X., & Garrell, J. M. (2002).** XCS and GALE: A comparative of two learning classifier systems and six other learning on classification tasks. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), Advances in Learning classifier systems: 4th International workshop, IWLCS 2001 (pp. 113-131). Berlin Heildelberg: Springer-Verlag.

[9] **Booker, L.B. (1982).** Intelligent Behavior as an Adaptation to the Task Environment. Doctoral Dissertation. Department of Computer and Communication Science, University of Michigan, Ann Arbor.

[10] **Booker, L.B. (1985).** Improving the performance of genetic algorithms in classifier systems. In J. J. Grefenstette, editor, Proceedings of an International Conference on Genetic Algorithms and Their Applications, pages 80-92. Lawrence Erlbaum Assoc., Hillsdale, New Jersey.

[11] **Booker, L.B. (1987).** Improving search in genetic algorithms. *Genetic algorithms and Simulated Annealing*. L. Davis (Ed.). Morgan Kauffmann, Los Altos,CA, 61-73.

[12] **Booker, L.B. (1990).** Representing Attribute-Based Concepts in a Classifier System. Proceeding of the First Workshop on Foundations of Genetic Algorithms, Morgan Kaufmann, , 115-127.

[13] **Booker, L.B. (1992).** Viewing Classifier Systems as an Integrated Architecture. Paper presented at The First Conference on Learning Classifier Systems, Houston, Texas.

[14] **Booker, L.B., Goldberg, D.E., Holland, J.H, (1989).** Classifier Systems and Genetic Algorithms. Artificial Intelligence, 40, 235-282.

[15] **Bridges, C.L., Goldberg, D. E. (1987).** An analysis of reproduction and crossover in a binary-coded genetic algorithm. *Proc. Second Intl. Conf. On Genetic Algorithms and their Applications*. 9-13.

[16] **Bull, L. (2001).** Simple markov models of the genetic algorithm in classifier systems: Accuracy-based fitness. In Lanzi, P.L., Stolzmann, W., & Wilson, S. W. (Eds.), Advances in Learning Classifier Systems: Proceeding of the Third International Workshop, LNAI 1996 (pp. 21-28). Berlin Heildelberg: springer-Verlag.

[17] **Butz, M. V. (1999).** An Implementation of the classifier system in C (Technical Report 99021). The Illinois Genetic algorithms Laboratory.

[18] **Butz, M. V., & Pelikan, M. (2001).** Analysing the evolutionary pressures in XCS. In Spector, L., Goodman, E. d., Wu, A., Langdom, W. B., Voigt, H. M., Gen, M., Sen, s., Dorigo, M., Pezeshk, S., Garzon, M. H., & Burke, E. (Eds.) , Procedding of the Genetic and Evolutionary Computation Conference (GECCO-2001) (pp. 935-942). San Francisco, CA: Morgan Kaufmann.

[19] **Butz, M. V., & Wilson, S. W. (2001).** An algorithmic description of XCS. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), Advances in Learning Classifier systems: Proceedings of the Third International Workshop, LNAI 1996 (pp. 253-272). Berlin Heildelberg: Springer-Verlag.

[20] **Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2001).** How XCS evolves accurate classifiers. In Spector, L., Goodman, E. D., Wu, A., Langdom, W. B., Voigt, H. M., Gen, M., Sen, S., Dorigo, M., Pezeshk, s., Garzon, M. H., & Burke, E. (Eds.), Proceeding of the Genetic and Evolutionary Computation conference (GECCO-2001) (pp. 927-934). San Francisco, CA: Morgan Kaufmann.

[21] **Cantú-Paz, E. (1995).** A summary of research on parallel genetic algorithms. Technical Report 95007, University of Illinois at Urbana - Champaign, 1995.

[22] **Cantú-Paz, E. (1998).** A Markov Chain analysis of parallel genetic algorithms with

arbitrary topologies and migration rates. ILLIGAL 98010.

[23] **Cantú-Paz, E. (1999).** Designing Efficient and Accurate Parallel Genetic Algorithms. PdD thesis. University of Illinois at Urbana-Champaign.

[24] **Carruana, R.A., & Schaffer, J.D. (1988).** Representation and hidden bias: Gray vs. binary coding for genetic algorithms. *Proc. 5th Intl. Conf. on Machine Learning*. Morgan Kaufmann, Los Altos.CA.

[25] **Carse, B., & Fogarty, T.C., (1994).** A Fuzzy Classifier System Using the Pittsburgh Approach. In Davidor, Y., Schwefel, H.P., Manner (Eds.), Parallel Problem Solving from Nature - PPSN III, Springer-Verlag, 1994, 260-269.

[26] **Celikis, G., & Kuehni, R.G. (1983).** Colour Technology in the Textile Industry, US.

[27] **Chapman, D., & Kaelbling, P. (1991).** Input generalization in delayed reinforcement learning: An algorithm and performance Comparisons. *Proceedings of the International Joint Conference on Artificial Intelligence.*

[28] **Coello, Coello, C. (1999).** Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. Computer in Industry.

[29] **Davis, L. (1985).** Applying adaptive algorithms to epistemic domains. Proc. Intl. Joint Conf. On Artificial Intelligence. 162-164.

[30] **Davis, L. (1991).** Handbook of genetic algorithms. New York: Van Nostrand Reinhold.

[31] **Davis, T. E., & Principe, J. C. (1991).** A simulated annealing like convergence theory for the simple genetic algorithm. *Proc. 4th Intl. Conf. on Genetic Algorithms and their Applications*. Morgan Kaufmann. La Jolla, CA. 174-181.

[32] **De Jong, K. (1988).** Learning with Genetic Algorithms: An Overview. Machine Learning 3, 121-138.

[33] **De Jong, K., & Spears, W. (1989).** Using genetic algorithms to solve NP-complete problems. *Proc. of 3er Intl. Conference on Genetic algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos. 124-132.

[34] **De Jong, K., & Spears, W. (1990).** An analysis of the interacting roles of population size and crossover in genetic algorithms. *Proc. First Workshop Parallel Problem Solving from Nature*, Springer- Verlag, Berlin. 38-47.

[35] **De Jong, K., & Spears, W. (1992).** A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Maths. And AI*, 5, 1-26.

[36] **De Jong, K., & Spears, W. (1993).** On the state of evolutionary computation. *Proc. 5th Intl. Conf. on Genetic Algorithms and their Applications. Morgan Kaufmann*, San Mateo. CA.

617-623.

[37] **Dean, Givan, Thomas, D., & Robert, G., (1997).** Model minimization in markov decision processes. In *Proceedings of the American Association of Artificial Intelligence (AAAI-97)*. AAAI Press.

[38] **Dorigo M. (1992).** Using transporters to increase speed and flexibility of genetics-based machine learning systems. Micro processing and Microprogramming J. vol. 34 (1992) 147-152.

[39] **Dorigo M., & Schnepf, U., (1993).** Genetics-based machine learning and behavior base robotics: A new synthesis. IEEE Transactions on Systems, Man, and Cybernetics, 23, 1, 141-153.

[40] **Dorigo, M. H., & Bersini, (1994).** A comparison of Q-learning and classifier systems. In From Animals To Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior, pages 248-255. Brighton, UK, MIT Press.

[41] **Dorigo, M. (1991).** Message-based bucket brigade: An algorithm for the appointment of credit problem. In Yves Kodratoff, editor, *Machine Learning. European Workshop on Machine Learning*, LNAI 482, pages 235-244. Springer-Verlag.

[42] **Dorigo, M. (1991).** New Perspectives about Default Hier Formation in Learning Classifier Systems. Technical Report No. 91-002. Dipartimento di Elettronica, Politecnico di Milano, Italy.

[43] **Eshelman, L. J., & Schaffer, J.D. (1993).** Real-coded genetic algorithms and interval-schemata. In L. D. Whitley (Ed.). *Foundations of genetic algorithms, 2* . San Mateo: Morgan Kaufmann. 187-202

[44] **Fernandez, D. B., & Matellán, B. (1999).** VQQL applied to reinforcement learning. In *Proceedings of the Third International Workshop on RoboCup (IJCAI'99)*, Stockholm, Sweden, August 1999.To appears.

[45] **Fogel, D.B., (1993).** On the philosophical difference between evolutionary algorithms and genetic algorithms. In D. B. Fogel and W. Atmar, editors, Proceedings of the 2nd Annual Conference on Evolutionary Programming. Evolutionary Programming Society, La Jolla, CA.

[46] **Forrest, S. (1985).** A Study of Parallelism in the Classifier System and its Application to Classification in KL-ONE semantic network, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

[47] **Forrest, S. (1993).** Genetic Algorithms: Principles of natural selection applied to computation. *Science*. Vol. 261. August. 872-878.

[48] **Forrest, S. (1993).** Proc. of the Fifth Int. Conf. on Genetic Algorithms. University of Illinois at Urbana-Champaign.

[49] **Fox, B. R., & McMahon, M. R. (1991).** Genetic operators for sequencing problems. G.J.E. Rawlins (Ed.) Foundations of genetic algorithms. San Mateo: Morgan Kaufmann.284-300.

[50] **Furuhashi T., Nakaoka K., Morikawa K., & Uchikawa (1993).** Controlling excessive fuzziness in a fuzzy classifier system. Proceedings of the Fifth International Conference on Genetic Algorithms, 635.

[51] **Garay, M.R., & Johnson, D.S., (1979).** Computers and intractability. A guide to the theory of NP- Completeness. W. H. Freeman and Company, New York.

[52] **Giani, F. B., & Starita, A. (1995).** Q-learning and parallelism in evolutionary rule based systems. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms, pages 376-379. Springer-Verlag.

[53] **Glover, F. (1977).** "Heuristics for integer programming using surrogate constraints", *Decision Sciences*, No. 8, pp. 156-166.

[54] **Glover, F. (1990).** "Tabu Search : A Tutorial". *Interfaces*. Vol 20, No. 4, pp. 74-94, July-August.

[55] **Goldberg D. E., Korb, B., & Deb, K. (1989).** Messy genetic algorithms: Motivation, analysis, and first results. *Complex Syst. 3*, 493- 530.

[56] **Goldberg, D. E. (1989).** Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley.

[57] **Goldberg, D. E. (1989).** Sizing populations for serial and parallel genetic algorithms. *Proc. of 3er Intl. Conference on Genetic algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos. CA. 70-79.

[58] **Goldberg, D. E. (1989).** Genetic algorithms and Walsh functions: Part I, a gentle introduction. Complex Systems, 3, 129- 152.

[59] **Goldberg, D. E. (1989).** Genetic algorithms and Walsh functions: Part II, deception and its analysis. Complex Systems, 3, 153- 171.

[60] **Goldberg, D. E. (1991).** Real-coded genetic algorithms, virtual alphabets and blocking. *Complex Systems*, 5, 139-167.

[61] **Goldberg, D. E. (1991).** The theory of virtual alphabets. *Lectures Notes in Computer Science: Parallel Problem Solving from Nature*, 496, 13-22.

[62] **Goldberg, D. E., & Sastry, K. (2001).** A practical schema theorem for genetic algorithm design and tuning. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001), 328-335.

[63] **Goldberg, D. E., Deb, K., & Thierens, D. (1993).** Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, 32(1), 10-16.

[64] **Goldberg, D. E., & Lingle, R. (1985).** Alleles, loci and the TSP. *Proc. of First Intl. Conference on Genetic algorithms*. Lawrence Erlbaum Associates, Hilldale, N.J. 154-159.

[65] **Goldberg, D. E., & Rudnick, M. (1991).** Genetic algorithms and the variance of fitness. *Complex Systems, 5*, 265-278.

[66] **Goldberg, D.E. (1983).** Computer-aided gas pipeline operation using genetic algorithms and rule learning, Ph.D. Dissertation, University Microfilms No. 8402282, University of Michigan, Ann Arbor, MI.

[67] **Grefenstette, J.J. (1986).** Optimization of Control Parameters for Genetic Algorithms. IEEE Transactions on Systems, Man, and Cybernetics 16, 122-128.

[68] **Grefenstette, J.J. (1987).** Genetic Algorithms and Their Applications (Erlbaum, Hillsdale, NJ.).

[69] **Grefenstette, J.J. (1987).** Multilevel credit assignment in a genetic learning system. In J. J. Grefenstette, editor, Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, pages 202-209. Lawrence Erlbaum Assoc., Hillsdale, New Jersey.

[70] **Grefenstette, J.J. (1988).** Credit Assignment in Rule Discovery Systems based on Genetic Algorithms. Machine Learning 3, 225-245.

[71] **Grefenstette, J.J. (1994),**Genetic Algorithms for Machine Learning. Kluwer Academic, Boston.

[72] **Holland J.H. (1975).** Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press, (The MIT Press, London, 1992).

[73] **Holland, J. H. (1976).** Application in artificial systems with adaptation natural. Ann Arbor, MI: University of Michigan Press, second edition 1992.

[74] **Holland, J. H. (1977).** Processing and processors for schemata. In E. L. Jacks, editor, Associative Information processing, pages 127-146. American Elsevier, New York.

[75] **Holland, J.H. (1985).** Properties of the Bucket Brigade Algorithm. Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Pittsburgh, PA: Erlbaum , 1-7.

[76] **Holland, J.H. (1986).** Escaping brittleness: The possibilities of general-purpose learning algorithm applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. Machine learning: An artificial intelligence approach, volume 2. Morgan Kaufmann, Los Angeles CA.

[77] **Holland, J.H., & Reitman, J.S. (1978).** Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, Pattern-directed inference systems. Academic Press, New York.

[78] **Holmes, (1996).** Sentinel Features in Epidemiologic Surveilance, Ph.D. thesis, Drexel University.

[79] **Homaifar, A., Guan, Sh., & Liepins, G. E. (1993).** A new approach on the travelling salesman problem by genetic algorithms. *Proc. 5th Intl. Conf. on Genetic Algorithms and their Applications. Morgan Kaufmann*, San Mateo. CA. 460 - 465.

[80] **Huang, D. (1989).** The context-array bucket-brigade algorithm: An enhanced approach to credit-apportionment in classifier systems. Proceedings of the Third International Conference on Genetic Algorithms, 311-316.

[81] **Jog, P., Suh, J.Y., & Gucht, D.V. (1989).** The effect of population size, heuristic crossover and local improvement on a genetic algorithm for the travelling salesman problem. *Proc. of 3er Intl. Conference on Genetic algorithms.* J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos, CA. 110-115.

[82] **Kovacs, T. (1996).** Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, School of Computer Science, University of Bermingham, UK. Also tech. Report CSR-96-17 and CSRP-96-17 ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz.

[83] **Kovacs, T. (1999).** Deletion schemes for classifier systems. In Banzhaf, w., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., & Smith, R. E. (Eds.). Proceeding of the Genetic and Evolutionary Computation Conference (GECCO-99) (pp. 329-336). San Francisco, CA: Morgan Kaufmann.

[84] **Kovacs, T. (1999).** What Makes a Problem Hard for XCS?. GECCO-99.

[85] **Kovacs, T. (2000).** Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), Learning Classifier Systems:

From Foundations to Applications, LNAI 1813 (pp. 143-160). Berlin Heildelberg: Springer-Verlag.

[86] **Kovacs, T. (2001).** What should a classifier systems learn? In Proceeding of the 2001 Congress on Evolutionary Computation (CEC01) (pp. 775.782). IEEE Press.

[87] **Kovacs, T., & Kerber, M. (2001).** What makes a problem hard for XCS? In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), Advances in Learning Classifier Systems: Proceedings of the Third International workshop, LNAI 1996 (pp. 80-99). Berlin Heildelberg: Springer Verlag.

[88] **Koza, J. R. (1992).** Genetic programing: On the programming of computer by means of natural selection. Cambrdge, MA: MIT Press.

[89] **Kuri M. A. (1999).** A statistic Genetic Algorithms. ENC-1999, Pachuca, México.

[90] **Laguna, M. (1990).** "A guide to implementing Tabu Search", Technical Report, Graduate School of Business.

[91] **Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.) (2000).** Learning Classifier Systems. From Foundations to Applications, Volume 1813 of LNAI. Berlin: Springer-Verlag.

[92] **Lecan, L.F. (1998).** GAT I have come to understand about classifier systems, Department of Electrical Engineering and Computer Science, University of Michigan.

[93] **Mahfoud, S. W., & Goldberg, D. E. (1995).** Niching method for GA. *IlliGAL Report No. 95001, May.* The Illinois Genetic Algorithm Laboratory. University of Illinois at Urbana-Champaign.

[94] **Matellán, V. (1998).** *ABC^2: Un Modelo para el Control de Robots Autónomos.* PhD thesis, Universidad Politécnica de Madrid.

[95] **Matellán, V., Borrajo, D. (1998).** *ABC^2*: An architecture for intelligent autonomous systems. In *Proceedings of the Third IFAC Symposium on Intelligent Autonomous Vehicles*, pages 57-61, Madrid, Spain.

[96] **Mejía Olvera, M., & Cantú-Paz, E. (1994).** DGENESIS – Software para la ejecución de algoritmos genéticos distribuidos. Memorias de la XX Conferencia Latinoamericana de Informática. CLEI Panel. (pp. 935-946). Atizapan de Zaragoza, México.

[97] **Mejía Olvera, M., & Cantú-Paz, E. (1995).** Algoritmos Genéticos paralelos. Soluciones Avanzadas. Vol. 3, N° 17. pp. 12-19.

[98] **Michalewicz, Z. (1992).** Genetic algorithms + data structures = evolutionary programs, Springer-Verlag, Berlin.

[99] **Mitchalski, R. S., Mitchalski, J. G., Carbonel, T. & Mitchell, M. (1994).** *Machine Learning. An Artificial Inteligence Approach*, volume 1. Springer-Verlag.

[100] **Moore, A. W. (1991).** Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. *Proceedings in Eighth International Machine Learning Workshop.*

[101] **Moore, A. W. (1994).** The party-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, pages 711-718, San Mateo, CA, Morgan Kaufmann.

[102] **Mühlenbein, H., Gorges-Schleuter, M., & Krmer, O. (1988).** Evolution algorithms in combinatorial optimization. *Parallel Computing, Vol. 7.* 65-85.

[103] **Oliver, I. M., Smith, D. J., & Holland, J. R. (1987).** A study of permutation crossover operators on the travelling salesman problem. *Proc. Second Intl. Conf. On Genetic Algorithms and their Applications.* 224-230.

[104] **Padilla, D, F. (2001).** Technical report - Learning of Classifier Systems vs. knowledge cognitive in the Classifier Systems, UIUC, November, 2001.

[105] Padilla, D. F. (2002). Technical note and software (LCS). Models and learning in LCS, ILLIGAL, January, 2002.

[106] **Pelikan, M. Goldberg, D.E., & Cantú-Paz, E. (1999).** Linkage problem, distribution estimation, and Bayesian networks. ILLIGAL 98013.

[107] **Pelikan, M. Goldberg, D.E., & Cantú-Paz, E. (2000).** Bayesian Optimization Algorithms, population size, and time to convergence. In Whitley, D., Goldberg, D.E., Cantú-Paz, E., GECCO-2000: Proceeding of the Genetic of the Genetic and Evolutionary Computation Conference. (pp. 275-282). San Francisco, CA: Morgan Jaufmann.

[108] **Pelikan, M., Goldberg, D.E., & Cantú-Paz, E. (1999).** BOA: The Bayesian optimization algorithms. ILLIGAL 99003.

[109] **Poon, P. W., & Carter, J. N. (1995).** Genetic algorithm crossover operators for ordering applications. *Computers Ops. Res.* Vol. 22, No. 1. 135-147.

[110] **Reeves, C.R. (1993).** Modern heuristic techniques for combinatorial problems. John Wiley and Sons, Inc. New York.

[111] **Riolo, R. L. (1988).** Empirical studies of default hierarchies and sequences of rules in learning classifier systems. PhD thesis, University of Michigan

[112] **Riolo, R.L., & Robertson, G. (1988).** A tale of two classifier systems. Machine Learning, 3, 139-159.

[113] **Rodríguez, Katya, V., & Peter J. Fleming (1999).** Genetic Algorithms for subset selection in system identification. ENC-1999, Pachuca, México.

[114] **Samuel, L. (1959).** Some studies in machine learning using the game of checkers. IBM Journal on Research and Development, 3:210-229.

[115] **Schaffer, J.D., Caruana, R.A. , Eshelman, L.J., & Das, R. (1989).** A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proc. of 3er Intl. Conference on Genetic Algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos, CA. 51-60.

[116] **Shaefer, C.G. (1987).** Proc of 2nd Intl. Conference on Genetic Algorithms. J.J. Grefenstette(Ed.). Erlbaun, Hills-dale, NJ. 50-58.

[117] **Smith, R.E. (1991).** Default Hierarchy Formation and Memory Exploitation in Learning Classifiers Systems. Ph. D. thesis, The University of Alabama, Tuscaloosa, Al.

[118] **Smith, S. F. (1980).** A Learning System Based on Genetic Adaptive Algorithms. Doctoral Dissertation, Department of Computer Science, University of Pittsburgh, PA.

[119] **Smith, S. F. (1983).** Flexible learning of problem solving heuristics through adaptive search. In Proceedings of the 8th Joint Conference on Artificial Intelligence.

[120] **Starkweather, T., McDaniel, S., Mathias, K., Whitley, C., & Whitley, D. (1991).** A comparison of genetic sequencing operators. *Proc. 4th Intl. Conf. on Genetic Algorithms and their Applications*. Morgan Kaufmann. La Jolla, CA. 69-76.

[121] **Sutton, Richard, S. (1984).** Temporal credit assignment in reinforcement learning. PhD thesis, Dept. of Computer & Information Sciences, Univ. of Massachusetts, Amherst, MA.

[122] **Sutton, Richard, S. (1988).** Learning to predict by the methods of temporal differences. Machine Learning, 3.

[123] **Sutton, Richard, S., & Barto, A. G. (1998).** Reinforcement Learning. An Introduction MIT Press, 1998.

[124] **Suzuki, J. (1995).** A Markov chain analysis on a simple genetic algorithm. *IEEE Trans On Systems, Man and Cybernetics*, Vol. SMC-25, No. 4,655-659.

[125] **Syswerda, G. (1989).** Uniform crossover in genetic algorithms. *Proc. of 3er Intl Conference on Genetic Algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos, CA. 2-9.

[126] **Syswerda, G. (1991).** Schedule optimization using genetic algorithms. *Handbook of*

Genetic Algorithms. L David (Ed.). Van Nostrand Reinhold, New York. 332-349.

[127] **Ulder, N. L. J., Aarts, E. H. L., Bandelt, H. J., Laarhoven, P. J. M., & Pesch, E. (1990).** Genetic local search algorithms for the travelling salesman problem. *Proc. First Workshop Parallel Problem Solving from Nature*, Springer- Verlag, Berlin. 109-116.University of Colorado, Boulder, Colorado, August, 1994.

[128] **Valenzuela-Rendón, M. (1991).** The Fuzzy Classifier System: A Classifier System for Continuously Varing Variables. Proc. of the Fourth Int. Conf. on Genetic Algorithms, San Mateo, CA: Morgan Kauffmann, 346-353.

[129] **Watkins C. H. (1989).** Learning with delayed rewards. Ph. D. dissertation, Psychology Department. University of Cambridge, England.

[130] **Watkins, C. H, & Dayan, P. (1992). Technical note** - Q-learning. *Machine Learning*, 8(3/4).279-292, May 1992.

[131] **Whitley, D. (1989).** The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. *Proc. of 3er Intl. Conference on Genetic Algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos, CA. 116-121.

[132] **Whitley, D., & Kauth, J. (1988).** GENITOR: A different genetic algorithm. *Proceeding Rocky Mountain Conf. on Artificial Intelligence.*

[133] **Whitley, D., Mathias, & Fitzhorn, P. (1991).** Delta coding: An iterative search strategy for genetic algorithms. *Proc. 4th Intl. Conf. on Genetic Algorithms and their Applications*. Morgan Kaufmann. La Jolla, CA. 77-84.

[134] **Whitley, D., Starkweather, & Fuquay, D. T. (1989)**. Squeduling problems and travelling salesman: the genetic edge recombination and operator. *Proc. of 3er Intl. Conference on Genetic Algorithms*. J. D. Schaffer (Ed.). Morgan Kaufmann, Los Altos, CA. 133-140.

[135] **Whitley, D., Starkweather, & Shaner, D. (1991).** The travelling salesman and sequence scheduling: quality solutions using genetic edge recombination. *Handbook of Genetic Algorithms*. L David (Ed.). Van Nostrand Reinhold, New York. 350-372.

[136] **Whitley, L.D. (1993).** Foundations of genetic algorithms 2. San Mateo: Morgan Kaufmann.

[137] **Wilson S.W., & Goldberg D.E. (1989)**. A critical review of classifier systems. Proceedings of the Third International Conference on Genetic Algorithms, 244-255.

[138] **Wilson, S. W.** (1987). Classifier Systems and the Animat Problem. Machine Learning Journal 2, 199-228.

[139] **Wilson, S. W.** (1987). Hierarchical Credit Allocation in a Classifier System. In Genetic Algorithm and Simulated Annealing. L. Davis (Ed.). (CA: Morgan Kaufmann, Los Altos, 1987) 104-105.

[140] **Wilson, S. W. (1988).** Bid competition and specificity reconsidered. Complex Systems, 2(6).705-723.

[141] **Wilson, S. W.** (1994). ZCS: A Zeroth Level Classifier System. Evolutionary Computation Vol. 2, No 1, 1-18.

[142] **Wilson, S. W. (1995).** Classifier fitness based on accuracy. Evolutionary Computation, 3(2), 149-175.

[143] **Wilson, S. W. (1996).** Generalisation in the XCS classifier system. In Koza, J. R., Banzhaf, W., Chllapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M. H., Goldberg, D. E., Iba, H., & riolo, R. (Eds.), Genetic Programming 1998. Proceeding of the Third annual Conference (pp. 665-674). Morgan Kaufmann.