UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

FACULTAD DE INGENIERÍA
MAESTRÍA EN CIENCIAS DE LA INGENIERÍA

Sistema genérico de clasificación extendida con aprendizaje hebbiano (GXCS-H) para apoyo a la generación de comportamientos autónomos

TESIS

QUE PARA OBTENER EL GRADO DE:

Maestra en Ciencias de la Ingeniería

PRESENTA:
Ing. Bertha Eugenia Ordoñez Guadarrama

DIRIGIDA POR:
Dr. Marco Antonio Ramos Corchado

Dr. Erick Vresnev Castellanos Hernández

Toluca, México, 2017

COMITÉ TUTORIAL

Dr. Marco Antonio Ramos Corchado

Dra. Vianney Muñoz Jiménez

Dr. José Raymundo Marcial Romero

AGRADECIMIENTOS

Agradezco primeramente al Creador por llenar mi vida de bendiciones y por guiarme hacia el estudio de la maestría, pues el tema aquí presentado me pareció muy interesante y de ello aprendí muchísimo para mi vida en general.

A mi mamá, por estar siempre ahí, por escuchar mis pláticas sobre mi tema de tesis. Gracias por inculcarme el hábito de la lectura, por enseñarme a razonar las cosas y contagiarme las ganas de aprender siempre más.

To my love Bob, thank you for being in my life, you arrived just in time to make me belive in myself and make me feel stronger. Thank you for listening about my thesis theme and give me some ideas to complete my explanation.

A familiares y amigos, por el apoyo que me otorgaron de diferentes maneras.

Profesores y personal de la Facultad de Ingeniería de la UAEMex y Cinvestav, por los conocimientos que me transmitieron y el apoyo recibido, respectivamente, en particular al Dr. Marco Antonio Ramos Corchado y al Dr. Erck Vresnev Castellanos Hernández, por dirigirme en la elaboración de la tesis.

A Conacyt, por el apoyo económico recibido a través de la beca de investigación asociada al CVU 702163.

A Cinvestav y al Dr. Félix Francisco Ramos Corchado, por los conocimientos que me compartieron en mi estancia en el semestre en el que me permitieron realizar mi estancia de investigación en sus instalaciones del campus Guadalajara, en el estado de Jalisco.

RESUMEN

Los sistemas clasificadores de aprendizaje LCS, son herramientas de apoyo para los agentes virtuales. Estos sistemas les permite generar comportamientos autónomos en ambientes virtuales basados en un conjunto de reglas y acciones que hacen posible encontrar una solución a un problema dado a través de dos tipos de algoritmos: uno de generación de reglas y otro para evaluar las existentes por medio del aprendizaje que ellos obtienen.

Los LCS generan un resultado aproximado al que espera el usuario, pero su algoritmo de aprendizaje está lejos de emular el aprendizaje humano, este aprendizaje se logra a través de la sinapsis neuronal generada al tomar una decisión y aprender de ella, la cual puede ser emulada a partir del aprendizaje hebbiano, ya que éste se basa en los estudios realizados sobre la conexión de una red de neuronas cuando ocurre el proceso de sinapsis, el cual da como resultado que el ser humano reaccione con un comportamiento que puede repetir si se ha generado la misma sinapsis con aprendizajes parecidos.

A partir de los estudios sobre la sipnasis neuronal y de la necesidad de cambiar el algoritmo de aprendizaje en un sistema clasificador, se decidió implementar el aprendizaje hebbiano en el sistema clasificador de aprendizaje GXCS (Generic eXtended Classifier System) con el objeto de generar comportamientos basados en la sinapsis neuronal.

El LCS implementado durante esta investigación se probó, al igual que se realiza en otros sistemas de la misma índole, sobre ambientes computacionales (woods y multiplexor) propuestos por la comunidad que los desarrolla [12]. Esto permite realizar la comparación entre los LCS deseados y verificar en este caso, si el aprendizaje hebbiano genera la convergencia esperada en la generación de soluciones a los problemas utilizados.

En el presente trabajo de investigación se muestra que es posible introducir la ecuación de la sinapsis neuronal dentro de un LCS. Esto se puede observar en el desarrollo de éste, donde se muestran resultados de varias pruebas y se demuestra que al modificar el aprendizaje actual por el hebbiano, el LCS tiene un comportamiento que mejora el rendimiento del mismo. Dentro de este documento se describirá a detalle este resultado.

TABLA DE CONTENIDO

1. Introducción **1**
- 1.1. Descripción . . . 1
- 1.2. Planteamiento del problema . . . 2
- 1.3. Justificación . . . 3
- 1.4. Objetivos . . . 3
  - 1.4.1. Objetivo general . . . 3
  - 1.4.2. Objetivos particulares . . . 3
- 1.5. Hipótesis . . . 4
- 1.6. Estructura del documento . . . 4

2. Marco teórico **6**
- 2.1. Agentes inteligentes y sistemas multiagentes . . . 6
  - 2.1.1. Agente inteligente . . . 6
  - 2.1.2. Enfoque multiagente . . . 7
  - 2.1.3. Sistemas multiagente . . . 7
- 2.2. Tipos de Aprendizaje . . . 8
- 2.3. Algoritmos genéticos . . . 9
- 2.4. Aprendizaje por clasificación . . . 12
  - 2.4.1. El sistema de mensajes y de reglas . . . 12
  - 2.4.2. Sistema clasificador de aprendizaje . . . 12
  - 2.4.3. Sistema de clasificación Michigan . . . 12
  - 2.4.4. Sistema de clasificación Pittsburg . . . 13
- 2.5. Arquitecturas de sistemas clasificadores de aprendizaje . . . 14
  - 2.5.1. LCS . . . 14
  - 2.5.2. ZCS . . . 16
  - 2.5.3. XCS . . . 17
- 2.6. Mecanismos de aprendizaje en clasificadores . . . 18
  - 2.6.1. Algoritmo del bombero . . . 18
  - 2.6.2. Aprendizaje rápido . . . 20

3. Estado del arte **23**
- 3.1. GXCS . . . 23
- 3.2. Aprendizaje hebbiano . . . 26
  - 3.2.1. Reglas de plasticidad sináptica . . . 27

3.2.1.1. Normalización multiplicativa 30
3.2.1.2. Regla instar 31
3.2.1.3. Regla outstar 31
3.3. Modelos de evaluación para LCS 32
3.3.1. Ambiente woods 33
3.3.2. Ambiente multiplexor booleano 34
3.3.3. Resumen 34

4. Sistema clasificador de aprendizaje GXCS-H 35
4.1. Propuesta: Aprendizaje hebbiano en un GXCS 35
4.2. Arquitectura del GXCS-H 36
4.3. Reglas hebbianas estables 40
4.3.1. Regla de normalización multiplicativa 40
4.3.2. Regla de instar 41
4.3.3. Regla de outstar 42
4.4. Desarrollo 43
4.5. Pruebas y resultados 52
4.5.1. Pruebas 52

5. Conclusiones y trabajo futuro 63
5.1. Conclusiones 63
5.2. Trabajo futuro 64

A. Publicación 65

Índice de figuras

2.1. Agente inteligente, basado en [36] 7
2.2. Algoritmo genético, basado en [25] 11
2.3. Arquitectura LCS, basado en [20] 16
2.4. Arquitectura ZCS, basado en [43] 17
2.5. Arquitectura XCS, basado en [44] 18
2.6. Algoritmo de aprendizaje rápido 22

3.1. Arquitectura de GXCS, basado en [33] 24
3.2. Diagrama de flujo del aprendizaje rápido en el clasificador 26
3.3. Aprendizaje hebbiano, basado en [14] 27

4.1. Arquitectura del GXCS-H 37
4.2. Diagrama de flujo del aprendizaje hebbiano en GXCS-H 39
4.3. Arquitectura en normalización 41
4.4. Arquitectura en instar 42
4.5. Arquitectura en outstar 43
4.6. Diagrama de clases UML 48
4.7. WeightVector.java en UML 49
4.8. Ejemplo de ambiente woods 54
4.9. Ejemplo de ambiente multiplexor 55
4.10. Comparación GXCS y GXCS-H con Oja 58
4.11. Comparación GXCS y GXCS-H con instar 59
4.12. Comparación GXCS y GXCS-H con outstar 60
4.13. Comparación GXCS y GXCS-H con Oja 61
4.14. Comparación GXCS e instar 61
4.15. Comparación GXCS y Oustar 62

Índice de tablas

3.1. Vertientes hebbianas 29

4.1. Valores de las constantes 46
4.2. Principales clasificadores de aprendizaje 51
4.3. Vertientes hebbianas estables 52
4.4. Comparación sin el uso de la degradación 57
4.5. Comparación con degradación 62

Capítulo 1

Introducción

En el presente capítulo se exponen las bases que sustentan la propuesta de investigación de este trabajo. El contenido incluye el planteamiento del problema que se abordó, la justificación de su desarrollo, los objetivos que sirvieron de guía y meta a alcanzar, así como la hipótesis a comprobar con el desarrollo basado en los conceptos que se revisaron a lo largo de la investigación.

1.1. Descripción

Como se menciona en [21], las herramientas informáticas de la inteligencia artificial se han ido incorporando en las actividades diarias del ser humano al grado de tener sistemas donde el usuario interactúa directamente con un ambiente creado y asistido por computadora por medio de los agentes virtuales, que son entidades con comportamiento autónomo que reaccionan al ambiente.

Para lograrlo, la inteligencia artificial se vale de mecanismos que permiten que los agentes virtuales adopten un comportamiento dentro del ambiente simulado en el que se encuentran.

Los comportamientos que los agentes realizan son posibles gracias a que existen algunos mecanismos de aprendizaje, tales como el algoritmo del bombero [18] o de aprendizaje rápido [40], que permiten a los agentes tomar decisiones a partir de la experiencia que van adquiriendo.

Dicho aprendizaje puede ser replicado y mejorado al tomar como herramientas de apoyo a los Sistemas Clasificadores de Aprendizaje (LCS, por sus siglas en inglés) que son máquinas de aprendizaje que apoyan en su reacción a las condiciones del ambiente. Este tipo de máquinas son conocidas como adaptativas o evolutivas debido a la integración de un mecanismo de evolución que permite que el agente vaya aprendiendo y, por lo tanto, evolucionando. La capacidad de percibir el ambiente y poder utilizar un conjunto de reglas para seleccionar la

que mejor resuelva el problema, permite generar comportamientos autónomos inteligentes como los descritos en [20].

Aunque los LCS han permitido que los agentes tengan comportamientos aceptables generados por medio de algoritmos de aprendizaje aun se obtienen modelos poco intuitivos para el usuario [21] al no tener base neurocientífica y por lo tanto, no se acercan a los resultados obtenidos de estudios realizados sobre el comportamiento humano.

Existe un método de aprendizaje que está basado en la manera en que el cerebro humano genera sinapsis neuronal, esta sinapsis ocurre cuando el ser humano aprende, es decir, cuando se enfrenta a ciertas situaciones en la vida, realiza un comportamiento que lo lleva a un resultado, a partir de este resultado aprende y toma decisiones. Este aprendizaje se conoce como hebbiano.

Al incluir el aprendizaje hebbiano en un LCS, un agente puede adoptar un aprendizaje basado en la sinapsis, con lo que éste tomará decisiones similares a las que toma un ser humano al interactuar con su ambiente. Con ello, el comportamiento del agente imita el comportamiento humano de una manera aproximada. Así, los sistemas de inmersión virtual que se basen en un LCS que cuente con aprendizaje hebbiano serán más intuitivos para el usuario y más llamativos, ya que sentirá que interactúa con un sistema que se comporta de manera similar a su propio comportamiento.

1.2. Planteamiento del problema

Dentro de la creación de comportamientos virtuales embebidos en agentes es necesario realizar mejoras, pues algunos modelos de esta índole son poco intuitivos[29], generan desinterés en el usuario [3] o emulan inadecuadamente el comportamiento humano, de tal manera que no se logra el resultado esperado, como por ejemplo, en automóviles autónomos, un vehículo autónomo aun no logra tomar decisiones de tal manera que se conduce solo en situaciones controladas, donde se tiene una ruta ideal [22].

Es por esto que se considera necesario que los agentes virtuales generen comportamientos autónomos que emulen el comportamiento humano a partir del estudio de la sinapsis neuronal que da como resultado el aprendizaje humano. Con esto se evita la generación de comportamientos poco intuitivos o repetitivos.

El tipo de comportamiento basado en la sinapsis neuronal puede ser emulado a través de las ecuaciones del aprendizaje hebbiano, y este aprendizaje a su vez puede ser implementado en un LCS con el objeto de emularlo y generar comportamientos cercanos a los generados al ser humano al aprender de las situaciones que vive.

1.3. Justificación

Los comportamientos autónomos que realizan los agentes virtuales se han generado a través de diferentes algoritmos de aprendizaje, algunos de ellos se utilizan dentro de los LCS. Se ha podido observar que estos comportamientos son repetitivos y no captan la atención del usuario que interactúa con ellos a través de un dispositivo electrónico. Por ello, se tuvo la inquietud de investigar sobre algún mecanismo de aprendizaje apegado al del ser humano.

El mecanismo elegido fue tomado del área de neurociencias y es llamado aprendizaje hebbiano, el cuál está basado en el comportamiento cerebral de las neuronas al momento en que el ser humano realiza una actividad, generando un comportamiento. Al realizarla, las neuronas se conectan, si el ser humano vuelve a realizar la misma actividad, esas mismas neuronas se fortalecen y el comportamiento se vuelve a repetir cuando sucede una situación similar. Ese fortalecimiento neuronal hace que el ser humano aprenda y tome decisiones.

En el presente trabajo se decidió unir ambos conocimientos para explotar las ventajas que tiene un LCS, en este caso el GXCS (*Generic eXtended Classifier System*), así como incorporar en él, el aprendizaje hebbiano con el objeto de emular la sinapsis cerebral humana implementando las ecuaciones hebbianas en el GXCS y así generar un comportamiento aproximado al del ser humano.

1.4. Objetivos

1.4.1. Objetivo general

Implementar el aprendizaje hebbiano dentro del Generic eXtended Classifier System para obtener un comportamiento similar al generado a través de la sinapsis cerebral tal como sucede en el proceso de aprendizaje de un individuo.

1.4.2. Objetivos particulares

- Recavar información sobre las diferentes vertientes relacionadas con el aprendiaje hebbiano.
- Realizar pruebas en el código del GXCS desarrollado en Java con el objeto de comprender la implementación y posteriormente modificarla.
- Generar e implementar un algoritmo con base en las ecuaciones que pertenecen a las vertientes del aprendizaje hebbiano en el GXCS.
- Comparar los resultados de la versión GXCS y los generados por el GXCS-H (GXCS con hebbiano) en un ambiente virtual utilizando los ambientes woods y multiplexor.

1.5. Hipótesis

Incorporar el aprendizaje hebbiano en un GXCS permite que los agentes virtuales se adapten a las condiciones del ambiente en el que se encuentran a partir de un comportamiento basado en la ecuación que describe la sinapsis neuronal.

Con esta incorporación, el agente virtual se comporta de una manera aproximada al comportamiento humano, pues se toma la base de la sinapsis neuronal al implementar la ecuación de hebb y adecuándola al conjunto de condición/acción con la que se trabaja en un LCS. A partir de ahí genera un aprendizaje que permite tomar la acción que mejor se acople a la condición y que genere un resultado esperado por el usuario.

El resultado esperado se compara con los resultados del GXCS, los cuales se toman del rendimiento, el cual a su vez está basado en la aptitud, el castigo y el premio de cada par condición/acción.

Si los resultados obtenidos por el GXCS-H generan una aptitud mayor al que genera el GXCS por lo menos en un 80 %, es decir, que el rendimiento reportado sea mayor a 0.6, entonces se considera como válida esta hipótesis.

1.6. Estructura del documento

Este trabajo de investigación consta de cinco capítulos y un anexo. El presente capítulo contiene una pequeña introducción, el planteamiento del problema, la justificación de su realización, objetivos del mismo e hipótesis correspondiente.

El segundo capítulo consta del marco teórico, donde se describen temas que sirvieron de base para el trabajo de investigación contenido en este documento. Entre algunos temas podemos encontrar la definición de un agente, una breve descripción de los sistemas multiagente, sistemas clasificadores de aprendizaje y sus respectivos mecanismos de aprendizaje.

En el tercer capítulo se describe el estado del arte, que contiene información del LCS que se utiliza de base para esta investigación, es decir el GXCS, así como los modelos de prueba usados también en los clasificadores y la información recabada sobre algunas de las vertientes del aprendizaje hebbiano.

En el cuarto capítulo se aborda la propuesta de investigación, donde se puede observar que se han utilizado los conceptos abordados en los capítulos anteriores y se ha implementado el aprendizaje hebbiano en el clasificador GXCS-H. También contiene los resultados de las pruebas realizadas sobre el desarrollo de software en ambos ambientes.

En el quinto capítulo se pueden consultar las conclusiones a las que se llegó durante el proceso de pruebas y obtención de resultados y se sugieren algunas ideas para trabajo futuro.

El documento también cuenta con un anexo donde se menciona la publicación relacionada al tema de investigación.

Capítulo 2

Marco teórico

En este capítulo se abordan temas de investigación que sustentan el desarrollo de la propuesta hecha para este trabajo. Se decidió incluir la definición de agente inteligente, una breve descripción de lo que es un sistema multiagente, conceptos de apredizaje y los dos tipos de clasificación bajo los que se encuentran los LCS que sirven como base al sistema clasificador desarrollado en el presente trabajo.

2.1. Agentes inteligentes y sistemas multiagentes

2.1.1. Agente inteligente

Como se describe en [36], un agente inteligente es cualquier entidad capaz de percibir su ambiente por medio de sensores y que actúa en el ambiente a través de sus efectores. Un humano tiene ojos, oídos, y otros órganos que fungen como sus sensores y, manos, piernas, brazos y otras partes del cuerpo que hacen la función de efectores. Un agente robótico los emula por medio de cámaras y buscadores de rango infrarrojo que fungen como sensores y varios motores que fungen como efectores. Un agente de software contiene cadenas de bits codificadas que funcionan como sensores y acciones que pasan a su vez por esos sensores que se transforman en efectores.

Se puede decir entonces que un agente inteligente es una entidad virtual que se mueve dentro de un ambiente, por ejemplo, un robot con objetivos, acciones y conocimientos aprendidos [38]. Un agente inteligente debe ser autónomo en la medida en que su comportamiento sea determinado por su propia experiencia, es decir, cuando sea capaz de trabajar adecuadamente en una gran variedad de ambientes una vez que se ha adaptado a ellos [36].

Esto se puede ejemplificar en la Figura 2.1, donde se puede ver la interacción con el ambiente a través de los efectores y sensores, es decir, el receptor y el

emisor. A partir de las condiciones que el agente recibe del ambiente puede reconocer las acciones que va a realizar.

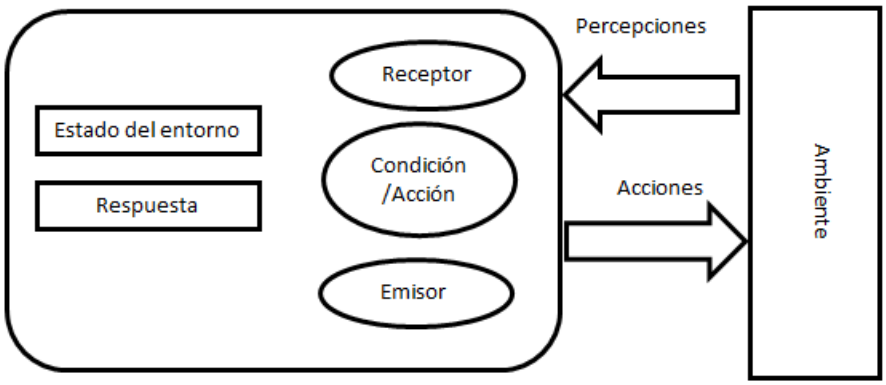

Figura 2.1: Agente inteligente, basado en [36]

Los agentes se utilizan en gran medida en sistemas llamados multiagentes los cuales tienen la capacidad de manipular más de un agente en un ambiente [16].

2.1.2. Enfoque multiagente

Un sistema multiagente tiene como objetivo, proporcionar los principios de construcción de sistemas complejos que involucra múltiples agentes y mecanismos de coordinación para los comportamientos de los agentes independientes [38]. En un sistema multiagente interactúan los agentes de tal manera que se puedan cumplir objetivos y realizar ciertas acciones, por ejemplo, reconocer un obstáculo o a otro agente en el mismo ambiente.

Existen algunos enfoques multiagente importantes para un LCS, los cuales pueden ser consultados en [13] para el enfoque según J. Ferber, [23] para el enfoque desarrollado por P. Maes y [6] para el enfoque semiótico que identificó P. Cariani.

2.1.3. Sistemas multiagente

J. Ferber [13] constata que un sistema multiagente depende directamente de su ambiente y de los mensajes internos que existen en sus agentes, es decir, la comunicación. Un sistema multiagente está compuesto por los siguientes elementos:

- Un ambiente **M**. Espacio que dispone de una métrica particular para cada ambiente.

- Un conjunto de objetos **O**. La mayoría de estos objetos son pasivos y pueden ser creados, eliminados o modificados por los agentes y se encuentran en cierto punto del ambiente **M**.
- Un conjunto de agentes **A**. Objetos particulares del conjunto **O** que representan las entidades activas del sistema.
- Un conjunto de relaciones **R**. Estas relaciones permiten la interacción entre los objetos.
- Un conjunto de operaciones **Op**. Permiten a los agentes percibir, consumir, producir, transformar y manipular los objetos de **O**.
- Un conjunto de operadores encargados de representar el uso de sus operaciones y el resultado obtenido de dicho uso en el ambiente.

En esta definición se toma al conjunto **O** como el formado por los objetos clasificados según el contexto general del ambiente en el que se encuentran. Aunque J. Ferber [13] dice que es posible que **M** se encuentre vacío y que $\mathbf{A} = \mathbf{O}$, define también una red donde los agentes son puramente comunicativos.

El acto de comunicación no es un simple intercambio entre agentes que influirá en sus creencias y objetivos sino que también permite cambiar la forma en que cooperan. La comunicación es un problema crítico en los sistemas multiagentes, por lo que es necesario definir las leyes de la interacción entre los agentes y de medio ambiente en el sistema donde se deseen replicar.

2.2. Tipos de Aprendizaje

Existen varios tipos de aprendizaje [32] que han sido utilizados en ámbitos computacionales, específicamente en redes neuronales y LCS, entre ellos se encuentran: el aprendizaje automático [27], por simulación montecarlo [32], corrección de error [6], estocástico [6], asociativo [32], competitivo [32], aprendizaje supervisado y no supervisado [11]. Algunos tipos de aprendizaje se describen a continuación:

- Automático. Es un campo de la inteligencia artificial involucrado en métodos y algoritmos que permiten que las máquinas aprendan. Estos algoritmos ayudan a que el conocimiento adquirido sea entendido a través de un número limitado de observaciones, de su objetivo o por medio de una descripción de una tarea.

- Por corrección de error. Calcula los pesos entre las conexiones de una red a partir de la diferencia entre los valores que se desean obtener y los que realmente se obtienen al salir de la red.

- Asociativo. Relaciona una sola entrada con su respectiva salida para que la siguiente vez que se presente la misma entrada se asocie con la misma salida.
- Competitivo y cooperativo. En este tipo de aprendizaje, las entradas y salidas realizan una tarea en conjunto para la cual cooperan y a la vez compiten para ver qué par entradas-salidas se van a activar dependiendo de la entrada que tome.
- Estocástico. Permite generar cambios en los valores de los pesos entre las conexiones neuronales de manera aleatoria y después evalúa su efecto dependiendo de lo que se desea alcanzar. Su característica principal es que utiliza distribuciones probabilísticas para la evaluación de los pesos.
- Montecarlo. Es de tipo estocástico y utiliza diversos algoritmos para simular situaciones al azar a partir de las entradas. Utiliza distribuciones probabilísticas cuyos resultados solo afectan a la entrada actual y no afecta a las demás entradas.
- Aprendizaje supervisado. En éste se tiene un conjunto de pares entrada/salida donde la función entre las entradas y sus salidas asociadas debe ser aprendida. Dada una nueva entrada, la relación aprendida puede ser usada para predecir la salida correspondiente. Un ejemplo de una tarea de aprendizaje supervisado es la clasificación donde se busca la relación entre las propiedades y el tipo asociado que nos permite predecir el tipo de objeto a partir de un conjunto de propiedades.
- Aprendizaje no supervisado. A diferencia del aprendizaje supervisado, las salidas no están disponibles. En lugar de aprender la relación entre las entradas y las salidas se va construyendo el modelo a partir de las entradas. Considérese una tarea donde muchas propiedades de algún objeto son dadas y se requiere agrupar los objetos por la semejanza entre ellas. Los objetos sólo contienen sus propiedades pero no el agrupamiento asignado para ellos.

Hay dos tipos de aprendizaje que se utilizan en los LCS y que serán descritos en secciones posteriores. Estos dos tipos de aprendizaje trabajan en conjunto con un algoritmo genético que permite crear más opciones de prueba. En la siguiente sección se describe lo que es un algoritmo genético.

2.3. Algoritmos genéticos

Los algoritmos genéticos, AG, proveen un método de aprendizaje inspirado en la simulación de un proceso de la evolución. Surgieron del interés de la comunidad científica por construir sistemas capaces de resolver problemas complejos basados en los procesos de evolución natural definidos por Darwin. Los biólogos han

descubierto que este proceso es realizado por los cromosomas, que poseen información genética de los individuos. A partir de esas investigaciones se considera que [19]:

- La evolución es un proceso construido fundamentalmente en los cromosomas.
- El proceso de selección natural permite a los individuos adaptarse al ambiente para sobrevivir y reproducirse, en algunos casos.
- La reproducción da lugar a la evolución. Los procesos de mutación producen cambios en los cromosomas.

John Holland fue el primero en aplicar las características de la vida en el campo de la informática al diseñar sistemas artificiales con un comportamiento análogo a un sistema de evolución natural. Él diseñó los AG pensando en imitar el proceso de la evolución, los cuáles consisten en hacer evolucionar una población de criaturas artificiales (con cadenas de caracteres) con ayuda de la selección, cruce y mutación de los cromosomas, como sucede en la teoría de la evolución de Darwin [41].

Los AG tienen 4 puntos principales:

- Trabajan las codificaciones del espacio de búsqueda con valores determinados.
- Efectúan la búsqueda en una población de soluciones posibles a través de un solo valor.
- No utilizan derivados ni otras propiedades de la función objetivo, sólo la función objetivo por sí misma.
- Son regidos por medio de reglas de transición probabilistas, no deterministas.

Los AG requieren que todos los valores del espacio de búsqueda sean codificados por medio de cadenas de caracteres finitas sobre un alfabeto de caracteres que también es finito y predeterminado para un caso particular.

Los AG pueden ser aplicados en múltiples tareas [2]: Optimización de funciones numéricas (discontinuas, multimodales, ruidosas, etc.) [26], tratamiento de imágenes (alineación de fotos satelitales, reconocimiento de sospechosos, etc.), optimización del empleo del tiempo, optimización de diseño, control de sistemas industriales [2], administración de un sistema de control de tiempos (cadena de producción, central nuclear, etc.), determinar la configuración de energía mínima de moléculas o modelar el comportamiento animal.

Los AG son igualmente utilizados para optimizar las redes (cables, fibras ópticas, también de agua o gas, etc.), los circuitos VLSI (Very Large Scale Integration ó Integración a muy grande escala) [34], antenas, etc. Se pueden utilizar también para encontrar los parámetros de un modelo de pequeña señal a partir de medidas experimentales [26] y los conmutadores ópticos adiabáticos se pueden optimizar con la ayuda de las estrategias de evolución [28].

Dentro del enfoque de los AG no se trata de encontrar una solución analítica exacta o una buena aproximación numérica, pero sí encontrar soluciones que satisfagan mejor los diferentes criterios que a veces son confusos o contradictorios. Las soluciones generadas con un AG son mejores que las obtenidas por métodos clásicos y se realizan con el mismo tiempo de cálculo [1]. La Figura 2.2 resume los pasos que realiza un algoritmo genético.

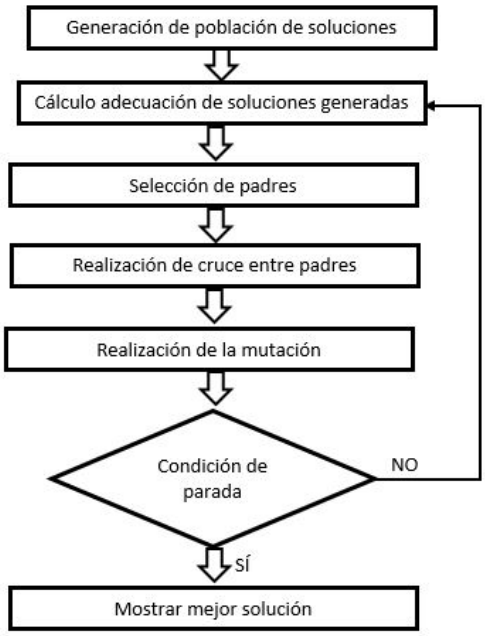

Figura 2.2: Algoritmo genético, basado en [25]

Algunos de los tipos de aprendizaje mencionados son la base para el algoritmo empleado en algunos de los LCS. En la siguiente sección se mencionan los conceptos principales de un sistema clasificador y la base de su funcionamiento en conjunto con el sistema de reglas que se requiere para llegar a un resultado.

2.4. Aprendizaje por clasificación

El CS-1, primer sistema clasificador de aprendizaje, fue creado en 1978 por John H. Holland como marco de trabajo donde se pueblan clasificadores de tipo condición - acción que evolucionan colectivamente en los sistemas basados en reglas usando algoritmos genéticos [20].

A partir de este clasificador fueron creadas otras vertientes, todas ellas basadas en el sistema de mensajes y reglas, pero con diferencias significativas al realizar el proceso de aprendizaje. La información relevante del sistema de reglas, así como las vertientes de los clasificadores más importantes para el desarrollo del presente trabajo serán descritas en el presente capítulo.

2.4.1. El sistema de mensajes y de reglas

Un sistema de clasificadores está compuesto de una interfaz de entrada (captores) que permite conocer la información proveniente del ambiente (mensajes). La información está representada como dos cadenas binarias de longitud definida que son insertadas en la lista de mensajes. Entonces, el módulo de reglas analiza el mensaje por evaluar que desencadenará una acción realizada gracias a la interfaz de salidas (efectores). La lista no influye directamente al ambiente, solamente la acción que se activará. El tamaño de esa lista de mensajes es fija, lo que implica el uso de un método de selección de mensajes que borre las que ya no se requieren [5].

2.4.2. Sistema clasificador de aprendizaje

Un LCS puede ser usado en minería de datos, sistemas difusos, sistemas neuronales, eventos nucleares controlados, etc. [5]. Estos LCS pertenecen a alguno de los dos enfoques de sistemas de clasificación: Michigan y Pittsburg, los cuales se describen en las siguientes secciones.

2.4.3. Sistema de clasificación Michigan

Este enfoque fue propuesto por primera vez por Holland con el primer LCS. En él se utiliza un algoritmo genético para generar la evolución de las reglas y un algoritmo de recompensa, como el algoritmo del bombero, para la evaluación de éstas [8]. En este caso, todos los clasificadores cooperan colectivamente para llegar a una solución. Cada clasificador elegido representa una solución completa del problema, donde un solo elemento es capaz de producir una clasificación de los datos por sí mismo a partir del par condición/acción que se genera [8].

Este tipo de clasificadores generalmente utilizan el mecanismo de aprendizaje por refuerzo, donde los clasificadores o reglas parciales útiles son recompensados de tal manera que incrementan la posibilidad de ser la solución final [8]. Cada regla o clasificador manipula un mensaje generando una condición y una acción.

El sistema interactúa con el ambiente mandándole los mensajes por medio de reglas dando como resultado una acción a partir de una condición que el mensaje satisface. En general, todos los mensajes son del mismo tamaño y pertenecen a un alfabeto definido para un ambiente en particular. Sus componentes son:

- Interfaz de entrada: Convierte el estado del ambiente en mensajes.
- Clasificadores o reglas: Definen el proceso de mensajes.
- Lista de mensajes: Mensajes de entrada dados por las reglas disparadas.
- Interfaz de salida: Traduce los mensajes en acciones para modificar el ambiente.

Lo primero que realiza el clasificador es colocar todos los mensajes de la interfaz de entrada en la lista de mensajes, después realiza una iteración donde se comparan los mensajes con la lista de condiciones de los clasificadores disparando todos los clasificadores que satisfacen sus condiciones y tienen suficiente fuerza para ser tomados en cuenta formando con cada acción una lista de mensajes nuevos. Al final reemplaza todos los mensajes anteriores por los que se han generado.

El ambiente proporciona el refuerzo mediante un premio o un castigo utilizando algún método para asignar crédito a las reglas que intervienen en la solución. A cada regla se le asigna una cantidad o fuerza que se ajusta y después se utiliza para competir. La competencia se basa en la oferta donde las reglas que son mas fuertes son las que ganan [8].

2.4.4. Sistema de clasificación Pittsburg

Introducido por Smith S. F. en 1980, este tipo de sistemas utiliza para su evaluación y evolución, un algoritmo genético. La diferencia con los otros sistemas clasificadores es que en éste, cada individuo representa una solución a un problema tratado y es literalmente una célula con su composición genética creada a partir de varios cromosomas o estructura de conocimiento, esto con el fin de responder al ambiente con el tamaño de su genoma adaptado. El tamaño de un sistema de clasificadores se expresa por el número de éstos que lo componen, así como por un número de reglas variable de cada individuo. Los individuos que componen el sistema de clasificadores son evaluados independientemente y en paralelo. Un individuo dispone de una fuerza que es dada por una función de aptitud, generalmente conocida como *fitness*, que a su vez representa el valor del individuo, fruto de k tentativas de solución del problema abordado. Es esta misma fuerza de adaptación la que va a permitir al algoritmo genético mezclar a los individuos entre ellos [37].

En este entorno, el clasificador tipo Pittsburg [37] se compone de 3 módulos:

- El sistema de solución del problema que utiliza las reglas que componen cada individuo para responder a un problema propuesto.
- La función de aptitud que mide el rendimiento de cada individuo de k ensayos de la solución del problema.
- El sistema de aprendizaje que está compuesto de un algoritmo genético con individuos de tamaño diferente.

El mecanismo de solución proporciona el uso de un lenguaje complejo así como de un espacio para colocar los mensajes internos. Cada regla dispone de una parte para la condición y de una parte para recibir la señal del ambiente. La señal se puede descomponer en muchas señales representando diferentes variables. Los captores de señal pueden ser completados con un bit simbolizado por "#" para expresar la función "no importa" para que la regla coincida si uno de los bits de valor binario no coincide con todos los bits de la condición correspondiente. Por otro lado, se pueden manipular las variables del sistema de mensajería interna incluso cuando se llegue a la acción final. Estas variables son consideradas como argumentos de la parte condicional que pueden ser instancias que ayudan a la mensajería interna con la comunicación al exterior del ambiente.

En la siguiente sección se describen algunos de los clasificadores que se generaron a partir del modelo de Holland y que son la base de invetigación del presente trabajo.

2.5. Arquitecturas de sistemas clasificadores de aprendizaje

2.5.1. LCS

El trabajo de Holland sirvió como base para Martin Butz, Wilson, Stolzmann y Goldberg quienes generaron otros modelos entre los años de 1985 y 1995, de estos trabajos realizados por ello se se tomaron el ZCS y el XCS, ya que ambos son la base principal del GXCS, clasificador que a su vez se tomó como base para este trabajo de investigación [21] [5].

Los LCS son sistemas basados en reglas donde el aprendizaje es visto como un proceso de adaptación de un ambiente parcialmente desconocido a través de un algoritmo genético y una diferencia de aprendizaje temporal [24].

John H. Holland describe el LCS como el marco de trabajo que utiliza algoritmos genéticos para estudiar el aprendizaje en sistemas basados en reglas y en el par "condición/acción". Los captores transportan el estado actual del ambiente en mensajes, los efectores transportan mensajes seleccionados a las acciones de ese ambiente. El conjunto de condiciones-reglas de acción, se construyen utilizando la siguiente notación: # Símbolo de "*don't care*" o "no importa", ? es el

símbolo "*fits all*" = "se ajusta a todo" 0 = falso, 1 = verdadero. Los símbolos # y ? se utilizan de manera indistinta para efectos de la implementación en el LCS, pues en ambos casos actúan como comodines que aceptan cualquier valor permitido en el ambiente.

Pasos de un clasificador:

- Un mensaje m es localizado en la interfaz de entrada.
- m es comparado con la condición de todos los clasificadores en la lista actual del clasificador. Los clasificadores que coinciden generan su propio conjunto.
- Un clasificador fuera del conjunto es seleccionado por el método de la "ruleta rusa" o alguna otra técnica.
- El clasificador seleccionado se activa. Esta parte interactúa con la interfaz de salida y causa una acción motora en el LCS.
- El reforzamiento de aprendizaje es aplicado al evaluar las reglas y calcular una recompensa o castigo, según sea necesario.

Una vez hecho esto, la generación de reglas es realizada para producir nuevos clasificadores mientras se reemplazan los que tienen menos fuerza. Hay dos formas de aprender en un LCS:

- Primer nivel. Asignación de créditos (reforzamiento del aprendizaje en los clasificadores). Un clasificador es reforzado (la fuerza de la regla es modificada) en dependencia de una recompensa ambiental llamado saldo (Algoritmo del bombero, aprendizaje rápido, etc).
- Segundo nivel. Usualmente independiente del primer nivel, consiste en el descubrimiento de la regla con el objeto de generar nuevos clasificadores (para ello, los LCS típicamente usan algoritmos genéticos).

De manera general, podemos ver en la Figura 2.3 cómo funciona un LCS, donde se puede observar una regla en el ambiente que entra al clasificador por medio de los detectores, esta regla se agrega a una lista de mensajes, después se compara con las reglas existentes, las cuales ya tienen asociadas una acción cada una. Esta comparación permitirá elegir sólo las reglas que son compatibles con la que está entrando desde el ambiente y las almacena en otro conjunto para ser evaluadas por el algoritmo de asignación de créditos, o para crear más a partir reglas por medio de un algoritmo genético.

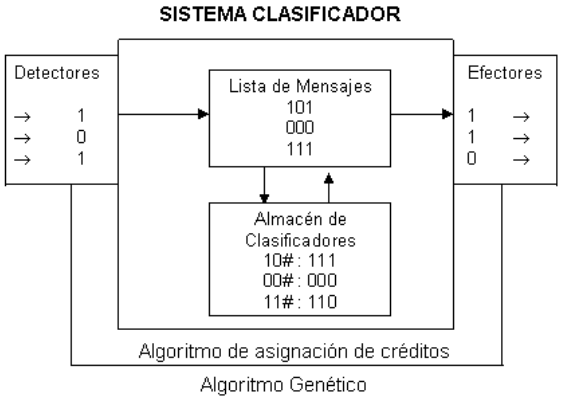

Figura 2.3: Arquitectura LCS, basado en [20]

A partir del primer clasificador desarrollado por Holland, se crearon varias versiones generadas para casos particulares, algunas de ellas son el ZCS, XCS y GXCS. Como se comentó anteriormente, estos tres clasificadores son la base para el desarrollo del clasificador del presente trabajo y se describirán en las siguientes secciones.

2.5.2. ZCS

Zeroth level Classifier System, ZCS por sus siglas en inglés, ignora elementos complejos al tomarlos de manera muy general y utiliza un algoritmo de bajo rendimiento conocido como aprendizaje rápido.

Una de las desventajas del ZCS es que no tiene lista interna de mensajes y distribuye la recompensa entre los clasificadores con la técnica combinada con los algoritmos del bombero y de aprendizaje rápido. Estos serán descritos en la siguiente sección. El ZCS fue uno de los primeros clasificadores en usar aprendizaje rápido como técnica de asignación de recompensas a los clasificadores [24].

Su funcionamiento se muestra en la Figura 2.4, donde se puede observar que la regla de ejemplo 0011 entra del ambiente. Esta regla es tomada por los captores y pasada al conjunto de población o lista de clasificadores. Entonces se compara con los clasificadores existentes en este conjunto. Después de la comparación se toman todas las reglas compatibles con la regla que acaba de entrar a la lista inicial de clasificadores y se pasan a un conjunto de coincidencias, llamado conjunto **M** de donde se toman para asignarle a cada una una posible acción y se copian a un conjunto de acciones **A**.

Se tiene a su vez un conjunto A_{-1} que contiene las acciones precedentes, las cuales se toman como punto de partida para el cálculo del reforzamiento de cada regla, este reforzamiento se calcula verificando si la regla ya fue evaluada

y elegida por tener un valor de recompensa alto y se refuerza asignándole un valor mayor al que tenía en la iteración anterior. Una vez hecho esto, este par condición/acción se envía al ambiente por medio de los efectores y es ahí donde se le da una retribución a la regla.

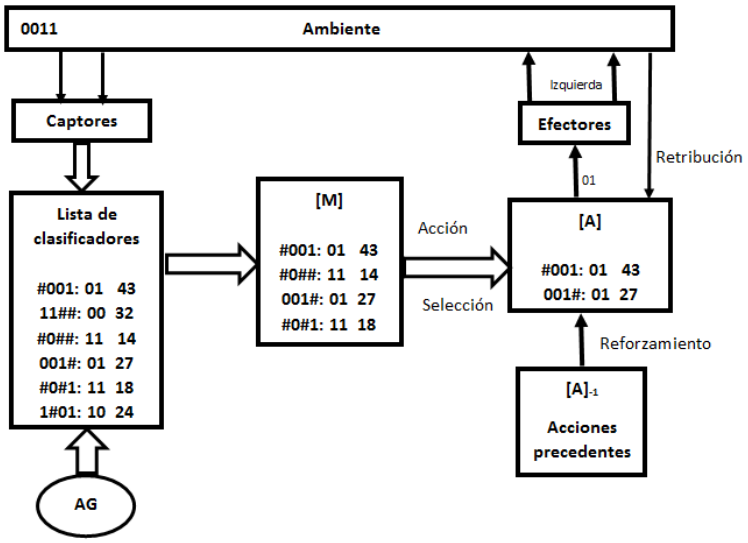

Figura 2.4: Arquitectura ZCS, basado en [43]

2.5.3. XCS

En 1995, Wilson creó el XCS, en él mantiene las ideas básicas del clasificador de Holland pero realiza la mejora del rendimiento. Sus reglas están basadas en la precisión y en una asociación completa de la base de reglas. Utiliza el algoritmo de aprendizaje rápido para distribuir las recompensas y el algoritmo genético actúa sólo sobre una parte de la población. La capacidad de los clasificadores está basada en la exactitud de las predicciones del clasificador en lugar de la predicción de sí misma. Usando tanto la exactitud del clasificador como la capacidad del algoritmo genético, XCS es capaz de evolucionar clasificadores con una predicción exacta de la recompensa esperada y empatando tantas situaciones como sean posibles sin caer en su generalización [24].

El funcionamiento del XCS se puede apreciar en la Figura 2.5 donde se puede ver que la arquitectura del XCS es muy similar a la del ZCS descrita en la sección anterior. La diferencia entre estas dos arquitecturas es que, en la del XCS, se incluye un arreglo de predicciones que se calcula sobre las reglas del conjunto de concidencias **M**.

El arreglo de predicciones contiene una recompensa inicial asignada al par

condición/acción. A partir de este arreglo, se pasan las reglas prevaloradas al conjunto de acciones **A**, las reglas que están en este conjunto se actualizan y se calcula el error en la predicción. La regla con mayores valores en la predicción se pasa a los efectores para que estos la envíen a su vez al ambiente y se pueda calcula la recompensa que se le asigna.

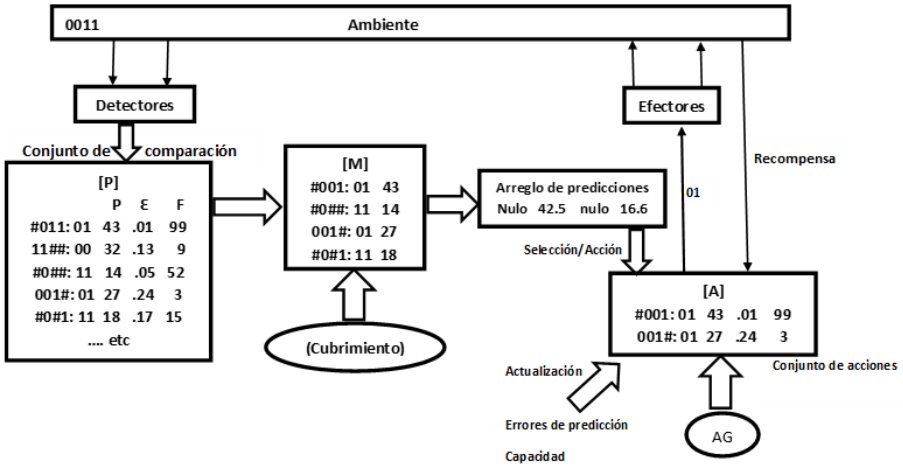

Figura 2.5: Arquitectura XCS, basado en [44]

En la siguiente sección se describen los dos mecanismos de aprendizaje utilizados en los LCS mencionados.

2.6. Mecanismos de aprendizaje en clasificadores

2.6.1. Algoritmo del bombero

El algoritmo de repartición de créditos llamado algoritmo del bombero, *Bucket brigade* o *human chain* fue definido por John Holland. La idea de éste es manipular un sistema de subasta para recompensar a los clasificadores eficaces y desplazar poco a poco a los que no aportan buenos resultados [18].

Este algoritmo está constituido de dos componentes principales: La subasta y la compensación. Si un clasificador puede ser activado por un mensaje se da a conocer al sistema y se hace visible para las subastas. Cuando cada clasificador es consultado, la subasta comienza y todos los clasificadores registrados hacen una sola oferta de valor proporcional a su fuerza. Un clasificador se elige por este método y puede a su vez producir un nuevo mensaje. Después de usar el mensaje, el clasificador es elegido para pasar a la cámara de compensación y

remunerar de manera equitativa a los clasificadores candidatos a ser activados ofreciendo algo de su fuerza. Esta recompensa puede modificar la cadena de clasificadores, de manera similar a la que los bomberos pasaban las cubetas, buscando la forma más rápida de apagar un incendio. En este caso, lo que se desea es encontrar los mejores clasificadores que cumplan con la regla.

Cuando el número de los caminos de activación crecen exponencialmente, la técnica del algoritmo del bombero deja de depender en retrospectiva del análisis pero sigue trabajando localmente durante la operación y se sigue enfocando en la fortaleza de la transacción entre los pasos, con el mejor clasificador de cada paso seleccionado estadísticamente en cada momento. El principio del algoritmo del bombero sería apropiado para los sistemas, como los animales y robots autónomos en interacciones en marcha con ambientes inciertos, donde el almacenamiento y el análisis de la falta de experiencia es caro o impráctico [17].

El algoritmo del bombero implementa una economía en la cual la información fluye hacia enfrente con inferencia y la fortaleza fluye hacia atrás en un patrón de distribución por goteo. Esto puede ser considerado como información económica donde el derecho de comerciar es comprado y vendido por los clasificadores. Los clasificadores forman una cadena de intermediarios desde el fabricante (ambiente) al consumidor de información (efectores). Dos funciones principales son realizadas por dicho algoritmo:

- Asignar crédito a cada individuo
- Resolver el problema en conflicto

Para implementar el algoritmo, cada clasificador es asignado a una cualidad llamada fuerza. Este algoritmo ajusta la fuerza para reflejar en general la utilidad para el sistema y la fortaleza es usada como base de la competencia [17].

El algoritmo del bombero simula una cadena de personas que pasa cubetas de agua a partir de un punto particular a fin de extinguir un incendio. El agua simboliza la recompensa y extinguir el incendio corresponde a resolver el problema. Los miembros de la cadena que lleguen a extinguir el incendio rápido obtienen una buena recompensa, es decir, la cadena más corta que cumpla con este trabajo es la que obtiene mejor recompensa. Las personas se convierten en los miembros del sistema clasificador y el agua es una recompensa que va de miembro en miembro a cada generación. Cada participante tendrá una recompensa cuando el último de la cadena entregue el agua al ambiente. Para ser el primero de la cadena, debe ser capaz de leer el mensaje "fuego", para ser parte de la cadena de solución, debe interpretar el nuevo mensaje que el último miembro de la cadena le transmite, nadie puede interpretar el mensaje emitido después de él, sin embargo, si el último mensaje enviado está mal, no pasará el agua y toda la cadena corre el riesgo de desaparecer. Por el contrario, si hay agua, la cadena puede subsistir. La recompensa es retro propagada a los

individuos. Un individuo del sistema de clasificadores que comprende la señal hace una oferta para solucionar el problema.

2.6.2. Aprendizaje rápido

Introducido por Watkins, el aprendizaje rápido, conocido también como *Q Learning*, es un método de aprendizaje por refuerzo que permite hacer evolucionar a un sistema de aprendizaje para evaluar el rendimiento de todas las acciones posibles al ser cruzada con la mejor. Visto desde una manera formal [40]:

- Sea **E** todos los estados posibles del sistema.
- Sea **A** todas las acciones posibles.
- Sea β y γ los factores de aprendizaje.
- Sea **R** la recompensa asociada a un estado **e** $\in$ **E** y a una acción **a** $\in$ **A**.
- Sea **e'** $\in$ **E** el estado resultante de la aplicación de la pareja **(e, a)** y **a'** $\in$ **A** la mejor acción del estado **e'**.
- Sea **Q** el resultado de la ecuación. Se utilizó dicha nomenclatura para hacer referencia al algoritmo por su nombre en inglés: *Q Learning*

Tenemos la ecuación 2.1.

$$Q(e,a) \leftarrow (1-\beta).Q(e,a) + \beta x(R + \gamma.Q(e',a')) \tag{2.1}$$

Igualmente obtenemos una tabla de predicciones **P** $\in$ [0, 1] que para cada tripleta (**e, e', a**) indica la probabilidad de elegir **a** como acción del estado **e** dividido entre las predicciones de los estados **e"** y las acciones **a"** posibles, como se puede ver en la ecuación 2.2.

$$P(e,e',a) = Q(e,a) / \sum_{e'' \in E}^{a'' \in A} Q(e'',a'') \tag{2.2}$$

Para adaptar el aprendizaje rápido con los sistemas de clasificadores:

- **E** son todos los estados. Se convierten en **C** todas las condiciones posibles, y c_j es el primer clasificador **c** $\in$ **C**.
- β y γ, los factores de aprendizaje son reemplazados por α
- F_t es la función que calcula el valor de la fuerza del clasificador **i** al instante **t**

Esto se puede observar en la ecuación 2.3.

$$F_{t+1}(c_i, a_i) = (1 - \alpha).F_t(c_i, a_i) + \alpha.(R + F_{t+1}(c_j, a_j)) \tag{2.3}$$

En este caso, la activación máxima del clasificador i conduce a la activación del clasificador j para el siguiente paso para las tripletas (e, e', a), del caso general (e_j, e'_j, a_j), de los sistemas de clasificadores, donde i y j son únicas. El aprendizaje rápido no funciona idealmente en un ambiente markoviano [24] aunque ofrece un algoritmo relativamente caro pero muy eficiente.

Esta es la razón por la que la mayoría de los juegos de ajedrez usan este tipo de algoritmo para determinar qué tirada sigue. Sólo los tiempos de cálculo permiten tener una profundidad suficiente para su oponente. Aunque el aprendizaje rápido ofrece muy buenos resultados en entornos markovianos, es necesario ayudar al sistema a conocer el futuro con el fin de anticipar la situación no determinista en el que se encuentra pues la debilidad de este sistema es la función de maximización de la recompensa que crea la incapacidad de disponer de una estrategia a mediano y largo plazo con cualquier tipo de problema.

Una de sus desventajas es que la solución hallada no siempre es la óptima aunque tiende a estar muy cerca de la misma. Para que el aprendizaje sea más efectivo, el algoritmo debe evaluar la mayor cantidad de estados posibles [31]. Su algoritmo se muestra en el diagrama de flujo de la Figura 2.6:

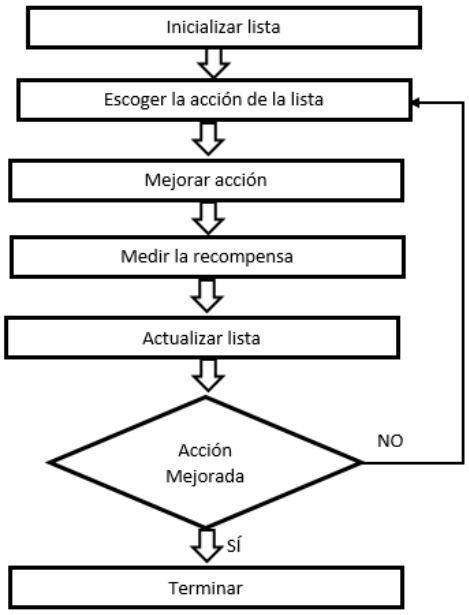

Figura 2.6: Algoritmo de aprendizaje rápido

Capítulo 3

Estado del arte

Este capítulo consta de la base de conocimientos utilizados para el desarrollo del presente trabajo. En éste podemos encontrar el GXCS, clasificador al que se le modifica su mecanismo de aprendizaje, también se habla de los dos modelos de evaluación que se han utilizado desde el primer clasificador creado y se utilizan actualmente para evaluar los clasificadores de nueva creación. Se incluye una descripción del aprendizaje hebbiano y algunas de sus vertientes. Este aprendizaje se incorpora al GXCS con el objeto de lograr que su comportamiento se acerque al del comportamiento humano al emular en él la sinapsis neuronal.

La regla básica de Hebb y sus vertientes han sido probadas en redes neuronales con resultados aceptables. Es por eso que se decidió probar las vertientes estables en el ámbito de los LCS, ya que el aprendizaje hebbiano no había sido evaluado de esa manera [14].

3.1. GXCS

El GXCS, *Generic eXtended Classifier System* o sistema clasificador genérico extendido, basado en el XCS y realizado por el Dr. Ramos, acepta un rango más amplio de entradas que los clasificadores mencionados en el capítulo anterior: binaria, real, booleana e intervalos de enteros o reales. Cuenta con un conjunto completo de clasificadores, por lo tanto, tiene la habilidad de producir comportamientos complejos, tanto individuales como colectivos, en ambientes dinámicos (markovianos y no markovianos), como son los ambientes de prueba woods y el multiplexor booleano, que serán explicados más adelante en el presente capítulo.

La principal ventaja del GXCS es que está basado en la pertinencia de la predicción de los clasificadores, para lo cual cuenta con un arreglo de predicciones. Esto permite construir un conjunto completo y compacto de todos los **X*A**→ **P**, es decir, todas las correspondencias del estado del sistema **X**, las acciones inducidas **A** y las recompensas predichas del ambiente **P** [33].

Su desventaja es la persistencia de reglas sub generalizadas, por lo que la predicción sigue considerándose muy general. El sistema puede tener una respuesta aceptable con nuevas situaciones en un ambiente dinámico y fijo, pero si se agregan nuevas reglas o elementos que realizan una importante modificación en el mundo virtual, el GXCS se llega a desestabilizar. Cuando este tipo de situaciones se presentan, las condiciones del GXCS no son suficientes para generar situaciones dentro de ambientes markovianos, por lo tanto, no logra converger en un nuevo comportamiento y no puede determinar cual es el curso de acciones a tomar. Aunque una condición muy descriptiva no interfiere con el proceso de convergencia.

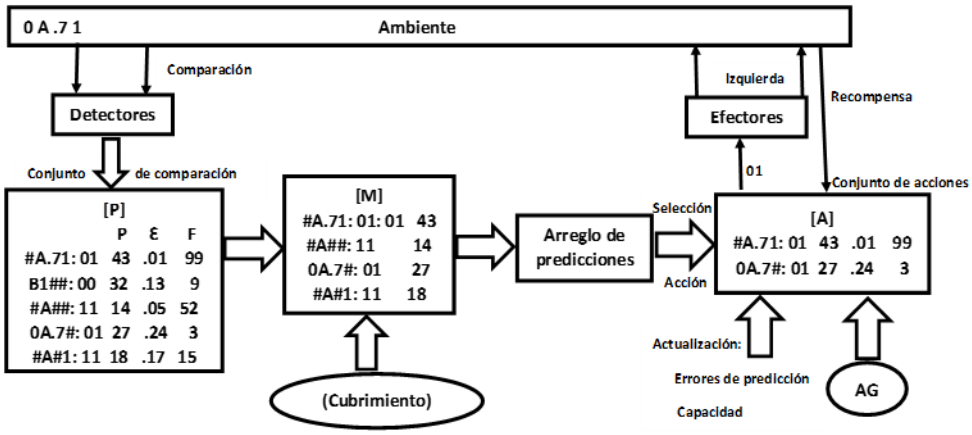

Figura 3.1: Arquitectura de GXCS, basado en [33]

En la Figura 3.1 muestra la arquitectura del GXCS donde se puede ver que el ambiente cuenta con reglas formadas de caracteres alfanúmericos. Como ejemplo se obtiene la regla 0 A .7 1, donde se puede observar el uso de dichos caracteres, los cuales son tomados del ambiente por los detectores, que envían la regla al conjunto de población y entonces se realiza la evaluación con el objeto de verificar cuáles reglas son similares. Si encuentra similtud en algunas, las pasa al conjunto **M**, donde se realiza el **cubrimiento** ó ***covering***, que es el proceso de crear reglas a partir de las existentes cuando ya no hay más reglas qué comparar en el conjunto **M**. A partir del conjunto de coincidencias **M** se crea el arreglo de predicciones, donde se va a calcular la predicción de que la regla obtenga una acción con mayor recompensa que las otras con respecto a su acción asociada. El conjunto de reglas con una buena predicción pasa al conjunto de acciones, donde se actualiza el valor del error de la predicción y donde puede dar a lugar el uso del AG, cuando sea necesario crear otras reglas a partir de dos reglas padre. Se toman las reglas de este conjunto para enviarlas al ambiente ya con una acción selecionada la cual es recompensada o castigada. El

proceso es muy parecido a los sistemas clasificadores descritos en los capítulos anteriores, a excepción de que en este caso se modificó el modelo para que el cálculo de la recompensa y del error acepten entradas alfanuméricas, así como también el arreglo de predicciones y el algoritmo genético se adaptan a los tipos de entrada actuales.

El diagrama de flujo del algoritmo de aprendizaje utilizado en el GXCS, se muestra en la Figura 3.2. En él se pueden apreciar los diferentes módulos que se realizan durante el aprendizaje rápido. El primero es crear la población inicial con valores pseudoaleatorios y tomar los mensajes del ambiente. Después se verifican cuales de las reglas de la población tienen coincidencias con la que entra del ambiente. Una vez realizado esto, se inicializan los estados de las acciones y se calcula la aptitud por medio de ecuaciones de aprendizaje rápido, las cuales se describen en el presente capítulo. Se toma una regla del conjunto de coincidencias y se le asigna una acción, se verifica el resultado de esa acción, es decir, la aptitud y exactitud que se le asignó, si ambos valores son los mejores encontrados por medio de la ecuación de aprendizaje rápido, entonces se le asigna una recompensa o castigo. Si la recompensa es la esperada entonces se termina la ejecución de los pasos, en caso contrario se repite el flujo hasta que se encuentre una regla que tenga una recompensa que genere una acción esperada.

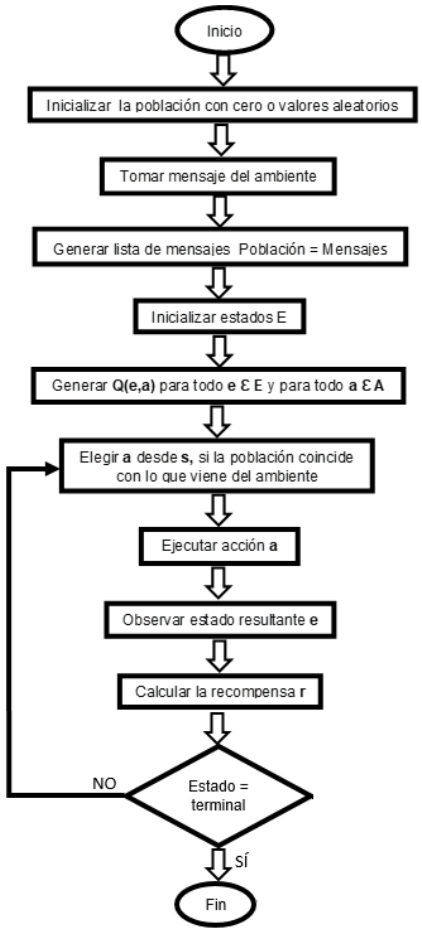

Figura 3.2: Diagrama de flujo del aprendizaje rápido en el clasificador

El GXCS carece de la emulación del comportamiento humano. Por ello, en este trabajo de investigación se propuso incorporar esta característica a través del aprendizaje hebbiano, que será explicado en la siguiente sección.

3.2. Aprendizaje hebbiano

Donald O. Hebb fue considerado el padre de la psicobiología cognoscitiva debido a los estudios que realizó sobre el funcionamiento neuronal del cerebro humano [9]. En 1949, a partir de estos estudios enunció el postulado que lleva su nombre, el cual ha sido aplicado en redes neuronales con prealimentación, circuitos integrados VLSI, reconocimiento de voz, reconocimiento de imágenes y control de motores, ecuaciones, grafos, optimización, etc [34].

El postulado de Hebb dice que: " Cuando un axón de una celda **A** está suficientemente cerca como para conseguir excitar una celda **B** y persistentemente toma parte en su activación, algún proceso de crecimiento tiene lugar en una o ambas celdas, de tal forma que la eficiencia de **A**, cuando la celda a activar es **B**, aumenta"[9].

Este postulado nos dice que los pesos de las conexiones sinápticas se ajustan por medio de la correlación entre los valores de activación de las dos neuronas conectadas. Las neuronas son activadas por los pensamientos, las decisiones y las experiencias a partir de la estimulación externa, entonces el cerebro verifica si ya ha habido un estímulo similar o no y toma las características de los datos de entrada generando una asociación entre las entradas y las salidas del sistema. La conexión de las neuronas generan un peso sináptico. Si un peso contribuye a la activación de la neurona, el peso incrementa, si se inhibe, el peso se decrementa por lo que la fuerza de la sinapsis en el cerebro cambia proporcionalmente según el disparo de las neuronas de entrada y salida. Este proceso remodela las viejas redes hebbianas, (las que ya no son agradables) o crea nuevas redes (que sí sean agradables) generando la neuroplasticidad. La neurona de entrada es conocida como presináptica y la de salida como post sináptica [14]. En la Figura 3.3 se observa el esquema de la generación de la neuroplasticidad, descrito anteriormente.

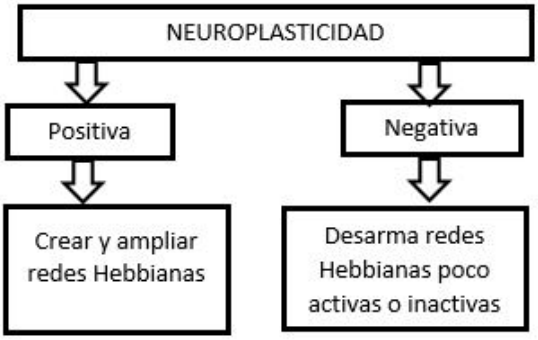

Figura 3.3: Aprendizaje hebbiano, basado en [14]

El aprendizaje hebbiano está regido por las reglas de plasticidad sináptica, las cuales son producto de las investigaciones hechas por Hebb a partir de su postulado.

3.2.1. Reglas de plasticidad sináptica

Una regla de plasticidad sináptica puede verse como un conjunto de ecuaciones diferenciales que describen el promedio de pruebas de los pesos sinápticos, la

función de las actividades presinápticas, possinápticas y otros factores. En algunos modelos de plasticidad, la actividad de cada neurona se describe como una variable continua que pueden tomar valores positivos y negativos. El valor del peso sináptico se incrementa si los valores de entrada y los de salida son positivos o negativos, y se decrementa si alguno de ellos es positivo y el otro negativo. Una actividad variable que toma ambos valores, tanto el positivo como el negativo, puede ser interpretada como la diferencia entre el promedio de pruebas y una tasa de fondo modificada, o como que está entre las tasas de disparo de dos neuronas que son tratadas como una unidad [9].

La regla de plasticidad creada por Donald Hebb es conocida como regla básica de Hebb, a partir de ella se crearon varias vertientes. Las más conocidas y utilizadas por la comunidad científica son: la regla básica, de covarianza, con degradación [10], BCM, normalización sináptica por susbtracción y multiplicativa, instar, Kohonen [42] [10] y outstar. Algunas de ellas no fueron descritas en el presente trabajo debido a que la literatura científica reporta que son inestables cuando se tienen muchos parámetros o elementos, dichas reglas pueden ser revisadas más a detalle en [4], [11], [10] y [39].

En la Tabla (3.1) se muestra la comparación entre las características de las vertientes de la regla de Hebb que se encontraron en la literatura. En ella se muestran las características principales tanto de las vertientes inestables como de las estables y se explica a grandes rasgos qué es lo que causa la inestabilidad en las reglas en las que se genera dicha inestabilidad al introducirse pesos grandes. El concepto de pesos será descrito en las siguientes secciones, así como se detallará las vertientes utilizadas para el presente trabajo.

Reglas de Hebb			
Vertiente	Aprendizaje	Ventajas	Desventajas
Básica	No supervisado	Buena aproximación con pesos muy pequeños.	Inestable con pesos muy grandes. No decrementa los pesos.
Covarianza	No supervisado	Buena aproximación con pesos muy pequeños.	Inestable con pesos muy grandes No decrementa los pesos
BCM	No supervisado	Buena aproximación con θ variable.	θ con valor constante se vuelve inestable Problema en generación de sinapsis.
Normalización por sustracción	No supervisado	Evita que se disparen los pesos.	Se vuelve inestable en saturación de pesos.
Normalización multiplicativa (R. de Oja)	Supervisado y no supervisado	Restricción dinámica. Estabilidad con pesos grandes. Olvida pesos pequeños. Introduce tasa de degradación.	Sin restricción en tasa de degradación se vuelve inestable.
Con degradación	Supervisado y no supervisado	Olvida rápidamente los pesos viejos.	Requiere de estímulos para que la asociación no decaiga
Kohonen	No supervisado	Buena aproximación con pocas reglas.	Sólo los pesos del ganador son ajustados. Algunas neuronas no se entrenan y no ganan
INSTAR	No supervisado	Evita la degradación de pesos sin estímulos. El valor de pesos está en las entradas.	Sin restricción en entradas se vuelve inestable.
OUTSTAR	Supervisado	Evita degradación de pesos sin estímulos. La degradación de pesos es proporcional a la entrada.	Sin restricción en salidas se vuelve inestable.

Tabla 3.1: Vertientes hebbianas

3.2.1.1. Normalización multiplicativa

La regla multiplicativa, también conocida como regla de Oja, fue creada en 1982 por Erkki Oja para aprendizaje no supervisado. Esta introduce la restricción dinámicamente donde se suma el cuadrado de los pesos sinápticos.

El uso de la normalización tiene el efecto de introducir competición entre las sinapsis de la neurona sobre recursos limitados, lo que es esencial para la estabilización. Desde un punto de vista matemático, la ecuación (3.1) describe una forma conveniente de normalización [35].

$$w_{ij}(q) = w_{ij}(q-1) + \alpha.y_i(q).[x_j(q) - y_i(q).w_{ij}(q-1)] \tag{3.1}$$

donde:
q es el par entrada/salida al que se le da un peso.
x contiene los valores de entrada.
y contiene los valores de salida.
α es la tasa de aprendizaje con valores entre 0 y 1.
y_i es la salida de la neurona i.
x_j es la entrada de la neurona j.
w_{ij} es la conexión entre el elemento de i y el elemento de salida j, es decir el vector de pesos.

Si el valor de α se aproxima a 1, la red tendrá pesos pequeños, pero recordará poco de lo que aprendió en ciclos previos. El término $\alpha y(n)x_i(n)$ representa las modificaciones hebbianas al peso sináptico w_i.

$-y_i(q)w_{ij}(q-1)$ es la responsable de la estabilización y está relacionado con el factor de olvido o tasa de degradación que se utiliza generalmente en las reglas de aprendizaje. En este caso, la tasa de degradación toma un valor mayor ante una respuesta más fuerte. La elección de la tasa de degradación tiene un gran efecto sobre la rapidez de convergencia del algoritmo, en particular, no puede ser demasiado grande, pues de otro modo, el algoritmo se haría inestable. En la práctica, una buena estrategia es utilizar al principio un valor de α relativamente grande para asegurar una convergencia inicial e ir haciendo α gradualmente pequeño hasta lograr la precisión deseada [35].

La magnitud de los pesos se restringe entre 0, que corresponde a no tener peso, y 1, que corresponde a la neurona de entrada con cualquier peso. Esto se realiza para que el vector de pesos tenga longitud 1. [30].

Existe otra vertiente que es estable y se describe en la siguiente sección.

3.2.1.2. Regla instar

La regla de instar es una mejora de la regla de Hebb con degradación [10] que evita que los pesos se vayan degradando cuando no se encuentra un estímulo. La regla de Hebb con degradación puede ser consultada en [10]. Instar es considerada como una neurona con un vector de entradas y es la red más simple capaz de tener un reconocimiento de patrones. Instar estará activa siempre que el producto interno del vector de pesos y la entrada sea mayor o igual a -b, donde b es considerado como la "tendencia" y debe tener un valor entre 0 y 1.

Esta regla fué diseñada por Grossberg en 1969, pertenece al aprendizaje no supervisado y trabaja de manera local. A diferencia de la regla de Hebb con degradación, ésta sólo permite la degradación del peso cuando la neurona instar está activa. Es decir, cuando $a \neq 0$. Para lograr esto se agrega un término que evita el olvido, el cual es proporcional a $y_i(q)$ [42]. Esto se aprecia en la ecuación (3.2).

$$w_{ij}(q) = w_{ij}(q-1) + \alpha.y_i(q).(x(q) - w_{ij}(q-1)) \tag{3.2}$$

La descripción de los parámetros fueron descritos en la sección anterior, con excepción de $x(q)$, que contiene todos los valores del vector de salidas.

α toma valores entre 0 y 1 para asegurar que los valores de los pesos se disparen, si α se aproxima a cero se vuelve inestable y, por el contrario, si se aproxima a 1 la red empieza a olvidar rápidamente las entradas viejas y sólo recordará los patrones más recientes permitiendo tener un límite en el crecimiento de la matriz de pesos. El valor máximo en el peso w_{ij}^{max} está determinado por α, lo cual sucede cuando y_i y x_j tienen valores $= 1$ para todos los q, lo cual maximiza el aprendizaje.

Cuando la neurona instar está activa, el vector de pesos se acerca al vector de entrada entre el vector de pesos viejo y el vector de entradas. La distancia entre los vectores depende de α. Si $\alpha = 0$, el nuevo vector de pesos es igual al vector de pesos viejo, es decir, no hubo movimientos. Si $\alpha = 1$, el vector de pesos es igual al vector de entradas, es decir, el movimiento obtuvo el valor máximo. Si $\alpha = 0.5$, el nuevo vector de pesos será la mitad del viejo vector y la otra mitad el vector de entradas. Una característica muy útil de la regla instar es que si el vector de entradas es normalizado entonces w_i será también normalizado una vez que éste ha aprendido un vector particular q.

3.2.1.3. Regla outstar

Este tipo de red tiene una entrada escalar y un vector de salida. Esto puede mejorar el patrón de llamada asociando un estímulo con el vector de respuesta. La regla outstar, al contrario de la instar, hace que la degradación del peso sea

proporcional a la entrada de la red x_j. La ecuación (3.3) muestra la regla de outstar [10]:

$$w_{ij}(q) = w_{ij}(q-1) + \alpha.(y(q) - w_{ij}(q-1)).x_j(q) \tag{3.3}$$

$y(q)$ contiene todos los valores del vector de salidas.
La regla outstar fue creada por Steven Grossberg en 1969 como complemento de la regla instar y pertenece al aprendizaje supervisado. En ella, el aprendizaje ocurre siempre que $x_j \neq 0$. Cuando la regla ha aprendido algo, la columna w_{ij} se acerca al vector de salida, es decir, cuando algunos pesos son iguales con los de la salida deseada [7].

Las señales de la neurona de entrada generan pesos para aprender y volver a llamar patrones arbitrarios a través de la neurona de salida. Permite que el peso cambie sólo si la neurona j está activa. Los demás pesos de i no cambian cuando la categoria j es seleccionada, entonces el aprendizaje inicial se mantiene.

Una vez descrito el GXCS y las vertientes de la regla de plasticidad hebbiana se explican los dos modelos de evaluación que la comunidad científica que trabaja con los LCS utiliza y que, por lo tanto, se utilizarán en el desarrollo del presente clasificador.

3.3. Modelos de evaluación para LCS

Es importante tener ambientes virtuales donde se puedan realizar las pruebas de los sistemas clasificadores para así poder contar con resultados que nos permitan medir el alcance de cada uno [12].

Existen dos tipos de ambiente sugeridos y utilizados por la comunidad de los sistemas clasificadores: los ambientes estáticos y los dinámicos.

Se considera un ambiente estático a aquel que no cambia en el transcurso del tiempo, como por ejemplo, que un animal encuentre su comida en la salida de un laberinto, donde el animal debe localizar el camino más corto al mismo tiempo que sortea obstáculos y así llegar a la comida puesta en alguna casilla del laberinto aleatoriamente. Visto desde el lado de los LCS, la tarea consiste en aprender todas las acciones posibles para cada señal de entrada y poder calificar dicha señal por medio de su aptitud, tomando en cuenta los valores de los obstáculos y también de las celdas libres.

Por el contrario, un ambiente dinámico es un sistema que evoluciona conforme el tiempo transcurre. Un ejemplo de este ambiente es el control del tráfico aéreo, donde los vuelos pueden ser reprogramados o cancelados, entre otras cosas, que suceden de manera dinámica. Por lo tanto, para un clasificador, se debe

obtener una buena solución tomando en cuenta que las variables del ambiente cambian en cada instante t.

Para la evaluación de los sistemas clasificadores se pueden utilizar los ambientes woods y el multiplexor booleano para los ambientes estático y dinámico, respectivamente. Estos ambientes ofrecen grados de dificultad relativamente controlables y facilitan la comparación de los diferentes modelos, como se comenta en [12].

3.3.1. Ambiente woods

El ambiente woods es un mundo en 2D representado por una rejilla propuesto por Wilson para hacer pruebas confiables sobre los sistemas clasificadores. Este ambiente puede ser modelado como un autómata, aunque no se utiliza mucho esta representación debido a que el número de posibilidades se vuelve exponencial en función del tamaño del ambiente, por lo que resultaría difícil hacer un cálculo de la solución. Por el contrario, este tipo de ambientes es muy utilizado en los sistemas evolucionistas de aprendizaje. Un ejemplo de su uso son las pruebas realizadas con el GXCS, el cual se estabiliza con 280 reglas con una tendencia a aumentar el número de reglas [12].

En este ambiente, los estados son caracterizados por atributos que representan muchas de las propiedades perceptibles del ambiente y está compuesto de celdas que pueden marcarse las celdas con valores que representen los espacios donde el agente puede moverse, así como representar obstáculos o comida. Los obstáculos impiden el paso del agente y la comida es la meta a la que éste debe llegar. El agente puede moverse hacia cualquier celda vacía contigua a el, donde hace una evaluación de las celdas que se encuentran a su alrededor para verificar cual de ellas está vacía. A partir de ello se crea una situación, que puede verse como un vector de muchos valores discretos entre ceros y unos, donde el cero indica la ausencia de obstáculos y el 1 es presencia de estos. Cada valor es asociado con un atributo del ambiente, que pueden contener la dirección hacia donde se puede mover el agente. Cuando el agente llega a la casilla que contiene la comida, recibe una retribución y es reemplazado aleatoriamente sobre la rejilla.

Los mundos woods se pueden clasificar en dos: markoviano y no markoviano. Este último puede ser estático o dinámico. Un ambiente es markoviano si por cada mensaje dado en el ambiente hay una acción que conduce al éxito. Un ambiente no markoviano puede generar situaciones aparentemente idénticas pero la acción exitosa es diferente, es decir, no determinista, el cual requiere mucho tiempo para poder olvidar una respuesta incorrecta que, a la vez puede ser considerada buena para otro tipo de ambiente. Los ambientes no markovianos aplicados a los sistemas sin memoria son parte de problemas NP completos, por lo que Wilson demostró que era posible transformar cualquier ambiente no markoviano en un ambiente markoviano con el objeto de aumentar las posibilidades

sensoriales del sistema para agregar las partes condición y acción en situaciones críticas.

3.3.2. Ambiente multiplexor booleano

Los multiplexores booleanos son utilizados en los sistemas de aprendizaje como los sistemas clasificadores, algoritmos genéticos y redes neuronales. El multiplexor booleano es un problema de referencia que se utiliza en los ambientes markovianos. Un multiplexor permite seleccionar de una lista de entradas, datos que serán indexados para otras entradas. El multiplexor es una caja negra que puede realizar operaciones booleanas con compuertas lógicas utilizando los símbolos **.** + y **'** que significan **Y, O** y **NO** lógicos, respectivamente, donde se van combinando los valores de las entradas para obtener la respuesta que se considera como mejor opción de salida. El objetivo del sistema de aprendizaje es calcular la salida del multiplexor, que nos puede generar un cero o un uno en función de las entradas del sistema. Para calcular el número de operaciones a realizar se tiene un valor llamado **k**, que representa el número de entradas en el sistema, otro valor **n** que es una entrada elegida y la fórmula: $\mathbf{k} = \mathbf{n} + 2^n$ que es verificada en este ambiente. Una vez obtenida la respuesta, ésta se envía al sistema clasificador como salida al ambiente [12].

3.3.3. Resumen

Como se pudo ver dentro de esta sección, los clasificadores XCS y GXCS utilizan el algoritmo de aprendizaje rápido, y un algoritmo genético para la adaptación de las reglas al ambiente, por lo que se sabe que el GXCS carece de emulación del comportamiento humano. Es por ello que nuestra propuesta incluye el estudio de las vertientes de la reglas de Hebb, y la implementación de las que la literatura indica que son estables. Estas vertientes tienen su base en neurociencias y en psicología, por lo que permiten emular el aprendizaje humano a partir de la sinapsis neuronal. De esta unión del GXCS con los conceptos hebbianos surge nuestra propuesta: el GXCS-H.

Capítulo 4

Sistema clasificador de aprendizaje GXCS-H

Dentro de este capítulo se plasma el resultado de las investigaciones realizadas. Se implementó el desarrollo del LCS que involucra el aprendizaje hebbiano. Se inicia con una descripción de la nueva arquitectura y los detalles del desarrollo de ésta por medio de diagramas de flujo y el diagrama propio de la arquitectura, así como un diagrama y descripción de los archivos fuente principales que se implementaron o modificaron, según sea el caso.

4.1. Propuesta: Aprendizaje hebbiano en un GXCS

A partir de las investigaciones realizadas sobre los temas de LCS y aprendizaje hebbiano se decidió poner a prueba el comportamiento de dicho aprendizaje sobre un LCS. Se eligió el GXCS debido a que éste tiene su base en los clasificadores ZCS y XCS de los cuales se habló en el capítulo anterior y además permite generar entradas de diversos tipos de datos, como pueden ser letras, reales, enteros, etc [33].

Como se comentó en los capítulos anteriores, el aprendizaje hebbiano ha sido probado sobre redes neuronales [14], en ellas ha recibido una buena aceptación, sin embargo, las redes neuronales requieren ser entrenadas para poder generar un resultado y no son tan flexibles en ambientes cambiantes, aunado a esto, las redes neuronales tampoco generan comportamientos cercanos al comportamiento humano. Por el contrario, la capacidad principal de un LCS es que es reactivo y trabaja dentro de ambientes estáticos y dinámicos.

Actualmente aún se utilizan diferentes tipos de aprendizaje dentro de los LCS basados en algoritmos que trabajan con recompensas y castigos sobre las reglas, entre otras cosas, pero estos no emulan un comportamiento humano. Dado que el aprendizaje hebbiano lo emula a partir de la ecuación tomada de la

sinapsis neuronal, permite generar una aproximación del comportamiento humano. Esto es posible afirmarlo debido a que la base del aprendizaje que cada ser humano tiene es la conexión neuronal que genera al realizar sus actividades, tomar decisiones y memorizarlas. Justo cuando se realizan esos procesos, las neuronas se conectan o refuerzan su conexión.

La adaptación del sistema al medio ambiente esta dado por un algoritmo genético. La arquitectura fue modificada para incorporar el aprendizaje hebbiano. La razón principal de esto es poder dotar al agente (agentes) de un aprendizaje basado en las conexiones que tiene el cerebro humano provistas teóricamente por neurociencias permitiendo tener control sobre la cantidad de reglas que compiten para encontrar una solución sobre un problema dado. A este clasificador lo llamamos **GXCS-H** *(Generic eXtended Classifier System with Hebbian learning).*

En las secciones siguientes se explica como se implementaron las ecuaciones del aprendizaje hebbiano y su incorporación dentro de un GXCS.

4.2. Arquitectura del GXCS-H

En la Figura 4.1 se muestran las secciones de la arquitectura del GXCS en la que se realizaron cambios fundamentales para lograr el GXCS-H. Estos módulos en específico permiten al clasificador calcular un vector de pesos compuesto por condiciones del ambiente así como evaluar cada una de estas condiciones para aprender con respecto a la dinámica del ambiente, contrario a utilizar un arreglo de predicciones como en los aprendizajes tradicionales.

En la Figura 4.1 se integra un módulo en específico para los valores de x y y, que son directamente asignados a las condiciones de entrada del ambiente que están embebidas en cada una de las reglas que compiten para realizar una determinada acción. Los valores que reciben x y y pueden ser escalares o vectoriales de acuerdo a una condición hebbiana como Oja, instar y outstar, que son reglas estables, esto es, que no generan estados no deseables en el entorno, como lo indica la literatura [4]. Dichas reglas se detallan en las siguientes secciones.

La tasa de degradación permite evaluar la permanencia o borrado de las reglas que se encuentran dentro del conjunto de acciones al controlar el número de reglas producidas por la interacción del ambiente evitando así la sobre explotación de reglas que pudieran ralentizar el sistema.

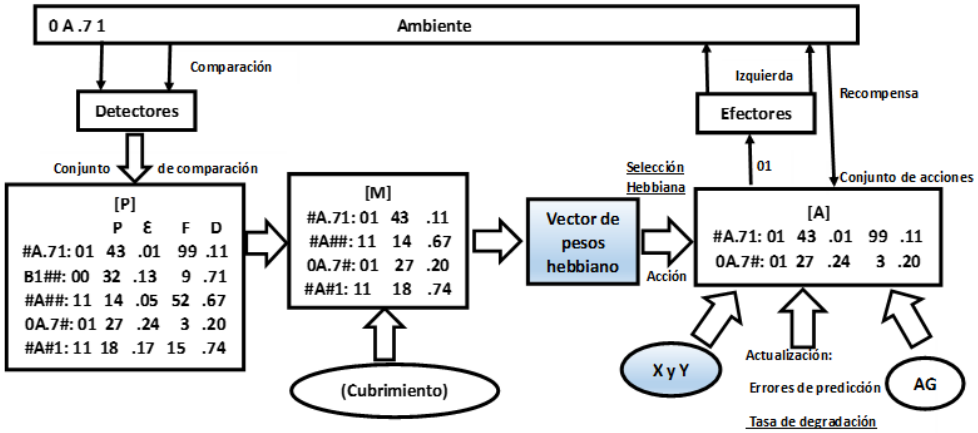

Figura 4.1: Arquitectura del GXCS-H

De igual manera construimos el diagrama de flujo para la arquitectura propuesta con el fin de identificar las partes en donde está involucrado el proceso de aprendizaje del sistema. En la Figura 4.2 se muestran con recuadros sombreados, las modificaciones hechas en el GXCS-H. En el diagrama se puede ver la inicialización del vector de pesos sinápticos, así como también la introducción de la ecuación de aprendizaje hebbiano representado con $H(e, a)$. Se puede ver que también se agrega la tasa de degradación como parámetro de borrado o conservación de alguna regla específica.

Los primeros cuatro módulos del diagrama se realizan de la misma manera que en los clasificadores anteriores, ya que en ellos se inicializan los valores de manera pseudoaleatoria, la cual fue tomada de los LCS respetando la creación de reglas de manera heurística, pues esta es la forma en que el ser humano realiza la toma de decisiones dentro de un ambiente.

Con ello se crean las primeras reglas a partir del ambiente, éstas forman parte de la población inicial. Después se toman los mensajes creados desde el ambiente para generar la población y se inicializan todos los estados. Inicialmente sólo se tienen esas dos reglas, que serán evaluadas posteriormente y se les asigna una posible acción a cada regla, la cual, en el sistema representa un valor asignado que significa algo para el ambiente. El significado se le da según el tipo de ambiente que se está emulando, por ejemplo, puede dar como resultado una acción con $valor = 01$, lo cual podría indicar la salida por la izquierda dentro de un laberinto.

El siguiente paso es elegir la acción asociada a una regla, que coincide con lo que entra en el ambiente y se ejecuta. Posteriormente se llena el vector de pesos

hebbiano con valores positivos muy cercanos a cero y se inicializan los valores de las reglas de entrada $\mathbf{x}$ y las posibles acciones $\mathbf{y}$, los cuales son propuestos por la literatura a partir de pruebas realizadas en redes neuronales [4].

Para ello, se calculan los pesos asociados de manera pseudoaleatoria sobre los dos primeros clasificadores tomando en cuenta los valores de $\mathbf{x}$ y $\mathbf{y}$ que se asignaron. Después de esto se revisa el estado resultante y se calcula la recompensa e inmediatamente después se calcula la tasa de degradación, la cual toma un valor inicial cercano a cero, valor propuesto por la literatura sobre las reglas de Hebb y probado también en redes neuronales.

Ya que se ha dado un valor a la tasa de degradación con valores pseudoaleatorios con rango entre cero y uno, éste se utiliza para evaluar la persistencia de los clasificadores en el conjunto de la población. Si la tasa de degradación es menor a 0.5 entonces se conserva, si es menor, se elimina. Este valor se crea de manera pseudoaleatoria dentro del rango de cero y uno, valor que también fue propuesto en la literatura para facilitar su cálculo y evitar que los valores en los pesos hebbianos se disparen y se vuelvan inestables [4].

El valor asignado a la tasa de degradación se realiza tomando en cuenta el rango sugerido por la literatura [4], ya que se demostró en redes neuronales que si se le da otro valor fuera del rango $(0, 1)$ la estabilidad de los pesos hebbianos se rompe. El valor específico de 0.5 fue escogido a partir de las pruebas realizadas, ya que si se escogían valores cercanos a 1, se tenían demasiadas reglas en el conjunto de población y se saturaba el GXCS-H, y si se tenían valores muy cercanos a 0 se borraban reglas que tenían valores de pesos sinápticos válidos para siguientes pruebas, o se borraban reglas que aun no habían sido probadas. Por lo que se encontró un equilibrio entre el borrado y la aceptación de reglas con una tasa de degradación de 0.5.

Estos pasos se realizan de manera continua hasta que el GXCS-H encuentra un estado final esperado, es decir, que cumpla con el objetivo que se espera como comportamiento de un agente dentro del ambiente.

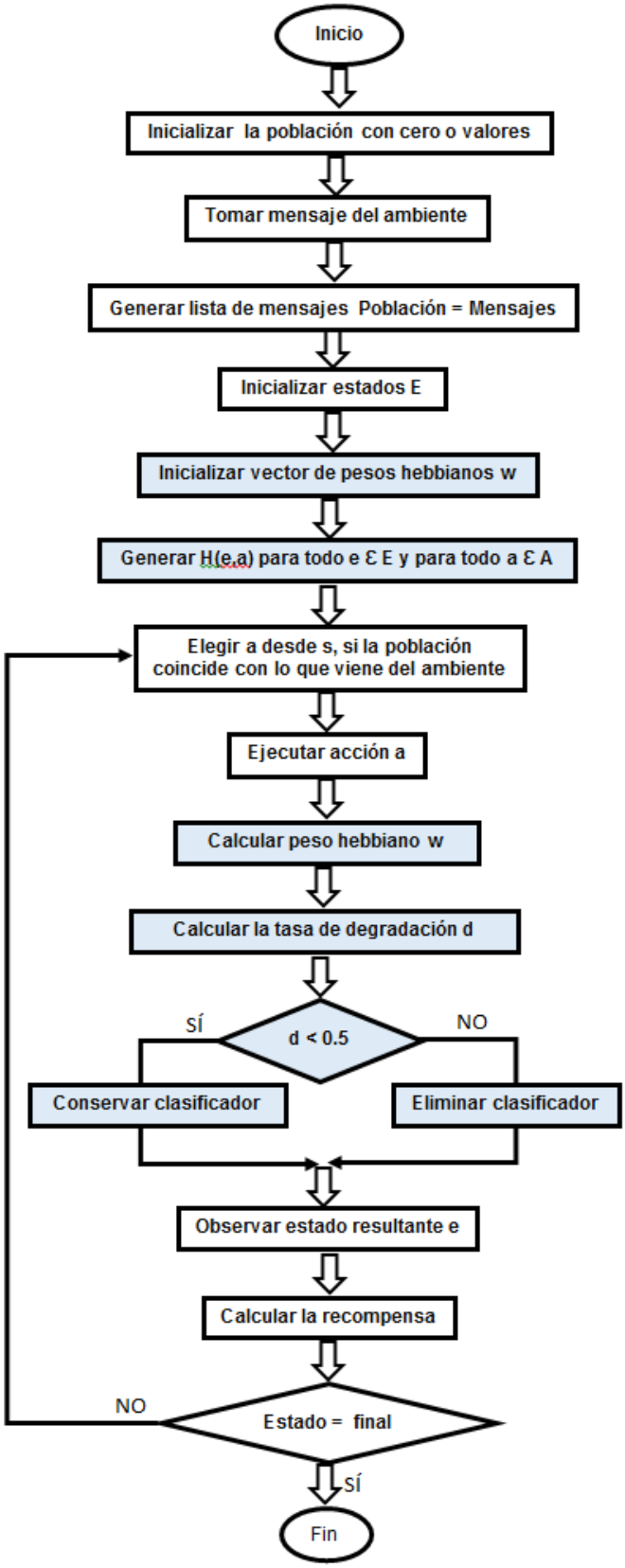

Figura 4.2: Diagrama de flujo del aprendizaje hebbiano en GXCS-H

Una de las dificultades encontradas dentro de algunas vertientes del aprendizaje hebbiano es que la variación en el número de reglas no controlada provoca rápidamente inestabilidad en los valores de los pesos cuando se tienen demasiadas reglas o pesos con valores negativos o muy grandes, por lo tanto, dichas reglas fueron descartadas en el GXCS-H. Para resolver el problema de la inestabilidad, se utilizan las reglas de normalización multiplicativa (también llamada de Oja), instar y outstar, lo que permite tener reglas lo más estables posibles [4].

4.3. Reglas hebbianas estables

Dentro de las reglas hebbianas encontradas en la literatura [4] [14] se eligieron las tres que cuentan con parámetros que estabilizan la ecuación de la regla básica de Hebb y ayudan a la convergencia dentro de los clasificadores. Estos parámetros permiten que los valores se encuentren en el rango entre 0 y 1, valores recomendados en la misma literatura para que las reglas no vayan a tener valores muy grandes o negativos y puedan ser manipulables en las ecuaciones y sobretodo, puedan ser utilizados y leídos para generar una salida esperada. Las reglas estables son:

- Regla de normalización multiplicativa u Oja.
- Regla de instar.
- Regla de outstar.

Las reglas de **instar, outstar y multiplicativa (Oja)**, contienen un parámetro de decaimiento que controla la degradación de pesos. Este parámetro evita que se eliminen reglas que son útiles en futuras condiciones del ambiente y permite eliminar aquellas que no han competido o que tienen aptitud con valor cercano a cero, evitando así que se sature la memoria. En la siguiente sección se describe cada una de ellas.

4.3.1. Regla de normalización multiplicativa

Toma la información de la sinapsis que se está modificando evitando que los pesos hebbianos crezcan indefinidamente. El producto del valor de la salida asociada a la regla y del peso hebbiano de cada regla es el responsable de la estabilización de los pesos, este producto se conoce como factor de olvido o tasa de degradación y permite que sólo se eliminen las reglas que no han sido utilizadas o que tienen valores muy bajos, cercanos a cero o que son cero [14].

En la Figura 4.3 se puede ver un fragmento de la arquitectura del GXCS-H donde se aprecia el vector de pesos que se asignan a las reglas que estarán en el conjunto de acciones y que son tomadas para re-calcular los valores de entradas y salidas de cada una de ellas. El valor asociado a la regla que entra del ambiente

se almacena en un escalar **x**. La salida **y** es un valor escalar que toma su valor de manera aleatoria. Ambos valores fluctúan entre cero y uno, rango que se explico en la sección de estado del arte. La tasa de degradación **decay** es también calculada con valores aleatorios que se encuentren entre cero y uno y se asigna a un elemento del conjunto **A** de reglas y acciones. En caso necesario, se invoca el algoritmo genético que utilizan los LCS, incluyendo esta versión. Una vez que se obtuvieron los valores de los pesos **w**, la tasa de degradación **decay**, las entradas **x** y las salidas **y**, se calcula el valor del siguiente peso con la fórmula de la normalización multiplicativa (o de Oja), como se puede apreciar en la Figura 4.3.

La tasa de degradación **decay** es calculada a partir de un fragmento de la ecuación de normalización (**decay** = **y.w**[**0**]), como se mencionó en el estado del arte.

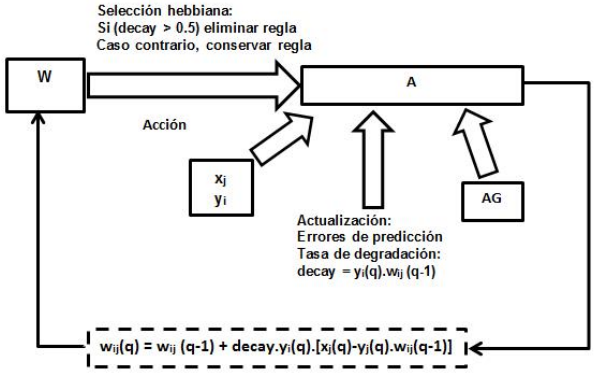

Figura 4.3: Arquitectura en normalización

4.3.2. Regla de instar

Evita que los pesos se degraden cuando no hay estímulo suficiente al calcular los valores de las entradas y las salidas. Permite la degradación sólo si la regla está activa, es decir, que la salida tenga un valor mayor a 0.5. Los valores de peso son aprendidos al mismo tiempo que se van degradando los valores viejos de las entradas y las salidas así como del peso hebbiano, todos ellos asignados a cada conjunto de reglas y acciones dentro del clasificador.

El vector de pesos tiene valores similares a los del vector de entradas cuando la regla tiene asociada un valor en la aptitud mayor a la media. En este caso, si la tasa de degradación es igual a cero el peso actual es igual al peso nuevo asignado a la regla, si la tasa de degradación obtiene el valor de uno, entonces el peso hebbiano tiene su valor máximo, si la tasa de degradación tiene el valor de 0.5 entonces se le agrega el valor del vector de entradas al peso hebbiano.

En la Figura 4.4 se puede ver un fragmento de la arquitectura del GXCS-H.

El proceso es muy parecido al descrito en la sección anterior sobre la regla de Oja. Su diferencia principal reside en que el valor de y de las salidas asociadas a las acciones se toma de un conjunto de valores que pertenecen a un vector de salidas, del cual se toma la más apta.

Su segunda diferencia con respecto a la regla de Oja es que la tasa de degradación *decay* no se toma directamente de la ecuación, sino que se calcula de manera pseudoaleatoria con valores que fluctúan entre cero y uno, acorde a lo que se describió en los capítulos anteriores.

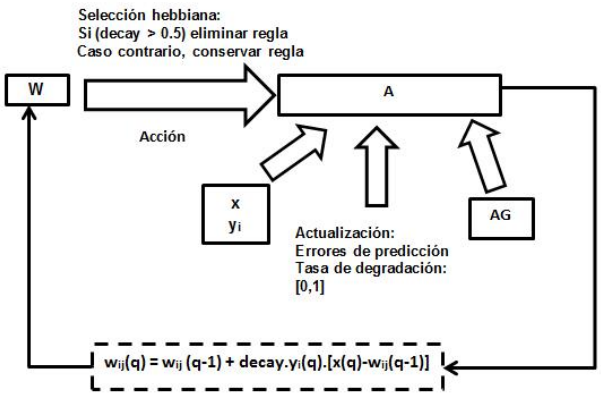

Figura 4.4: Arquitectura en instar

4.3.3. Regla de outstar

En este caso la entrada es un valor escalar y la salida un vector. El valor del peso hebbiano se acerca a los valores que se encuentran en el vector de salida cuando la regla ha aprendido algo, entonces algunos pesos adquieren el mismo valor que la salida. El peso cambia cuando la regla tienen una aptitud mayor o igual al promedio de aptitudes.

En la Figura 4.5 se puede ver un fragmento de la arquitectura del GXCS-H la cual tiene un proceso similar a las dos vertientes descritas. Su diferencia con la regla de instar reside en que el valor de x de las entradas asociadas a las reglas se toma de un conjunto de valores que pertenecen a un vector de entradas, del cual se toma la más apta. El cálculo de la tasa de degradación *decay* se realiza de la misma manera que en la regla de instar.

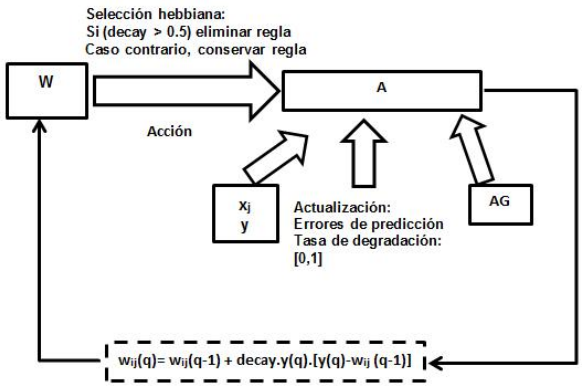

Figura 4.5: Arquitectura en outstar

4.4. Desarrollo

La implementación de las ecuaciones hebbianas en el clasificador se realiza partiendo de los conceptos genéticos interpretados de la siguiente manera: las reglas que llegan del ambiente como condición son los cromosomas, el fenotipo es la salida o acción y genotipo, la condición. Cada cromosoma está compuesto de genes, que se pueden ver como los valores que van conformando la regla. Dicha regla es asociada a una posible acción, las cuales son evaluadas en conjunto por medio de un peso sináptico calculado con las ecuaciones de la regla de normalización multiplicativa, instar y outstar, y del cálculo de una recompensa o castigo y de la aptitud asignada a la regla.

Introduciendo los conceptos del aprendizaje hebbiano en el LCS, podemos entender como estimulación externa a las percepciones del ambiente, la neurona presináptica contiene la regla percibida en el ambiente y la postsináptica la acción a realizar. Los pesos sinápticos son los valores que otorgan un valor a la sinapsis para hacerla más fuerte o más débil, según sea necesario.

El cálculo de la recompensa/castigo y de la aptitud fueron tomados del GXCS y se incluyeron dentro del GXCS-H para evaluar la regla en conjunto con el peso hebbiano asignado y los valores de las entradas y salidas de las reglas. La aptitud se utilizó en conjunto con la tasa de degradación para poder realizar la eliminación de las reglas, es decir, si se tenía una aptitud aceptable con una tasa de degradación cercana a cero se permitió re-calcular el peso hebbiano, las entradas, salidas y la misma tasa de degradación.

La constante **delta** es tomada para el cálculo de la tasa de degradación inicial, en este caso es necesaria para evitar la inestabilidad de la regla. También se agregaron las variables **x** y **y**, y el vector **w** para almacenar los valores cal-

culados de pesos sinápticos y los valores asignados a las reglas de entrada y a la salida.

Se han agregado algunas constantes conforme se han ido creando nuevas versiones. En este caso se mostrarán las necesarias en la implementación del GXCS-H. Los parámetros definidos pueden ser modificados en **XCSConstants.java** y se listan a continuación:

- **maxPopSize** Especifica el valor máximo que pueden tener los micro clasificadores en la población.
- **alpha** Se utiliza para distinguir los clasificadores con la variable de precisión o sin ella.
- **beta** Tasa de aprendizaje del clasificador.
- **gamma** Factor de descuento para los problemas de múltiples pasos, también se utiliza como tasa de degradación o decaimiento para eliminar o mantener las reglas en el aprendizaje hebbiano.
- **delta** Aptitud media en la población de clasificadores sobre la cual la aptitud de un sólo clasificador es considerada en el método de borrado de éste.
- **nu** Evaluación del clasificador según su aptitud.
- **theta_GA** Especifica el límite del algoritmo genético.
- **epsilon_0** Límite del error de la precisión de que a un clasificador se le asigne el valor de UNO. Esta variable será calculada de acuerdo al pago máximo posible en el ambiente.
- **theta_del** Límite considerado para saber si un clasificador debe ser eliminado o no. Utilizada en conjunto con la constante de la tasa de degradación.
- **pX** Probabilidad de aplicar el método de cruzamiento al aplicar el algoritmo genético.
- **pM** Probablidad de aplicar el método de mutación al aplicar el algoritmo genético.
- **P_doncare** Probabilidad de utilizar el símbolo de # en un atributo durante el proceso de cubrimiento.
- **predictionErrorReduction** Reducción del error en la predicción al crear un nuevo clasificador.
- **fitnessReduction** Reducción del valor de la aptitud al crear un nuevo clasificador.

- **theta_sub** Especifica la experiencia requerida que debe tener un clasificador para ser candidado a subgeneralización para otros clasificadores.
- **teletransportation** Número máximo de pasos ejecutados en las pruebas para problemas de múltiples pasos.
- **doGASubsumption** Bandera que indica si se ejecutó el algoritmo genético al momento de la subgeneralización.
- **doActionSetSubsumption** Bandera que indica si la subgeneralización debe ser activada para el conjunto de acciones.
- **predictionIni** Especifica el valor inicial de la predicción de un nuevo clasificador.
- **predictionErrorIni** Especifica el error en la predicción inicial de un nuevo clasificador.
- **fitnessIni** Especifica el valor de la aptitud inicial de un nuevo clasificador.
- **experience** Problemas aprendidos por el clasificador.

La constante **theta_del** no se utilizó en el GXCS-H, ya que ésta indicaba el momento en que se debía borrar las reglas. Ya que en el sistema actual el borrado se hace a partir de la regla de degradación, entonces no fue necesario incluir dicha variable.

Así como las variables más importantes generadas específicamente para los cálculos de las reglas de aprendizaje hebbiano:

- **nr** La suma de las aptitudes de los clasificadores que representan cada peso en el vector de pesos.
- **aa** Arreglo que guarda las acciones asociadas a las reglas.
- **x** Valor hebbiano asociado a las reglas de entrada.
- **y** Valor hebbiano asociado a la mejor acción registrada.
- **w** Vector de pesos hebbiano que reemplazó el arreglo de predicciones.

Las variables **x** y **y** contienen valores escalares para la regla de Oja, pero para la regla de instar **x** es tomado del vector de entradas y **y** se mantiene como escalar y, por el contrario, para la regla de Oustar, **x** se mantiene con un valor escalar mientras que **y** se toma de un vector de salidas. **w** es el vector que contiene los pesos hebbianos para las tres vertientes. Estas variables inician con valores muy cercanos a cero, valor recomendado en la literatura [14].

Esto se realiza de esta manera para garantizar que las siguientes reglas incrementen sus valores de pesos de manera gradual y suficiente para saber cuáles

son elegibles para obtener una mejor recompensa. La Tabla 4.1 muestra los valores de las constantes del clasificador propuestos como estándares dentro de los LCS por Olivier Heguy [15].

Valores constantes	
Constante	Valor
gamma	0.95
delta	0.10
maxPopSize	6800
alpha	0.1
beta	0.2
nu	5
theta_GA	25
epsilon_0	10
pX	0.8
pM	0.04
P_dontcare	0.2
predictionErrorReduction	0.25
fitnessReduction	0.1
teletransportation	50
predictionIni	10.0
predictionErrorIni	0.0
fitnessIni	0.01

Tabla 4.1: Valores de las constantes

Se generó un diagrama de clases general que contiene la información sobre las clases del GXCS que se modificaron para incluir el código del GXCS-H. Como se puede apreciar en la Figura 4.6, los archivos fuente .java que se modificaron parcialmente a excepción de la clase GXCSHWeightVector que es nueva, son:

- GXCSH. Es el archivo principal del GXCS-H. En él se encuentra la invocación al método **main()**, el cual también realiza una invocación a los métodos implementados para la inicialización de los parámetros y atributos respectivo. El método que a su vez también invoca a los siguientes métodos es **runGXCS()**. Desde este método también se realizan invocaciones a los métodos que pertenecen a las implementaciones de otros archivos java, como son los métodos para el cálculo de pesos, del valor de degradación y de recompensas, por ejemplo. Dentro de esta clase se modificó el borrado de los clasificadores. Anteriormente se hacía con el método de selección de ruleta y actualmente se toma el valor de decaimiento mencionado anteriormente.
- GXCSHClassifier. En este archivo se agregaron las variables de entrada **x**, de salida **y**, así como la de la tasa de decaimiento **decay**. Los valores

de estas variables se modifican en otros archivos, pero se crearon en éste debido a que cada clasificador creado debe tener asociado un valor de estas variables independiente de los valores que se obtengan para otros clasificadores. El método **getDelProp()** se ha eliminado de GXCS-H, por la misma razón sobre la tasa de decaimiento.

- GXCSHClassifierSet. Se modificó el método **updateSet()** para realizar el cálculo de los valores **x** y **y** cuando la acción ha sido la ganadora.
- GXCSHConstants. Se modificaron algunas constantes y se eliminaron otras. **theta_del** y **theta_sub** fueron eliminadas, ya que éstas se utilizaban para evaluar las reglas que debían ser eliminadas. Ahora no son necesarias pues se utiliza el parámetro de degradación para este fin.

En la Figura 4.6 sólo se aprecian las clases que fueron modificadas para incorporar la implementación de las ecuaciones de las reglas del aprendizaje hebbiano seleccionadas. La notación UML utilizadas en ella sólo muestra la división de la clase en cuanto al contenido de atributos y métodos java asociados a cada una de ellas, se omitió incluir la relación entre clases por facilidad de lectura del diagrama.

Cabe señalar que se agregó la H al final de cada clase para indicar que pertenecen al nuevo clasificador, por ejemplo, el archivo fuente de la clase principal se llama **GXCSH**.

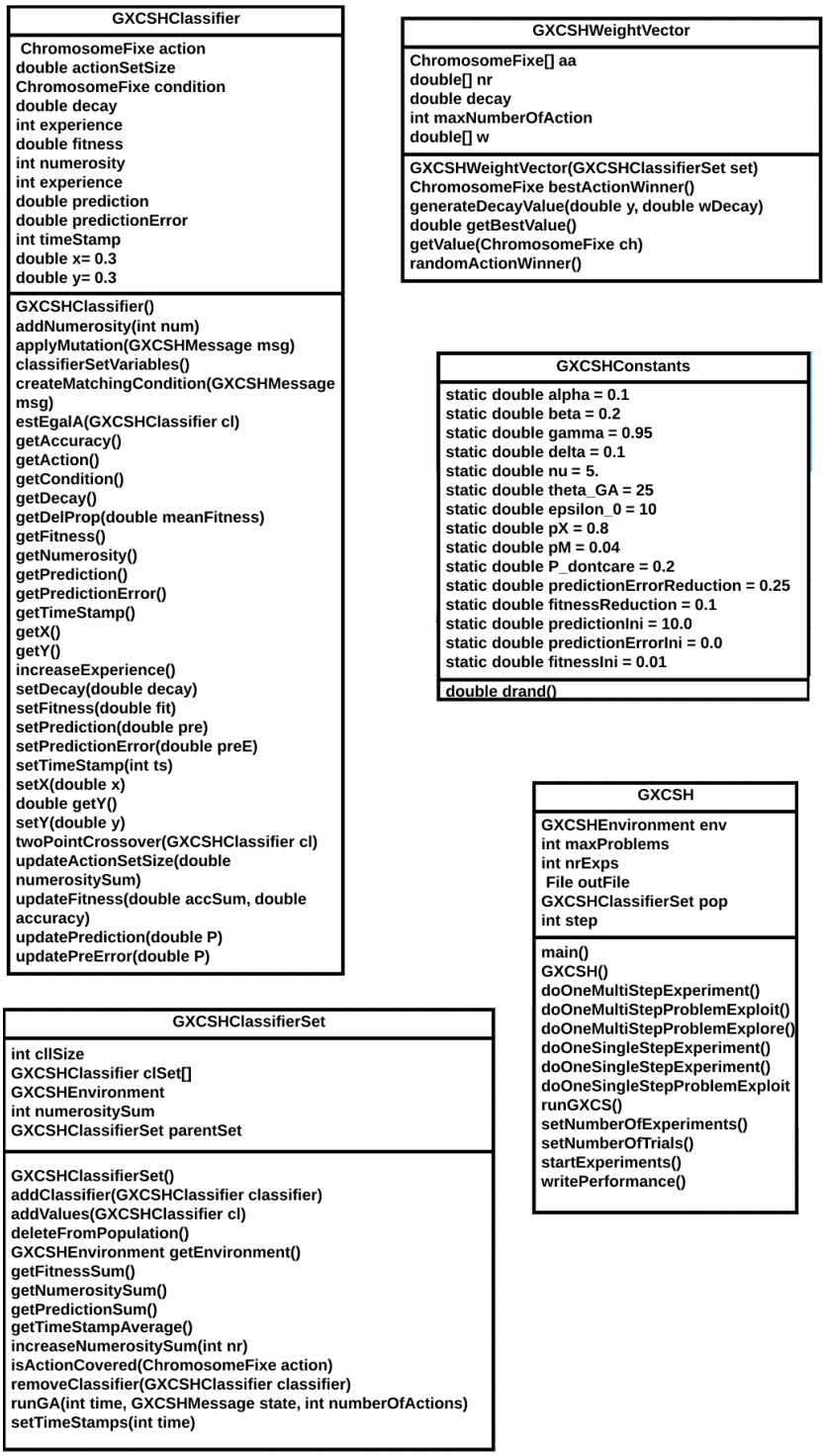

Figura 4.6: Diagrama de clases UML

La Figura 4.7 muestra la clase que se implementó para el cálculo de los pesos hebbianos que se almacenan en el vector, así como de la recompensa y castigo para las reglas; la aptitud y rendimiento de las que son seleccionadas como ganadoras. Al implementar el archivo fuente de esta clase se eliminó **GXCSPredictionArray**, ya que éste contenía el cálculo del arreglo de predicciones que se utiliza en el GXCS.

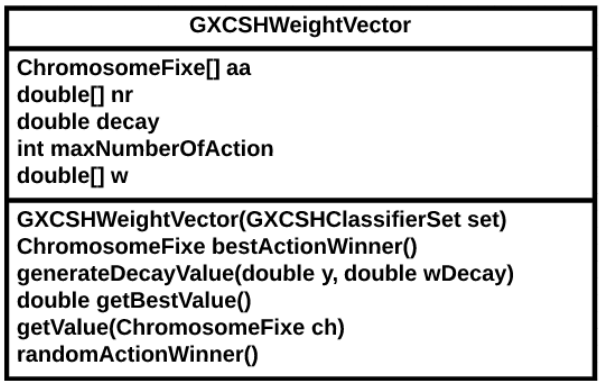

Figura 4.7: WeightVector.java en UML

El archivo WeightVector contiene los métodos y atributos necesarios para el cálculo del vector de pesos, así como los nuevos valores que van adquiriendo **x** y **y**.

En este archivo se creó el vector de pesos **w** y se siguió utilizando el vector de acciones **aa**, cálculo que se realiza de la misma manera que en el GXCS. Ambos valores se asignan a cada regla después de que se ha hecho la selección de reglas de la población que cumplen con los valores de las reglas que se obtienen desde el ambiente. El vector de pesos se inicializa con valores pseudoaleatorios cercanos a cero mientras que el de acciones se inicializa en cero. Los valores para ambas inicializaciones fueron propuestas por la literatura [14], debido a que han sido probadas en redes neuronales y se ha comprobado que ellas permiten que los valores se mantengan dentro de un rango promedio entre cero y uno. Después se realiza el cálculo de cada uno de estos elementos.

Los valores que va tomando el vector de pesos se calculan por medio de las fórmulas hebbianas mientras que el de acciones se calcula como se hace en el GXCS. Se decidió dejar la versión GXCS de este cálculo ya que se consideró que no afecta al resultado final, aunque es deseable que posteriormente se pruebe con los datos tomados totalmente de las ecuaciones de las vertientes hebbianas. También se calcula la tasa de degradación **decay**, que también se inicializa con valores cercanos a cero de manera pseudoaleatoria y que cambia a partir de la fórmula del aprendizaje hebbiano.

El método **generateDecayValue()** es el responsable de realizar el cálculo de la tasa de degradación de la regla, mientras que el método **bestActionWinner** busca la acción con mejor recompensa, **getBestValue()** entrega el mejor valor para la recompensa y **randomActionWinner()** genera un valor pseudoaleatorio para las acciones que se encuentran en el conjunto de coincidencias **M**, esta forma de generar el cálculo para obtener la acción ganadora se realiza de la misma manera que GXCS.

El GXCS toma algunas de las características de los clasificadores más importantes para el desarrollo de este trabajo, como son: ZCS y XCS. Del ZCS toma el cálculo de la exactitud del algoritmo de bombero, y del XCS el cálculo de la aptitud, la recompensa y el castigo. En la Tabla 4.2 se muestra la comparación entre los principales clasificadores de aprendizaje, incluyendo el GSCX-H.

Después de tomar la elección de las reglas que se implementaron se revisó el código del GXCS para evaluar qué código de éste podría ser re-utilizable y cual era mejor desechar. Al crear el GXCS-H se eliminó el código principal del aprendizaje rápido y se incluyó el de las reglas de Hebb, sin embargo, se reutilizó el manejo del error en la predicción con sus respectivas variables y ecuaciones así como la variable ALFA que puede ser utilizada como factor de aprendizaje, ya que tiene las mismas características que el utilizado en las ecuaciones de la regla de Hebb.

Características de los clasificadores					
	Entrada	A. markoviano	A. no markoviano	Ventajas	Desventajas
LCS	Binario	Sí	No	Concatenación en ciclo interno.	Subgeneralización.
ZCS	Binario	Sí	No	Mejor convergencia. Repartición de créditos a las reglas de solución.	Predominio de reglas subgeneralizadas.
XCS	Binario, cadena de bits	Sí	Sí	Control de errores. Aptitud basada en el rendimiento.	Problemas en ambientes no markovianos.
GXCS	Binario, real, booleano, vectorial	Sí	Sí	Conjunto completo de clasificadores. Pertinencia en la predicción.	Persistencia de reglas subgeneralizadas.
GXCS-H	Binario, real, booleano, vectorial.	Sí	Sí	Basado en GXCS. Aprendizaje hebbiano. Tasa de degradación para estabilización de reglas.	Precisión del algoritmo de bombero. Aptitud y recompensa de aprendizaje rápido.

Tabla 4.2: Principales clasificadores de aprendizaje

En el presente trabajo sólo se realizaron pruebas con la simulación de un solo agente por cada prueba debido a que es la primer fase de prueba de un clasificador, pero esto no quiere decir que no tenga la capacidad de trabajar con mas de un agente pues sus características inducen a que se pueda usar en un sistema multi-agente.

En la Tabla (4.3) se muestra la comparación entre las características de las vertientes de la regla de Hebb estables.

Reglas de Hebb			
Vertiente	Valores	Características	Ecuación
Normalización multiplicativa (R. de Oja)	x escalar y escalar *decay* escalar	Restricción dinámica. Estabilidad con pesos grandes. Olvida pesos pequeños. Tasa de degradación.	$w_{ij}(q) =$ $w_{ij}(q-1)+$ $\alpha.y_i(q)(x_j(q) - y_i(q).w_{ij}(q-1))$
INSTAR	x vector y escalar *decay* escalar	Evita la degradación de pesos sin estímulos. El valor de pesos está en las entradas.	$w_{ij}(q) =$ $w_{ij}(q-1) + \alpha.(y_i(q).$ $(x(q) - w_{ij}(q-1))$
OUTSTAR	x escalar y vector *decay* escalar	Evita degradación de pesos sin estímulos. La degradación de pesos es proporcional a la entrada.	$w_{ij}(q) = w_{ij}(q-1)+$ $\alpha.(y(q) - w_{ij}(q-1)).y(q)$

Tabla 4.3: Vertientes hebbianas estables

4.5. Pruebas y resultados

En esta sección se explica la manera en que se realizaron las pruebas del GXCS-H y los resultados que se obtuvieron de ellas.

4.5.1. Pruebas

El algoritmo de aprendizaje hebbiano se implementó en el lenguaje de programación Java versión 8 dentro del entorno de desarrollo integrado Eclipse JEE Neon en una computadora Hewlett Packard HP Prodesk 600 G1 TWR con procesador Intel(R) Core(TM) i7-4770 CPU, velocidad de 3.40 GHz, 8 GB en RAM, sistema operativo Windows 10 de 64 bits. Así como también se realizaron las mismas pruebas en un equipo Hewlett Packard modelo HP Pavilion dv51142la Entertainment PC con 4 GB en RAM procesador Intel(R) Pentium(R) Dual Core con el entorno de desarrollo Eclipse Juno con Java version 7.

Es importante puntualizar que se eligió el lenguaje de programación Java debido a que las implementaciones de los sistemas clasificadores a los que se tenía acceso se encuentran desarrollados en dicho lenguaje, incluyendo el GXCS, del cual se tomó el código, se utilizó como base y se modificó para incluir el aprendizaje hebbiano.

Los ambientes utilizados para probar la propuesta presentada son, **multiplexor** y **woods**, respectivamente. Estos ambientes fueron propuestos por la comunidad cientifífica en numerosas versiones de los LCS ya que ambos permiten obtener una solución fácil de interpretar. El ambiente multiplexor se conoce como estático y woods como dinámico. El ambiente woods es usado para probar la regla de Oja con aprendizaje supervisado y outstar. En el ambiente del multiplexor se probó la regla de normalización multiplicativa (Oja) para aprendizaje no supervisado e instar.

El ambiente **woods** puede verse como un laberinto donde se tienen obstáculos, un agente que lo recorra para llegar a un objetivo y el objetivo en sí. En la Figura 4.8 se ilustra un ejemplo de cómo se puede desarrollar un laberinto. El agente se muestra como una persona que va a recorrer el laberinto para llegar al objetivo final que en este caso está ejemplificado con un regalo. La persona deberá pasar por el camino más corto siempre que no haya obstáculos en él.

Los obstáculos en este caso están ilustrados por medio de pequeñas bardas que impiden el paso del agente. Este ambiente, computacionalmente hablando, se puede ver como una matriz con valores, donde cada obstáculo tiene asignado el valor de 1 y la celda que no tiene obstáculo tiene el valor de 0. Se construye una cadena de ceros y unos acomodados por posición, donde cada posición en la cadena indica si hay un obstáculo o está libre el paso, por ejemplo, la primer posición de la cadena puede indicar la posición arriba del agente, donde se almacenó un uno para indicar que hay un obstáculo.

En el ejemplo se tomó la celda que está justo arriba del agente como la primera a agregarse a la regla, y de ahí se van tomando los valores en sentido de las manecillas del reloj, como resultado de este ejemplo, la regla generada es: 111100111, donde se puede observar que hay obstáculos exactamente arriba del agente, en la esquina superior derecha y del lado derecho de la misma, después se encuentran dos ceros que indican que no hay obstáculos en la esquina inferior derecha y exactamente abajo y después hay más obstáculos en las siguientes tres posiciones que están a su lado izquierdo.

Cabe señalar que el agente sólo ve las ocho celdas que están justo al rededor de él, no alcanza a ver las que están más allá de las 8 más cercanas. A partir de ello toma la decisición de avanzar por alguna de las celdas que están libres de obstáculos y nuevamente hace una revisión de las celdas que se convierten en las 8 más cercanas. Cada paso que dé le permite ver el siguiente conjunto de celdas y seguir tomando decisiones para avanzar.

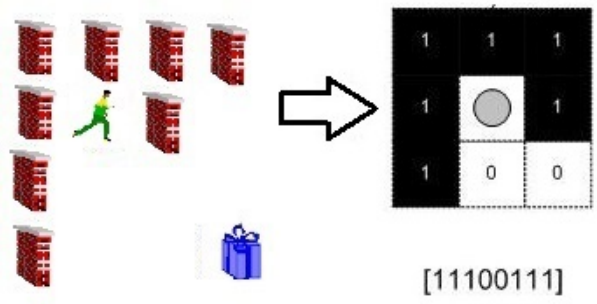

Figura 4.8: Ejemplo de ambiente woods

El ambiente **multiplexor** se puede ilustrar por medio de compuertas lógicas donde se tienen varias entradas, en este caso 6, y una sola salida. En la Figura 4.9 se observa que las entradas están asociadas a las variables x_i donde i= 0, 2,... ,5. Estas entradas entran a un circuito con compuertas lógicas acomodadas similares al multiplexor booleano, y se va mezclando los valores que van tomando con operadores lógicos de **y** y **o**, consiguiendo una sola salida F_6.

Para un LCS, las entradas del multiplexor son parte de la regla que se va a evaluar, donde cada entrada representa un parámetro en particular de esa regla y la salida es la acción asociada a toda la regla. Esos datos se introducen al multiplexor, el cual evalúa como si tuviera compuertas lógicas y permite ir asignando los valores al par regla/acción correspondiente.

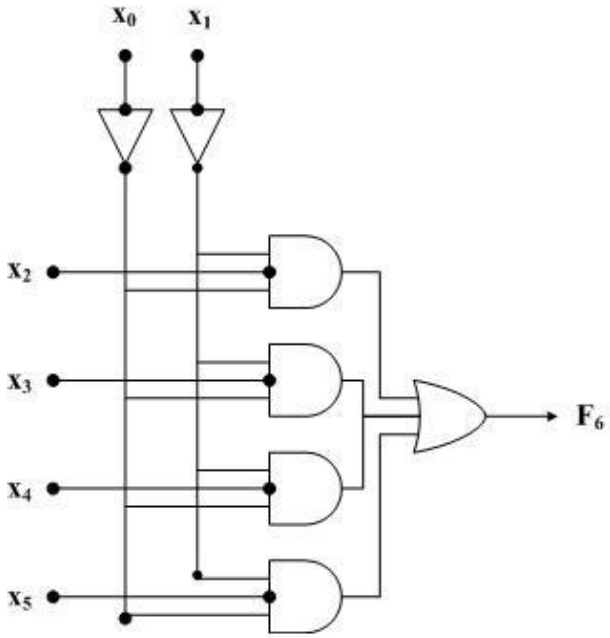

Figura 4.9: Ejemplo de ambiente multiplexor

Para propósitos de este trabajo, se realizaron pruebas en ambos ambientes con 5,000, 10,000, 15,000, 20,000, 25,000 y 50,000 pasos, con 50 y 100 pasos promediados para cada una en ambos ambientes. Se realizaron más de 50 pruebas para cada vertiente con cada uno de este número de reglas. Se observó que los resultados de cada una de estas pruebas tienen resultados muy similares entre ellas en ambos ambientes, los datos de salida caen en los mismos rangos. Para efectos prácticos, en el presente capítulo sólo se incluyeron los resultados de las comparaciones de las 3 vertientes hebbianas contra los del GXCS con 20,000 reglas y 100 pasos en el ambiente multiplexor.

Las comparaciones hechas de las pruebas de 20,000 reglas en el GXCS dentro del ambiente del multiplexor y las tres vertientes probadas en el GXCS-H se pueden observar en las figuras 4.10, 4.11 y 4.12. El factor de comparación principal mostrado fue el rendimiento, pues éste es calculado a partir de los valores obtenidos sobre el vector de pesos, la recompensa y la aptitud de cada regla. La tasa de degradación o decaimiento nos permitió eliminar las reglas que tenían un menor peso y por lo tanto, no era candidatas a ser tomadas en cuenta para futuros cálculos.

Dentro de la implementación de GXCS y de GXCS-H se genera un archivo de salida para almacenar los resultados de las pruebas, es decir, el rendimiento, número de pruebas realizadas, margen de error en la predicción y número de

elementos por cada iteración. El GXCS-H utiliza el mismo formato de salida, donde se pueden apreciar los mismos tipos de datos. En estos resultados se puede observar que el rendimiento en el GXCS-H es más cercano a cero que el rendimiento del GXCS.

El valor del rendimiento se calcula a través de la aptitud de cada regla para el GXCS, y a través de esa misma aptitud en conjunto con los pesos hebbianos y los valores asociados con las reglas de entrada y las acciones de salida para el GXCS-H. Es por ello que se toma a este parámetro como base de comparación entre ambos clasificadores. Como se verá más adelante en esta misma sección, el GXCS-H mejoró el rendimiento del GXCS.

El rendimiento es un parámetro que refleja el promedio de la aptitud de un conjunto de reglas, en este caso, 100 reglas, y éste debe caer en el rango entre cero y uno. Si está cercano a cero, el rendimiento y, por lo tanto la aptitud, son muy bajas. Si está cercano a uno, el rendimiento es muy bueno, por lo que las reglas que pertenecen a ese conjunto y sus respectivas acciones son consideradas como respuestas válidas que se enviarán al ambiente y serán evaluadas en él.

Aunado a esto se encontró que el tiempo de ejecución es menor, ya que en el GSCS-H se eliminaron algunos métodos en los archivos fuente, como son, el uso de un ciclo para eliminar las reglas que ya no se requerían, así como la subgeneralización, que también pudo ser eliminada. El cálculo en el tiempo de ejecución se generó a partir del número de repeticiones y operaciones y su orden de complejidad.

Las primeras pruebas se hicieron con un peso inicial muy cercano a cero, con valores pseudoaleatorios entre 0.5 y 0.6 pero se vio que se obtenían mejores resultados con un rango más amplio, entonces se decidió utilizar el rango $0.1 - 0.6$, los cuales mejoraron el rendimiento obtenido anteriormente. Este cálculo se utilizó para las 3 vertientes hebbianas probadas, ya que para las tres se encontraron mejores valores en los pesos y en el rendimiento del clasificador.

La variable que se utilizó para la tasa de degradación se inició con valores cercanos a cero. El rango utilizado fue entre 0.1 y 0.7 y luego se aplicó para el borrado de las reglas. Inicialmente se borraron las reglas que tenían tasa de degradación mayor a 0.5 pero esto hizo que el sistema no borrara suficientes reglas, ya que había muy pocas con una tasa con esas características y por lo tanto, saturó el conjunto de población, por lo que se optó por borrar reglas con valores de degradación desde 0.25. Esto se aplicó para las 3 reglas hebbianas que se probaron en el sistema.

A continuación se presentan las gráficas obtenidas a partir de las primeras pruebas realizadas con el cálculo de pesos hebbianos solamente, en ellas se muestran los primeros resultados obtenidos, es decir, sin tasa de degradación y aun con la subgeneralización. Estas gráficas son realizadas sobre las pruebas de las

vertientes hebbianas estables, es decir, las que tienen vector de pesos hebbianos estandarizados entre cero y uno, como lo indica la literatura. Estas vertientes son: instar, outstar y Oja. Para todas las pruebas mostradas se tiene un conjunto de 20,000 reglas agrupadas de 100 en 100 generando un rendimiento promedio para cada uno de dichos grupos.

En la Tabla (4.4) se muestra la comparación entre las características de las vertientes de la regla de Hebb con pruebas que no incluían la tasa de degradación. En esta tabla se pueden observar los valores promedio de rendimiento mínimo y máximo encontrados al realizar las pruebas.

GXCS y GXCS-H				
Rendimiento	GXCS	Oja	instar	outstar
Mínimo	0.37	0.30	0.28	0.38
Máximo	0.65	0.75	0.67	0.63

Tabla 4.4: Comparación sin el uso de la degradación

Estos mismos resultados se pueden observar de una manera más amplia en las gráficas 4.10, 4.11 y 4.12. Los resultados que se muestran en la tabla son valores promedio generados de las diferentes pruebas que se hicieron. Todas estas gráficas muestran la evidencia de una corrida con 20,000 pasos, respectivamente. Es decir, son una muestra representativa del comportamiento de todas las pruebas que se realizaron en las diferentes vertientes.

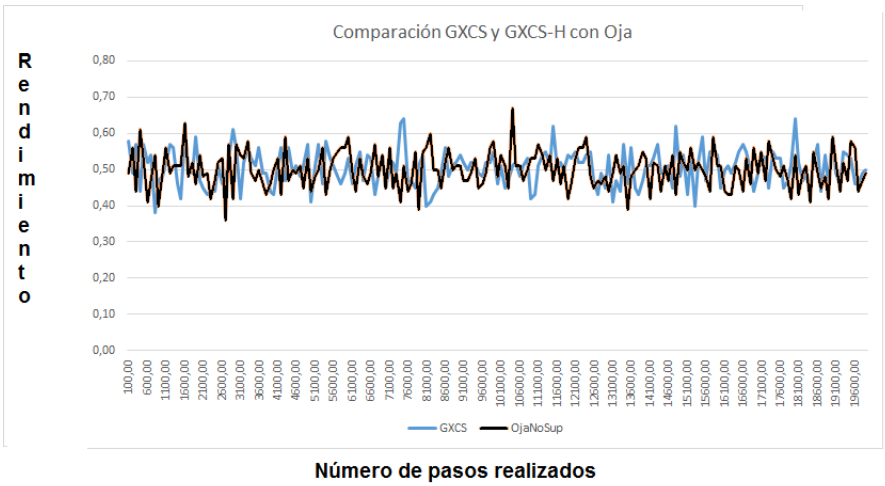

Figura 4.10: Comparación GXCS y GXCS-H con Oja

Como se puede apreciar en la Figura 4.10, la regla de Oja en GXCS-H lleva un comportamiento muy similar al GXCS, sobretodo al inicio de las iteraciones. Poco a poco se empieza a estabilizar mejor que el GXCS con el aumento del número de pasos, en los cuales va agregando reglas por medio del algoritmo genético y va eliminado las que tienen una tasa de degradación mayor a 0.5. Hay momentos en los que el rendimiento, y por lo tanto la aptitud y los pesos hebbianos, aumentan y se estabilizan, lo que quiere decir que se cuenta con más reglas que son aptas para resolver un problema a través de las acciones que se le asignan.

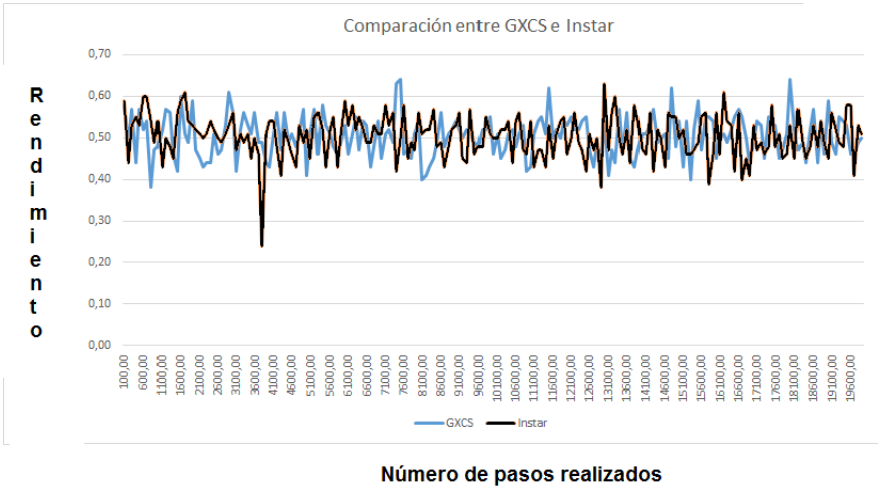

Figura 4.11: Comparación GXCS y GXCS-H con instar

La Figura 4.11 muestra que la regla de instar tiene comportamientos ligeramente menores que GXCS. En este ejemplo, se puede ver que la regla instar tiene una caída en el rendimiento en el paso 3500, esto se debe a que se creó una nueva regla que no había sido probada y, por lo tanto se le dieron pesos hebbianos y recompensas muy pequeñas al inicio. Posteriormente se puede ver que el GXCS-H con regla de instar se vuelve a recuperar y las reglas son ligeramente más estables. En general, la regla instar genera rendimientos ligeramente mayores a los generados por el GXCS.

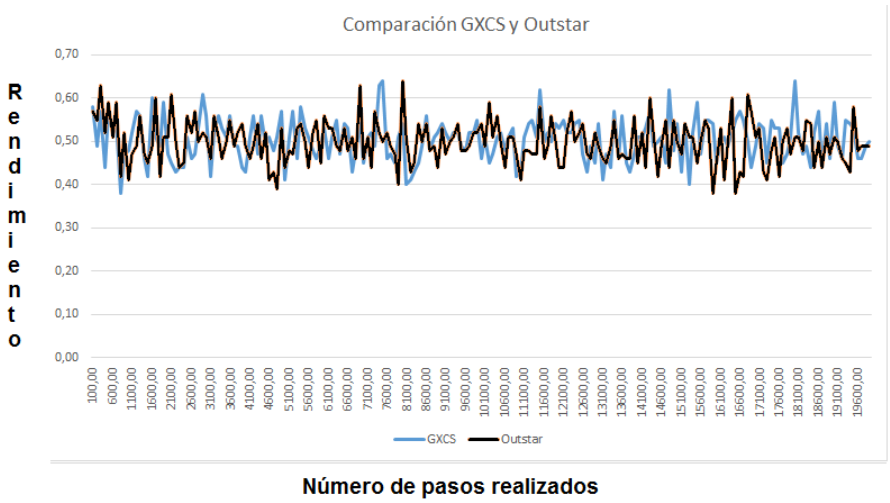

Figura 4.12: Comparación GXCS y GXCS-H con outstar

Como se puede apreciar en la Figura 4.12, el comportamiento de la regla outstar en el GXCS-H es similar al comportamiento de las reglas de instar y de Oja.

Una vez agregada la tasa de decaimiento para eliminar reglas con valores de peso hebbiano muy bajo y eliminada la subgeneralización de la implementación, se realizaron las pruebas respectivas las cuales generaron los resultados que se muestran en las Figuras 4.13, 4.14 y 4.15. En los tres casos se observa que GXCS genera un rendimiento menor que las reglas de Oja, instar y outstar, respectivamente.

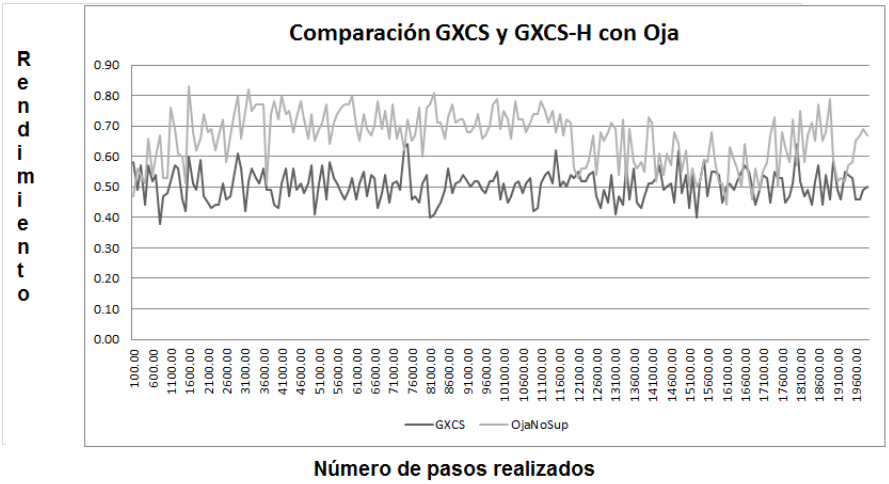

Figura 4.13: Comparación GXCS y GXCS-H con Oja

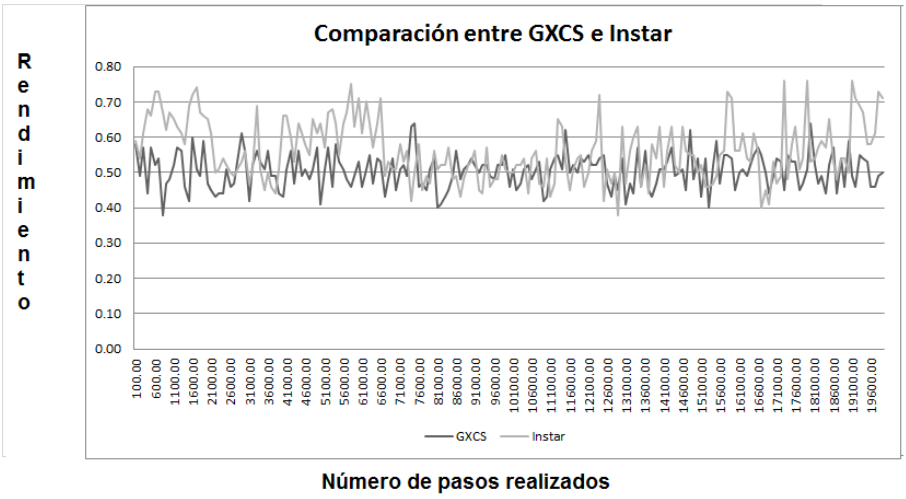

Figura 4.14: Comparación GXCS e instar

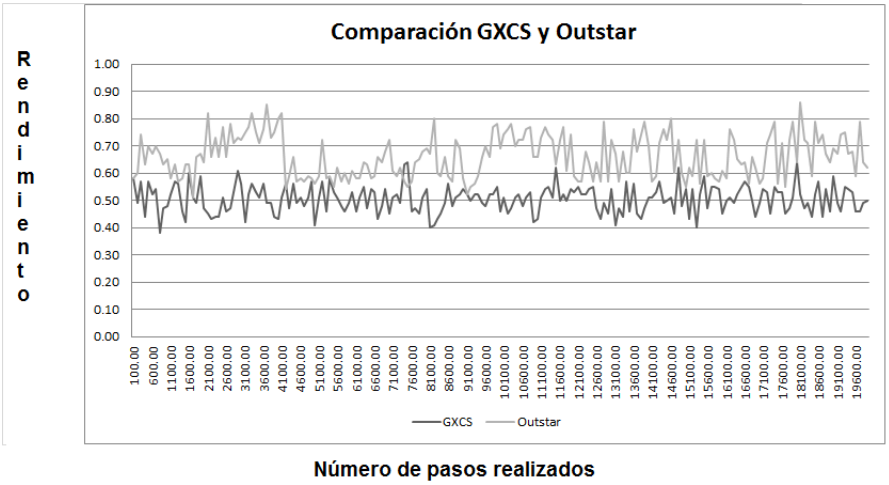

Figura 4.15: Comparación GXCS y Oustar

En la Tabla (4.5) se muestra la comparación entre las características de las vertientes de la regla de Hebb que incluye la tasa degradación.

GXCS y GXCS-H				
Rendimiento	GXCS	Oja	instar	outstar
Mínimo	0.37	0.44	0.44	0.55
Máximo	0.65	0.83	0.75	0.85

Tabla 4.5: Comparación con degradación

Los resultados obtenidos a partir de la incorporación del aprendizaje hebbiano comparados con los resultados del GXCS muestran una mejora en la aptitud, premio y castigo por cada par condición-acción con lo que nuestra hipótesis inicial se cumple de acuerdo a las condiciones evaluadas y los resultados obtenidos en el GXCS-H. Cabe señalar que nuestra hipótesis se cumple para las tres vertientes probadas.

Capítulo 5

Conclusiones y trabajo futuro

El presente capítulo incluye las conclusiones observadas en el desarrollo del trabajo de investigación realizado, así como algunas sugerencias sobre lo que se detectó como futuro desarrollo.

5.1. Conclusiones

Una vez realizadas las pruebas y obtenidos los resultados se describen en la presente sección las conclusiones a las que se llegaron.

- La incorporación del aprendizaje hebbiano en un GXCS mostró que el clasificador puede converger a la solución.
- Dentro de un ambiente dinámico el clasificador mostró que el control en las reglas es más eficiente debido a que el factor de decaimiento permite eliminar reglas que no son utilizadas y solo mantener aquellas que están en constante competición.
- El uso del aprendizaje hebbiano evita la subgeneralización de reglas, con lo que el sistema ya no requiere evaluar las reglas de esta manera, ahorrando tiempo de procesamiento.
- Se encontró una diferencia significativa entre las pruebas realizadas a las vertientes de Hebb cuando se agregó la tasa de degradación y se eliminó la subgeneralización.
- El uso de la variable de degradación permite eliminar sólo aquellas reglas que no han sido utilizadas en la búsqueda de una solución sin afectar el rendimiento del clasificador mismo.

- Los resultados obtenidos hasta el momento son satisfactorios en los ambientes de pruebas, sin embargo, se requiere de mayor experimentación en otro tipo de ambientes.
- A partir del presente trabajo se puede asegurar que el aprendizaje hebbiano puede ser incorporado en un sistema clasificador llevando a este tipo de sistemas a contar con un mecanismo de aprendizaje más natural, entendiendo esto como lo realiza un ser humano.
- El estudio del aprendizaje hebbiano y de los LCS me permitió tener un panorama más amplio sobre la aplicación de las neurociencias en el área de inteligencia artificial, con ello verifiqué que este aprendizaje mejora los resultados esperados de un LCS.
- A pesar de que se ha incluido el aprendizaje hebbiano en el presente trabajo, se sugiere realizar pruebas con ambientes multiagente.

5.2. Trabajo futuro

A partir de las conclusiones enumeradas anteriormente se pueden identificar algunas de las tareas que se recomienda realizar como trabajo futuro.

- Es deseable realizar pruebas en ambientes donde se pueda emular más de un agente, es decir, con sistemas multiagentes.
- Realizar pruebas del GXCS-H en casos de estudio específicos.
- Eliminar el cálculo de la exactitud, tomada en el presente trabajo del algoritmo de Bombero.
- Eliminar el cálculo de la aptitud y recompensa tomada en el presente trabajo del aprendizaje rápido.
- Realizar el cálculo de recompensa y aptitud a partir de las ecuaciones de las vertientes de Hebb tomadas en el presente trabajo. Esto nos crearía un sistema que se comportará con características mas apegadas al comportamiento humano real durante su fase de aprendizaje.

Apéndice A

Publicación

En el presente anexo se incluye el trabajo relacionado que se publicó a la par del desarrollo de investigación que se muestra en el presente documento.

Se publicó el artículo en el mes de Octubre de 2015, es decir, al inicio del desarrollo del presente trabajo, titulado *"Evolutive Autonomous Behaviors for Agents in Serious Games"* en "2015 International Conference on Computational Science and Computational Intelligence (CSCI)", con un formato IEEE. Su DOI es 10.1109/CSCI.2015.175. Este artículo nos permitió tener una visión general de lo que se puede hacer en el área de los sistemas clasificadores y el comportamiento de los agentes y nos sirvió de pauta para el desarrollo de la investigación.

Evolutive Autonomous Behaviors for Agents System in Serious Games

Marco A. Ramos*, Vianney Muñoz-Jiménez*, Félix F. Ramos†, José R. Marcial Romero*, Aurelio López López‡, Bertha E. Ordoñez G.*

*Universidad Autónoma del Estado de México Cerro de Coatepec, s/n Ciudad Universitaria
Toluca, México, México, 50130
marco.corchado@gmail.com, vmunozj@uaemex.mx, jrmarcialr@uaemex.mx, beordonezg@uaemex.mx

†Cinvestav-IPN, Unidad Guadalajara Av. del Bosque 1145, colonia el Bajo, Zapopan , 45019
Jalisco, México
framos@gdl.cinvestav.mx

‡INAOE, Puebla Luis Enrique Erro 1, Tonantzintla, Puebla, México C.P. 72840
Puebla, México
allopez@inaoep.mx

***Abstract*—This article describes how to generate autonomous behavior to populate a virtual environment using Serious Games and Learning Classifier Systems. A serious game is a paradigm that simulates the real environment like a natural phenomenon. For example, people's behavior living an earthquake, fire, weather phenomenon or others. Into a serious game, the users are represented by virtual entities that have autonomous behavior taken from human's behavior. The principal interest to use serious games is that it's possible to obtain a tool with capabilities to predict, to plan and to train people involved in many natural phenomenons. The originality of this paper is that used a Learning Classifier Systems (LCS) inside in Serious games. It is possible to find a better simulation of the human's behavior into a real situation, using learning machine. The entities (agents) has autonomy and adaptability given by a genetic algorithm embedded in the LCS.**

Key words: Agents's Behavior, Artificial Intelligence, Virtual Environments, Serious Games, LCS, Learning Machine.

1. Introduction

Today, computer-based systems support the different activities of human, beginning from productive tasks until tasks of leisure. One of the most active areas of computer systems is Immersive Virtual Reality Systems. That is virtual environments where users interact with the environment using possibly specific interfaces like gloves, head-mounted displays, virtual sensors, etc. Human imagination only limits the applications of this technology. Currently, using this technology, we can do virtual visit to museums, factories, research centers, etc. However, one of the presents problems in this technology is the creation of realistic virtual environments.

In this paper, we are interested in the creation of realistic behavior using in scenarios where agents act in natural form but were the evolution of the agents is important. Model tools are used to satisfy this requirement. However, solutions using this approach are complex and very costly, mainly when the objective is the simulation of the evolution of a virtual world.

In our case, the humans are embed in a social environment, a social environment system is a set of individuals that interact mutually to the artifacts from its environment. The agents needed to recreate the social behavior, in fact, the agent evolves accorded to their belief, desire and intentions.

Serious games are an actual paradigm to recreate the real phenomenon that common people are living. The principal interest to recreate the real environment is to can predict a situation where the user's integrity can be affected by the external phenomenons. A clear example was the influenza pandemic, where Mexico had affected in 2008 leaving considerable damages in the population and the official organizations.

Computer Science, in particular, Artificial Intelligence, proposes methods to generate autonomous behavior to collaborates disciplinary with human behavior areas, thus, help to understand social behavior in individual that interact with the environment. The Multi-Agent Systems (MAS) are the representation of a virtual entity. These systems try to copy the social behavior similar to individual in real world [1];

The knowledge initial of agents is taken on goal in

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.175

concordance to desires, beliefs and intentions grouped to create autonomy. The LCS grants the possibility to evolve into the environment by itself, characteristics of communication and collaboration emerge in agents to meet your goals like observed in human society.

This article shows the way to create intelligent agents with autonomous social behavior to develop serious games, where phenomenon in daily life is shown. Serious games are a tool to predict not controlled events. In section 2 we presents the definition of an intelligent Agents, section 3 describe the Learning Classifier System and our proposal architecture, section 4 presents the development our proposal architecture and finally section 5 and 6 presents the results obtained with our proposal architecture and the conclusion, respectly.

2. Intelligent Agents

Intelligent Agent is an entity capable of perceiving its environment, processing such perceptions and to response or action in your environment in a rational manner [2]. In other words, Intelligent Agent gets the best performance to maximize the answer desired. Figure 1 represents the basic architecture of an agent and the ability to act autonomously in an environment. Therefore, an agent is an entity able to take their decisions about how to act in your environment, where there is no influence of a leader or a global plan.

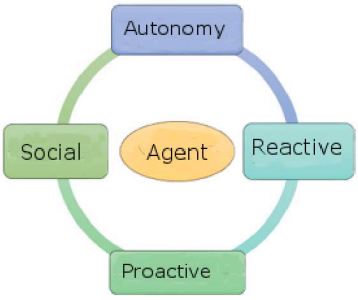

Figure 1. Architecture based on an intelligent agent

An intelligent agent has a behaviour; it's can go from the complex action of the human or animal's behaviour based on their instinct or desired [4]. An agent must be autonomous, reactive, proactive and social. Their characteristics be mention below:

- Autonomous: Agents have to operate without human intervention on their actions and decisions.
- Reactive: Agents can feel the change of the environment and to adapt at them.
- Proactive: Agents can focus on their actions and decisions to achieve their objectives.
- Social: Agents can communicate with them.

The characteristics of an agent are not more than an architecture that can be using, as a particular methodology for its construction. An agent has multiples components that interact with them to define the internal characteristics of the state of the agent. An architecture covers both technical and algorithms that support this methodology. Consequently, an architecture depends on the objectives, the tasks to be carried out and the environment [5].

In other words, an agent can have many components that interact to define the state of its internal characteristics. Architecture covers techniques and algorithms that support this methodology. As a consequence, architecture depends on environment; goals and tasks that have to be done [5].

In our case, we will use architectures based on BDI (Belief-Desire -Intention). This type of architectures takes decisions through a process of reasoning, which starts with the beliefs of the world and desires that the agent seeks to achieve. The intentions are building as a result of the beliefs and desires.

3. Learning Classifier System

The LCS is a powerful learning machine that uses ambient conditions to suit the objectives and achieve this through an evolutionary mechanism that allows find new solutions to help meet complex tasks [12].

The agent must learn from its environment, and take actions to react in some situations that can have unexpected events. Classifier systems can help giving tools on artificial life processes and will be used in our work because we want to simulate different behaviors on the environment. The LCS general structure is shown in Figure 2.

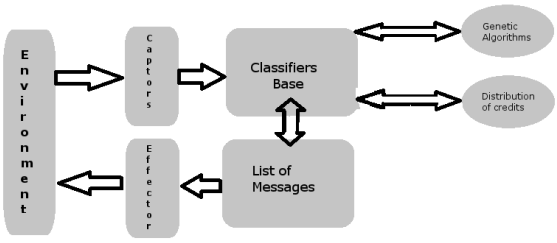

Figure 2. LCS general structure

A classifier has three parts: condition, action and weight. These are described below:

condition: the condition is a situation of an environment, and the captors sense this condition then its matches with

the base of classifiers; the best classifier is the winner to respond to a given situation.

action is the element that respond to a situation of the environment and modify this to a new situation.

weight is activated when there is more than one classifier that respond a situation of the environment, so, it activates the rule with higher weight using genetics algorithms. The rules are updating its status with different weights, so, worst rules are eliminated, and the environment can adapt itself to better rules.

In dynamic environments, the response to several situations should be fast, and then it is necessary to use mechanisms of prediction, correct classifiers have the same value and same changes to be selected. LCS most used as a Q-learning learning system [3]. With this advantage, all the status from the environment is saved updating the prediction of the classifiers to get a complete panorama to the environment. Q-learning is a process that helps to update the prediction of the classifiers from the previous step $p_{[A]i-1}$. Q-learning uses the previous rewards $[Pi-1]$ and the prediction $p_{[A]}$ and γ the influence is given to the next predictions. The equation (1) shows the mechanism to insert many classifiers, as much as we need to simulate a complex environment.

$$p_{[A]i-1} = [Pi-1] + \gamma MAX(p_{[A]}) wiht(0 \leq \gamma \leq 1) \quad (1)$$

Update the prediction of the next state in the environment is over-taken by the dynamics of the environment that necessitates the creation of new classifiers using a genetic algorithm.

The nature of the construction of the initial population of the genetic algorithm is a direct transition from the representation of the situation of the environment which makes the system evolutionary completely.

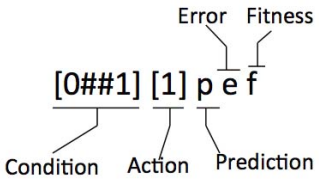

Figure 3. construction of classifier in LCS

The Figure 3 shows the construction of a binary classifier, this is the easiest way to creation, but not the best. We use a representation of the condition of the environment with abstract data.

4. Development

BDI model [7] is conceived as Practical Human Reasoning Theory, this model supports events based on reactive and proactive behavior or behavior to reach goals, this is a benefit of this model when is needed to implement Serious games. Figure 4 shows BDI model.

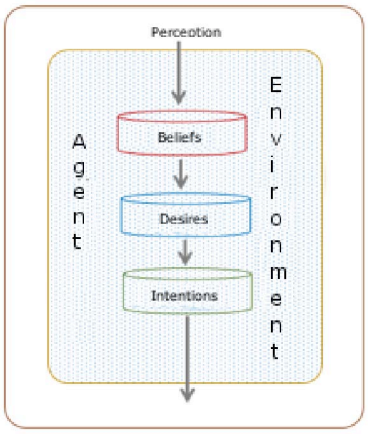

Figure 4. BDI Architecture model

Beliefs inform system status that represent information that an agent has about its environment and its internal status. Thus, desires are agents motivational status used to reach its goals. Finally, intentions help to take decisions about plans that are made to reach goals. However, a plan is not necessary a part of basic facts, the plan can include sub-goals.

Serious games take importance since the user has to be aided with a task. This task can be from an educative assistant to a training complex system to maintain an airplane turbine. Serious games contain entities that help the user and give knowledge to the user. Entities behavior must be as real as possible to make user feels attracted by the serious game all the time.

Figure 5 shows an environment based on a virtual city using Unity 3D. This city has main items for simulating agents as pedestrians.

Agents that pollute the environment are based on LCS and BDI model as follows:

- θ is a set that describes the agents environment.

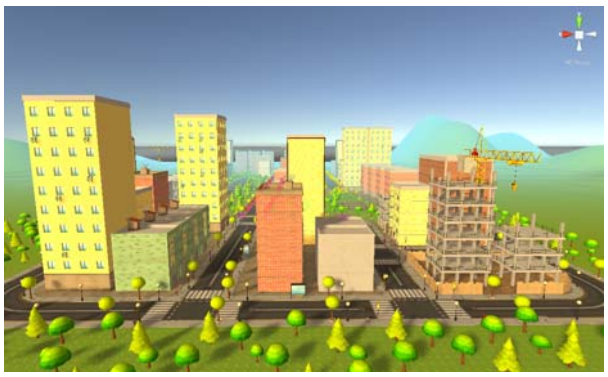

Figure 5. Environment in a virtual city

- T(θ) is a set of attributes $\{\tau 1, \tau 2 \ldots \tau n\}$ describes the environment

Definition 1: A belief c over θ is a n-tuple of T(θ) attributes. It is referred as c = ((τ1, τ2, ... τm) with $m \leq n$.

Definition 2: A set of beliefs over θ as C(θ) = {(τ1, τ2, τm) where $j = 1, 2, ..., m \leq n$}

Example: Assumption that T(θ) includes all attributes needed to characterize a touristic path in Salamanca city.

T(θ) = {τ1 = monument's, τ2 = schedule, τ3 = cost, τ4 = travelling time, ..., τn}

In this environment a belief, for example, monument, is an attribute vector of $T(\theta)$ that belongs to monument

Belief monument = {τ1 = monument's, τ2 = schedule, τ3 = cost, τ4 = quality indicator}

Definition 3: Being $\wedge$ an accessibility operator between m beliefs defined as ($c1$, $c2$, $c3$, ..., cm) $\wedge$ ($c1, c2, c3, ..., cm$) = ($c1$ $\wedge$ $c2 \wedge$.... cm), this is the operator to create new structures joining beliefs that supports each other.

$\wedge$ operator is null over beliefs if some of them is not accessible and it is denoted by $\wedge(c1, c2, c3, ..., cm) = 0$.

Definition 4: An intention i over is a s-tuple of beliefs each other compatible and is denoted by $i = (c1, c2, ..., cs)$ with s$\in$ N, $\wedge$ $(ci, cj) \neq 0$.

Definition 5: Set of intentions over θ , is defined by I(θ) = $\{(c1, c2, ..., ck)$ where k $\in$ N$\}$

Definition 6: Canonical variables of a set are any set I(θ) of independent linear parameters χ= (A_1, A_2, ..., A_v) that characterize i $\in$ I(θ) items. These variables are part of each problem P given.

Definition 7: A desire d over τ is a function defined by

d : I(θ) $\rightarrow$ $\Omega(\chi)$

i = ($c1 \wedge ... \wedge cr \rightarrow$ F(A_1, A_2, ..., A_v)

where $\Omega(\chi)$ is a function set over χ

The expression above refers to intentional attitudes that we call desires. These desires can be achieved through a plan to this effect, in other words, all information that beliefs give us are evaluated taking parameters related to desired goals.

Definition 8: A set of desires over θ is defined as

$D(\theta)$=$\{d$: I(θ) $\rightarrow$ $\Omega(\chi)\}$

Where I(θ) are intentions and $\Omega(\chi)$ are functions over χ.

5. Results

To watch physical and natural phenomenon like human watch them in his real world is a wide problem. Serious games have a virtual environment to simulate this phenomenon. Castle [8] said that a virtual environment is a way to do simulations of activities from real world thru time, this is done to understand its behavior and evaluate new possibilities.

To watch physical and natural phenomenon like human watch them in his real world is a wide problem. Serious games have a virtual environment to simulate this phenomenon. Castle [8] said that a virtual environment is a way to do simulations of activities from real world thru time, this is done to understand its behavior and evaluate new possibilities.

To this article, we used an LCS and BDI architecture for creating a Systeme Multi-Agents to pedestrian simulation in a city for watching agents behavior and how agents modify the internal states to reach their goals. Movement of agents is based on vehicle concept proposed by Reynolds [10]. So, is possible to find autonomous agents with movements made in three layers: I) Choose action, II) Direction and III) Locomotion. These layers are described below:

I) Choose an action: Agent has one or more goals, so, it can select one or more actions to reach its goals. (For example, communicate with other agents, avoid an obstacle, take a route, etc.). First of all, an agent will broker its environment and later, it will take an action based on a desire.

II) Direction: After the action was chosen, the agent has to do next movement under a direction force.

III) Locomotion: This layer sometimes is not considered because is related with agents body movement.

Figure 6 shows hierarchy between layers described above:

Figure 6. Hierarchy of movement behavior

Figure 7 shows an agent embedded in a virtual city. On this stage, a maquette is used to simulate the environment for validating agents behaviors.

Agents can walk on the blocks, visit attraction parks, avoid obstacles, and overall, have knowledge of pedestrian zone, such as sidewalks, corridors, etc. Agents have to go to corners and wait till traffic light is green for crossing streets. On this stage our virtual environment doesn't have cars, it will be included later.

Figure 7. Agent embedded on a virtual environment

The Figure 8 shows an agent called Jazmin, she has to walk thru a virtual city, if she wishes, she can go inside of a building, only if this building is available. Non-specific characteristic blocks compound the virtual environment, but it will contain doors that can be open, access to buildings, etc.

Figure 8. Agent Jazmin walking thru virtual city

LCS architecture and BDI paradigm let us watch autonomous behavior, even though architecture can change depending on the autonomous action that we want to represent or study.

6. Conclusion

Serious games are an efficient tool to watch interesting phenomenon in different society areas and productive sectors thru real environment simulations.

LCS shows high reactivity in the environment, which allows us to observe behavior in the most natural agents, according to the ambient pressure.

The BDI model allows agents to have an initial knowledge base upon the simulation and can be observed behaviors encouraged if not based on tasks and objectives to be achieved in the environment.

The possibility to change the agent's knowledge base to implement an adaptive process with real autonomy in entities that integrate virtual environment. We have to generate behavior based on emotions created by virtual agents, just to study the social phenomenon and another kind of behavior.

In this first stage, we have an SMA that aim to walk in the street of the city to stop at some point if there is a change of attention, for example, see a sideboard.

We currently work in more complex behaviors for a disaster such as an earthquake.

References

[1] Cynthia Nikolai y Gregory Madey. Tools of the trade: A survey of various agent based modeling platforms. Journal of Artificial Societies and Social Simulation, 12 (22), 2009.

[2] Yves Damazeau, Machael Pechoucek, Juan M. Corchado, y Javier Bajo Prez. Advances on Practical Applications of Agents and Multiagent Systems. Springer, 2011.

[3] Marco A. Ramos, Flix Ramos, Yves Duthen. System Multiagents whit Character Anticipative. Five International symposium and Shool on Advance Distrivuted Systems ISSADS 2005, Gadalajara Jalisco, México. Ed by Springer in a Special Number of LNCS.

[4] Michael Wooldridge. An Introduction to MultiAgent Systems. UK: John Wiley & Sons,ltd., 2009

[5] Daniel Shiffman. The Nature of Code, Simulating Natural Systems with Processing. 2012

[6] J. P. Muller. The design of intelligent agents. Springer, 2012.

[7] J. Harland y Morley D. N. An operational semantics for the goal life-cycle in BDI agents. Springer, 2013.

[8] Huang Xiang. La reconstruccin de la teora del razonamiento de dignaga por medio de la lgica formal. una crtica a la propuesta formalista del razonamiento. Departamento de Filosofa, Universidad Autnoma del Estado de Morelos, 2003.

[9] Christian J. E. Castle y Andrew T. Crooks. Principles and concepts of agent- based modelling for developing geospatial simulations. Working Papers Series, 2006.

[10] Craig W. Reynolds. Steering behaviors for autonomous characters. Sony Com- puter Entertainment America, 2005.

[11] Lanzi, Pier L and Stolzmann, Wolfgang and Wilson, Stewart W, Learning classifier systems: from foundations to applications, 2000

[12] Olivier, Heguy and Stéphane, Sanchez and Alain, Berro and Hervé, Luga, Generic classifiers systems and learning behaviours in virtual worlds, 2004

7. Authors

Marco A. Ramos, Researcher professor in Artificial Inteligence and Virtual Reality in Universidad Autónoma del Estado de México. He got his PhD from Toulouse University on 2007, France. His research themes are: Artificial Life, animation techniques, distributed systems, Intelligent agents, etc.

Vianney Muñoz Jiménez, Researcher professor in image processing and Computational vision at Universidad Autónoma del Estado de México. In 2009, she received her PhD from Paris 13 University, France. Her research work is about computational vision, image processing, video compressing, etc.

Félix Ramos, Received his PhD from Technology University Of Compiegne France in 1997. The M.Sc. degree from the Cinvestav, México City, México, in 1986, a DEA in Distributed Systems from the CNAM in Paris in 1994. Currently, he is Professor of Computer Sciences with CINVESTAV campus Guadalajara, Guadalajara, México. His research interest includes Distributed Complex Systems including Multi-agent Systems and nature and Brain inspired Distributed Systems. He has more than 90 technical publications in Conferences, Book chapters, and Journals. He is advisor of PhD and M.Sc thesis in Computer Sciences.

José R Marcial, received his PhD in Computational Science from Birmingham University in 2005. His recent papers in indexed magazines are: A threshold for a Polynomial Solution of #2SAT en Fundamenta Informaticae, Estimating the relevance on communication lines based on the number of edge covers en la revista Electronic Notes on Discrete Mathematics, First order definability of LRTp en la revista Artificial Intelligence, Sequential Real Number Computation and Recursive Relations in Mathematical logic Quarterly magazine, and Semantics of a sequential language for exact real number computation in Theoretical Computer Science magazine. Is a SNI, articles evaluator to national and international events. He has participated in evaluating committees from CONACyT and SEP. In 2006 he was awarded by Cience Mexican Asociation with a scholarship to study at United States of America. In 2009 he received a CONACyT scholarship to do his postdoctoral studies at Benemérita Universidad Autónoma de Puebla. SEP gave him the PROMEP perfil recognition.

Aurelio López López, Received his PhD from Syracuse University, Syracuse Nueva York, U.S.A. His research interest includes aknowlegde representation, retrieval information, data mining and natural language. Currently, he is Professor of Computer Sciences at INAOE-Puebla.

Bertha Ordoñez G, is a master student in Artificial Intelligence line at Universidad Autónoma del Estado de México. She is a computer Engineer from Universidad Autónoma del Estado de México

Referencias

[1] Thomas Back, Ulrich Hammel, and H-P Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[2] David Beasley, RR Martin, and DR Bull. An overview of genetic algorithms: Part 1. fundamentals. *University computing*, 15:58–58, 1993.

[3] Meurig Beynon, Russell Boyatt, and Zhan En Chan. Intuition in software development revisited. 2008.

[4] Meg Broderick. Hebbian learning rules. *Kluwer Academics Journal on Analog integrated circuits and signal processing*, 2002.

[5] Cédric Buche and Pierre De Loor. Generic model for experimenting and using a family of classifiers systems: description and basic applications. In *Artificial Intelligence and Soft Computing*, pages 299–306. Springer, 2010.

[6] Peter Anthony Cariani. *On the design of devices with emergent semantic functions*. PhD thesis, State University of New York Binghamton, NY, 1989.

[7] Gail A Carpenter. A distributed outstar network for spatial pattern learning. *Neural Networks*, 7(1):159–168, 1994.

[8] Alejandro Cervantes, I Galvan, and Pedro Isasi. A comparison between the pittsburgh and michigan approaches for the binary pso algorithm. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 290–297. IEEE, 2005.

[9] Peter Dayan and LF Abbott. Theoretical neuroscience: computational and mathematical modeling of neural systems. *Journal of Cognitive Neuroscience*, 15(1):154–155, 2003.

[10] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[11] Jan Drugowitsch. Design and analysis of learning classifier systems. *Sringer, Berlin*, 2008.

[12] Nicolas Durand. *Algorithmes Génétiques et autres méthodes d'optimisation appliqués à la gestion de trafic aérien*. PhD thesis, INPT, 2004.

[13] Jacques Ferber and Jean-François Perrot. *Les systèmes multi-agents: vers une intelligence collective*. InterEditions, 1995.

[14] Carlos A Logatt Grabner. Neuroplasticidad y redes hebbianas: las bases del aprendizaje. *Asociación Educar. Buenos Aires (Argentina)*, 2011.

[15] Olivier Heguy. *Architecture comportementale pour l'émergence d'activités coopératives en environnement virtuel*. PhD thesis, Toulouse 3, 2003.

[16] JC Heudin. La vie artificielle, collection systèmes complexes. *Hermès, Paris*, 1994.

[17] Nabil M Hewahi and KK Bharadwaj. Bucket brigade algorithm for hierarchical censored production rule-based system. *International Journal of Intelligent Systems*, 11(4):197–225, 1996.

[18] John H Holland. Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7. L. Erlbaum Associates Inc., 1985.

[19] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[20] John H Holland, Lashon B Booker, Marco Colombetti, Marco Dorigo, David E Goldberg, Stephanie Forrest, Rick L Riolo, Robert E Smith, Pier Luca Lanzi, Wolfgang Stolzmann, et al. What is a learning classifier system. In *Learning Classifier Systems*, pages 3–32. Springer, 2000.

[21] Pier Luca Lanzi and Rick L Riolo. A roadmap to the last decade of learning classifier system research (from 1989 to 1999). In *Learning Classifier Systems*, pages 33–61. Springer, 2000.

[22] Tomás Lozano-Perez. *Autonomous robot vehicles*. Springer Science & Business Media, 2012.

[23] Pattie Maes. Modeling adaptive autonomous agents. *Artificial life*, 1(1_2):135–162, 1993.

[24] Worth Martin. *Foundations of Genetic Algorithms 2001*. Morgan Kaufmann, 2001.

[25] C Mendaña-Cuervo and E López-González. La gestión presupuestaria de distribución con un algoritmo genético borroso. *Información tecnológica*, 16(3):45–56, 2005.

[26] R Menozzi and A Piazzi. Hemt and hbt small-signal model optimization using a genetic algorithm. In *High Performance Electron Devices for Microwave and Optoelectronic Applications, 1997. EDMO. 1997 Workshop on*, pages 13–18. IEEE, 1997.

[27] Thomas M Mitchell. Machine learning. *Machine Learning*, 1997.

[28] R Moosburger, C Kostrzewa, G Fischbeck, and K Petermann. Shaping the digital optical switch using evolution strategies and bpm. *IEEE Photonics Technology Letters*, 11(9):1484–1486, 1997.

[29] Peter Naur. Intuition in software development. *Formal Methods and Software Development*, pages 60–79, 1985.

[30] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[31] Hilda Osorio. Estudio comparativo de escuelas de ingeniería en telecomunicaciones en universidades de venezuela. *Tekhné*, 1(14), 2016.

[32] FJG Quesada, MAF Graciani, MTL Bonal, and MA Díaz-Mata. Aprendizaje con redes neuronales artificiales. *Ensayos: Revista de la Facultad de Educación de Albacete*, (9):169–180, 1994.

[33] Marco Antonio Ramos Corchado. *Etude et proposition d'un système comportemental autonome anticipatif*. Thèse de doctorat, Université de Toulouse, Toulouse, France, décembre 2007.

[34] Alain Reineix, Daniel Eclercy, and Bernard Jecko. Fdtd/genetic algorithm coupling for antennas optimization. *Annals of Telecommunications*, 52(9):503–508, 1997.

[35] Piedad Tolmos Rodríguez-Piñero. Redes neuronales artificiales para el análisis de componentes principales. la red de oja. *Vniversitat de Valéncia*, 2000.

[36] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[37] Stephen Frederick Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, Pittsburgh, PA, USA, 1980. AAI8112638.

[38] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[39] Isasi Viñuela, LEON PedroGALVAN, INES M Pedro Isai Viñuela, and Inés M Galván León. *Redes de neuronas artificiales: un enfoque práctico*. Pearson, 2004.

[40] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[41] Darrell Whitley and A Genetic Algorithm Tutorial. Technical report cs-93-103. *Colorado State University*, pages 1–37, 1993.

[42] B Wilamowski. Neural networks and fuzzy systems. *The Microelectronic Handbook*, 2002.

[43] Stewart W Wilson. Zcs: A zeroth level classifier system. *Evolutionary computation*, 2(1):1–18, 1994.

[44] Stewart W Wilson, SW Wilson, Generalization Xcs, et al. Generalization in the xcs classifier system. *Citeseer*, 1998.