INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

Título de la tesis

Metodología heurística y autómatas celulares
para la clasificación de patrones

Que para obtener el grado de:
Maestría en Ciencias de la Computación.

P R E S E N T A:
Ing. Jaime Jonathan Martínez Chepe

Director de Tesis.
M. en C. Germán Téllez Castillo

AGOSTO 2015

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 16 del mes de julio de 2015 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Metodología heurística y autómatas celulares para la clasificación de patrones"

Presentada por el alumno:

MARTÍNEZ	**CHEPE**	**JAIME JONATHAN**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: **B 1 3 0 1 0 5**

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

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

LA COMISIÓN REVISORA

Director de Tesis

M. en C. Germán Téllez Castillo

Dr. Sergio Suárez Guerra

Dr. Grigori Sidorov

Dr. José de Jesús Medel Juárez

Dr. Carlos Fernando Aguilar Ibáñez

Dr. Salvador Godoy Calderón

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITECNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 23 del mes julio del año 2015, el (la) que suscribe Jaime Jonathan Martínez Chepe alumno (a) del Programa de Maestría en ciencias de la computación con número de registro B130105, adscrito a Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de M. en C. Germán Téllez Castillo y cede los derechos del trabajo intitulado Metodología heurística y autómatas celulares para la clasificación de patrones, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección jonathanzar7@hotmail.com . Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Jaime Jonathan Martinez Chepe

Jaime Jonathan

Nombre y firma

Resumen

El interés en el reconocimiento de patrones y su clasificación ha crecido debido a su amplio rango de aplicaciones tales como, la minería de datos, biométrica, reconocimiento de caracteres y documentos, diagnostico asistido por computadora, bioinformática, entre otros, plantea un desafío computacional al tener que definir las características (atributos) y el número de ellas para un patrón, así como el número de patrones, de tal forma que la dimensión del espacio sea un problema tratable computacionalmente, por lo que el diseño e implementación de clasificadores eficaces y eficientes es hoy un problema actual.

En este trabajo de tesis se desarrolló un clasificador de patrones, que ha sido llamado MCJJ2 (iniciales del autor de esta tesis y el 2 significa que es un clasificador binario), además se desarrollaron los algoritmos necesarios para el clasificador de dos etapas DS-DV (del ingles Dependency String (DS) y Dependency Vector (DV)). Este trabajo está basado en los conceptos de autómatas celulares, algoritmos genéticos y árboles de decisión. Las pruebas y resultados muestran una eficiencia en tiempos de cómputo lineal, para el MCJJ2 y cuadrático para el DS-DV en la etapa de entrenamiento, y al ser simulados con diferentes bases de datos, estos muestran una precisión de al menos 90 % para el MCJJ2 y para el DS-DV de al menos 83 %.

Abstract

The interest in pattern recognition and classification has grown because of its wide range of applications such as data mining, biometrics, character and document recognition, computer aided diagnosi, bioinformatics, among others, it presents a computational challenge having to define the attributes, and the number of them for a pattern, and the number of patterns, so that the dimension of space is a treatable problem computationally, so the design and implementation of effective and efficient classifiers is now a current problem.

The thesis a new classifier of patterns called MCJJ2 (MCCJ comes from name of the author of this thesis and the number two means that it's a binary classifier) was developed, and all the algorithms necessary for the classifier of two stages DS-DV (Dependency String and Dependency Vector) were developed. This work is based on the concepts of cellular automata, genetic algorithms and decision trees. Tests and results show a efficient computation for the MCJJ2 is linear and for the DS-DV is quadratic, and when tested with different databases show an accuracy of at least 90 % for MCJJ2 and at least 83 % for DS-DV.

Agradecimientos

Al Instituto Politécnico Nacional por darme la oportunidad de estudiar y haberme proporcionado las herramientas necesarias para enfrentarme a las adversidades.

A mi director de tesis, M. en C. Germán Téllez Castillo por su esfuerzo y dedicación, quien con sus conocimientos, su experiencia, su paciencia y su motivación ha logrado en mí que pueda terminar mis estudios con éxito.

A mi padre Jaime Martínez Morales y a mi madre Angelina Chepe Zuñiga por el gran ejemplo que representan y por todo su apoyo incondicional. A mi esposa Guadalupe León Reyes por su cariño y comprensión.

Índice general

Capítulo 1

PLANTEAMIENTO DEL PROBLEMA

1.1. Introducción

El reconocimiento de patrones es una disciplina científica cuyo uno de sus objetivos es la clasificación de patrones (también conocido como instancias, tuplas o ejemplos) en un conjunto de categorías que también se conocen como clases o etiquetas. Usualmente la clasificación se basa en modelos estadísticos que son inducidos a partir de un conjunto representativo de patrones preclasificados; alternativamente, la clasificación utiliza conocimiento suministrado por algún experto en el campo de aplicación. Un patrón se compone generalmente de un conjunto de mediciones que caracterizan un objeto determinado, entonces surgen las siguientes interrogantes: ¿cuantas características debemos medir?, ¿cuál sería la mejor?. El problema es que en cuanto más se mide mayor es la dimensión del espacio de características y más complicada es la clasificación, además de incrementarse el tiempo de cálculo y el costo de almacenamiento.

En este trabajo de tesis se diseño un clasificador llamado MCJJ2, que haciendo uso de autómatas celulares (AC), algoritmos genéticos (AG) y árboles de decisión permite obtener una respuesta satisfactoria al problema planteado en este trabajo de tesis. Así también se diseñaron los algoritmos necesarios para un clasificador DS-DV.

Los AC son sistemas en donde el espacio, el tiempo y el conjunto de estados son variables discretas. La evolución de un AC se determina por un conjunto de reglas que toman en consideración el estado en que se encuentran los vecinos de una célula en el tiempo t para obtener el nuevo estado de la misma célula en el tiempo $t + 1$. El conjunto de estados que puede tomar una célula del autómata es finito. El universo en el que evoluciona el AC es una *lattice* de dimensión $\mathcal{D}$, donde $\mathcal{D}$ es un entero no negativo, cada componente de la *lattice*

es llamada célula. La vecindad de una célula c_i es el conjunto de células que influyen en el comportamiento de la célula c_i.

En esta tesis se utilizo un tipo particular de AC llamado autómata celular de múltiples sumideros (ACMS). Se ha establecido analíticamente que un ACMS simula una clase especial de familia de funciones *Hash* conocidas como HHF (por sus siglas en ingles: *Hamming Hash Family*) [19]. Entonces, la característica inherente de HHF proporciona el poder de clasificación al clasificador DS-DV y al clasificador MCJJ2 propuesto en esta tesis.

En la etapa de entrenamiento de ambos clasificadores es necesario encontrar un ACMS que haga la tarea de clasificar los patrones siguiendo la muestra de supervisión, para realizar esta busqueda del ACMS se utilizo un AG.

Un AG es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Para ello, se parte de una población inicial de la cual se seleccionan los individuos más capacitados para luego reproducirlos y mutarlos para finalmente obtener la siguiente generación de individuos que se espera que estén más adaptados que la anterior generación.

La aplicación más común de los AG ha sido la de obtener una solución aproximada y factible a problemas para los cuales no se puede obtener la solución óptima ya sea porque no hay solución analítica o porque la misma tiene un alto costo computacional.

Un árbol de decisión es una estructura dinámica de datos cuyas propiedades matemáticas son las de ser un grafo conexo y acíclico, el cuál nos ayuda a elegir un camino entre varios posibles.

Un árbol de decisión puede ser visto también como un clasificador cuyo modelo forma una partición recursiva del espacio de atributos. El modelo se describe como un árbol con raíz, es decir, un árbol dirigido con un nodo distinguido llamado “raíz” para el cuál ninguna arista llega a él.

1.2. Problema

Actualmente el manejo de información tiende a incrementarse, y los tipos de patrones tienden a ser de varias clases y con muchos atributos, la necesidad de tener un algoritmo que pueda clasificar con buena precisión sin importar el tipo de atributo y una metodología que lo haga eficiente.

En esta tesis se resuelve el siguiente problema: ¿Es posible clasificar un universo de patrones con AC y AG de manera eficiente y eficaz, sin importar el tipo de patrón de tal forma que el costo computacional sea un problema tratable?.

1.3. Justificación

El diseño de clasificadores eficaces y eficientes es un problema actual y relevante, dada la gran cantidad de datos que se generan hoy en día por ejemplo, las cámaras digitales, los sistemas de vídeo vigilancia, redes sociales, entre otros, generan una gran cantidad de datos que es necesario clasificar de tal forma que se pueda obtener una representación compacta y permita una mejor toma de decisiones.

La solución propuesta en este trabajo de tesis tiene el mérito de ser eficiente con un alto nivel de precisión y además combina diferentes herramientas del tipo heurístico tales como: AC y AG.

El aporte de este trabajo de tesis es el diseño e implementación del clasificador MCJJ2 para el cuál hubo necesidad de desarrollar diferentes algoritmos tanto para el AG como para el árbol de decisión, lo que permitió tener respuestas eficientes para su respectiva tarea. También se diseñaron todos los algoritmos necesarios para crear un clasificador DS-DV.

1.4. Objetivos

Objetivo General

Diseñar e implementar un clasificador de patrones eficiente y eficaz basado en los conceptos de autómatas celulares y algoritmos genéticos.

Objetivo Particulares

- ⋆ Diseñar un algoritmo genético para la creación del clasificador de dos etapas DS-DV.
- ⋆ Diseñar algoritmos para: la creación de la población inicial, la función de evaluación, cruce y mutación de un individuo del AG para la etapa de entrenamiento del clasificador DS-DV.
- ⋆ Diseñar el clasificador MCJJ2.
- ⋆ Diseñar un AG que obtenga un DV para cada nodo del árbol de decisiones del clasificador MCJJ2.
- ⋆ Diseñar un algoritmo para la creación de un árbol de decisiones binario.

1.5. Contribuciones

Implementación del DV, el cuál es un concepto de ACMS que nos permite agrupar en dos conjuntos, aquellos patrones binarios para los cuales su distancia Hamming es menor.

Diseño e implementación de algoritmos que nos permiten obtener un clasificador DS-DV, tales como los algoritmos para la creación de la población inicial, funciones de evaluación, mutación, entre otros.

Una metodología que nos permite clasificar patrones con alto nivel de eficiencia utilizando los conceptos de AC y AG.

Un nuevo clasificador que fue nombrado MCJJ2 que mejora al clasificador de dos etapas DS-DV en los siguientes aspectos:

1. Tiempo de ejecución.
2. Precisión en la clasificación.

1.6. Estructura de la tesis

Este trabajo de tesis esta organizado de la siguiente forma:

En el capitulo 2 se dan los conceptos básicos de AC aditivos, de AG, árboles de decisión y clasificación de patrones, los cuales se ocuparan a lo largo de este trabajo de tesis y además se dan ejemplos con el objetivo de aclarar los conceptos.

En el capitulo 3, se describen los principales trabajos relativos al problema planteado en esta tesis.

En el capitulo 4, se describe la propuesta de solución al problema planteado en este trabajo de tesis.

El capitulo 5, se muestran los escenarios de simulación y los resultados obtenidos en los experimentos planteados.

El capitulo 6, describen las conclusiones y el trabajo futuro de esta tesis.

Finalmente el ANEXO A, contiene el diagrama de los sistemas que realizan la etapa de entrenamiento de los clasificadores DS-DV y MCJJ2.

Capítulo 2

ANTECEDENTES

En este capítulo se definen los conceptos básicos de AC lineales, ACMS, AG y clasificación de patrones, los cuales son necesarios para esta tesis.

2.1. Autómatas Celulares

Los AC son sistemas espacialmente descentralizados, consistentes de un gran número de componentes idénticos con conectividad local. Tales sistemas tienen el potencial de ejecutar cálculos complejos con alto grado de eficiencia y robustez.

Los AC surgieron como un área de estudio dentro del área de la ciencia teórica de la computación. Son estudiados como modelos para fenómenos físicos y biológicos, tales como, dinámica de fluidos, formación de patrones biológicos, sismos, formación de galaxias,... además de ser considerados como objetos matemáticos con propiedades formales, así como ser usados como dispositivos de cómputo paralelo para simulación de modelos.

La mayoría de los modelos con autómatas celulares poseen las cinco características siguientes:

- Una ***lattice* de células discretas**: El sistema consiste de una estructura llamada *lattice* la cual puede ser de dimensión d, donde $d \in \mathbb{N} \cup \{0\}$.
- **Homogeneidad**: Todas las células son equivalentes.
- **Estados discretos**: Cada célula toma un estado de un conjunto finitos de estados discretos.
- **Interacciones locales**. Cada célula interactua sólo con las células que están en su vecindad.
- **Sistemas dinámicos discretos**. En cada paso de tiempo discreto, cada célula actualiza su estado actual de acuerdo a una función o regla de evolución que toma como entrada los estados de las células vecinas y da como salida un estado del conjunto de estados.

***Definición* 2.1.1** *Un Autómata Celular (AC) es una 5-tupla $(L, \mathcal{D}, S, H, f)$ donde:*

- *L es una lattice de dimensión d.*
- *$\mathcal{D} \in \mathbb{N} \cup \{0\}$ y es la dimensión del autómata.*
- *S es un conjunto finito de elementos llamados estados y es denotado por:*
  $S = \{s_k : k \in \{0, \ldots, |S| - 1\}\}$
  donde $|S|$ es la cardinalidad del conjunto de estados S.
- *H es un subconjunto finito de Z^d llamado vecindad y es denotado por:*
  $\{v_j = x_{1,j}, \ldots, x_{d,j} : j \in 1, \ldots, |H|\}$, donde los elementos v_j son llamados vectores vecindad.
- *f es una función de $S^{|H|}$ en S, llamada la función de evolución o regla.*
  i.e. $f : S^{|H|} \to S$

***Definición* 2.1.2** *En un AC se dice que posee fronteras nulas, si el vecino de la izquierda (o de la derecha) de la célula del extremo izquierdo (o derecho) se considera siempre como cero.*

***Definición* 2.1.3** *En un AC se dice que posee fronteras periódicas, si los extremos derecho e izquierdo son adyacente el uno del otro.*

Ejemplo 2.1.1 ***AC de una dimensión [1].***

Para el AC de una dimensión, el valor de la $i-$ésima célula en el tiempo t denotado por $c_i(t)$ de acuerdo a la función de evolución f que es una función de $c_i(t)$ y otras celdas que están dentro del radio r (a la derecha e izquierda) de $c_i(t)$:

$$c_i(t) = f\left(c_{i-r}(t-1), c_{i-r+1}(t-1), ..., c_{i+r-1}(t-1), c_{i+r}(t-1)\right)$$

Cada célula puede tomar un sólo valor de los k posibles valores, $c_i(t) \in \{0, 1, 2, ..., k-1\}$ la función de evolución f es definida completamente especificando el valor que se le debe de asignar a cada una de las k^{2r+1} posibles configuraciones de las $(2r+1)$-tuplas dado el radio r de la vecindad.

Como f asigna cualquiera de los k valores a cada una de las k^{2r+1} posibles configuraciones de las $(2r+1)$-tuplas, hay en total $k^{k^{2r+1}}$ posibles funciones de evolución.

Para el caso particular con $r = 1$ y $k = 2$ (ie. $c_i \in \{0, 1\}$), hay $2^8 = 256$ posibles funciones de evolución.
Observemos el comportamiento de una de estas reglas.

Como $r = 1$, tenemos que la vecindad esta conformada por una 3-tupla como se muestra en la figura 2.1.

Figura 2.1: AC uno dimensional donde el recuadro rojo indica la vecindad de radio $r = 1$ para la célula marcada con una estrella.

En la tabla 2.1 se muestran todas las configuraciones para una vecindad de cardinalidad tres. La regla de evolución mostrada en la tabla 2.1 puede ser interpretada como la regla de evolución 109 si consideramos que la función de evolución es un número escrito en base dos, esto es: $01101101_2 = 109_{10}$.

Tabla 2.1: Todas las posibles configuraciones de vecindades.

Vecindad	**Resultado**	**Vecindad**	**Resultado**
000	1	100	0
001	0	101	1
010	1	110	1
011	1	111	0

En la figura 2.2 se muestra el AC de una dimensión para la regla 109.

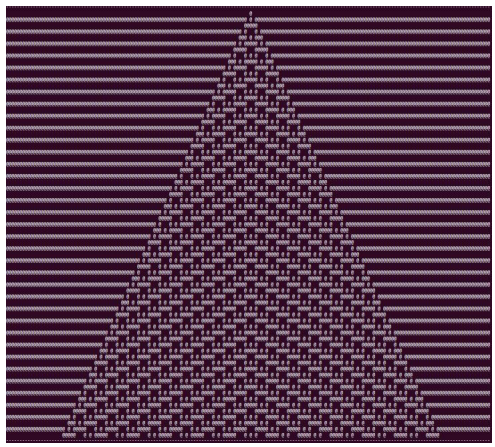


Figura 2.2: Evolución de la regla 109.

Clases de Wolfram

El comportamiento de un AC es muy parecido al comportamiento de sistemas dinámicos continuos. Existe evidencia que sugiere que los patrones generados por todos los AC de una dimensión que evolucionan a partir de cualquier estado inicial caen en uno de 4 comportamientos básicos llamados clases:

1. Atractor punto. La evolución del AC conduce a un estado homogéneo, en el cual todas las celdas eventualmente llegan a un mismo estado. En la figura 2.3a se muestra un ejemplo de esta clase.

2. Atractor cicló. La evolución del AC conduce a estados simples y estables o periódicos y estructuras separadas. Después de un cierto número de pasos, el AC caé en configuraciones periódicas. En las figuras 2.3b y 2.3c se muestran ejemplos de esta clase.

3. Caos. La evolución del AC conduce a un estado caótico y no periódico, ninguna configuración se repite. En las figuras 2.3d y 2.3e se muestran ejemplos de esta clase.

4. Comportamiento emergente. La evolución del AC conduce a configuraciones complejas. Cualquier mezcla de las anteriores, la dinámica que se genera no es uniforme. En la figura 2.3f se muestra un ejemplo de esta clase.

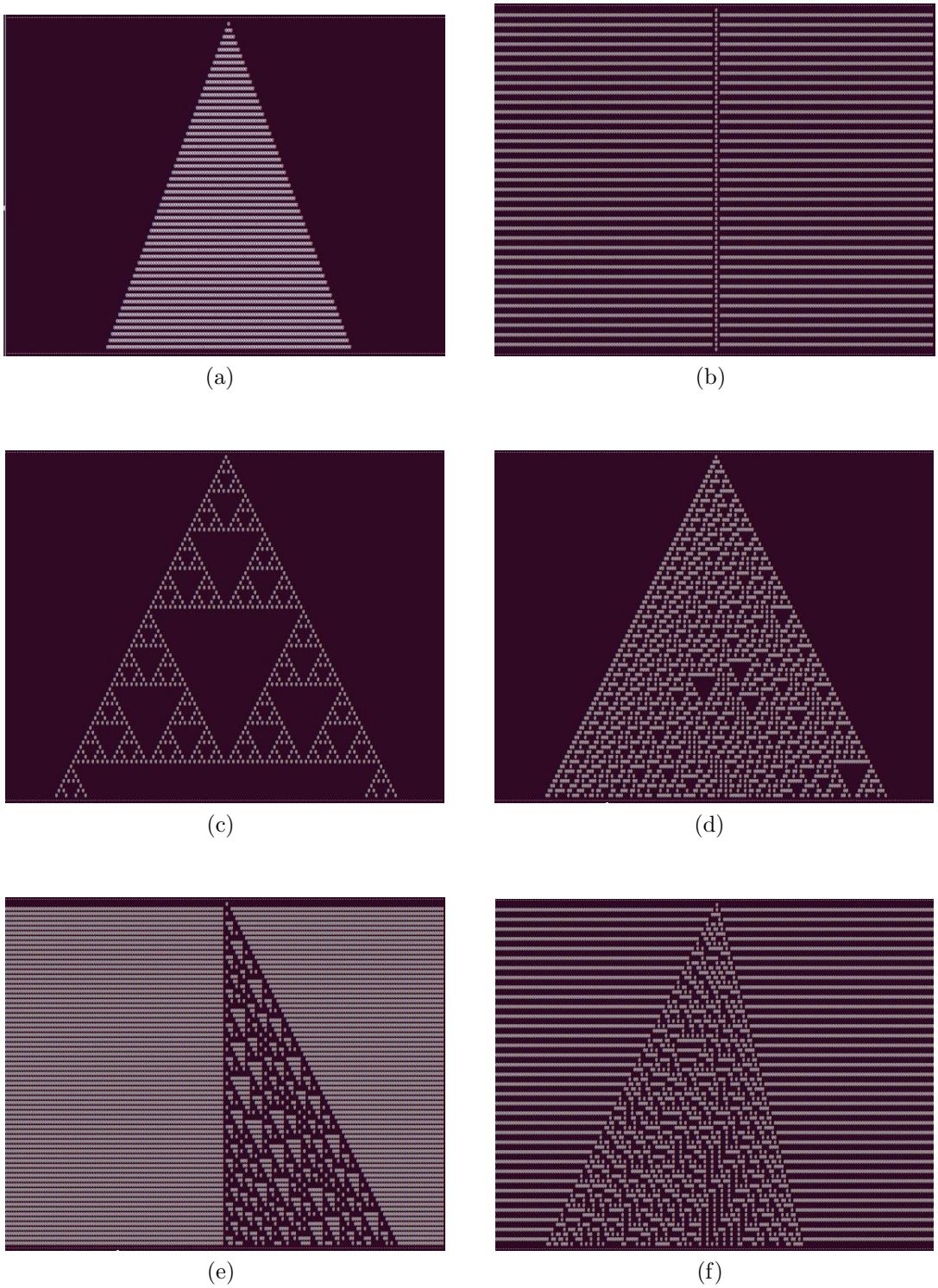


Figura 2.3: Autómatas celulares de uno-dimensinal. (a) regla = 254 (b) regla = 5 (c) regla = 90 (d) regla = 30 (e) regla = 193 (f) regla = 101

Ejemplo 2.1.2 ***AC de dos dimensiones (Juego de la Vida)***

El juego de la vida es un AC diseñado por el matemático británico John Horton Conway en 1970. El juego de la vida de Conway es un autómata binario de dos dimensiones cuya vecindad de Moore consta de nueve células; la célula del centro y las ocho células que le son adyacentes (ver figura 2.4). El criterio para escoger la regla de evolución fue que el campo de células no debería disminuir hasta la nada (todas cero), ni llenarse completamente (todas uno).

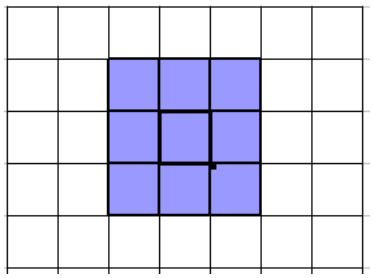


Figura 2.4: Vecindad de Moore con radio igual a uno.

El Juego de la vida consiste en:

1. Una retícula donde cada cuadro de la *lattice* representa una célula que puede estar viva o muerta.

2. Simultáneamente (y repetidamente) se aplican en pasos de tiempo discreto las siguientes tres reglas a cada célula de la *lattice*:

   - ⋆ **Nacimiento**. Reemplazar una célula muerta por una viva, si exactamente 3 de sus vecinas están vivas.
   - ⋆ **Muerte**. Reemplazar una célula viva por una muerta, si alguna de las siguientes situaciones se dan (i) si tiene uno o menos células vecinas vivas, muere por aislamiento y (ii) si tiene más de tres células vecinas vivas, muere por sobrepoblación.
   - ⋆ **Supervivencia**. La célula permanece viva si tiene exactamente dos o tres vecinas vivas.

Con estas reglas se puede obtener una gran variedad de comportamientos de este AC, en las figuras 2.5 y 2.6 se muestra la evolución en diferentes etapas del tiempo del autómata del juego de la vida.

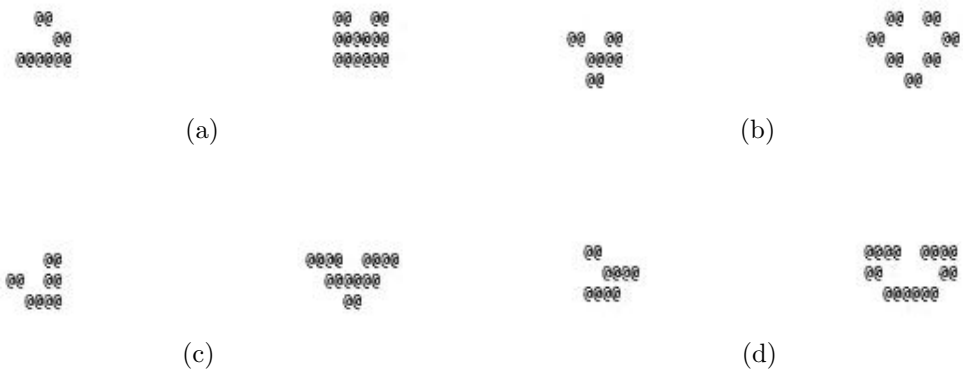
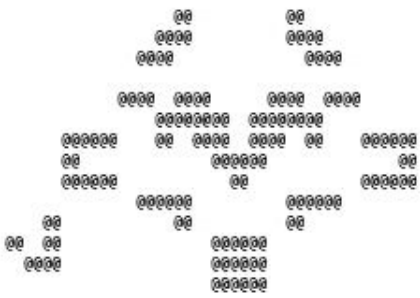
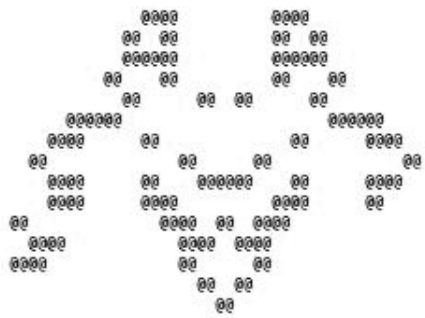


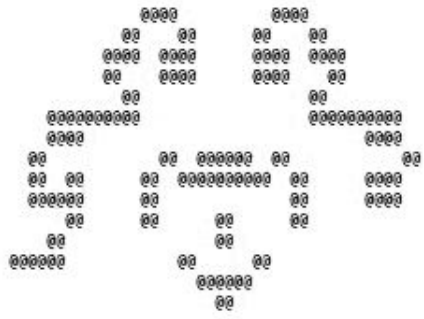
Figura 2.5: Evolución del Juego de la vida para los primeros valores de tiempo (a) tiempo = 0, (b) tiempo = 1, (c) tiempo = 2, (d) tiempo = 3



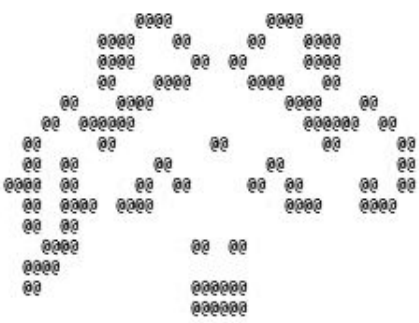
(a)



(b)



(c)



(d)

Figura 2.6: Evolución del Juego de la vida para otros valores de tiempo; (a) tiempo = 26, (b) tiempo = 27, (c) tiempo = 28, (d) tiempo = 29

2.2. AC Lineales

***Definición* 2.2.1** *Si la regla de un AC involucra sólo a el operador lógico XOR, entonces es llamada regla lineal. Si la regla involucra al operador XNOR lógico (en donde existe una inversión de la lógica XOR) entonces es llamada regla complementaria. Un AC en donde todas las células tienen reglas lineales es llamado AC lineal. Si en un AC tiene una combinación de reglas lineales y complementarias es llamado AC aditivo [2].*

***Definición* 2.2.2** *Si en un AC la misma regla aplica para todas las celdas, entonces el AC es llamado uniforme o regular, de otra forma es llamado híbrido.*

En una vecindad de tamaño tres, el siguiente estado $q_i(t+1)$ de una célula depende sólo de si misma y de sus dos vecinas (de la derecha e izquierda) y se denota por:

$$q_i(t+1) = f\left(q_{i-1}(t), q_i(t), q_{i+1}(t)\right)$$

Donde:
$q_i(t)$ representa el estado de la $i-$ ésima célula en el tiempo t.
f es la función de evolución para obtener el siguiente estado.

De ahora en adelante se utilizarán autómatas celulares híbridos (ver definición 2.2.2) y con fronteras nulas.

Los AC cuya función de evolución se puede expresar sólo con XOR son llamados AC lineales (ver definición 2.2.1). A continuación, en la tabla 2.2 se muestran las siete reglas del AC lineal que se pueden expresar con el XOR lógico.

Tabla 2.2: Reglas para los AC lineales.

Número de regla	**Función de evolución**
Regla 60	$q_i(t+1) = q_{i-1}(t) \oplus q_i(t)$
Regla 90	$q_i(t+1) = q_{i-1}(t) \oplus q_{i+1}(t)$
Regla 102	$q_i(t+1) = q_i(t) \oplus q_{i+1}(t)$
Regla 150	$q_i(t+1) = q_{i-1}(t) \oplus q_i(t) \oplus q_{i+1}(t)$
Regla 170	$q_i(t+1) = q_{i+1}(t)$
Regla 204	$q_i(t+1) = q_i(t)$
Regla 240	$q_i(t+1) = q_{i-1}(t)$

2.2.1. Caracterización de un AC lineal

El estado S_t de un AC de n células en el tiempo t es una cadena binaria de símbolos de tamaño n. El siguiente estado S_{t+1} del AC esta dado por:

$$S_{t+1} = T \cdot S_t$$

Dónde T es una matriz característica de tamaño $n \times n$, también conocida como: Matriz de Dependencia.

$$T_{ij} = \begin{cases} 1 & \text{Si el siguiente estado de la celula } i^{th} \text{depende del presente estado de la celula } j^{th}. \\ 0 & \text{cualquier otro caso.} \end{cases} \tag{2.1}$$

Ejemplo 2.2.1 ***Para un AC con el siguiente vector de estados:*** $< 204, 170, 90, 240, 204 >$***, y considerando fronteras nulas, obtener la matriz característica.***

La matriz característica de este AC se obtiene como sigue:

i,j	0	1	2	3	4	Regla que se aplica
0	1	0	0	0	0	$q_i(t+1) = q_i(t)$
1	0	0	1	0	0	$q_i(t+1) = q_{i+1}(t)$
2	0	1	0	1	0	$q_i(t+1) = q_{i-1}(t) \oplus q_{i+1}(t)$
3	0	0	1	0	0	$q_i(t+1) = q_{i-1}(t)$
4	0	0	0	0	1	$q_i(t+1) = q_i(t)$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Así se obtiene la siguiente matriz de dependencia T, con la cual se puede obtener los estados siguientes del AC; por ejemplo, para un estado inicial $S_0 = [0, 1, 1, 0, 0]$, si se quiere obtener los dos estados siguientes se procede como sigue a continuación:

$$S_1 = TS_0^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0, & 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0, & 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0, \\ 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0, & 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 & \end{bmatrix} = [0, 1, 1, 1, 0]$$

Donde: S_0^* es la transpuesta de S_0

$$S_2 = TS_1^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0, & 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0, & 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0, \\ 0 \oplus 0 \oplus 1 \oplus 0 \oplus 0, & 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 & \end{bmatrix} = [0, 1, 0, 1, 0]$$

Si restringimos a una vecindad de tres vecinos, entonces T_{ij} va a tener valores no ceros sólo para $j = (i-1), i$ y $(i+1)$. Así, T se vuelve una matriz tridiagonal. El polinomio $\phi(x)$

del cual T es raíz es llamado: polinomio característico del AC. El polinomio característico se obtiene de T calculando $det(T + Ix)$, donde I es la matriz identidad.

Si todos los estados de un AC vistos en un diagrama de transiciones caen en algún ciclo, entonces se dice que es un AC grupo, mientras un AC no grupo, el diagrama de transiciones tiene ambos comportamientos cíclico y no cíclico. Para un AC grupo $det[T] \neq 0$; mientras para un AC no grupo, $det[T] = 0$.
En este trabajo se va a ocupar una clase especial de AC no grupo llamado autómata celular con múltiple sumideros (ACMS) para el diseño del clasificador.

2.2.2. Autómata celular de múltiples sumideros (ACMS)

El grafo de transición de un ACMS consiste en un número de estados cíclicos y no cíclicos. El conjunto de estados no cíclicos de un ACMS forman un árbol invertido con la raíz en los estados cíclicos . Los estados que se ciclan a si mismos son llamados sumideros. En la figura 2.7 se muestra el diagrama de estados de transición de un ACMS de cinco células, su vector de reglas es $< 102, 60, 204, 204, 240 >$. Los cuatro estados cíclicos $00000(0), 00011(3), 00100(4), 00111(7)$ son llamados sumideros y el conjunto de estados que conforman los árboles invertidos correspondientes a cada sumidero son α ($\alpha = 0, 3, 4, 7$) a los cuales les llamaremos cuencas [7].

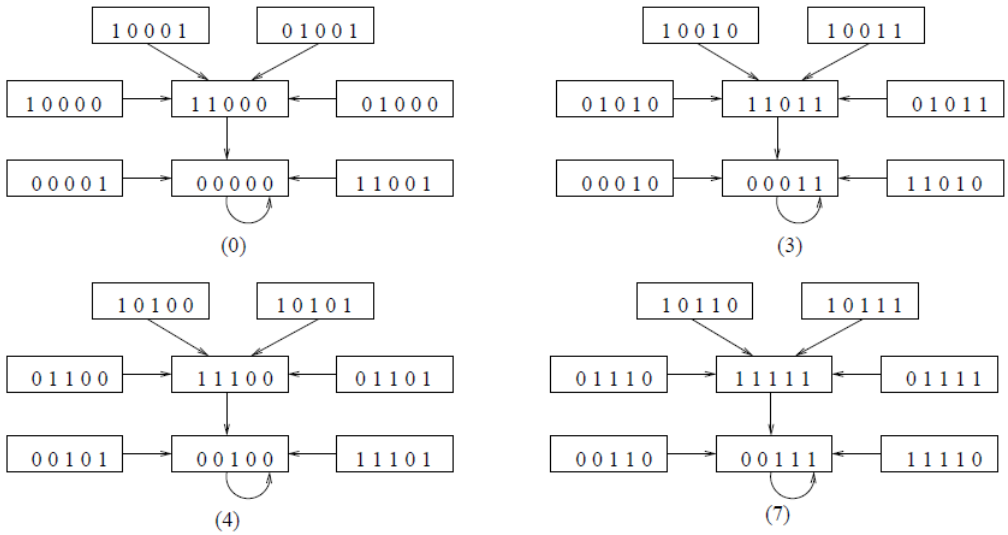


Figura 2.7: Diagramas de transición de estados.

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix}, \text{ Donde } T_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ y } T_2 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \quad (2.2)$$

***Definición* 2.2.3** *La profundidad d de un ACMS es el número de estados entre un determinado estado j y un sumidero.*

A continuación se muestran algunos resultados fundamentales para un ACMS de n células con k número de sumideros [6]:

- **Resultado I:** El polinomio característico de un ACMS de n células y k sumideros es $((x^{n-m}(1+x)^m)$, donde $m = \log_2(k)$.
- **Resultado II:**

  ***Teorema* 2.2.1** *Para un ACMS de n células y $k = 2^m$ sumideros, existen m bits para las cuales los sumideros generan 2^m patrones exhaustivos (PE).*

- **Resultado III:** El conjunto de patrones exhaustivos (CPE) de un sumidero sirven como un identificador de su respectivo sumidero. Para identificar la clase a la que pertenece el estado P, el ACMS es inicializado con P y operado hasta un máximo de d (profundidad) número de iteraciones para encontrar un sumidero. Después, los bits del PE pueden ser extraídos para identificar la clase de P. En general, d es definido como el número de iteraciones que un ACMS necesita para encontrar un estado de sumidero cuando este es inicializado con un estado sin clasificar.
- **Resultado IV:**

  ***Teorema* 2.2.2** *La suma modulo-dos de dos estados es un predecesor del estado cero (patrón de sólo ceros), si y sólo si, los dos estados pertenecen a la misma cuenca.*

  ***Teorema* 2.2.3** *La cuenca cero de un ACMS de n bits con 2^m sumideros forman un vector de dimención $(n-m)$.*

- **Resultado V:** La matriz tridiagonal de dependencia (T) posee el siguiente polinomio característico $((x^{n-m}(1+x)^m)$ puede ser obtenido de m matrices de dependencia T_i $(i = 1, 2, ..., m)$ acomodados en una diagonal de bloques como se muestra en la ecuación 2.3, donde cada T_i tiene su polinomio característico $(x^{n_i-1}(1+x))$ y $n_1+n_2+\ldots+n_m = n$.

$$T = \begin{bmatrix} [T_1] & & & \\ & [T_2] & & \\ & & \ddots & \\ & & & [T_m] \end{bmatrix} \tag{2.3}$$

Ejemplo 2.2.2 ***Utilizando todos estos resultados para el ACMS de la figura 2.7 que tiene como matriz de dependencia la mostrada en 2.2 se tiene lo siguiente:***

- ⋆ Es un ACMS de cinco células ($n = 5$) que tiene cuatro sumideros ($k = 4$) y una profundidad $d = 2$.
- ⋆ Como $k = 4$ entonces, $m = \log_2(k) = \log_2(4) = 2$ por lo tanto, $m = 2$ por lo tanto, el polinomio característico queda como sigue:

$$\left(x^{n-m}(1+x)^m\right) = x^3(1+x)^2$$

- ⋆ De la figura 2.7, los bits de las posiciones tres y cuatro forman el CPE.
- ⋆ Considere el estado 10010.
  El ACMS tiene como estado inicial el estado 10001 y se cicla dos veces (dado que su profundidad es $d = 2$) y encuentra al sumidero 00011. Para el cual los bits CPE son 01.
- ⋆ De acuerdo al teorema 2.2.2, la suma modulo-2 de un par de estados 10110 y 11110 que pertenecen al mismo sumidero 00111, es: $(10110) \oplus (11110) = (01000)$, el cual es un estado que pertenece a la cuenca cero. Por el contrario, si tomamos dos estados 10100 y 10111 que pertenecen a cuencas diferentes 00100 y 00111 respectivamente, su suma modulo-2 es: $(10100) \oplus (10111) = (00011)$ el cual es un estado que no pertenece a la cuenca cero ya que pertenece a la cuenca tres (que pertenece al sumidero 00011).
- ⋆ En la cuenca cero,(i) el vector cero esta siempre presente; (ii) la suma modulo-2 de cualquiera dos estados que se encuentren en la cuenca cero pertenece sólo a la cuenca cero. Estas dos propiedades hacen la cuenca cero un vector subespacio.

2.3. ACMS como clasificador de patrones

Un clasificador es entrenado con ciertos conjuntos de patrones $P = \{P_1, \ldots, P_i, \ldots, P_k\}$ que pertenecen a ciertas clases $\{C_1, \ldots, C_i, \ldots, C_k\}$ respectivamente. A esta fase se le llama fase de entrenamiento del clasificador.

Un ACMS de tamaño n bits con k sumideros puede ser visto como un clasificador. Un ACMS clasifica un conjunto de patrones en k clases distintas; cada clase contiene un conjunto de estados que van a caer en ese sumidero. En otras palabras, un ACMS con k sumideros puede ser diseñado para clasificar un conjunto de patrones que estén divididos en a lo más k clases.

El ACMS del ejemplo, el cual se diseño para ser un clasificador de 2 clases, donde la clase I es representada por el conjunto de sumideros $(00001, 10001, 10000)$ y la clase II representada por el sumidero 00000. Cuando el ACMS es inicializado con un patrón $p_i = (00010) \in P$ y el AC es ejecutado por un número de ciclos igual a su profundidad d, el AC llegara a una configuración (00000) que representa la clase II, que es la clase a la que pertenece p_i. Los bits CPE (el primer bit y el ultimo, que es 00) es el identificador de la clase a la que pertenece p_i.

Los bits CPE nos sirven para etiquetar una clase, es decir es suficiente con identificar cuales son los bits CPE para saber a que clase pertenece p_i.

A continuación veremos las restricciones que debe de cumplir el ACMS para el diseño de un clasificador que clasifica en dos clases.

2.3.1. Condiciones de diseño

Para el diseño de un clasificador de dos clases basado en un ACMS se deben de cumplir las siguientes condiciones:

- **DC1**. Dado dos conjuntos de patrones S_1 y S_2 que requieren ser clasificados (en dos clases), los elementos de cada conjunto deben de caer en diferentes sumideros del ACMS.
- **DC2**. Cualquier patrón P_{inc} debe de caer en la clase para la cual su distancia de *Hamming* es menor. La distancia de *Hamming* entre un patrón P_{inc} y la clase C_1 o C_2 es medida como el promedio de las distancias de P_i con cada elemento del conjunto de patrones S_1 y S_2 y es llamada como AHD.

  $$AHD\left(P_{inc}, C_{1(2)}\right) = \frac{\sum_{i=1}^{k_{1(2)}} \left(HD(P_{inc}, P_{1(2)i})\right)}{k_{1(2)}} \qquad \forall P_{1(2)i} \in P_{1(2)}$$

  Donde:
  $P_{1(2)}$ es un conjunto de patrones que conforman la clase $C_{1(2)}$
  y $k_{1(2)} = |P_{1(2)}|$.

  Entonces se debe de cumplir la siguiente relación:
  $AHD\left(P_{inc}, S_1\right) < AHD\left(P_{inc}, S_2\right)$, si P_{inc} tienen la menor distancia con respecto a S_1

Cumplimiento de **DC1**: Para el diseño de un clasificador basado en un ACMS que clasifica dos conjuntos de patrones S_1 y S_2 se debe asegurar que todos los elementos de una clase, digamos S_1, pertenecen al conjunto de sumideros que no contienen ningún elemento de la clase S_2. Es decir, cualesquiera dos patrones $a \in S_1$ y $b \in S_2$ deberían de caer en diferentes sumideros.
Para asegurar que cualquiera dos patrones $a \in S_1$ y $b \in S_2$ caerán en diferentes sumideros, el patrón obtenido de la suma modulo-2 de a y b $(a \oplus b)$ debe de caer en una cuenca que no sea la cuenca cero (De acuerdo al teorema 2.2.2). Sea X un conjunto cuyos elementos son resultado de la suma modulo-2 de cada elemento de S_1 con cada elemento de S_2. Entonces, cada elemento de X debe de caer un una cuenca no cero. Esto implica que la siguiente ecuación debe de cumplirse:

$$T^d \cdot x \neq 0 \qquad \forall x \in X, \qquad X = \{x \mid x = (a_i \in S_1) \oplus (b_j \in P_2) \forall i, j\}$$

Donde: T es un ACMS que se empleara para el diseño de un clasificador de dos clases, y d es la profundidad del ACMS.

Cumplimiento de **DC2**: Un ACMS mantiene una relación inversa entre la colisión de dos patrones y sus distancias de *Hamming* [19].
La distancia de *Hammning* es medida entre dos conjuntos ordenados cuyos elementos son valores discretos. Entonces, como se ve en la definición 2.3.1 distancia de *Hammning* indica cuantas posiciones de dos vectores son diferentes.

***Definición* 2.3.1** *Dado dos conjuntos,* $X = \{x_1, x_2, \ldots, x_n\}$ *y* $Y = \{y_1, y_2, \ldots, y_n\}$ *la distancia de Hamming* (∂_H) *se define como [1]:*

$$\partial_H(X, Y) = \sum_{i=1}^{n} \mathcal{X}_i \tag{2.4}$$

Donde: $\mathcal{X}_i = 1$, *si* $x_i \neq y_i$ *y* $\mathcal{X}_i = 0$ *en otro caso.*

El concepto de HHF proporciona una propiedad importante a las cuencas de sumideros del ACMS. Como las llaves(patrones) que están más cercanas se dirigen hacia el mismo sumidero, las llaves (patrones) que están más cercanas una de la otra en términos de la distancia *Hamming* es más probable que pertenezcan a la misma cuenca. Por esta razón cuando un patrón P_{inc} es dado como entrada, es más probable que este vaya a caer en el mismo sumidero en el cual caen los patrones con los que tiene menor distancia de *Hammning*.

Ejemplo 2.3.1 ***Consideremos los siguientes conjunto de patrones:***
$S_1 = \{11111, 01111, 11010\}$ ***y*** $S_2 = \{01000, 01010\}$
Se debe clasificar S_1 ***y*** S_2 ***en dos clases distintas, Clase I y Clase II respectivamente, para lograr esto necesitamos diseñar un ACMS tal que, cada patrón de cada clase caiga en diferente cuenca de sumidero del ACMS.***

El ACMS con el vector regla $< 204, 170, 90, 240, 204 >$ puede clasificar S_1 y S_2 en distintas cuencas de sumidero, donde la clase I es representada por el conjunto de sumideros: $\{00001, 10001, 10000\}$, mientras que la Clase II es representada por el sumidero: $\{00000\}$.
Cuando el ACMS es inicializado con un patrón $x = 00010$ y iterado d ciclos, el ACMS llega hasta un sumidero.

La distancia de *Hamming* entre el patrón x y la clase I es:

$$\partial_H(x, P_{11}) = 4, \quad \partial_H(x, P_{12}) = 3, \quad \partial_H(x, P_{13}) = 2 \quad \Rightarrow AHD(x, C_1) = \frac{4+3+2}{3} = 3$$

$$\partial_H(x, P_{21}) = 2, \quad \partial_H(x, P_{22}) = 1, \quad \Rightarrow AHD(x, C_2) = \frac{2+1}{2} = 1{,}5$$

Como $AHD(x, C_2) < AHD(x, C_1)$ la entrada x cae en el sumidero 00000, que es el sumidero que pertenece a la clase II.

- ⋆ **DC3**. Para un clasificador que distingue entre dos clases diferentes, es necesario emplear sólo un bit como CPE, para hacer la distinción entre las dos clases en cuestión. El ejemplo 2.3.1 pone en evidencia una condición de diseño adicional; ya que para

clasificar dos clases diferentes hay en total 4 cuencas de sumidero, es decir, se necesitan de dos bits de CPE para hacer la clasificación, lo cual significa un desperdicio de memoria. Es por esto que en el diseño de un clasificador ACMS, hay que tomar en cuenta la minimización de desperdicio de memoria.

2.4. Vector de dependencia (DV) y Cadena de dependencia (DS)

En un subespacio de dimensión $(n - m)$, si el conjunto de vectores de n bits, es representado como un sistema de ecuaciones de n variables, entonces va a existir un total de m variables dependientes.

Ejemplo 2.4.1 ***Relaciones dependientes***

Consideremos el subespacio vectorial $V = \{00000, 01001, 10001, 11000, 00110, 01111, 10111, 11110\}$. Todos los vectores $v_i \in V$ son de cinco bits de longitud $(n = 5)$. La dimensión del subespacio vectorial es tres. Por lo tanto $m = 5-3 = 2$. Si el conjunto vectorial V es representado como un sistema de ecuaciones lineales de cinco variables (a, b, c, d, e), entonces, los elementos se pueden representar como se muestra en la tabla 2.3. Para el conjunto de ecuaciones de la tabla 2.3 [2],
(1) $a \oplus b \oplus e = 0$ para todos los vectores $v_i \in V$; esto es, que a depende de b y e, y
(2) $c \oplus d = 0$ para todos los vectores $v_i \in V$; esto es, que c depende de d. Por lo tanto, existen $2m$ número de relaciones dependientes en el subespacio vectorial.

Tabla 2.3: Sistema de ecuaciones correspondiente al conjunto V

Vector	Ecuación correspondiente
00000	$0 \cdot a + 0 \cdot b + 0 \cdot c + 0 \cdot d + 0 \cdot e$
01001	$0 \cdot a + 1 \cdot b + 0 \cdot c + 0 \cdot d + 1 \cdot e$
10001	$1 \cdot a + 0 \cdot b + 0 \cdot c + 0 \cdot d + 1 \cdot e$
11000	$1 \cdot a + 1 \cdot b + 0 \cdot c + 0 \cdot d + 0 \cdot e$
00110	$0 \cdot a + 0 \cdot b + 1 \cdot c + 1 \cdot d + 0 \cdot e$
01111	$0 \cdot a + 1 \cdot b + 1 \cdot c + 1 \cdot d + 1 \cdot e$
10111	$1 \cdot a + 0 \cdot b + 1 \cdot c + 1 \cdot d + 1 \cdot e$
11110	$1 \cdot a + 1 \cdot b + 1 \cdot c + 1 \cdot d + 0 \cdot e$

***Definición* 2.4.1** *Vector de dependencia (DV por sus siglas en ingles Dependency Vector) representa la dependencia lineal de todos los elementos del subespacio V [19].*

Cada uno de los n bits del DV representan el orden de todas las variables en secuencia $< abcde >$. Los dos vectores de dependencia para el ejemplo anterior son:

- 11001 que corresponde a las variables dependientes a, b y e.
- 00110 que corresponde a las variables dependientes c y d.

Los 1's en el DV denotan las variables dependientes; que son,

$$DV \cdot v_i^t = 0 \quad \forall v_i \in V$$

***Definición* 2.4.2** *Cadena de dependencia (DS por sus siglas en ingles Dependency String) representa la múltiple dependencia lineal, si es que existe en el subespacio (V). La cadena de dependencia en el ejemplo anterior es:* 11221 *donde los 1's indican la relación entre a, b y e mientras que los 2's indican la relación entre c y d. En esencia, los dos vectores de dependencia (DV) se unen para formar la cadena de dependencia (DS).*

El DV y el DS tienen una connotación importante con respecto a la cuenca 0 del ACSM. La cuenca 0 de un ACSM$(n, 1)$, que tiene dos cuencas de sumidero, es un subespacio de dimensión $(n-1)$. De esta forma, existe una variable dependiente en el subespacio de vectores y entonces la caracterización de la cuenca 0 puede ser únicamente realizado por un DV. La cuenca 0 de un ACSM(n, m) es un subespacio de dimensión $(n - m)$. Es por eso que hay m variables dependientes en el subespacio. El DS que representa varios DV, es necesario para la caracterización de un ACSM(n, m).

***Definición* 2.4.3** *Vector de dependencia válido (LDV Legal Dependency Vector) es un vector el cual no puede tener 2 o más ceros entre dos 1's.*

Ejemplo 2.4.2 ***DV's válidos***

Algunos ejemplos de DV's válidos son: 101011, 11010111, 001100, 0101000 etc. mientras que algunos ejemplos de DV que no pueden ser generados por un ACSM de tres vecinos son: 10011, 1001001. En ambos casos existen dos o más ceros entre un par de 1's.

2.4.1. Cadena de dependencia (DS) para identificación del CPE de un sumidero.

Si DV es un vector de dependencia de n bits de dimensión $(n - 1)$ y P es un patrón de tamaño n bits, entonces la suma modulo 2 (XOR) de las variables dependientes de P (donde DV contiene 1's) es igual a cero, si P pertenece a la cuenca 0; y es 1 en otro caso, esto es:

$$DV \cdot P = \begin{cases} 0 & \text{Si } P \in \text{cuenca 0} \\ 1 & \text{Si } P \in \text{cuenca no 0} \end{cases} \tag{2.5}$$

Ejemplo 2.4.3 *Si $DV = 10111$ y $P = 11001$, entonces $DV \cdot P = 10111 \cdot 11001 = 0$; por lo tanto $P \in$ cuenca 0. Entonces un ACSM con polinomio característico $x^{n-1}(1 + x)$ puede ser representado por un DV de tamaño n bits.*

Un DS que consiste de m números de DV's: $DV_1, DV_2, \ldots, DV_m$, representan un ACSM con polinomio característico $x^{n-m}(1+x)^m$ (figura 2.8). De acuerdo al teorema 2.2.1, el número de PE bits va a ser m. Cada DV contribuye en un bit (0 ó 1) al CPE de un sumidero.

Un DS de n bits es generado por la concatenación de m número de DV's de tamaño $n_1, n_2, \ldots, n_m$ respectivamente (figura 2.8), donde $n_1 + n_2 + \ldots + n_m = n$ y P es un patrón de tamaño de n bits para el cual el sumidero va a ser definido. Entonces, para cada DV_i (de tamaño n_i), las variables dependientes de los correspondientes n_i bits de P (P_i).

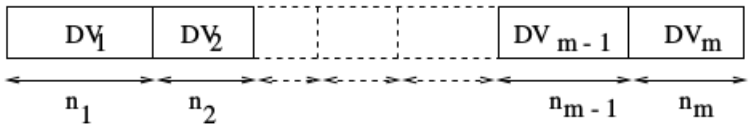


Figura 2.8: Un DS de tamaño $n-$bits que consta de m DVs. Polinomio característico $x^{n-m}(1+x)^m$

$$DV_i \cdot P_i = \begin{cases} 0 & \text{Si } P_i \in \text{cuenca 0 de } DV_i \\ 1 & \text{Si } P \in \text{cuenca no 0 de } DV_i \end{cases} \tag{2.6}$$

Finalmente, el bit resultante es el valor del i_a bit del CPE. Entonces una cadena de m bits puede ser obtenida de m números de DVs. Y esta cadena binaria de m bits es el CPE del sumidero al que pertenece P.

Así, el CPE del sumidero al que pertenece P esta dado por:

$$CPE = DS \cdot P \tag{2.7}$$

Ejemplo 2.4.4 ***Para el ACSM mostrado en la figura 2.7. Supongase que se se desea identificar el sumidero para el patrón*** $P = 10101$***.***

Primero, obtenemos el DS que es 00120. El DS esta formado a su vez por dos DVs (001 y 10) de tamaño 3 y 2 respectivamente. La suma modulo 2 (XOR) de las 3 variables dependientes que corresponde a los primeros tres bits de P (P_1) contribuye al primer bit del CPE y los bits restantes contribuyen al segundo bit del CPE, como se muestra a continuación:

$$\left.\begin{array}{rcccl} (101)\cdot(001) & = & 0\oplus 0\oplus 1 & = & 1 \\ (01)\cdot(10) & = & 0\oplus 0 & = & 0 \end{array}\right\} \quad CPE = 10$$

2.4.2. Clasificador de dos etapas basado en un ACSM

Un clasificador de 2 clases basado en un ACSM debe de asegurar que para dos conjuntos S_1 y S_2 que contienen patrones de tamaño n bits los elementos de una clase (S_1) se encuentran en los sumideros que no incluyen algún miembro de la clase S_2 y vice versa. En consecuencia, para cualquier patrón de n bits $P_1 \in S_1$ y $P_2 \in S_2$ deben de caer en diferente cuencas de sumidero.

Sea un ACSM que tiene el siguiente polinomio característico: $x^{n-m}(1+x)^m$ que puede clasificar dos patrones S_1 y S_2, esto es:

$$DS \cdot P_1 \neq DS \cdot P_2 \tag{2.8}$$

Donde DS es una cadena de dependencia de n bits que consiste de m DV's. Entonces, el número total de cuencas de sumideros va a ser 2^m, el CPE de cada cuenca de sumidero

va a ser de tamaño m bits. La primera etapa del clasificador mapea el patrón de n-bits a un patrón de m bits que es el CPE de la cuenca de sumidero al que pertenece el patrón.

Después para la segunda etapa el patrón de m bits es clasificado en dos diferentes clases. Sean k_1 y k_2 dos conjuntos de patrones de m-bits que son los CPE de los sumideros de dos patrones de n bits que pertenecen a los conjuntos S_1 y S_2 respectivamente. Cualquier patrón de m bits $p_1 \in k_1$ y $p_2 \in k_2$ debería de caer en diferentes cuencas (mientras uno cae en la cuenca cero el otro cae en la cuenca no cero), esto es:

$$DV \cdot p_1 \neq DV \cdot p_2 \tag{2.9}$$

Donde DV es un vector de dependencia de m bits.
En la figura 2.9 se muestra la arquitectura del clasificador de dos etapas. Este consiste en tres capas: entrada, oculta y salida, denotadas como $x_i(i = 1, 2, \ldots, n)$ y $y_j(j = 1, 2, \ldots, m)$ respectivamente.

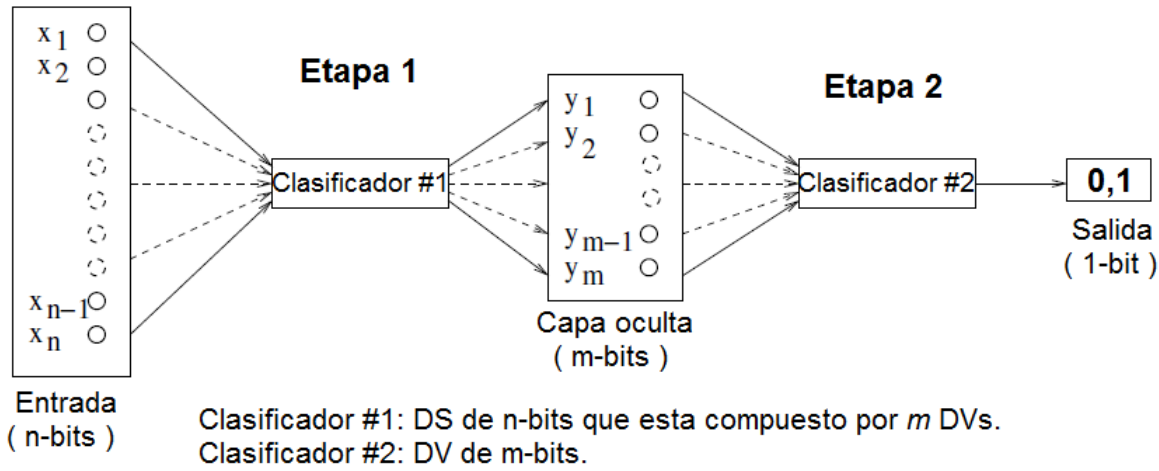


Figura 2.9: Clasificador de dos etapas.

El primer clasificador mapea un patrón de n bits a un patrón de m bits (CPE) de la capa escondida, el segundo clasificador mapea un patrón de m bits a un solo bit (0 ó 1) de la salida.

Sea x un patrón de entrada de n bits, primero se carga x en el primer clasificador el cuál nos da como salida un patrón y de m bits (EP de la cuenca al cual x pertenece):

$$y = DS \cdot x \tag{2.10}$$

Luego, y es cargado al segundo clasificador el cual da como salida un sólo valor o (CPE de la cuenca al cual y pertenece) que determina la clase del patrón de entrada x.

$$o = DV \cdot y \tag{2.11}$$

2.5. Algoritmos Genéticos (AG)

Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Charles Robert Darwin (1859). Por imitación de este proceso, los algoritmos genéticos (AG) son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores cada vez más aproximados al óptimo del problema, depende en buena medida de una adecuada codificación de las mismas.

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes "superindividuos", cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los AG usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor o puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos – descendientes de los anteriores – los cuales comparten algunas de las características de sus padres.

Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el AG ha sido bien diseñado, la población convergerá hacia una solución cercana a la óptima del problema.

En la actualidad, los AGs son preferentemente utilizados como métodos de búsqueda de soluciones factibles que simulan la evolución natural y han sido usados con éxito en la solución de problemas de optimización combinatoria, optimización de funciones reales y como

mecanismos de aprendizaje de máquinas (*machine learning*).
Los AG se compone de las siguientes etapas [3]:

- **Codificación del dominio.** Para un AG lo primero que ser requiere es determinar en qué espacio se encuentran las posibles soluciones al problema que se pretende resolver. Los AG operan sobre códigos genéticos , sobre genotipos que se deberán mapear al espacio de soluciones. Es decir,es necesario codificar de alguna manera el dominio del problema para obtener estructuras manejables que puedan ser manipuladas por el AG. Cada una de estas estructuras constituye el equivalente al genotipo de un individuo en términos biológicos. Es frecuente que el código de los elementos del dominio del problema utilice un alfabeto binario (0's y 1's). Los códigos que representa a un individuo serán llamados cromosomas.

- **Población inicial.**Crear de alguna forma una población inicial de individuos que representaran las soluciones posibles al problema. La población inicial se representa por $P(0)$.

- **Evaluación de la población.** Al igual que en la naturaleza es, en los AG, es necesario establecer algún criterio que permita decidir cuáles de las soluciones propuestas en una población son mejores respecto del resto de las propuestas y cuáles no lo son. Es necesario establecer, para cada individuo, una medida de desempeño relativa a la población a la que pertenece. Para determinar cuáles de estos individuos corresponden a buenas propuestas de solución y cuáles no, es necesario calificarlos de alguna manera. Cada individuo de cada generación de un algoritmos genético recibe una calificación o, para usar en términos biológicos, una medida de su grado de adaptación(*fitness*). Éste es un número real no negativo, tanto más grande la solución propuesta por dicho individuo será mejor. $eval(v) = f(x)$,
Donde: el cromosoma v representa el valor real de x.

- **Selección**. Una vez calificados todos los individuos de una generación, el algoritmo debe, al igual que lo hacen la naturaleza, seleccionar a los individuos más calificados, mejor adaptados al medio, para que tengan mayor oportunidad de reproducción . De esta forma se incrementa la probabilidad de tener individuos buenos (con alta calificación) en el futuro.

- **Cruzamiento.** Dados dos individuos seleccionados en función de su grado de adaptación, éstos pasen a formar parte de la siguiente generación o, al menos, mezclen sus códigos genéticos para generar hijos que posean un código híbrido. Es decir, los códigos genéticos de los individuos se cruzan. El mecanismo de cruzamiento más común es el llamado cruzamiento de un punto.

- **Mutación.** Ocasionalmente algunos elementos del cromosoma de ciertos individuos de una AG se alteran a propósito. Éstos se seleccionan aleatoriamente. El objetivo es generar nuevos individuos, que exploren regiones del dominio del problema que probablemente no se han visitado aún. Esta exploración no presupone conocimiento alguno, no es sesgada. Aleatoriamente se buscan nuevas soluciones posibles que quizá superen las encontradas hasta el momento.

2.5.1. Algoritmo genético simple (AGS)

El algoritmo 1 es un AGS, en donde, t representa las generaciones y $P(t)$ representa la población P en la generación t. Al principio el algoritmo inicia con la generación 0 ($t = 0$) y se crea la población inicial ($P(0)$) de manera aleatoria, después comienza un bucle que se detiene dada una condición de paro, mientras no se de esta condición de paro el bucle seguirá cruzando y mutando individuos de una generación para obtener la siguiente generación [4].

Algoritmo 1 Algoritmo genético simple

```
1: t ← 0
2: Inicializar P(t) con N individuos.
3: Evaluar P(t)
4: while No se cumpla alguna condición de terminación do
5:     t ← (t + 1)
6:     while |P(t)| < N do
7:         Seleccionar dos individuos de P(t − 1) con probabilidad proporcional a su evaluación.
8:         if con probabilidad p_c then
9:             Cruzar a los dos individuos para obtener dos individuos nuevos.
10:            Con probabilidad p_m mutar a los dos individuos.
11:        end if
12:        Inserta a los dos individuos en P(t)
13:    end while
14:    Evaluar P(t)
15: end while
```

2.5.2. AGT (AGS + Elitismo)

En ocasiones el AG encuentra, en alguna generación, una buena aproximación a la solución del problema que se le ha planteado y que en la siguiente generación esta buena propuesta desaparece; en el AGS no existe ningún medio para asegurar que esa buena aproximación que ya se encontró se preserve. Puede ocurrir que ni siquiera fuera seleccionada para dejar descendientes en la siguiente generación. Es cierto que el mejor individuo de la población tiene la más alta probabilidad de ser seleccionado por la ruleta,pero hay una probabilidad distinta de cero de que esto no ocurra. También puede ser que sea elegido y se pierda al cruzarse con algún otro seleccionado distinto de él mismo, o bien que logre sobrevivir a la cruza pero la mutación lo altere. En fin, no se provee de un mecanismo que asegure que el mejor individuo de la generación t pase intacto a la generación $t + 1$. Esto trae como consecuencia que la mejor calificación de cada generación del AGS pueda fluctuar. Sería deseable que el comportamiento de la calificación máxima fuera monótono. En particular no decreciente. Para asegurar que el mejor individuo de la generación $t + 1$ tenga una calificación mayor o igual que la del individuo de la generación t. Para lograr esto se inventó el elitismo.

En términos generales este mecanismo consiste en conservar, en la generación t de un AG, a los k mejores individuos de las últimas r generaciones. Hay dos variantes de modelos elitistas:

- ⋆ Elitismo parcial. En una población de tamaño n mantenemos una copia de los mejores $\tau < n$ individuos hasta la generación k.
- ⋆ Elitismo total. En una población de tamaño n mantenemos una copia de los mejores n individuos hasta la generación k. Es decir, dado que hemos evaluado nk individuos hasta la generación k, nuestra población consistirá de los mejores n hasta ese punto.

Entonces si al algoritmo 1 se le agrega la cualidad de elitismo, tendremos un AG que nos garantiza converger a la mejor solución encontrada hasta el momento, a este algoritmo se le conoce como AGT.

2.6. AG para crear un clasificador de dos etapas (DS-DV)

2.6.1. Cromosoma

El cromosoma propuesto consiste en dos partes (ver figura 2.10).

1. Cadena de dependencia (DS) para el clasificador 1, una cadena de dígitos.
2. Vector de dependencia (DV) para el clasificador 2, una cadena binaria de formato valido.

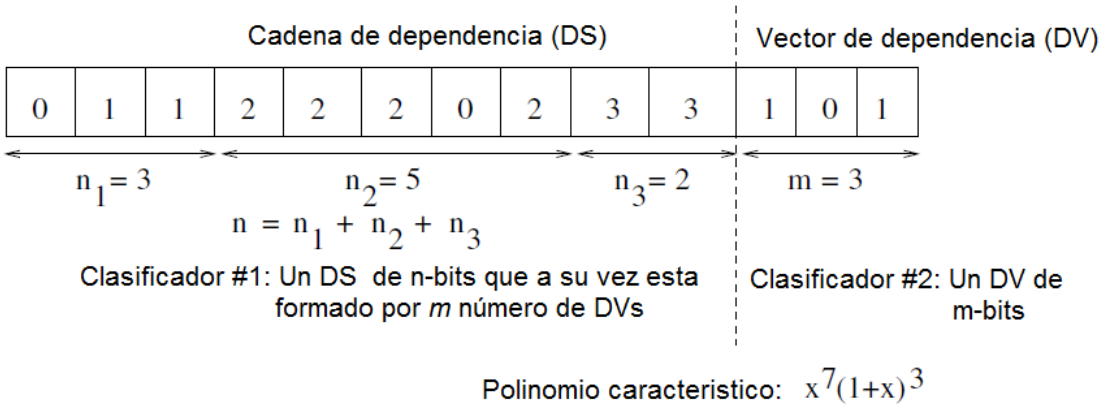


Figura 2.10: Ejemplo de un cromosoma.

Así, el tamaño del cromosoma es igual a $(n + m)$ donde n es el número de bits en un patrón y m es el numero de bits del CPE. En la figura 2.10 se muestra la representación de un cromosoma. El cromosoma consiste de 10-bits $(n = 10)$ $DS = 011|22202|33$ (clasificador 1) y 3-bits $(m = 3)$ $DV = 101$ (clasificador 2).
El DS esta dividido en 3 partes que corresponden a 3 DVs de tamaño $n_1 = 3$, $n_2 = 5$ y $n_3 = 2$ respectivamente [6].

2.6.2. Generación de la población inicial

Para la creación de la población inicial, se debe de asegurar que cada solución generada aleatoria mente es una combinación de un DS de n-bits con 2^m números de cuencas diferentes y un DV de m bits, los cromosomas son generados de forma aleatoria de acuerdo a los siguientes pasos [6]:

1. Dividir n de manera aleatoria en m enteros tal que $n_1 + n_2 + \ldots + n_m = n$
2. Para cada n_i generar de manera aleatoria un DV valido.
3. Crear un DS a través de la concatenación de los DV's creados anteriormente para el clasificador 1.

4. Crear de manera aleatoria un DV de m bits para el clasificador 2.

5. Crear el cromosoma completo mediante la concatenación del clasificador 1 y clasificador 2.

2.6.3. Algoritmo para la cruza

El algoritmo toma 2 cromosomas de la población actual (PP) y forma un cromosoma nuevo. Se escoge un sólo punto de cruce y se parten en dos (de acuerdo al punto de cruce) ambos cromosomas y se toman las dos partes de los dos cromosomas para formar el nuevo cromosoma. En la figura 2.11 se muestra un ejemplo del cruce de cromosomas.

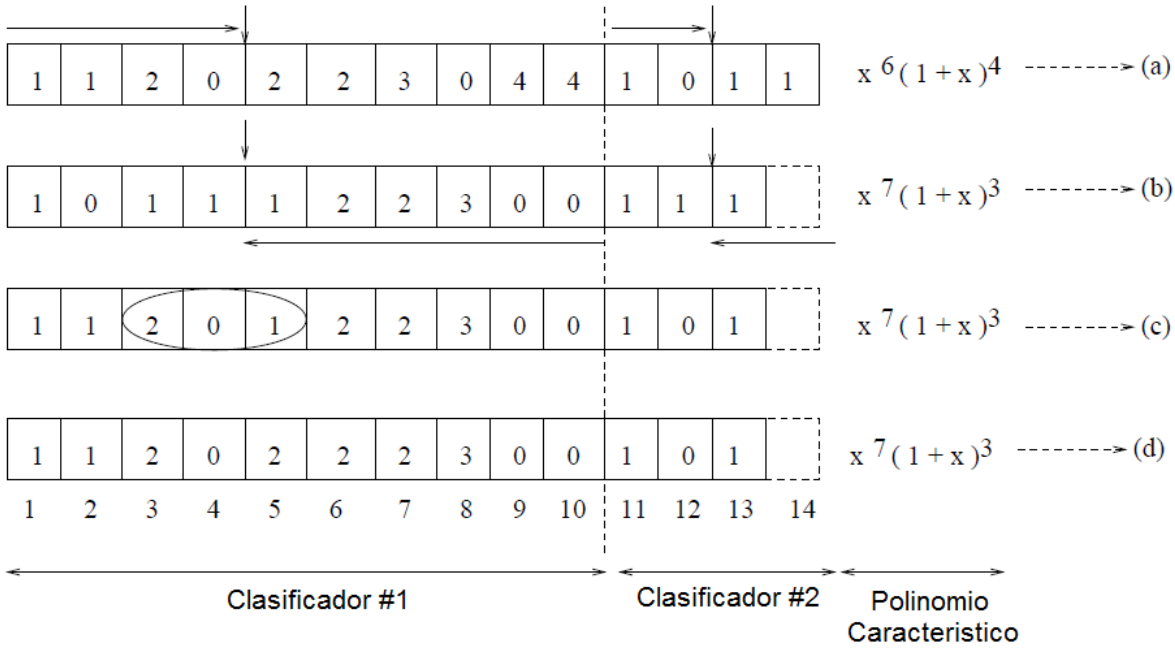


Figura 2.11: Ejemplo del algoritmo de cruce.

Dos cromosomas se tienen al principio 2.11(a) y 2.11(b). El punto de cruce es seleccionado aleatoriamente en este caso es 4 para la primera parte (clasificador 1) y 12 para la segunda parte (clasificador 2). Así, los primeros cuatro símbolos son tomados del primer cromosoma y los restantes seis símbolos son tomados del segundo cromosoma para formar la primera parte del nuevo cromosoma. De manera similar, los bits 11 y 12 son tomados del primer cromosoma y el bit 13 es tomado del segundo cromosoma para tener la segunda parte del nuevo cromosoma. Pero para este cruce el cromosoma obtenido genera un DS invalido debido a los símbolos que se encuentran en las posiciones 3, 4 y 5 (ver figura 2.11(c)).
En la figura 2.11(c), el DS es 1120122300, lo cual es invalido porque el 1 y 2 están intercalados. Entonces es necesario corregir ese error, en la figura 2.11(d) se muestra el cromosoma ya corregido.

2.6.4. Algoritmo para la mutación

El algoritmo de mutación realiza algunos cambios mínimos en los cromosomas de la población presente para así formar un nuevo cromosoma para la generación siguiente. El cromosoma es mutado en un punto único como se muestra en la figura 2.12. Los puntos de mutación para el clasificador 1 y clasificador 2 son 4 y 12 respectivamente.

Si alguna anomalía se presenta en el DS o DV al momento de la mutación, esta debe de ser corregida para que el DS o DV siga siendo valido.

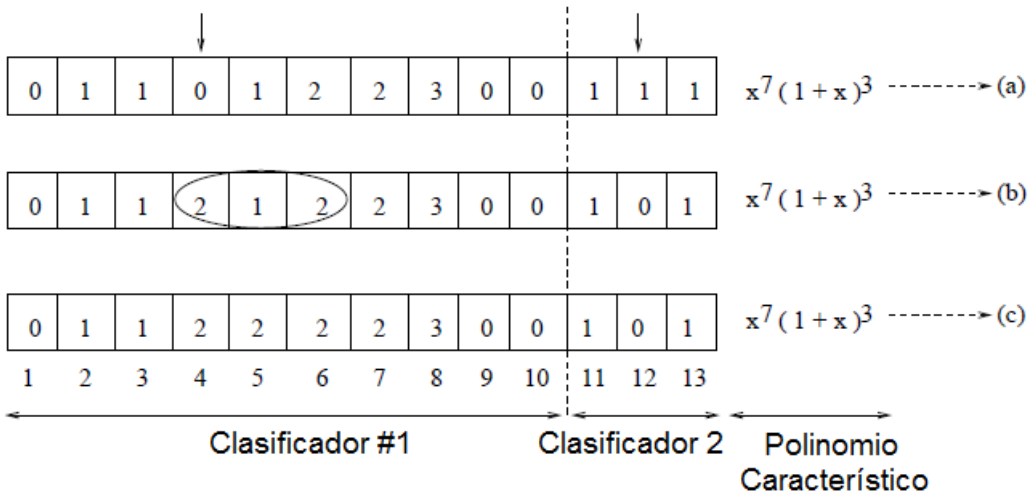


Figura 2.12: Ejemplo de la mutación de un cromosoma.

2.6.5. Función de evaluación

La función de evaluación F para un cromosoma en particular es determinada por dos factores.

1. La capacidad del DS para clasificar los conjuntos de patrones de entrada S_1 y S_2 en diferentes conjuntos de sumideros, y nos referiremos a esta como F_1.

2. La capacidad del DV para clasificar los conjuntos de CPEs k_1 y k_2 en diferentes sumideros (uno en la cuenca cero y el otro en la cuenca no cero), y nos referiremos a esta como F_2.

La función de evaluación de un cromosoma en particular esta dada por:

$$F = F_1 * F_2 \tag{2.12}$$

2.7. Clasificación

La clasificación es también referida como aprendizaje supervisado; es un método que asigna una clasea un conjunto de patrones bajo la supervisión de una muestra de entrenamiento (llamada muestra de supervisión). Los patrones son inicialmente divididos en varios conjuntos de clases de supervisión, entonces el clasificador es entrenado de manera que vaya siguiendo a la muestra de supervisión. La finalidad es predecir la clase $C_i = f(x_1, \ldots, x_n)$, donde $x_1, \ldots, x_n$ son los atributos de entrada. La entrada para un algoritmo de clasificación, es usualmente, una base de datos de entrenamiento (muestra de supervisión) que contiene patrones y a su vez cada patrón tiene un número determinado de atributos. Un atributo puede ser:

- **Numérico**: Es aquel que tiene valores numéricos reales o enteros.
- **Categórico**: Son valores discretos o un conjunto de símbolos definidos.

Existe un atributo que es llamado atributo dependiente. Si el atributo dependiente es categórico, entonces el problema es llamado clasificación y este atributo dependiente es llamado la etiqueta de clase, la cuál indica a que clase pertenece el patrón con este atributo.

2.7.1. Árboles de decisión

Un clasificador basado en un árbol de decisiones es uno de los métodos de aprendizaje supervisado. Los arboles de decisiones pueden ser representados con reglas $IF - THEN - ELSE$ [5].

Los arboles de decisión están formados por:

- **Nodos**: Nombres o identificadores de los atributos.
- **Ramas**: Posibles valores del atributo asociado al nodo.
- **Hojas**: Conjuntos ya clasificados de ejemplos y etiquetados con el nombre de una clase.

El proceso para construir árboles de decisión comienza con el conjunto de muestra de supervisión completo, y este procede de manera general como se muestra a continuación:

1. Si todo el conjunto de muestra de supervisión en el nodo actual t pertenecen a la clase C_i, crear una hoja y etiquetarla con la clase C_i.
2. De otra manera, evaluar un conjunto de posibles divisiones S, utilizando alguna medición.
3. Escoger la mejor división S' como prueba del nodo actual.
4. Crear tantos nodos hijos como distintos resultados existan de S', y dividir la muestra de supervisión usando S'.

5. Se dice que un nodo hijo t es puro, si toda la muestra de supervisión en el nodo t pertenecen a la misma clase. Repetir los pasos 1, 2, 3, 4 y 5 en todos los nodos hijos impuros.

La construcción de un árbol de decisión se puede hacer maximizando la información mutua del árbol entero, o puede ser maximizando la ganancia de información local. Para dividir un nodo se tienen los siguientes pasos:

- ⋆ Escoger un atributo que mejor separe patrones de diferentes clases.
- ⋆ Cuantificar el factor de impureza $I(S)$ de un conjunto S con l clases, el cual puede calcularse con la ecuación 2.16 o 2.14.

$$Entropia(S) = -\sum_{i=1}^{l}(\frac{n_i}{N})\log(\frac{n_i}{N}) = -\sum_{i=1}^{l} p_i \log(p_i) \tag{2.13}$$

La entropía es igual a cero cuando todos los patrones pertenecen sólo a una clase, y es igual a uno cuando todas las clases tienen el mimo número de patrones.

$$Gini(S) = 1 - \sum_{i=1}^{l}\left(\frac{n_i}{N}\right)^2 \tag{2.14}$$

- ⋆ Calcular la ganancia que se obtiene al dividir S en r conjuntos, con la ecuación:

$$Ganancia(S, S_1, \ldots S_r) = I(S) - \sum_{j=1}^{r}\frac{|S_j|}{|S|}I(S_j) \tag{2.15}$$

- ⋆ El atributo que posea la mayor ganancia es elegido para dividir el nodo.

Los principales algoritmos para arboles de decisión son los siguientes:

- ⋆ Para clasificación en *machine learning*: ID3, C4.5 y CART.
- ⋆ Para clasificación de grandes bases de datos: SLIQ, SPRINT, SONAR y RainForest.

2.7.2. Algoritmo ID3

De manera general el ID3 elige en cada momento el atributo que tiene la mayor ganancia en información, o lo que es lo mismo, la menor entropía. La entropía esta dada por la ecuación 2.13.

El caso general en el que se tienen N patrones que se dividen en varios conjuntos que representan diferentes clases C_i, $i = 1, 2, 3, \ldots, l$. El número de patrones pertenecientes a la clase C_i es n_i. Cada patrón tiene n atributos y cada atributo puede tener 2 o más valores [5].

1. Calcular el valor inicial de la entropía, con la ecuación 2.16.

$$Entropia = -\sum_{i=1}^{l}(\frac{n_i}{N})\log_2(\frac{n_i}{N}) = -\sum_{i=1}^{l} p_i \log_2(p_i) \tag{2.16}$$

2. Seleccionar el atributo para el cual la entropía es menor o tenga mayor ganancia de información (de acuerdo a la ecuación 2.15), para que sea el nodo raíz del árbol.

3. Construir el siguiente nivel del árbol de decisión haciendo que le entropía disminuya.

4. Repetir los pasos del 1 al 3. Continuar el procedimiento hasta que todos los patrones pertenezcan a una clase y la entropía del sistema sea igual a cero.

Ejemplo 2.7.1 ***Dados los patrones de la tabla 2.4, construir un árbol de decisión utilizando el algoritmo ID3.***

Tabla 2.4: Base de datos para el ejemplo del algoritmo ID3.

Clase	**Altura**	**Cabello**	**Ojos**	**Cabello**	**Ojos**	**Altura**	**Clase**
C_1	Alto	Rubio	Marrón	Rubio	Marrón	Alto	C_1
C_1	Alto	Negro	Azul	Rubio	Marrón	Pequeño	C_1
C_1	Alto	Negro	Marrón	Rubio	Azul	Pequeño	C_2
C_1	Pequeño	Negro	Azul	Rubio	Azul	Alto	C_2
C_1	Pequeño	Rubio	Marrón	Negro	Azul	Alto	C_1
C_2	Alto	Pelirrojo	Azul	Negro	Marrón	Alto	C_1
C_2	Alto	Rubio	Azul	Negro	Azul	Pequeño	C_1
C_2	Pequeño	Rubio	Azul	Pelirrojo	Azul	Alto	C_2

Lo primero es calcular el valor inicial de la entropía con la ecuación 2.16.

$$Entropia = -\frac{5}{8}\log_2\frac{5}{8} - \frac{3}{8}\log_2\frac{3}{8} = 0{,}954$$

Dividiendo el conjunto de patrones con el atributo Altura, tendremos dos ramas una para Alto y otra para Pequeño, calculamos la entropía de cada uno como sigue:

$$-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0{,}971 \qquad \text{y} \qquad -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} = 0{,}918$$

La ganancia de información obtenida con el atributo Altura es (usando ecuación 2.15):

$$Ganancia_{altura} = 0{,}954 - \left(\frac{5}{8} * 0{,}971 + \frac{3}{8} * 0{,}918\right) = 0{,}954 - 0{,}951 = 0{,}003$$

Dividiendo el conjunto de patrones con el atributo Cabello, tendremos tres ramas una para Rubio, Negro y otra para Pelirrojo, calculamos la entropía de cada uno como sigue:

$$-\frac{2}{4}\log_2\frac{2}{4}-\frac{2}{4}\log_2\frac{2}{4}=1, \qquad -\frac{3}{3}\log_2\frac{3}{3}=0 \qquad \text{y} \qquad -\frac{1}{1}\log_2\frac{1}{1}=0$$

La ganancia de información obtenida con el atributo Cabello es (usando ecuación 2.15):

$$Ganancia_{cabello}=0{,}954-\left(\frac{4}{8}*1+\frac{3}{8}*0+\frac{1}{8}*0\right)=0{,}454$$

Dividiendo el conjunto de patrones con el atributo Ojos, tendremos dos ramas una para Marrón y otra para Azul, calculamos la entropía de cada uno como sigue:

$$-\frac{3}{3}\log_2\frac{3}{3}=0 \qquad \text{y} \qquad -\frac{2}{5}\log_2\frac{2}{5}-\frac{3}{5}\log_2\frac{3}{5}=0{,}970951$$

La ganancia de información obtenida con el atributo Ojos es (usando ecuación 2.15):

$$Ganancia_{ojos}=0{,}954-\left(\frac{3}{8}*0+\frac{5}{8}*0{,}9709\right)=0{,}3471$$

Por lo tanto el atributo Cabello sera el atributo del nodo raíz, ya que fue el que obtuvo la mayor ganancia en información.

Con este primer nodo creamos 3 ramas diferentes que corresponden a los 3 valores diferentes del atributo Cabello. Estos tres valores son Rubio, Negro y Pelirrojo, como se muestra en la tabla 2.4, las ramas con los valores Negro y Pelirrojo pertenecen a una sola clase por lo tanto, no es necesario procesar estos nodos porque ya son nodos hoja. En cuanto al valor Rubio es necesario aplicar de nuevo el procedimiento ID3 para este nuevo nodo.

Para la rama con el valor Rubio tenemos que el valor inicial de la entropía es:

$$Entropia=-\frac{2}{4}\log_2\frac{2}{4}-\frac{2}{4}\log_2\frac{2}{4}=1$$

Dividiendo el conjunto con el atributo Ojos:

$$-\frac{2}{2}\log_2\frac{2}{2}=0, \qquad \text{y} \qquad -\frac{2}{2}\log_2\frac{2}{2}=0$$

$$Ganancia_{ojos}=1-\left(\frac{2}{4}*0+\frac{2}{4}*0\right)=1$$

Dividiendo el conjunto con el atributo Altura:

$$-\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}=1 \qquad \text{y} \qquad -\frac{1}{2}\log_2\frac{1}{2}-\frac{1}{2}\log_2\frac{1}{2}=1$$

$$Ganancia_{altura}=1-\left(\frac{2}{4}*1+\frac{2}{4}*1\right)=0$$

Por lo tanto, este nodo divide con el atributo Ojos, ya que fue el que obtuvo una mayor ganancia. El árbol de decisión para este ejemplo queda como se muestra en la figura 2.13.

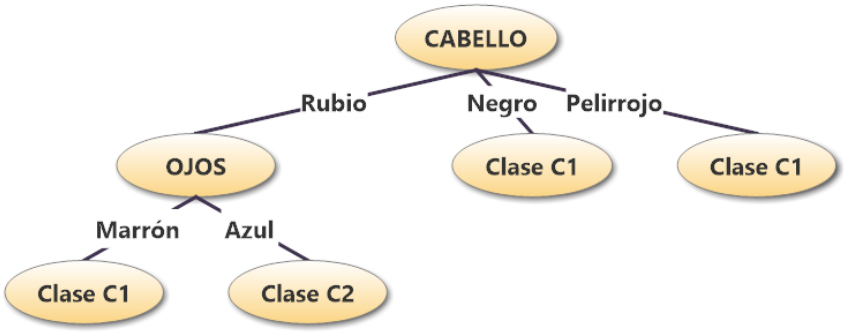


Figura 2.13: Árbol de decisión del ejemplo 2.7.1.

Capítulo 3

ESTADO DEL ARTE

La clasificación de patrones es el proceso de encontrar propiedades comunes entre un grupo de objetos entre una base de datos. Existen diferentes técnicas desarolladas para realizar la tarea de la clasificación tales como:

- Aproximación estadísticas
- Aproximaciones no métricas
- Aproximaciones cognitivas

3.1. Aproximación estadística

Enfoques estadísticos se caracterizan por su dependencia de un modelo de probabilidad explicito. Las características se extraen de los datos de entrada (objeto) y se utilizan para asignar a cada objeto una de las clases. Los límites de decisión se determinan por las distribuciones de probabilidad de los objetos que pertenecen a cada clase, que debe ser especificado o aprendido. Probabilidades a *priori* (es decir, probabilidades antes de la medición se describe por funciones de densidad de probabilidad) se convierten en una *a posteriori* (o probabilidades condicional de clase/medición) probabilidades (es decir, las probabilidades después de la medición). Las redes bayesianas es un ejemplo representativo de este enfoque de aprendizaje estadístico [15].

Un enfoque basado en el análisis discriminante, especifica una forma paramétrica de la frontera de decisión (por ejemplo, lineal o cuadrática), entonces se crea la mejor frontera de decisión, de esta forma se pueden clasificar los objetos dependiendo de en que lado de la frontera se encuentren. Dichos límites se pueden construir utilizando, por ejemplo, un criterio de error. En las técnicas de máxima entropía, el principio fundamental es que cuando no se sabe nada, la distribución debe ser lo más uniforme posible, es decir, tienen la entropía máxima. Los datos de entrenamiento etiquetados son usados para derivar un conjunto de

restricciones para el modelo.

Algoritmos de aprendizaje lento basados en instancia [19], así llamados porque ellos retrasan el proceso de inducción o generalización hasta que se realiza la clasificación. Estos algoritmos requieren menos tiempo de cálculo durante la fase de entrenamiento que los algoritmos de aprendizaje (tales como redes bayesianas, árboles de decisión, o redes neuronales), pero más tiempo de cálculo durante el proceso de clasificación. Uno de los algoritmos de aprendizaje basados en instancia es el algoritmo del vecino más cercano.

3.2. Aproximación no métrica

Enfoques no métricos como son: árboles de decisión, métodos sintácticos (o gramaticales) y clasificadores basados en reglas. Para clasificar patrones realiza una serie de preguntas, en el que la pregunta siguiente depende de la respuesta a la pregunta anterior. Este enfoque es útil para datos categóricos (no métricos), porque las preguntas se pueden plantear para obtener respuestas del tipo "sí/no" o "verdadero/falso". La secuencia de preguntas se puede mostrar usando una estructura de árbol, cuyos nodos son llamados nodos de decisión en los cuales cada uno se hace una pregunta y para cada respuesta se cuenta con otro nodo como respuesta. Estos nodos están conectados hasta llegar al nodo terminal u hoja el cual indica la clasificación.

En el caso de patrones complejos, un patrón puede ser visto como una composición jerárquica de subpatrones los cuales son también formados por otros subpatrones [21]. Los subpatrones más simples se denominan primitivas, y el patrón complejo se representa en términos de relaciones entre estas primitivas de una manera similar a la sintaxis de un lenguaje. Las primitivas son vistos como un lenguaje, y los patrones son frases generadas de acuerdo a una determinada gramática (es decir, un conjunto de reglas) que se infiere de las muestras de entrenamiento disponibles. Este enfoque es recomendable en situaciones donde los patrones tienen una estructura definida que puede ser codificado por un conjunto de reglas. Sin embargo, las dificultades para la segmentación de imágenes con ruido para detectar las primitivas y la inferencia de la gramática a partir de datos de entrenamiento a menudo impiden su aplicación. El enfoque sintáctico puede producir un número combinatorio de posibilidades ha ser investigado, lo que implica un alto costo en computo y conjuntos de entrenamiento grandes [22].

3.3. Aproximación cognitiva

La aproximación cognitiva incluye a las redes neuronales y máquinas de soporte vectorial (SVM). Las redes neuronales intentan replicar la organización del cerebro humano, en donde las células nerviosas (neuronas) están unidas entre sí por hilos de fibra (axones). Las redes neuronales son sistemas de computo paralelo que consta de un gran número de procesadores

con muchas interconexiones; son capaces de aprender relaciones entrada-salida no lineales y utilizar procedimientos de entrenamiento secuencial.

Máquinas de de soporte vectorial, SVM [23], representan los ejemplos de entrenamiento como puntos en el espacio p dimensional, donde los ejemplos de las clases de datos están separados por un hiperplano $(p-1)$ dimensional, el cual es elegido para maximizar los márgenes de cada lado del hiperplano.

3.4. Aproximación con AC

El diseño de clasificadores de patrones basados en AC ha sido publicado en diferentes trabajos ([11], [6], [12], [13], [14]). En el libro *Additive Cellular Automata* [2] se formalizan las bases matemáticas para el AC lineal y para un tipo de AC llamado ACMS.

En la tesis *Cellular Automata Evolution: Theory and Applications in Pattern Recognition and Classification* [19] se da una caracterización de los AC aditivos los cuales sirven como base para bosquejar un clasificador de patrones.

En el artículo *Evolving Cellular Automata as Pattern Classifier* [7] se reporta un clasificador de patrones binario basado en AC. En este trabajo en la parte del entrenamiento del AC es realizado por un AG que busca el mejor AC que pueda realizar la tarea de clasificar. Una vez encontrado el AC para clasificar un nuevo patrón es necesario realizar una multiplicación de matrices, lo que hace que la complejidad sea $O(n^3)$ cada vez que se desee clasificar un nuevo patrón.

En el artículo *Theory and Application of Cellular Automata For Pattern Classification* [6], baja la complejidad del clasificador en la etapa de clasificación a una complejidad lineal $O(n)$ introduciendo dos nuevas herramientas llamadas DS y DV, pero en la etapa de entrenamiento del clasificador se sigue utilizando un AG y la complejidad del mismo es cuadrática $O(n^2)$, lo que lo hace poco practico para muestras de supervisón de más de cientos de miles, en este artículo se hacen pruebas sólo con bases de datos pequeñas que tienen a lo más unos cuantos miles de patrones.

En el artículo *Data mining with cellular automata* [16], se menciona un AC de dos dimensiones para clasificación de patrones con dos atributos, pero tiene una gran desventaja y es que el número de dimensiones del AC es igual al número de atributos que tenga un patrón, lo cuál hace que esta propuesta de clasificación no sea tan versátil ya que al aumentar las dimensiones de un AC se hace mucho más complejo el cálculo de operaciones de AC.

Capítulo 4

PROPUESTA DE SOLUCIÓN

En este capítulo se muestran los algoritmos diseñados para resolver el problema planteado en esta tesis, esto es para obtener el clasificador DS-DV y el clasificador MCJJ2.

El primer clasificador que se diseño fue el clasificador DS-DV, por lo cual se muestran todos los algoritmos diseñados para el AG que encuentra el clasificador DS-DV en la etapa de entrenamiento.

El segundo clasificador diseñado es el MCJJ2, para este nuevo clasificador se diseñaron los algoritmos para crear un AGT y un árbol de decisiones basado en el concepto de DV, donde la etapa de entrenamiento de este clasificador consiste en construir un árbol binario de decisiones conformado de nodos que contienen sólo DV's y etiquetas que indican un clase.

Los clasificadores de patrones que se desarrollaron en esta tesis son clasificadores binarios, es decir, sólo clasifican en dos clases diferentes, y además son clasificadores que requieren patrones binarios, es por esto, que en caso de que los patrones no sean binarios; será necesario una etapa previa de discretización y binarización.

La figura 4.1 muestra el diagrama general para obtener el clasificador DS-DV. Al inicio se tiene un AG llamdo AGT-DSDV que se encarga de la etapa de entrenamiento del clasificador DS-DV. El AGT-DSDV requiere como entrada la muestra de supervisión y los parámetros propios de los algoritmos genéticos (como son población, número de generaciones, etc.).
La salida del AGT-DSDV será una población del tamaño que le hayamos indicado, donde cada individuo de la población representa un clasificador DS-DV. Seleccionamos el mejor individuo de la población. Finalmente, se hacen las pruebas con el clasificador DS-DV seleccionado, estas pruebas consisten en alimentar el clasificador DS-DV con la muestra de supervisión y la de control.

Este clasificador DS-DV es de dos etapas; la primera, es un clasificador DS y la segunda, es un clasificador DV. El AGT-DSDV busca estas dos etapas simultáneamente.

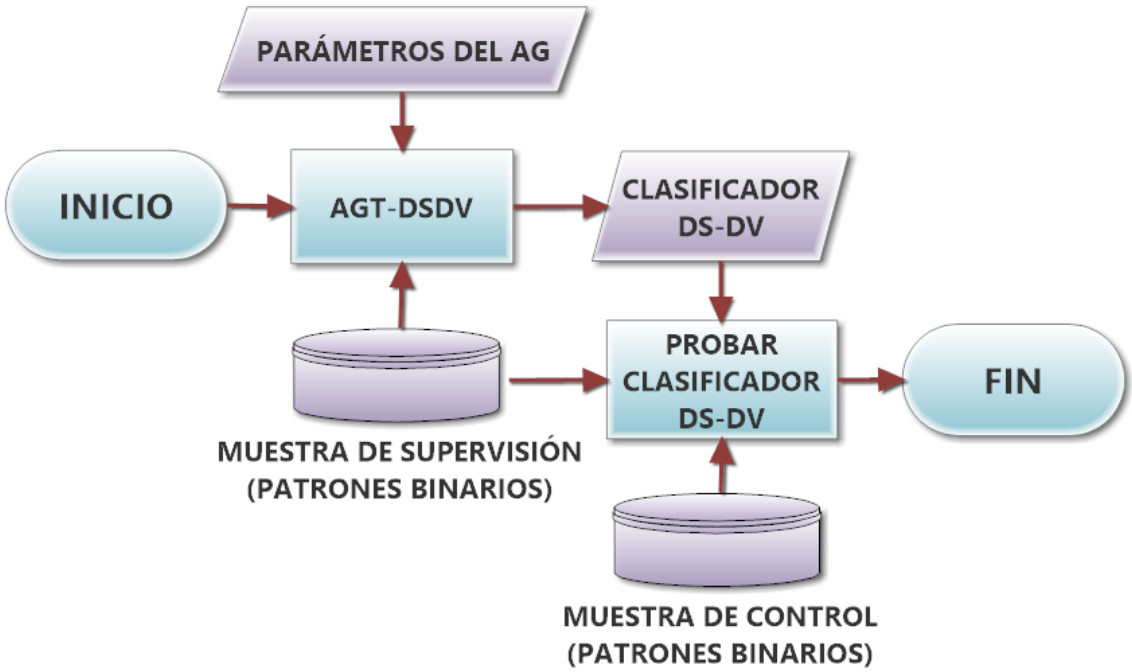


Figura 4.1: Diagrama general del sistema DSDV

La figura 4.2 muestra el diagrama general para obtener el clasificador MCJJ2. El MCJJ2 clasifica por medio de un árbol de decisión compuesto de DV's. El árbol de decisión se crea en la etapa de entrenamiento del clasificador y se hace por medio de un algoritmo parecido al ID3 llamado CREA-MCJJ2 y un AG llamado AGT-ADV (diseñados en esta tesis).

Después de la etapa de entrenamiento del clasificador MCJJ2, la forma de clasificar un patrón es que el patrón recorra el árbol decisión hasta llegar a un nodo hoja, el cuál estará etiquetado con la clase a la que pertenece. Finalmente, se hacen las pruebas con el clasificador MCJJ2 obtenido, estas pruebas consisten en alimentar el clasificador MCJJ2 con la muestra de supervisión y la de control.

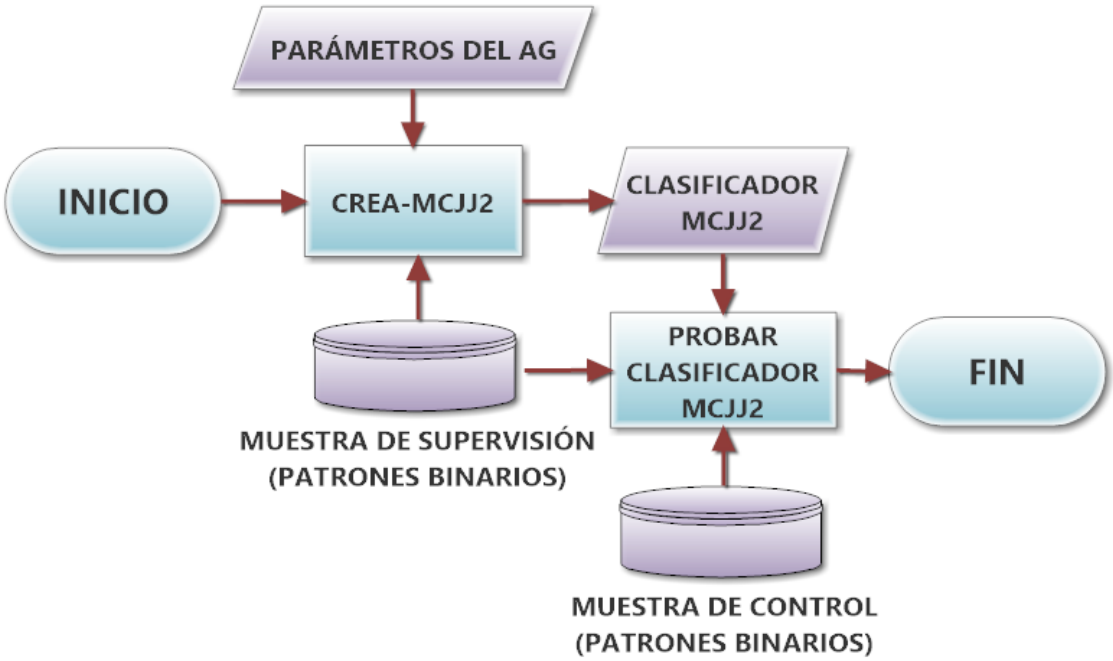


Figura 4.2: Diagrama general del sistema MCJJ2

4.1. Clasificador DS-DV

En esta sección se diseñan algoritmos necesarios para encontrar el DS y DV que hagan el trabajo de clasificación. Primero se definiera lo que se quiere obtener y la información con la que se cuenta.

A continuación se definirán dos operadores:

***Definición* 4.1.1** *Sean* $A = \{a_1, a_2, a_3, \cdots a_n\}$ *y* $B = \{b_1, b_2, b_3, \cdots b_n\}$ *dos cadenas binarias de tamaño* n*. El operador binario* $\odot$ *se define como:*

$$A \odot B = (a_1 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_3 \bullet b_3) \oplus \cdots \oplus (a_m \bullet b_m)$$

Donde:
$\bullet$ *: es la operación lógica AND.*
$\oplus$ *: es la operación lógica XOR.*

***Definición* 4.1.2** *Sean* A *y* B *dos cadenas binarias de tamaño* n*, tenemos la siguiente divisiones de* A *y* B *en* m *partes:* $\{A_1, A_2, A_3, \cdots, A_m\}$ *y* $\{B_1, B_2, B_3, \cdots, B_m\}$
Para las cuales se cumple lo siguiente:

1. $|B_1| = |A_1|, |B_2| = |A_2|, |B_3| = |A_3|, \cdots, |B_m| = |A_m|$
2. $|A_1| + |A_2| + |A_3| + \cdots + |A_m| = |A|$
3. $|B_1| + |B_2| + |B_3| + \cdots + |B_m| = |B|$

El operador binario $\diamond$ *se define como:*

$$A \diamond B = \{A_1 \odot B_1, A_2 \odot B_2, A_3 \odot B_3, \cdots, A_m \odot B_m\}$$

En un problema de clasificación de patrones, se debe de contar con:

1. **Muestra de supervisión**. Para nosotros esta muestra de supervisión consta de dos conjuntos:

$$S_1 = \{P_{11}, P_{12}, P_{13}, \ldots, P_{1x}\} \text{ y } S_2 = \{P_{21}, P_{22}, P_{23}, \ldots, P_{2y}\}.$$

Donde:
$x, y \in \mathbb{N}$
P_{ij} es un patrón conformado sólo de valores binarios.
$|P_{ij}| = n$

2. **Muestra de control.** Es el conjunto de patrones que se desea clasificar:

$$C = \{Q_1, Q_2, Q_3, \ldots, Q_z\}$$

Donde:
$z \in \mathbb{N}$
Q_i es un patrón conformado sólo de valores binarios.

Entonces el clasificador DS-DV es un clasificador de dos etapas la primera es un DS y la segunda es el DV, por lo tanto se desea obtener un DS y un DV:

$$DS = (DV_1, DV_2, DV_3, \ldots, DV_m) \tag{4.1}$$
$$DV = (i_1, i_2, i_3, \ldots, i_m) \tag{4.2}$$

Donde:
DV_i son válidos para $1 \leq i \leq m$
$|DS| = n$
$|DV| = m$
i_x puede ser 0 ó 1, para $1 \leq i \leq 1$.

Este DS y DV deben de poder dividir la muestra de supervisión en dos clases diferentes, siguiendo a la muestra de supervisión, esto es:

$$(DS \diamond P_{1i}) \odot DV = 0 \wedge (DS \diamond P_{2i}) \odot DV = 1 \tag{4.3}$$

$\forall P_{1i} \in S_1, \forall P_{2i} \in S_2$

Para poder encontrar el DS y DV en la fase de entrenamiento del clasificador DS-DV, se utilizará un AG más la característica de elitismo o sea un AGT. En la figura 4.3 se muestra el diagrama de flujo del AGT que se diseño.

En las siguientes secciones se muestra el diseño de los algoritmos necesarios para el AGT, como son algoritmos para la creación de la población inicial, selección, cruza, mutación, función de evolución y verificación.

Varios algoritmos diseñados utilizan una función para generar números pseudo-aleatorios. Esta función llamada $RANDOM$ se utiliza de las siguientes dos formas:

- $RANDOM()$: Genera un número real pseudo-aleatorio en el intervalo $[0, 1]$.
- $RANDOM(x)$: Genera un número entero pseudo-aleatorio entre $[0, x-1]$ con $x > 1$.

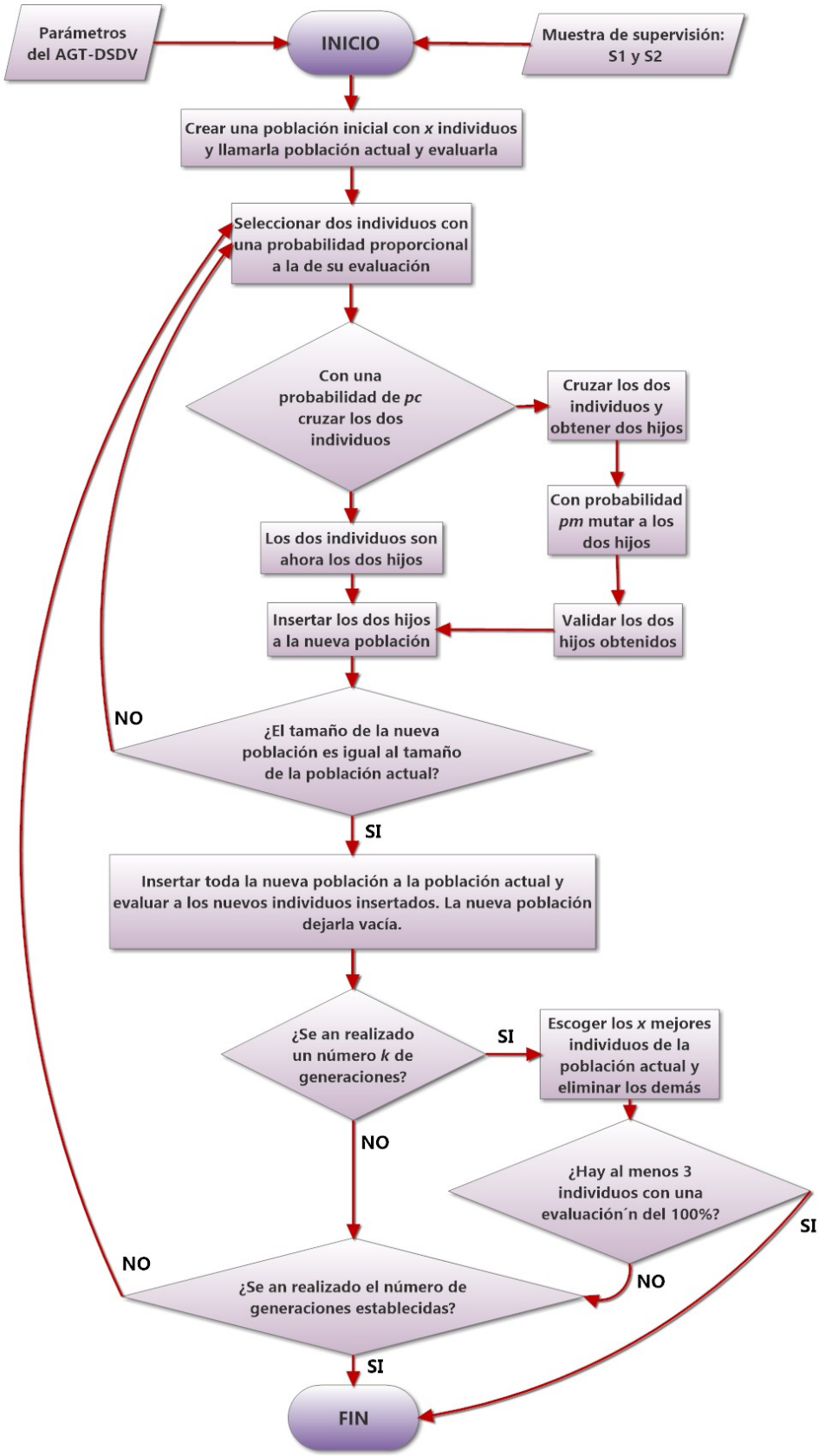


Figura 4.3: Diagrama de flujo del AGT

4.1.1. Codificación del dominio.

Para la codificación se emplearán las siguientes variables: DS, VP y DV que representarán a un individuo. Por ejemplo se tiene el siguiente clasificador:
DS =(0 0 0 1 1 0 2 2 2 0 2 3 0 3 0 0 0), DV =(1 0 1)
En el algoritmo genético es representado como:
DS =(0 0 0 1 1 0 1 1 1 0 1 1 0 1 0 0 0), VP =(6 5 7), DV =(1 0 1)
Se adiciona un elemento más que es el VP, el cual nos servirá para indicar las particiones del DS.

4.1.2. Población inicial.

La población inicial se creara de manera aleatoria de la siguiente manera:

1. Crear el VP. Cada elemento de un VP debe de ser mayor a 2 y la suma total de cada uno de sus elementos debe de ser n.

2. Crear el DS de acuerdo al VP. Se deben de crear varios DV's válidos y concatenarlos, el VP nos indicará cuantos y de que tamaño debe de ser cada DV.

3. Crear el DV. Crear un DV válido de tamaño m.

Para que un DV sea válido debe de haber a lo más un cero entre dos unos (definición 2.4.1). El algoritmo 3 válida si una cadena binaria es un DV válido.

El algoritmo 2 crea de manera aleatoria una cadena binaria, luego esta cadena binaria sirve como entrada del algoritmo 3, el cual regresara la misma cadena binaria si ésta es un DV válido, de lo contrario, creara un DV válido a partir de esta cadena binaria.

Algoritmo 2 Crea un DV válido de manera aleatoria

Entrada: La cardinalidad x del DV que se quiere.
Salida: Un DV de tamaño x

```
1:  procedure CREA-DV(x)
2:      if 0,1 < RANDOM() then
3:          a ←Crea aleatoremente un lista de 1's y 0's de tamaño x
4:          return VALIDAC-DV(a)
5:      else
6:          uno ← false
7:          inicio ← RANDOM(x − 2)
8:          salida ← Lista de 0's de tamaño inicio
9:          c ← (1 + RANDOM(x − inicio))
10:         Inserta c número de 1's en salida
11:         Inserta (c + inicio − x) número de 0's en salida
12:     end if
```

```
13:                                          ▷ Continuación del algoritmo CREA-DV
14:     if salida contiene sólo 0's then
15:         return una lista de puros 1's de tamaño x
16:     else
17:         if 0,5 < RANDOM() then
18:             return salida
19:         else
20:             return La lista salida invertida
21:         end if
22:     end if
23: end procedure
```

Algoritmo 3 Crea un DV válido a partir de una cadena binaria x.

Entrada: Una cadena binaria x.
Salida: Si x ya es un DV válido regresa x, de lo contrario crea un DV válido.

```
 1: procedure ValidaC-DV(x)
 2:     uno ← false
 3:     ceros ← 0
 4:     tope ← Posición del último 1 de x
 5:     salida ← x
 6:     if dv es una lista de puros ceros then
 7:         return Lista de puros 1's de tamaño |x|
 8:     else
 9:         i ← 0
10:         while i < tope do
11:             if dv[i] = 1 then
12:                 if uno then
13:                     ceros ← 0
14:                 else
15:                     uno ← true
16:                 end if
17:             else if uno then
18:                 ceros ← (ceros + 1)
19:                 if ceros > 1 then
20:                     salida[i] ← 1
21:                 end if
22:             end if
23:             i ← (i + 1)
24:         end while
25:     end if
26:     return salida
27: end procedure
```

4.1.3. Evaluación de la población.

La evaluación de cada individuo se realizara con cinco funciones de evaluación diferentes. En un AG la función de evaluación modifica en gran medida el comportamiento del AG. Por esta razón para lograr los mejores resultados posibles se probaron varias funciones diferentes para observar que resultados entregan cada una y poder decidir cuales son las mejores funciones.

Antes de ver los algoritmos de las funciones de evaluación, se muestran los algoritmos 4, 5 y 6 que serán utilizadas por las funciones de evaluación.

Algoritmo 4 Realiza las operación $\diamond$

Entrada: DS, VP y una cadena binaria P
Salida: Una cadena binaria de tamaño $|VP|$

```
1: procedure CLASIFICADOR1(DS, VP, P)
2:     salida ← ∅
3:     a ← 0
4:     for i ← 0, (|VP| − 1) do
5:         b ← (a + VP[i] − 1)
6:         c ← (DS[a : b] ⊙ P[a : b])
7:         Inserta c en salida
8:         a ← (a + VP[i])
9:     end for
10:    return salida
11: end procedure
```

Algoritmo 5 Función que obtiene el número de veces que aparece cada elemento del conjunto x.

Entrada: Conjunto x ordenado de menor a mayor.
Salida: Conjunto que indica cuantos elementos diferentes hay en x.

```
1: function CREA-PESOS(x)
2:     c ← 1
3:     s ← ∅
4:     for i = 0, (|x| − 2) do
5:         if x[i] = x[i + 1] then
6:             c ← (c + 1)
7:         else
8:             Inserta c en s
9:             c ← 1
10:        end if
11:    end for
12:    return s
13: end function
```

En general como se vio en 2.6.5, las funciones de evaluación deben de evaluar que tanto un individuo logra separar los conjuntos S_1 y S_2. Como se trata de un clasificador de dos etapas hay que asegurar que el clasificador 1 separe los conjuntos S_1 y S_2 en dos conjuntos diferentes digamos SS_1 y SS_2; finalmente, el clasificador 2 debe separar los conjuntos SS_1 y SS_2 en diferentes clases (cero o uno).
Hay varios caminos a seguir para llegar a separar completamente a S_1 y S_2; tan sólo para el clasificador 1 hay muchas formas de separar a S_1 y S_2. A continuación se explica la diferencia principal entre cada una de las funciones de evaluación diseñadas:

1. **Función de evaluación 1 (FE1).** Esta función de evaluación se diseño de acuerdo a lo visto previamente en la sección 2.6.5.

2. **Función de evaluación 2 (FE2).** Esta función contabiliza de manera independiente los patrones bien clasificados en la clase 0 ($f21$) y los bien clasificados en la clase 1 ($f22$), finalmente en el resultado estas dos variables se multiplican junto con $f1$ que en esta ocasión se calcula con una intersección. Con esta modificación hace que el AG se incline más por un individuo que clasifica de manera más equitativa.

3. **Función de evaluación 3 (FE3).** Esta función de evaluación es como FE1 con la diferencia de que tiene adicionado unos puntos extras para que los individuos con un $|VP|$ más pequeño tengan mejor calificación.

4. **Función de evaluación 4 (FE4).** Esta función de evaluación contempla dos opciones: **(a)** S_1 es asignado a la clase 0 y S_2 es asignado a la clase 1 Y **(b)** S_2 es asignado a la clase 0 y S_1 es asignado a la clase 1. De las cuales regresa el valor de la opción que obtuvo mayor puntuación.

5. **Función de evaluación 5 (FE5).** Combina las funciones de evaluación FE2, FE3 y FE4. Adicionalmente se agregan dos variables de entrada llamadas $fe-p1$ y $fe-p2$ que sirven para dar diferentes pesos a la evaluación que se realice.

Las funciones de evaluación FE4 y FE5 (algoritmos 10 y 11) tienen pasos en donde hay que eliminar elementos repetidos en los conjuntos $s1Aux$ y $s2Aux$, este paso es importante ya que en la mayoría de los casos siempre hay elementos repetidos en estos conjuntos, lo que da como resultado que el tiempo para evaluar un individuo se menor que si no se eliminaran estos elementos repetidos.

Los algoritmos 7, 8, 9, 10 y 11 muestran como calcular cada función de evaluación.

Algoritmo 6 Indica cuantos elementos de a no están en b .

Entrada: Dos listas ordenadas a, b y sus pesos pa y pb.
Salida: Valor numérico que indica cuantos elementos de a no están en b y viceversa

```
1: function MI-INTERSECCION(a, b, pa, pb)
2:     if (a[0] > b[|b| − 1])|(b[0] > a[|a| − 1]) then
3:         return (pa[0] + ... + pa[|pa| − 1]) + (pb[0] + ... + pb[|pb| − 1])
4:     else
5:         bandera ← true
6:         i ← 0
7:         s ← ∅
8:         while i < |a| do
9:             e ← x[i]
10:            j ← 0
11:            if y = ∅ then
12:                s ← (s + (pb[i] + pb[i + 1] + ... + pb[|b| − 1]))
13:                bandera ← false
14:                i ← (i + |a|)
15:            end if
16:            while bandera do
17:                aux ← b[j]
18:                if e = aux then
19:                    bandera ← false
20:                    Elimina al elemento pb[j] de pb
21:                    Elimina al elemento aux de b
22:                else if j < (|b| − 1) then
23:                    j ← (j + 1)
24:                else
25:                    bandera ← false
26:                    s ← (s + pa[i])
27:                end if
28:            end while
29:            bandera ← true
30:        end while
31:        return s
32:    end if
33: end function
```

Algoritmo 7 Función de evaluación número 1.

Entrada: Un individuo (DS,VP y DV) y la muestra de supervisión (S1 y S2)
Salida: La evaluación del individuo

1: **procedure** FE1($DS, VP, DV, S1, S2$)
2: $tamt \leftarrow (|S1| + |S2|)$
3: $indice \leftarrow 0$
4: $f1 \leftarrow 0$
5: $f2 \leftarrow 0$
6: $s1Aux \leftarrow \emptyset$
7: **for all** $s \in s1$ **do**
8: $cla1 \leftarrow$ Clasificador1(DS, VP, s)
9: $cla2 \leftarrow (cla1 \oplus DV)$
10: **Inserta** $cla1$ **en** $s1Aux$
11: **if** $cla2 = 0$ **then**
12: $f2 \leftarrow (f2 + 1)$
13: **end if**
14: **end for**
15: $s2Aux \leftarrow \emptyset$
16: **for all** $s \in s2$ **do**
17: $cla1 \leftarrow$ Clasificador1(DS, VP, s)
18: $cla2 \leftarrow (cla1 \oplus DV)$
19: **Inserta** $cla1$ **en** $s2Aux$
20: **if** $cla2 = 1$ **then**
21: $f2 \leftarrow (f2 + 1)$
22: **end if**
23: **end for**
24: **while** $indice < |s1Aux|$ **do**
25: **if** $s1Aux[indice] \in s2Aux$ **then**
26: **Elimina al elemento** $s1Aux[indice]$ **de** $s1Aux$
27: **Elimina al elemento** $s1Aux[indice]$ **de** $s2Aux$
28: **else**
29: $f1 \leftarrow (f1 + 1)$
30: **end if**
31: $indice \leftarrow (indice + 1)$
32: **end while**
33: $f1 \leftarrow (f1 + |s2Aux|)$
34: **return** $\left(\frac{f1 * f2 * 100}{tamt^2}\right)$
35: **end procedure**

Algoritmo 8 Función de evaluación número 2.

Entrada: Un individuo (DS,VP y DV) y la muestra de supervisión (S1 y S2)
Salida: La evaluación del individuo

1: **procedure** FE2($DS, VP, DV, S1, S2$)
2: $tam1 \leftarrow |S1|$
3: $tam2 \leftarrow |S2|$
4: $indice \leftarrow 0$
5: $f1 \leftarrow 0$
6: $f21 \leftarrow 0$
7: $f22 \leftarrow 0$
8: $s1Aux \leftarrow \emptyset$
9: **for all** $s \in s1$ **do**
10: $cla1 \leftarrow$ Clasificador1(DS, VP, s)
11: $cla2 \leftarrow (cla1 \oplus DV)$
12: **Inserta** $cla1$ **en** $s1Aux$
13: **if** $cla2 = 0$ **then**
14: $f21 \leftarrow (f21 + 1)$
15: **end if**
16: **end for**
17: $s2Aux \leftarrow \emptyset$
18: **for all** $s \in s2$ **do**
19: $cla1 \leftarrow$ Clasificador1(DS, VP, s)
20: $cla2 \leftarrow (cla1 \oplus DV)$
21: **Inserta** $cla1$ **en** $s2Aux$
22: **if** $cla2 = 1$ **then**
23: $f22 \leftarrow (f22 + 1)$
24: **end if**
25: **end for**
26: $f1 \leftarrow |s1Aux \cap s2Aux|$
27: **return** $\left(\frac{f21 * f22 * (tamt - f1) * 100}{(tam1 + tam2) * tam1 * tam2} \right)$
28: **end procedure**

Algoritmo 9 Función de evaluación número 3.

Entrada: Un individuo (DS,VP y DV) y la muestra de supervisión (S1 y S2)
Salida: La evaluación del individuo

1: **procedure** FE3($DS, VP, DV, S1, S2$)
2: $tamt \leftarrow (|S1| + |S2|)$
3: $indice \leftarrow 0$
4: $f1 \leftarrow 0$
5: $f2 \leftarrow 0$
6: $s1Aux \leftarrow \emptyset$
7: **for all** $s \in s1$ **do**
8: $cla1 \leftarrow$ Clasificador1(DS, VP, s)
9: $cla2 \leftarrow (cla1 \oplus DV)$
10: **Inserta** $cla1$ **en** $s1Aux$
11: **if** $cla2 = 0$ **then**
12: $f2 \leftarrow (f2 + 1)$
13: **end if**
14: **end for**
15: $s2Aux \leftarrow \emptyset$
16: **for all** $s \in s2$ **do**
17: $cla1 \leftarrow$ Clasificador1(DS, VP, s)
18: $cla2 \leftarrow (cla1 \oplus DV)$
19: **Inserta** $cla1$ **en** $s2Aux$
20: **if** $cla2 = 1$ **then**
21: $f2 \leftarrow (f2 + 1)$
22: **end if**
23: **end for**
24: **while** $indice < |s1Aux|$ **do**
25: **if** $s1Aux[indice] \in s2Aux$ **then**
26: **Elimina al elemento** $s1Aux[indice]$ **de** $s1Aux$
27: **Elimina al elemento** $s1Aux[indice]$ **de** $s2Aux$
28: **else**
29: $f1 \leftarrow (f1 + 1)$
30: **end if**
31: $indice \leftarrow (indice + 1)$
32: **end while**
33: $f1 \leftarrow (f1 + |s2Aux|)$
34: Elimina los elementos repetidos de $s1Aux$
35: Elimina los elementos repetidos de $s2Aux$
36: **return** $\left(\frac{f1 * f2 * 100}{tamt^2} + \frac{|s1Aux| + |s2Aux|}{2^{|VP|}} * 25\right)$
37: **end procedure**

Algoritmo 10 Función de evaluación número 4.

Entrada: Un individuo (DS,VP y DV) y la muestra de supervisión (S1 y S2)
Salida: La evaluación del individuo

1: **procedure** FE4($DS, VP, DV, S1, S2$)
2: $tamt \leftarrow (|S1| + |S2|)$
3: $f1 \leftarrow 0$
4: $f2s1 \leftarrow 0$
5: $f2s2 \leftarrow 0$
6: $s1Aux \leftarrow \emptyset$
7: **for all** $s \in s1$ **do**
8: $cla1 \leftarrow$ CLASIFICADOR1(DS, VP, s)
9: $cla2 \leftarrow (cla1 \oplus DV)$
10: **Inserta** $cla1$ **en** $s1Aux$
11: $f2s1 \leftarrow (f2s1 + cla2)$
12: **end for**
13: $s2Aux \leftarrow \emptyset$
14: **for all** $s \in s2$ **do**
15: $cla1 \leftarrow$ CLASIFICADOR1(DS, VP, s)
16: $cla2 \leftarrow (cla1 \oplus DV)$
17: **Inserta** $cla1$ **en** $s2Aux$
18: $f2s2 \leftarrow (f2s2 + cla2)$
19: **end for**

20: Ordenar $s1Aux$ de menor a mayor.
21: Ordenar $s2Aux$ de menor a mayor.
22: $pe1 \leftarrow$ CREA-PESOS(s1Aux)
23: $pe2 \leftarrow$ CREA-PESOS(s1Aux)
24: Elimina los elementos repetidos de $s1Aux$
25: Elimina los elementos repetidos de $s2Aux$
26: $f1 \leftarrow$ MI-INTERSECCION($s1Aux, s2Aux, pe1, pe2$)
27: $f1 \leftarrow (f1/tamt)$

28: **if** $f2s1 < f2s2$ **then**
29: $f2 \leftarrow (f2s2 + (|S1| - f2s1))$
30: **else**
31: $f2 \leftarrow (f2s1 + (|S2| - f2s2))$
32: **end if**
33: **return** $\left(\frac{f1 * f2 * 100}{tamt}\right)$
34: **end procedure**

Algoritmo 11 Función de evaluación número 5.

Entrada: Un individuo (DS,VP y DV), la muestra de supervisión (S1 y S2) y las variables $fep1$ y $fep2$ que sirven para modificar la evaluación.

Salida: La evaluación del individuo.

1: **procedure** FE5($DS, VP, DV, S1, S2, fep1, fep2$)
2: $tamt \leftarrow (|S1| + |S2|)$
3: $N \leftarrow |S1[1]|$
4: $f1 \leftarrow 0$
5: $f2s1 \leftarrow 0$
6: $f2s2 \leftarrow 0$
7: $s1Aux \leftarrow \emptyset$
8: **for all** $s \in s1$ **do**
9: $cla1 \leftarrow$ CLASIFICADOR1(DS, VP, s)
10: $cla2 \leftarrow (cla1 \oplus DV)$
11: **Inserta** $cla1$ **en** $s1Aux$
12: $f2s2 \leftarrow (f2s2 + cla2)$
13: **end for**
14: $s2Aux \leftarrow \emptyset$
15: **for all** $s \in s2$ **do**
16: $cla1 \leftarrow$ CLASIFICADOR1(DS, VP, s)
17: $cla2 \leftarrow (cla1 \oplus DV)$
18: **Inserta** $cla1$ **en** $s2Aux$
19: $f2s2 \leftarrow (f2s2 + cla2)$
20: **end for**
21: Ordenar $s1Aux$ de menor a mayor.
22: Ordenar $s2Aux$ de menor a mayor.
23: $pe1 \leftarrow$ CREA-PESOS(s1Aux)
24: $pe2 \leftarrow$ CREA-PESOS(s1Aux)
25: Elimina los elementos repetidos de $s1Aux$
26: Elimina los elementos repetidos de $s2Aux$
27: $f1 \leftarrow$ MI-INTERSECCION($s1Aux, s2Aux, pe1, pe2$)
28: $f1 \leftarrow (f1/tamt)$
29: **if** $f2s1 < f2s2$ **then**
30: $f2 \leftarrow (f2s2 + (|S1| - f2s1))$
31: **else**
32: $f2 \leftarrow (f2s1 + (|S2| - f2s2))$
33: **end if**
34: **return** $\left(\frac{f1 * f2 * 100}{tama} + \frac{|s1Aux| + |s2Aux|}{2^{|VP|}} * fep1 + \frac{N - |VP|}{N - 2} * fep2\right)$
35: **end procedure**

4.1.4. Selección.

El individuo se selecciona con una probabilidad proporcional a la de su evaluación. El algoritmo 12 se encarga de seleccionar a un individuo.

Algoritmo 12 Ruleta

Entrada: Un conjunto de valores x
Salida: El índice que se obtuvo con probabilidad proporcional a los valores de x

```
1: procedure RULETA(x)
2:     Generar un número aleatorio r ∈ [0, 1]
3:     S ← Suma de todos los elementos de x.
4:     c ← r * S
5:     i ← 0
6:     C_a ← x[i]
7:     while C_a < c do
8:         i ← (i + 1)
9:         C_a ← (C_a + x[i])
10:    end while
11:    return i
12: end procedure
```

4.1.5. Cruzamiento.

El cruzamiento se realiza en un punto y se hará en las 3 partes de cada individuo. Para el DV y DS el cruzamiento se realiza como se muestran en los algoritmos 13 y 14 respectivamente.

Algoritmo 13 Cruce del DV de dos individuos.

Entrada: Dos DVs.
Salida: Dos DVs hijos.

```
1: procedure CRUCE-DV(DVp, DVm)
2:     tam1 ← |DVp|
3:     tam2 ← |DVm|
4:     salida ← {∅, ∅}
5:     if tam1 < tam2 then
6:         corte ← tam2
7:     else
8:         corte ← tam1
9:     end if
10:    salida[0] ← {DVp[0; corte], DVm[(corte + 1); tam2]}
11:    salida[1] ← {DVm[0; corte], DVp[(corte + 1); tam1]}
12:    return salida
13: end procedure
```

Algoritmo 14 Cruce del DS de dos individuos.

Entrada: Dos DS's y el punto donde se realizara la cruza.
Salida: Dos DSs hijos.

```
1: procedure CRUCE-DS(DSp, DSm, corte)
2:     tam ← |DSp|
3:     salida ← {∅, ∅}
4:     salida[0] ← {DSp[0; corte], DSm[(corte + 1); tam]}
5:     salida[1] ← {DSm[0; corte], DSp[(corte + 1); tam]}
6:     return salida
7: end procedure
```

Para el VP primero hay que cambiar el formato de este mismo a una cadena que contenga el número 1 en la primera partición, el número 2 en la segunda y así sucesivamente, una vez que se tiene una cadena de números que representan al VP, ahora sólo se aplica el cruzamiento en un punto, el algoritmo 15 muestra el cruce de dos VP's.

Ejemplo 4.1.1 ***Se tienen 2 individuos que se desean cruzar:***
Individuo1: DS_1 =***(1 1 1 1 0 1 0 0 1 1 0),*** VP_1 = ***(4 2 5),*** DV_1 =***(1 1 0)***
Individuo2: DS_2 =***(1 0 1 0 1 1 1 0 1 1 1),*** VP_2 = ***(3 2 2 4),*** DV_2 =***(1 1 0 1)***

1. Cruce del DS y VP. De manera aleatoria se selecciona un punto de cruce, supongamos que es $corte = 5$.
   $DSh1$ =(1 1 1 1 0 1 1 0 1 1 1), $DSh2$ =(1 0 1 0 1 1 0 0 1 1 0)
   En el caso del VP antes de cruzar se necesita expandir como se muestra a continuación:
   VP_{1exp} =(1 1 1 1 2 2 3 3 3 3 3), VP_{2exp} =(1 1 1 2 2 3 3 4 4 4 4)
   Ahora que ya se tienen expandidos cada VP se procede a cruzar
   VPh_{1exp} =(1 1 1 1 2 2 3 4 4 4 4), VPh_{2exp} =(1 1 1 2 2 3 3 3 3 3 3)
   Quedando finalmente como: $VPh1$ =(4 3 4) y $VPh2$ = (3 2 6)

2. Cruce del DV. De manera aleatoria se selecciona un punto de corte, supongamos que es $corte = 0$
   $DVh1$ =(1 1 0 1), $DVh2$ =(1 1 0)
   Pero como se observa el $|DVh1| = 4$, mientras que en el cruce del VP se obtuvo que $|VPh1| = 3$, lo cual es incorrecto ya que la cardinalidad de $DVh1$ y $VPh1$ deben de ser iguales, entonces se prosigue a corregir al $DVh1$ quitando el elemento que sobra quedando finalmente: $DVh1$ = (1 1 0) y $DVh2$ =(1 1 0).

Algoritmo 15 Cruce del VP de dos individuos.

Entrada: Dos VPs y el punto donde se realizara la cruza.
Salida: Dos VPs hijos.

```
1: procedure CRUCE-VP(VPp, VPm, corte)
2:     aux1 ← EXPANDE-VP(VPp)
3:     aux2 ← EXPANDE-VP(VPm)
4:     salida ← {∅, ∅}
5:     salida[0] ← {aux1[0; corte], aux2[(corte + 1); tam]}
6:     salida[1] ← {aux2[0; corte], aux1[(corte + 1); tam]}
7:     salida[0] ← AUX-CRUCE-VP(salida[0])
8:     salida[1] ← AUX-CRUCE-VP(salida[1])
9:     return salida
10: end procedure
```

```
1: function EXPANDE-VP(VP)
2:     salida ← ∅
3:     for i = 0, (|VP| − 1) do
4:         Crea un conjunto con VP[i] número de i's y concatenalo a salida
5:     end for
6:     return salida
7: end function
```

```
1: function AUX-CRUCE-VP(VPe)
2:     salida ← ∅
3:     cuenta ← 1
4:     for i = 0, (|VPe| − 2) do
5:         if VPe[i] == VPe[i + 1] then
6:             cuenta ← (cuenta + 1)
7:         else if (cuenta == 1)&(VPe[i + 1] ≠ VPe[i − 1]) then
8:             Inserta cuenta en salida
9:             cuenta ← 1
10:        else
11:            cuenta ← (cuenta + 1)
12:        end if
13:        Inserta cuenta en salida
14:    end for
15:    return salida
16: end function
```

4.1.6. Mutación.

El proceso de mutación de un DV y un DS se realiza mediante el algoritmo 16 que por cada bit del DS o DV cambiará por su complemente con una probabilidad de pm.

Algoritmo 16 Mutación de un DS o un DV.

Entrada: Un DS o DV
Salida: Un DS o DV mutado.

```
1: procedure MUTA-DSDV(x, pm)
2:     for i = 0, (|x| − 1) do
3:         if RANDOM() < pm then
4:             if x[i] = 0 then
5:                 x[i] ← 1
6:             else
7:                 x[i] ← 0
8:             end if
9:         end if
10:    end for
11:    return x
12: end procedure
```

La mutación del VP se utiliza el algoritmo 17 el cuál con una probabilidad pm alterara un elemento de VP en una magnitud de $muta$.

Algoritmo 17 Mutación de un VP.

Entrada: Un VP.
Salida: Un VP mutado.

```
1: procedure MUTA-VP(VP, pm, muta)
2:     i_0 ← RANDOM()
3:     i_1 ← RANDOM()
4:     if RANDOM() < pm then
5:         if VP[i_0] > VP[i_1] then
6:             VP[i_0] ← (VP[i_0] − muta)
7:             VP[i_1] ← (VP[i_1] + muta)
8:         else if VP[i_1] > VP[i_0] then
9:             VP[i_0] ← (VP[i_0] + muta)
10:            VP[i_1] ← (VP[i_1] − muta)
11:        end if
12:    end if
13:    return VP
14: end procedure
```

Ejemplo 4.1.2 ***Tenemos el siguiente individuo que va sufrir una mutación:***
DS =***(0 1 1 1 0 1 1 1 0 1 0 0 1 1 0)***, VP =***(4 3 4 4)***, DV =***(1 1 0 1)***

Después de sufrir una mutación con las siguientes probabilidades $pm = 0{,}3$, $pm\_vp = 0{,}5$ y $muta = 2$ el individuo queda como sigue:
DS =(0 1 1 1 **1** 1 1 1 0 1 0 0 **0** 1 0), VP =(**2** 3 **6** 4), DV =(1 **0** 0 1)
Las partes que se vieron afectadas en la mutación se encuentran remarcadas y de color azul.

4.1.7. Validación de individuos.

En los ejemplos 4.1.1 y 4.1.2 se mostró como se lleva a cabo la cruza y la mutación de dos individuos, en ambos ejemplos después de realizar el cruce o la mutación, el individuo resultante es un individuo con un DS y un DV no válidos; es por esta razón que después de la cruza y mutación el individuo resultante debe de ser validado. El VP no es necesario validar porque se trata de sólo las particiones del DS y el DS se válida de acuerdo al VP.

Utilizando directamente el algoritmo 3 se puede validar un DV, mientras que para la validación de un DS se tiene el algoritmo 18, este algoritmo recibe dos DS's y regresa los mismos dos DS's si ambos son válidos de le contrario hace las modificaciones correspondientes para que sean válidos.

Algoritmo 18 Válida un DS

Entrada: Un DSp con su respectivo VP.
Salida: DSp si se trata de un DS válido, de le contrario regresa un DS válido modificado en las partes donde no era válido.

```
1: procedure ValidaC-DS(DSp, VP)
2:     a ← 0
3:     DSS ← DSp
4:     for i ← 0, (|VP| − 1) do
5:         b ← (a + VP[i] − 1)
6:         DS[a; b] ← ValidaC-DV(DSp[a; b])
7:         a ← (a + VP[i])
8:     end for
9: end procedure
```

4.1.8. AG + elitismo (AGT)

Ahora que ya se tienen todos los elementos de un AG, podemos realizar el diseño del algoritmo que implementa el AGT que se mostró al principio en el diagrama de flujo 4.3.

El algoritmo 19 muestra el AGT-DSDV, en este algoritmo tenemos como entrada la muestra de supervisión y varios parámetros más. A continuación se presenta la descripción de cada parámetro de entrada:

- p: Tamaño de la población.
- nG: Número de generaciones que va a calcular el algoritmo.
- pm: Probabilidad de mutación del DS y DV.
- pm_vp: Probabilidad de mutación del VP.
- pc: Probabilidad de cruzamiento.
- k: Variable referente a la característica de elitismo del AG, indica cada cuantas generaciones se van a seleccionar los mejores p individuos.
- $inicialm$: Indica el rango que puede tomar m para crear la población inicial, como se vio en la sección 2.3.
- $S1, S2$: Muestra de supervisión.

Se realizaron múltiples pruebas y simulaciones con el algoritmo 19 y se encontró que los mejores valores para los parámetros son los siguientes:

- $p \geq 100$ y $p \leq nG \leq p * 2$
- $0{,}1 < pm \leq 0{,}5$ y $0{,}2 < pm\_vp \leq 0{,}5$
- $pc > 0{,}7$ y $0{,}04nG < k < 0{,}2nG$
- $inicialm = (a, b)$, el rango que pueden tomar a y b es: $2 \leq a, b \leq \frac{n}{2}$. Lo más recomendable es comenzar con un $a = 2$ y $b = 0{,}2n$, pero esto no es recomendable para todos los casos ya que b depende también del número de patrones de la muestra de supervisión.

Algoritmo 19 Algoritmo genético AGT-DSDV (etapa de entrenamiento del clasificador DS-DV).

Entrada: $p, nG, pm, pm\_vp, pc, k, inicialm, S1, S2$
Salida: Población de p individuos con las mejores evaluaciones encontradas.

1: **procedure** AGT-DSDV($p, nG, pm, pm\_vp, muta, pc, k, inicialm, S1, S2$)
2: $P_{np} \leftarrow \emptyset$
3: $P_p \leftarrow \emptyset$
4: $E_{np} \leftarrow \emptyset$
5: $E_p \leftarrow \emptyset$
6: $G \leftarrow 1$
7: $C \leftarrow 1$
8: $incm \leftarrow 1/(nG * 0{,}9)$
9: Generar de forma pseudo-aleatoria p números que estén dentro del rango de $incialm$, estos serán el número m de los individuos de la población inicial.
10: Crear la población inicial (utilizando el algoritmo 2) que consta de p individuos y cada individuo tiene un DS, VP y DV.
11: Evaluar cada individuo de P y guardar las evaluaciones en E.
12: **while** $G \leq nGeneraciones$ **do**
13: **for** $j \leftarrow 0, (p/2)$ **do**
14: Seleccionar el individuo número RULETA(E) de P como padre.
15: El individuo padre esta conformado por pDS, pVP y pDV.
16: Seleccionar el individuo número RULETA(E) de P como madre.
17: El individuo madre esta conformado por mDS, mVP y mDV.
18: **if** $RANDOM() < pc$ **then**
19: $cr \leftarrow RANDOM()$
20: $hDS \leftarrow$ CRUCE-DS(pDS, mDS, cr)
21: $hVP \leftarrow$ CRUCE-VP(pVP, mVP, cr)
22: $hDV \leftarrow$ CRUCE-DV(pDV, mDV)
23: $hDS \leftarrow \{$MUTA-DSDV($hDS[0], pm$), MUTA-DSDV($hDS[1], pm$)$\}$
24: $hVP \leftarrow \{$MUTA-VP($hVP[0], pm\_vp, muta$), MUTA-VP($hVP[1], pm\_vp, muta$)$\}$
25: $hDV \leftarrow \{$MUTA-DSDV($hDV[0], pm$), MUTA-DSDV($hDV[1], pm$)$\}$
26: $hDS \leftarrow \{$VALIDAC-DS($hDS[0], hVP[0]$), VALIDAC-DS($hDS[1], hVP[1]$)$\}$
27: $hDV \leftarrow \{$VALIDAC-DV($hDV[0]$), VALIDAC-DV($hDV[1]$)$\}$
28: **Inserta** hDS, hVP y hDV **en** P_{np}
29: **else**
30: **Inserta** los individuos padre y madre **en** P_p
31: **Inserta** las evaluaciones del individuo padre y madre **en** E_p
32: **end if**
33: **end for**

34: ▷ Continuación del AGT-DSDV
35: Evaluar cada individuo de P_{np} y guardar las evaluaciones en E_{np}.
36: **Inserta** E_{np} **en** E
37: **Inserta** E_p **en** E
38: **Inserta** P_{np} **en** P
39: **Inserta** P_p **en** P
40: **if** $C * k = G$ **then**
41: Seleccionar los p mejores individuos de P.
42: Estos p individuos son ahora P.
43: Guardad todas las evaluaciones de P en E.
44: $C \leftarrow (C + 1)$
45: **end if**
46: **if** $G > (nG * 0{,}6)$ **then**
47: $pm \leftarrow (pm + incm)$
48: $pm\_vp \leftarrow (pm\_vp + incm)$
49: **end if**
50: $G \leftarrow (G + 1)$
51: $P_{np} \leftarrow \emptyset$
52: $P_p \leftarrow \emptyset$
53: $E_{np} \leftarrow \emptyset$
54: $E_p \leftarrow \emptyset$
55: **end while**
56: **return** P
57: **end procedure**

4.2. Complejidad del clasificador DS-DV

La complejidad del clasificador DS-DV en la etapa de entrenamiento es la siguiente:

El algoritmo 19 realiza nG generaciones y en cada generación se crea una población nueva de tamaño p; cada vez que se seleccionan dos individuos, después estos dos individuos pasan a la nueva generación o se cruzan (con una probabilidad de pc) y crean dos nuevos individuos. Por lo tanto se realizan el siguiente número de operaciones que realiza es:

$$\frac{nG * p * pc}{2} (O_{cruce} + pm * O_{muta} + O_{veri} + O_{FE}) \tag{4.4}$$

Donde:
$O_{cruce} = n + m$: es la complejidad de los algoritmos de cruzamiento
$O_{muta} = 2n + 2m$: es la complejidad de los algoritmos para la la mutación.
$O_{veri} = 2n + 2m$: es la complejidad de los algoritmos de verificación.
O_{FE}: es la complejidad del algoritmo de la función de evaluación.

La complejidad O_{cruce}, O_{muta} y O_{veri} son todas lineales para todos los casos (mejor, promedio y peor).

La complejidad de O_{FE} no es lineal y varia de acuerdo al número de patrones de la muestra de supervisión. Por lo tanto vamos a analizar la función de evaluación:

Primero hay operaciones que siempre se realizan, estas operaciones son las siguientes:

1. Se calcula la primera etapa del clasificador DS-DV para todos los patrones de $S1$ y $S2$ que consiste en la operación: $\diamond$; y el resultado de cada operación es guardado en $s1Aux$ y $s2Aux$ respectivamente. Después se calcula la segunda etapa del clasificador DS-DV a todos los elementos de $s1Aux$ y $s2Aux$ y el resultado es guardado en $f11$ y $f12$ respectivamente. Hasta este punto se han realizado $2x + 2y$ operaciones.

2. Después se ordenan los elemento de $s1Aux$ y $s2Aux$ de menor a mayor y se eliminan elementos repetidos, en esta parte se tienen que se realizan la siguiente cantidad de operaciones:
   $x_r \log_2(x_r) + y_r \log_2(y_r) + x + y$
   Donde:
   x_r es la cardinalidad de $s1Aux$ después de haber eliminado sus elementos repetidos.
   y_r es la cardinalidad de $s2Aux$ después de haber eliminado sus elementos repetidos.

Posteriormente hay que cuantificar cuantos elementos de $s1Aux$ también están en $s2Aux$, y esto se hace con los algoritmos 6, para este algoritmo analizaremos los siguientes casos:

Mejor caso. En el mejor de los casos se tiene que todos los elementos del conjunto $s1Aux$ no están en $s2Aux$ y todos los elementos de $s1Aux$ son menores al menor elemento de $s2AUx$ o viceversa, esto es: $s2Aux[0] > s1Aux[x - 1]$

Por lo tanto la complejidad es:

$$O_{FE} = 2x + 2y + x_r \log_2(x_r) + y_r \log_2(y_r) + x + y + 1$$

Caso medio. En el caso medio tendremos que todos elementos de $s1Aux$ no están en $s2Aux$, y se cumple lo siguiente: $s1Aux[\frac{y}{2}] > s2Aux[0]$
Por lo tanto la complejidad es:

$$O_{FE} = 2x + 2y + x_r \log_2(x_r) + y_r \log_2(y_r) + x + y + \frac{(x_r - 1) * (y_r - 1)}{4}$$

Peor caso. En el peor de los casos tendremos que ningún elemento de $s1Aux$ esta en $s2Aux$ y se cumplen lo siguiente: $s2Aux[y_r - 2] < s1Aux[l] < s2Aux[y_r - 1]$ con $0 \leq l < (x_r - 1)$
Por lo tanto la complejidad es:

$$O_{FE} = 2x + 2y + x_r \log_2(x_r) + y_r \log_2(y_r) + x + y + (x_r - 1) * (y_r - 1)$$

Por lo tanto sustituyendo estos valores en la ecuación 4.4, tenemos que la complejidad del algoritmo 19 (AGT-DSDV) es:

Mejor caso.

$$\frac{nG * p * pc}{2} (3n + 3m + 2pm * (n + m) + 3x + 3y + x_r \log_2(x_r) + y_r \log_2(y_r) + 1)$$

Caso medio.

$$\frac{nG * p * pc}{2} \left(3n + 3m + 2pm * (n + m) + 3x + 3y + x_r \log_2(x_r) + y_r \log_2(y_r) + \frac{(x_r - 1) * (y_r - 1)}{4}\right)$$

Peor caso.

$$\frac{nG * p * pc}{2} (3n + 3m + 2pm * (n + m) + 3x + 3y + x_r \log_2(x_r) + y_r \log_2(y_r) + (x_r - 1) * (y_r - 1))$$

4.3. Clasificador MCJJ2

En esta sección se diseñan los algoritmos necesarios para la etapa de entrenamiento del clasificador MCJJ2. La idea es construir un árbol de decisión que este conformados sólo de nodos que contienen un DV y los nodos hojas son las etiquetas que representan una clase. En un árbol DV cada nodo está constituido por un DV, cuando un patrón llegue a un nodo se le aplicará la operación $\odot$ (ver definición 4.1.1), como esta operación siempre nos entrega un 1 ó un 0 , el árbol DV será un árbol binario. Así un patrón recorrerá el árbol DV hasta llegar a un nodo hoja el cuál estará etiquetado con la clase a la que pertenece este patrón. La idea de crear un árbol DV es para mejorar el clasificador DS-DV, ya que como se ve en la sección de simulaciones el clasificador DS-DV tiene algunos inconvenientes. Veamos un ejemplo 4.3.1 pequeño que ilustra como el árbol DV es más versátil que sólo tener un DV (que sería las segunda etapa del clasificador DSDV).

Ejemplo 4.3.1 ***Tenemos los siguiente conjuntos (muestra de supervisión):***
$S_1 = \{(1001), (1010), (0101), (0010)\}$
$S_2 = \{(0111), (0001), (0110), (0100)\}$
Se requiere separar con un DV a S_1 ***y*** S_2***.***

Tenemos que $n = 4$, por lo tanto se tiene $2^4 = 16$ posibles combinaciones binarias, de las cuales 14 son DV's validos. En la tabla 4.1 los 14 DVs y el resultado de la operación $\odot$ con cada elemento de S_1 y S_2, como se observa ningún DV cumple con la ecuación 4.3.

Tabla 4.1: Resultados de todos los DVs posibles de tamaño 4.

DV	$P_{1i} \odot DV$	$P_{2i} \odot DV$
(0001)	1 0 1 0	1 1 0 0
(0010)	0 1 0 1	1 0 1 0
(0011)	1 1 1 1	0 1 1 0
(0100)	0 0 1 0	1 0 1 1
(0101)	1 0 0 0	0 1 1 1
(0110)	0 1 1 1	0 0 0 1
(0111)	1 1 0 1	1 1 0 1
(1000)	1 1 0 0	0 0 0 0
(1010)	1 0 0 1	1 0 1 0
(1011)	0 0 1 1	0 1 1 0
(1100)	1 1 1 0	1 0 1 1
(1101)	0 1 0 0	0 1 1 1
(1110)	1 0 1 1	0 0 0 1
(1111)	0 0 0 1	1 1 0 1

En la figura 4.4 se muestra el árbol de decisión que si logra dividir la muestra de supervsión en dos, que es lo que se desea. Para lograrlo se necesitaron 3 DVs.

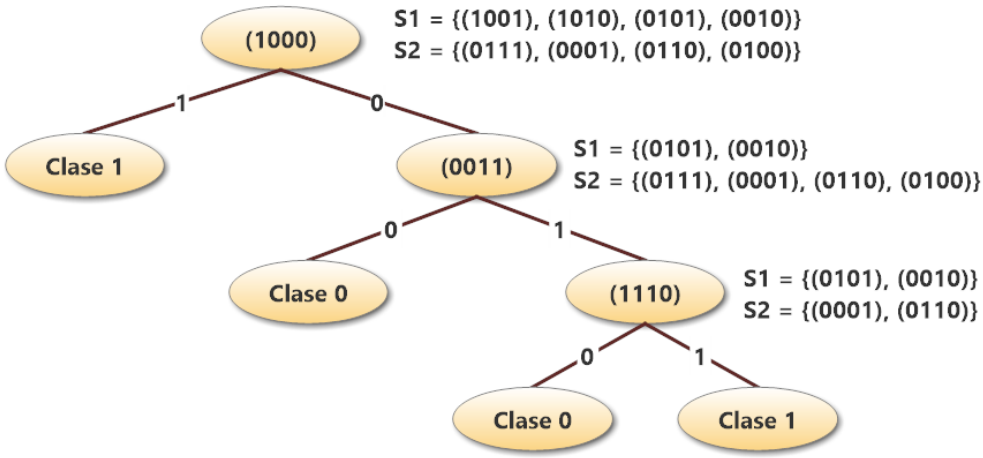


Figura 4.4: Árbol de decisiones para los conjuntos S_1 y S_2 del ejemplo 4.3.1

4.3.1. Algoritmos para crear el MCJJ2

Para crear el árbol de DV's se utilizará un algoritmo ID3 que se vio en la sección 2.7.1, con las siguientes consideraciones:

- ⋆ Cada nodo será un DV valido.
- ⋆ No hay atributos, en su lugar hay DVs validos de los cuales escoger para hacer la división del nodo.
- ⋆ El árbol DV es un árbol binario.
- ⋆ Para buscar el mejor DV de cada nodo, se utiliza un AG.

Para lograr que el árbol de DV's sea construido de manera que pueda realizar una clasificación correcta siguiendo el ejemplo de la muestra de supervisión, es necesario seleccionar el mejor DV para cada nodo. Para este fin se utiliza un AGT con una función de evolución que mida la ganancia de información mediante la entropia.
El algoritmo 20 muestra el AGT que realiza la búsqueda del mejor DV para una muestra de supervisión definida por $s1$ y $s2$.

El algoritmo 21 muestra la función de evolución utilizada por el AGT-ADV, esta función de evolución cálculo la ganancia de la ecuación 2.15, en dónde $I(s)$ se calcula con la entropía (ecuación 2.16).

Algoritmo 20 Algoritmo genético AGT-ADV para encontrar el mejor DV.

Entrada: Los parámetros del propios del AG p, nG, pm, pc, k y los conjuntos $s1, s2$ pertenecientes a clases diferentes.

Salida: Población de p DVs con las mejores evaluaciones encontradas.

procedure AGT-ADV($p, nG, pm, pc, k, s1, s2$)
  $P_{np} \leftarrow \emptyset$
  $P_p \leftarrow \emptyset$
  $E_{np} \leftarrow \emptyset$
  $E_p \leftarrow \emptyset$
  $G \leftarrow 1$
  $C \leftarrow 1$
  $P \leftarrow \emptyset$
  for $i \leftarrow 0, p$ **do** ▷ Creación de la población inicial.
    Inserta CREA-DV($|s1[0]|$) **en** P
  end for
  Evaluar cada individuo de P y guardar las evaluaciones en E.
  while $G \leq nGeneraciones$ **do**
    for $j \leftarrow 0, (p/2)$ **do**
      Seleccionar el individuo número RULETA(E) de P como padre y llamarlo pDV.
      Seleccionar el individuo número RULETA(E) de P como madre y llamarlo mDV.
      if $RANDOM() < pc$ **then**
        $cr \leftarrow RANDOM()$
        $hDV \leftarrow$ CRUCE-DV(pDV, mDV)
        $hDV \leftarrow \{$MUTA-DSDV($hDV[0], pm$), MUTA-DSDV($hDV[1], pm$)$\}$
        $hDV \leftarrow \{$VALIDAC-DV($hDV[0]$), VALIDAC-DV($hDV[1]$)$\}$
        Inserta hDS, hVP y hDV **en** P_{np}
      else
        Inserta pDV y mDV **en** P_p
        Inserta las evaluaciones del individuo pDV y mDV **en** E_p
      end if
    end for
    Evaluar cada individuo de P_{np} y guardar las evaluaciones en E_{np}.
    Inserta E_{np} **en** E
    Inserta E_p **en** E
    Inserta P_{np} **en** P
    Inserta P_p **en** P

▷ Continuación del AGT-ADV

if $C * k = G$ **then**
Seleccionar los p mejores individuos de P.
Estos p individuos son ahora P.
Guardad todas las evaluaciones de P en E.
$C \leftarrow (C + 1)$
end if

$G \leftarrow (G + 1)$
$P_{np} \leftarrow \emptyset$
$P_p \leftarrow \emptyset$
$E_{np} \leftarrow \emptyset$
$E_p \leftarrow \emptyset$
end while

return El mejor individuo de la población actual P.
end procedure

Algoritmo 21 Función de evaluación para el AGT-ADV.

Entrada: $s1, s2, DV$
Salida: Valor numérico entre 0 y 1.

function FE-GE($s1, s2, DV$)
$u1 \leftarrow 0$
$t \leftarrow (|s1| + |s2|)$
for all $s \in s1$ **do**
$u1 \leftarrow (u1 + s \odot DV)$
end for
$u2 \leftarrow 0$
for all $s \in s2$ **do**
$u2 \leftarrow (u2 + s \odot DV)$
end for
$c1 \leftarrow (|s1| - u1)$
$c2 \leftarrow (|s2| - u2)$
$ec \leftarrow \left(-\left(\frac{c1}{c2+c1}\right)\log_2\left(\frac{c1}{c2+c1}\right) - \left(\frac{c2}{c2+c1}\right)\log_2\left(\frac{c2}{c2+c1}\right)\right)$
$eu \leftarrow \left(-\left(\frac{u1}{u1+u2}\right)\log_2\left(\frac{u1}{u1+u2}\right) - \left(\frac{u2}{u1+u2}\right)\log_2\left(\frac{u2}{u1+u2}\right)\right)$
return $\left(-\left(\frac{|s1|}{t}\right)\log_2\left(\frac{|s1|}{t}\right) - \left(\frac{|s2|}{t}\right)\log_2\left(\frac{|s2|}{t}\right) + \left(\frac{c1+c2}{t}\right) * ec + \left(\frac{u1+u2}{t}\right) * eu\right)$
end function

Con los algoritmos 21 y 20 podremos obtener el mejor DV para cada nodo del árbol de DV's. Mientras que para crear el árbol de DV's completo se diseño el algoritmo 22.

El algoritmo 22 crea el árbol de DV's de forma recursiva, una vez que este algoritmo es ejecutado crea la estructura y el nodo raíz del árbol de DV's, después se vuelve a llamar a si mismo para crear los nodos que sean necesarios y deja de llamarse a si mismo cada vez que crea un nodo hoja.

Algoritmo 22 Crea un árbol binario de DV's (etapa de entrenamiento del clasificador MCJJ2).

Entrada: Conjuntos $s1$, $s2$, parámetros del AGT-ADV p, G, pm, k y el nodo ni.
Salida: Árbol de decisión binario de DV's.

1: **procedure** CREA-MCJJ2($s1, s2, p, G, pm, k, ni$)
2: $aDV \leftarrow$ AGT-ADV(p, G, pm, 0,85, k)
3: $cero\_s1 \leftarrow \emptyset$
4: $uno\_s1 \leftarrow \emptyset$
5: **for all** $s \in s1$ **do**
6: **if** $(s \odot aDV) = 0$ **then**
7: **Inserta** s **en** $cero\_s1$
8: **else**
9: **Inserta** s **en** uno_s1
10: **end if**
11: **end for**
12: $cero\_s2 \leftarrow \emptyset$
13: $uno\_s2 \leftarrow \emptyset$
14: **for all** $s \in s2$ **do**
15: **if** $(s \odot aDV) = 0$ **then**
16: **Inserta** s **en** $cero\_s2$
17: **else**
18: **Inserta** s **en** uno_s2
19: **end if**
20: **end for**
21: **if** ¿El árbol DV ya se creo? **then**
22: Etiquetar el nodo ni con aDV.
23: **else**
24: Crear la estructura del árbol DV con aDV como nodo raíz.
25: $ni \leftarrow$ nodo raíz.
26: **end if**

▷ Continuación del CREA-MCJJ2

if $((|uno\_s1| = |s1|)\,\&\,(|uno\_s2| = |s2|))\;OR\;((|cero\_s1| = |s1|)\,\&\,(|cero\_s2| = |s2|))$ **then**

    Elimina la etiqueta actual del nodo ni.
    El nodo ni ahora es un nodo hoja y etiquetarlo con la clase 1 ó 0.
else
    if $cero\_s1 = \emptyset$ **then**
        Adiciona al nodo ni una hoja del lado izquierdo con la etiqueta clase 1.
    else if $cero\_s2 = \emptyset$ **then**
        Adiciona al nodo ni una hoja del lado izquierdo con la etiqueta clase 0.
    else
        Crea un nodo vacío del lado izquierdo, este nodo a hora es el ni.
        CREA-MCJJ2$(cero_s1, cero_s2, p, G, pm, k, ni)$
    end if
    if $uno\_s1 = \emptyset$ **then**
        Adiciona al nodo ni una hoja del lado derecho con la etiqueta clase 1.
    else if $uno\_s2 = \emptyset$ **then**
        Adiciona al nodo ni una hoja del lado derecho con la etiqueta clase 0.
    else
        Crea un nodo vacío del lado derecho, este nodo a hora es el ni.
        CREA-MCJJ2$(uno_s1, uno_s2, p, G, pm, k, ni)$
    end if
end if
end procedure

Después de la etapa de entrenamiento del clasificador MCJJ2 se tendrá un árbol de decisiones binario y para el proceso de clasificar nuevos patrones, se tiene que recorrer al árbol comenzando con el nodo raíz hasta encontrar una nodo hoja que indique a que clase corresponde dicho patrón de entrada. El algoritmo 23 sirve para recorrer el árbol hasta llegar al nodo hoja.

Algoritmo 23 Devuelve la clase a la que pertenece el patrón P_e.

Entrada: Patrón P_e y el nodo n en el que se encuentra actualmente.
Salida: La etiqueta de la clase a la que corresponde P_e.

```
1: procedure R-arbolDV(P_e, n)
2:     Bajar al siguiente nivel del árbol por la rama etiquetada con (P_e ⊙ n).
3:     Asignar lo que se encuentre en este nodo a n.
4:     if El nodo n es un nodo hoja then
5:         return n
6:     else
7:         R-arbolDV(P_e, n)
8:     end if
9: end procedure
```

4.4. Complejidad del clasificador MCJJ2

A continuación se analiza la complejidad del clasificador MCJJ2 en la etapa de entrenamiento.

En cada nodo del árbol de DV's del clasificador MCJJ2, se ejecuta el algoritmo 20 para el cuál su complejidad es (para todos los casos mejor, medio y peor):

$$O_{AGT-ADV} = \frac{nG * p * pc}{2} (n + 2 * pm * n + 2 * n + x_i + y_i + 8) \tag{4.5}$$

Donde:
x_i es la cardinalidad del subconjunto de patrones de $s1$ en el nodo i.
y_i es la cardinalidad del subconjunto de patrones de $s2$ en el nodo i.

Mejor caso. En el mejor de los casos el clasificador MCJJ2 estara compuesto de solamente un nodo del árbol de DV's, es decir que tendrá un sólo DV que dividirá la muestra de supervisión en las dos clases deseadas. Y por lo tanto la complejidad será:

$$O_{MCJJ2} = O_{AGT-ADV} \tag{4.6}$$

Caso medio. En el caso medio tenemos que se va a crear un árbol de DV's con un número de nodos hojas igual a la mitad de la muestra total de supervisión, y la complejidad estaría dada por:

$$O_{MCJJ2} = \left(\frac{x + y}{2} - 1 \right) * O_{AGT-ADV} \tag{4.7}$$

Peor caso. En el peor de los casos el clasificador MCJJ2 tendrá que crear un nodo hoja para cada patrón de la muestra de supervisión. Entonces si tenemos un total de $x + y$ patrones en la muestra de supervisión, tendríamos en total $x + y - 1$ nodos en el árbol de DV's. Por lo tanto la complejidad en el peor de los casos estaría dada por:

$$O_{MCJJ2} = (x + y - 1) * O_{AGT-ADV} \tag{4.8}$$

Por lo tanto aunque se requiere llamar al AG más veces, la complejidad sigue siendo lineal.

En la practica es muy raro que se de el peor de los casos, de hecho el clasificador MCJJ2 tiende a acercarse más al mejor de los casos mientras que el peor de los casos es casi imposible que llegue a ocurrir.

Capítulo 5

PRUEBAS Y RESULTADOS

En este capítulo se mostrará la eficiencia de los algoritmos diseñados en esta tesis y se realizarán varias simulaciones con diferentes bases de datos para mostrar el comportamiento de cada clasificador.

Todas las simulaciones que se muestran en este capitulo se realizaron con una computadora con sistema operativo UBUNTU y con un procesador Intel a 1.2 GHz con 8 núcleos. Todos los algoritmos utilizados para las simulaciones fueron programados en el lenguaje de programación *COMMON LISP*.

5.1. Bases de datos

En esta sección se mostraran las características principales de las base de datos que se utilizaron en este trabajo para hacer las simulaciones.

5.1.1. Base de datos 1. *MONK's problem*

Descripción: Este conjunto de patrones se divide en tres problemas cada uno esta conformado por una muestra de control y una de supervisión. Cada patrón tiene un dominio representado por seis atributos discretos, en total son 8 atributos (incluyendo el atributo de clase). En la tabla 5.1 se muestra el dominio de cada atributo.

Tabla 5.1: Dominio de los atributos de MONK

Atributo	**Dominio**	**Codificación**
a0 (clase)	0, 1	N/A
a1	1, 2, 3	01, 10, 11
a2	1, 2, 3	01, 10, 11
a3	1, 2	01, 10
a4	1, 2, 3	01, 10, 11
a5	1, 2, 3, 4	001, 010, 011, 100
a6	1, 2	01, 10
a7	Etiqueta única para cada patrón	N/A

En la tabla 5.2 se muestra la información acerca de los 3 problemas de MONK que de ahora en adelante los llamaremos como MONK1, MONK2 y MONK3.

Tabla 5.2: Problemas de MONK

Problema	**Característica**	**Supervisión**	**Control**
MONK1	a1 = a2 ó a5 = 1	124	432
MONK2	exactamente dos atributos son igual a 1	169	432
MONK3	se añade 5 % de ruido a cada clase	122	432

5.1.2. Base de datos 2. Dígitos

Descripción: Imágenes normalizadas en formato bitmap de los dígitos del 0 al 9 que fueron escritas a mano por un total de 43 personas, 30 contribuyeron para el conjunto de entrenamiento y los restantes 13 contribuyeron para el conjunto de pruebas. Las entradas están centradas y normalizadas todas en bitmaps de 32x32.

La distribución de las clases se muestra en la tabla 5.3.

Tabla 5.3: Números de patrones por clase.

CLASE	SUPERVISION	CONTROL	TOTAL
0	189	87	276
1	198	97	295
2	195	92	287
3	199	85	284
4	186	114	300
5	187	108	295
6	195	87	282
7	201	96	297
8	180	91	271
9	204	89	293

Cada imagen es un patrón de tamaño $n = 1024$, que se obtiene mediante la concatenación de todas las filas que conforman la imagen.
En la figura 5.1 se muestra la configuración de una imagen dada, entonces el patrón para una imagen quedaría como se muestra a continuación:

$$P = \{f_0 f_1 f_2 \dots f_r\}$$

Dónde: f_i es la fila i de una imagen (ver figura 5.1).

	i_0	i_1	i_2	$\cdots$	i_c
f_0	x_{00}	x_{01}	x_{02}	$\cdots$	x_{0c}
f_1	x_{10}	x_{11}	x_{12}	$\cdots$	x_{1c}
f_2	x_{20}	x_{21}	x_{22}	$\cdots$	x_{2c}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
f_r	x_{r0}	x_{r1}	x_{r2}	$\cdots$	x_{rc}

Figura 5.1: Imagen.

5.1.3. Base de datos 3. Splice

Descripción: *Splice junctions* son partes de una secuencia de ADN, que son removidas durante un proceso de creación de proteínas en grandes organismos. El problema planteado en este conjunto de datos es, dada una secuencia de ADN reconocer las fronteras entre *exon* (las partes de la secuencia del ADN que se conservan después del empalme) e *introns* (las partes de la secuencia del ADN que se empalman). Este problema tiene dos subproblemas: reconocer las fronteras exon/intron (llamadas sitios EI), y reconocer las fronteras intron/exon (llamadas sitios IE).
En total hay 3190 patrones que se dividen en tres clases diferentes:

1. 767 patrones en la clase *EI*.
2. 769 patrones en la clase *IE*.
3. 1655 patrones en la clase *N* (Ninguna).

Cada patrón consta de 62 atributos como se muestra en la tabla 5.4.

Tabla 5.4: Dominio de los atributos de Splice

Atributo	**Dominio**
a0 (clase)	EI, IE, N
a1	Etiqueta única para cada patrón
a2-a62	A, G, T, C, D, N, S, R

5.1.4. Base de datos 4. Skin

Descripción: Este conjunto de datos esta formado por muestras al azar de tonalidades de la piel de la cara en formato RGB; estas muestras se seleccionaron de diferentes grupos de edades, grupos de razas (blanco, negro y asiático). En total hay 245057 patrones y se dividen en dos clases diferentes:

- ⋆ 50859 patrones de la clase que son tonalidades de piel de rostros.
- ⋆ 194198 patrones de la clase que no son tonalidades de piel de rostros.

Cada patrón tiene cuatro atributo; los primeros tres son valores enteros que corresponden a los valores B, G y R respectivamente; y el cuarto atributo es la etiqueta de la clase a la que pertenece.

La binarización se hace convirtiendo cada valor de cada atributo en su valor binario correspondiente.

5.2. Simulaciones con el sistema DS-DV

En esta sección se mostrarán las simulaciones realizadas con el clasificador DS-DV.

5.2.1. Escenario de simulación 1

A continuación se muestran los resultados que se obtuvieron con el sistema AGT-DSDV (algoritmo 19) del clasificador DS-DV, sobre las bases de datos descritas en la sección 5.1 y utilizando las cinco funciones de evaluación descritas en la sección 4.1.

Para observar el comportamiento de cada función de evaluación primero se probaron las bases de datos 1, 2 y 3 con las cinco funciones de evaluación.

Los parámetros del sistema AGT-DSDV se muestran en la tabla 5.5. Estos valores se mantendrán constantes y sólo se modificara la función de evaluación con que se evaluó cada individuo en el sistema. Todos los valores de estos parámetros se obtuvieron mediante varias simulaciones y pruebas previas del sistema AGT-DSDV, con el objetivo de obtener valores que permitieron un mejor rendimiento.

Tabla 5.5: Valores para los parámetros del AGT-DSDV.

PARÁMETRO	**VALORES**		
	Dígitos	**MONK**	**Splice**
Tamaño de población	100	100	100
Número de generaciones	100	100	100
Probabilidad de mutación	0.3	0.3	0.3
Probabilidad de mutación del VP	0.4	0.4	0.4
Parámetro de mutación	35	1	5
Probabilidad de cruzamiento	0.85	0.85	0.85
k	5	5	5
Intervalo de tamaño inicial del VP	$2-8$	$2-4$	$2-10$

Para la base de datos Dígitos se utilizaron seis dígitos diferentes para hacer tres simulaciones: una, con los dígitos 0 y 7, otra, con los dígitos 3 y 4 y una más con los dígitos 5 y 6; en las tablas 5.6, 5.7 y 5.8 se muestran los resultados de las simulaciones.

Las tablas 5.9, 5.10, 5.11 y 5.12 muestran los resultados para las base de datos de MONK1, MONK2, MONK3 y Splice respectivamente.

Todos los resultados obtenidos de las tablas 5.6, 5.7, 5.8, 5.9, 5.10, 5.11 y 5.12 se resumen en el gráfico de la figura 5.2, este gráfico muestra el promedio de cada función de evaluación obtenido en todas las pruebas anteriores. Como se puede ver la mejor función de evaluación fue la FE4, seguida de FE2.

Tabla 5.6: Resultados del AGT-DSDV con 5 funciones de evaluación, para los dígitos 0 y 7.

FUNCIÓN	**EVALUACIÓN**	**Tamaño del VP** (m)	**Precisión (%)**	
			Supervisión	**Control**
FE1	98.20	13	98.20	94.53
FE2	92.07	19	95.89	96.72
FE3	96.80	7	96.66	93.98
FE4	97.69	19	97.69	96.72
FE5	120.67	5	97.43	94.53

Tabla 5.7: Resultados del AGT-DSDV con 5 funciones de evaluación, para los dígitos 3 y 4.

FUNCIÓN	**EVALUACIÓN**	**Tamaño del VP** (m)	**Precisión (%)**	
			Supervisión	**Control**
FE1	72.93	23	73.50	70.85
FE2	61.55	28	78.44	80.90
FE3	111.62	19	82.33	83.41
FE4	86.75	25	86.75	83.91
FE5	77.50	5	64.67	64.82

Tabla 5.8: Resultados del AGT-DSDV con 5 funciones de evaluación, para los dígitos 5 y 6.

FUNCIÓN	**EVALUACIÓN**	**Tamaño del VP** (m)	**Precisión (%)**	
			Supervisión	**Control**
FE1	96.07	24	96.073	91.79
FE2	92.36	21	96.07	97.94
FE3	85.620	20	85.60	90.25
FE4	95.287	18	95.28	96.41
FE5	75.16	2	83.76	80.0

Tabla 5.9: Resultados del AGT-DSDV con 5 funciones de evaluación, para MONK1.

FUNCIÓN	EVALUACIÓN	Tamaño del VP (m)	Precisión (%)	
			Supervisión	Control
FE1	58.69	5	76.61	69.44
FE2	43.95	5	82.25	72.22
FE3	79.62	5	76.61	69.44
FE4	65.01	4	82.25	72.22
FE5	97.17	4	82.25	72.22

Tabla 5.10: Resultados del AGT-DSDV con 5 funciones de evaluación, para MONK2.

FUNCIÓN	EVALUACIÓN	Tamaño del VP (m)	Precisión (%)	
			Supervisión	Control
FE1	62.72	6	62.72	57.87
FE2	31.51	6	64.49	57.87
FE3	62.71	6	47.33	46.75
FE4	49.76	2	81.65	82.87
FE5	75.22	2	64.49	57.87

Tabla 5.11: Resultados del AGT-DSDV con 5 funciones de evaluación, para MONK3.

FUNCIÓN	EVALUACIÓN	Tamaño del VP (m)	Precisión (%)	
			Supervisión	Control
FE1	69.17	6	62.72	57.87
FE2	31.51	6	64.49	57.87
FE3	83.71	6	77.86	80.55
FE4	49.76	2	81.65	82.87
FE5	80.38	2	77.86	80.55

Tabla 5.12: Resultados del AGT-DSDV con 5 funciones de evaluación para Splice.

FUNCIÓN	EVALUACIÓN	Tamaño del VP (m)	Precisión (%)	
			Supervisión	Control
FE1	62.89	19	63.10	63.41
FE2	47.79	22	72.45	67.93
FE3	66.49	19	66.44	64.57
FE4	61.35	16	62.18	62.66
FE5	75.05	2	82.88	81.50

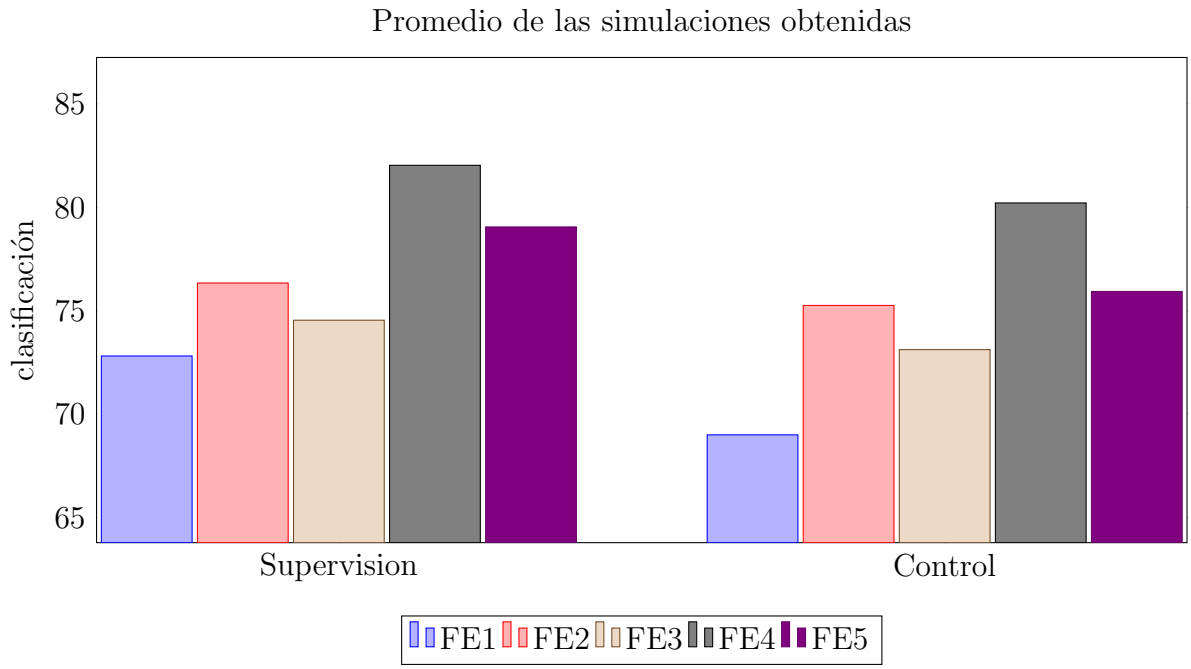


Figura 5.2: Promedio de las cinco funciones de evaluación

Ahora veamos el comportamiento del AGT-DSDV, en las gráficas de las figuras 5.3 y 5.4 podemos observar el comportamiento de cada función conforme avanzan las generaciones del AG.
Como podemos ver en las cuatro gráficas el promedio de las evaluaciones de la población del AGT-DSDV tiende a mejorar conforme pasan las generaciones lo cual es el comportamiento deseado.

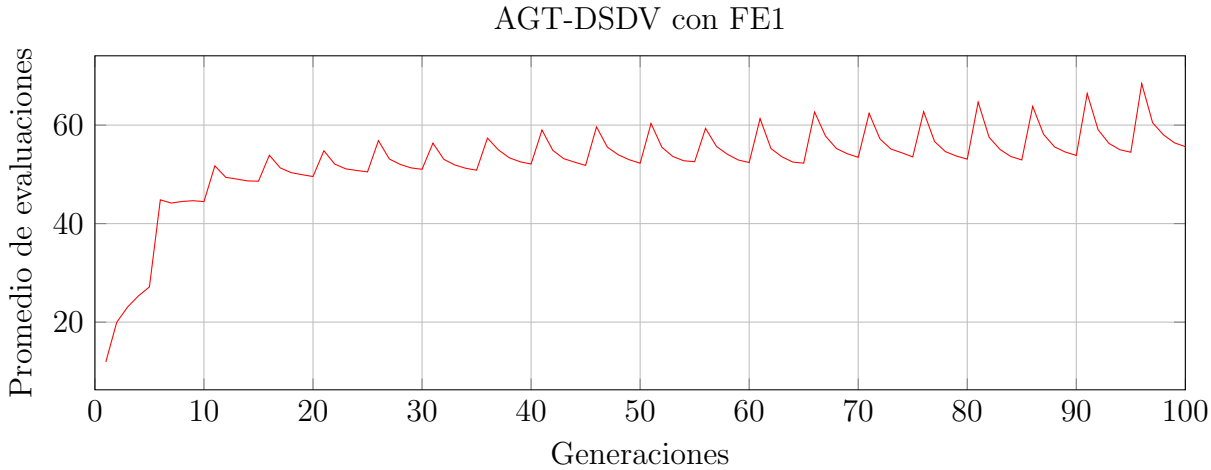


Figura 5.3: Gráfica del comportamiento del AGT-DSDV con FE1, simulación de los dígitos 0 y 7.

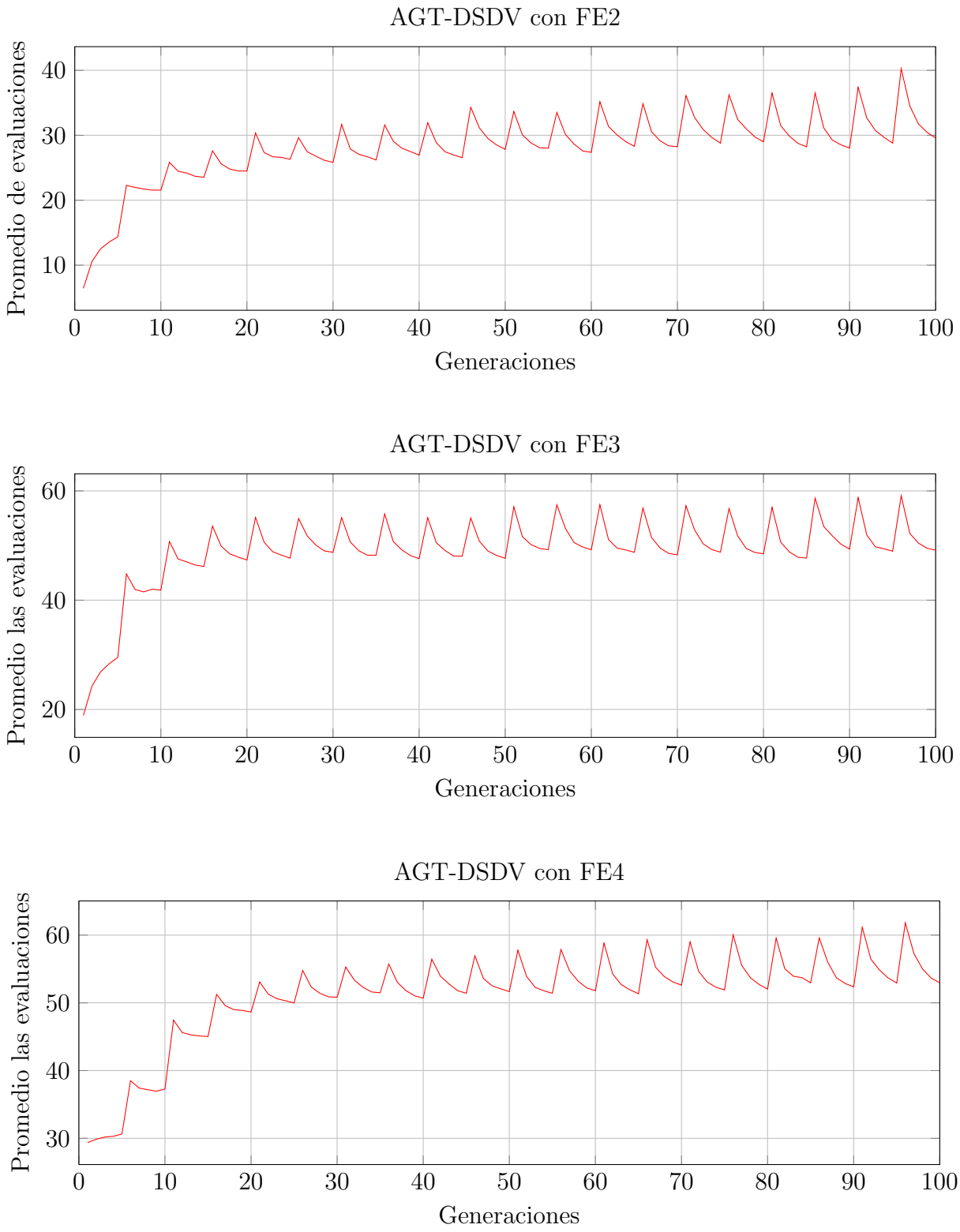


Figura 5.4: Gráficas del comportamiento del AGT-DSDV con FE2, FE3 y FE4, de las simulaciones de los dígitos 0 y 7.

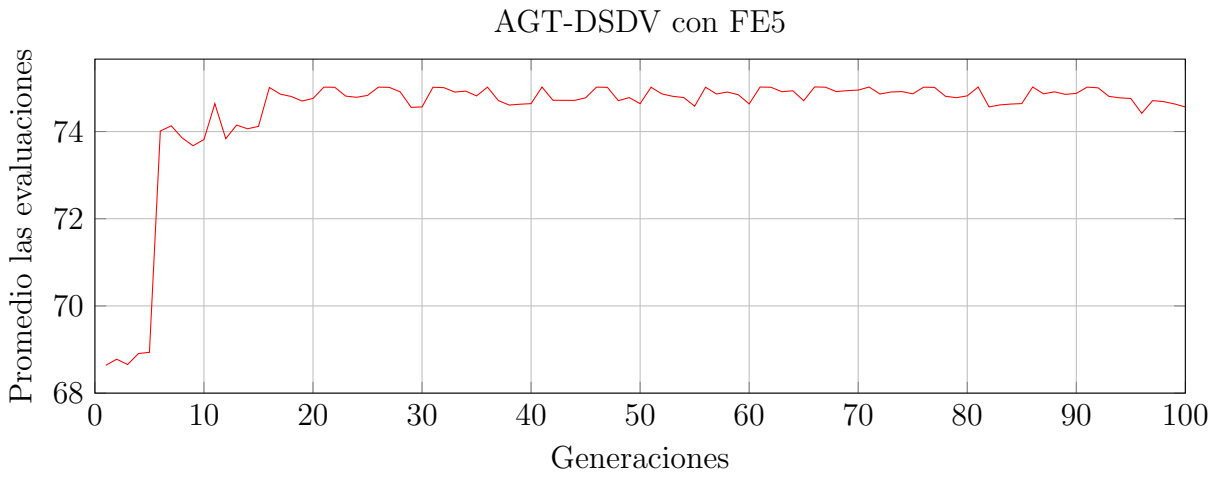


Figura 5.5: Gráficas del comportamiento del AGT-DSDV con FE5, de la simulación con Splice.

En la tabla 5.13 se muestran los mejores resultados obtenidos del clasificador DS-DV, para las base de datos Dígitos, Splice y skin. Para estas simulaciones se aumentaron los valores para la población y el número de generaciones, en cuanto a la función de evaluación utilizada, se selecciono la función con la que mejor se haya obtenido resultados así también se utilizo la combinación de tres funciones diferentes para que la población fuera más variada.

Tabla 5.13: Resultados obtenidos con **FE5**.

BASE DE DATOS	**EVALUACIÓN**	**TIEMPO (segundos)**	**Tamaño VP (m)**	**Precisión (%)**	
				Supervisión	**Control**
Dígitos 0 y 7	97.33	708	14	98.46	98.36
Dígitos 3 y 4	90.89	663	9	91.68	93.96
Dígitos 5 y 6	107.04	1059	8	97.64	98.46
Splice	68.06	1889	15	82.9	81.59
Skin	87.63	4109	9	91.51	87.91

El clasificador DS-DV para la base de datos de Splice, con el que se obtuvieron los resultados de la table 5.13 es:

DS = (1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 1
1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0
1 1 0 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1 1 1 1
1 0 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 1 1 0 1
1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 1 1)
VP = (34 33 18 6 13 10 4 19 7 9 6 20 4 28 26 3)
DV = (0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0)

El clasificador DS-DV para la base de datos de Skin, con el que se obtuvieron los resultados de la table 5.13 es:

DS = (1 0 0 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 1)
VP = (2 3 3 2 5 2 2 2 3)
DV = (0 0 0 1 0 1 1 0 0)

5.2.2. Escenario de simulación 2

En este escenario de simulación se probará el comportamiento del clasificador DS-DV con varios tamaños de la muestra de supervisión. Estas simulaciones se realizarán con la función de evaluación FE5 y con las bases de datos 3 y 4.
Se realizaron en total seis simulaciones con el 20 %, 40 % y 60 % del total de patrones para la muestra de supervisión, en todos los casos los patrones de la muestra de supervisión, seleccionaron de manera pseudo-aleatoria.

Tabla 5.14: Resultados del AGT-DSDV con varios tamaños de muestra de supervisión.

BASE DE DATOS	**MUESTRA DE SUPERVISIÓN**	**Tiempo (segundos)**	**Precisión (%)**	
			Supervisión	**Control**
Splice	20 %	1016	80.594	82.359
	40 %	2543	81.739	82.183
	60 %	5074	81.338	83.007
Skin	20 %	2368	81.67	81.41
	40 %	4199	63.15	62.81
	60 %	4945	82.89	82.95

A continuación se muestra el clasificador DS-DV que se obtiene del conjunto de patrones Splice, con el cual se obtuvo la mejor precisión que fue con la muestra de supervisión al 60 %.

DS = (0 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1
0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0
1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 1 1 1 0 1 1
1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 0 1 0 1 0 1 1 0 1
1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 0 0 0)
VP = (4 21 16 11 9 7 18 6 5 5 14 16 4 2 16 15 13 13 5 8 19 13)
DV = (0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0)

A continuación se muestra el clasificador DS-DV que se obtiene del conjunto de patrones Skin, con el cual se obtuvo la mejor precisión que fue con la muestra de supervisión al 20 %.

DS = (1 0 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 1 1 0)
VP = (2 12 5 3 2)
DV = (1 0 1 0 0)

5.2.3. Escenario de simulación 3

En este escenario de simulación se prueba el clasificador DS-DV, con una muestra de supervisión conformada por los patrones más representativos, menos representativos y una muestra aleatoria, los resultados de estas simulaciones se muestran en la tabla 5.15.

Los resultados de la tabla 5.18 se realizaron con la base de datos Dígitos y se eligieron los números 1 y 5. La muestra de supervisión contiene 50 patrones y la muestra de control contiene 245 patrones.

Para obtener los patrones más representativos y menos representativos se uso la distancia de *hamming* (definición 2.3.1).

Tabla 5.15: Simulaciones del clasificador DS-DV con diferentes muestras de supervisión

Muestra de supervisión	**Supervisión**				**Control**			
	Precisión (%)		**Recall (%)**		**Precisión (%)**		**Recall (%)**	
	C-0	**C-1**	**C-0**	**C-1**	**C-0**	**C-1**	**C-0**	**C-1**
Más representativos	100	100	100	100	82.857	90.204	89.427	84.030
Menos representativos	58	88	82.857	67.692	72.653	61.224	65.201	69.124
Aleatorio	94	86	87.037	93.478	80	67.346	71.014	77.102

5.3. Simulaciones con el sistema MCJJ2

En esta sección se mostrarán las simulaciones realizadas con el clasificador MCJJ2 que se diseño en esta tesis.

5.3.1. Escenario de simulación 1

En este primer escenario de simulación se prueba el clasificador MCJJ2 con las bases de datos 1, 2, 3 y 4, como se muestra en la tabla 5.16. Para las bases de datos 1 y 2 se toma la muestra de supervisión y la de control conforme se describe en la sección 5.1. Mientras que para Splice se tomo el 35 % del total de patrones para la muestra de supervisión y el restante para la muestra de control. Y para Skin se tomo el 30 % del total de patrones para la muestra de supervisión.

Tabla 5.16: Resultados obtenidos con el algoritmo 22.

BASE DE DATOS	**PARÁMETROS AGT-ADV población, generaciones, pm, k**	**TIEMPO (segundos)**	**Precisión (%)**	
			Supervisión	**Control**
MONK1	100, 80, 0.35, 5	5.69	100	100
MONK2	80, 100, 0.34, 5	6.59	100	94.07
MONK3	80, 100, 0.34, 5	6.10	100	93.26
Dígitos 0 y 7	200, 200, 0.34, 8	2593	100	100
Dígitos 3 y 4	200, 200, 0.34, 8	2400	100	100
Dígitos 5 y 6	200, 200, 0.34, 8	2639	100	100
Splice	120, 150, 0.35, 6	1198	100	92.69
Skin	120, 140, 0.3, 5	9389	99.98	99.61

En la figura 5.6 se muestra el clasificador MCJJ2 correspondiente al conjunto de patrones MONK1.

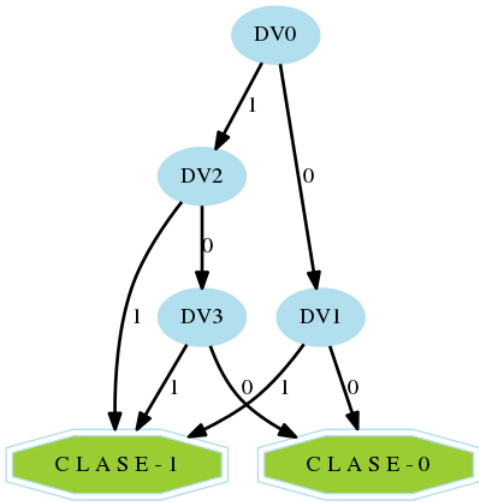


Figura 5.6: Clasificador MCJJ2 para los patrones de MONK1.

Donde: **DV0** = (1 0 1 0 1 1 0 0 0 0 0 0 0), **DV1** = (0 0 0 0 0 0 0 0 1 1 0 1 1), **DV2** = (1 1 1 1 1 1 0 0 0 0 0 0 0) y **DV3** = (0 0 0 0 0 0 0 0 1 1 0 1 1).

En la figura 5.7 se muestra el clasificador MCJJ2 correspondiente al conjunto de patrones Dígitos 0 y 7.

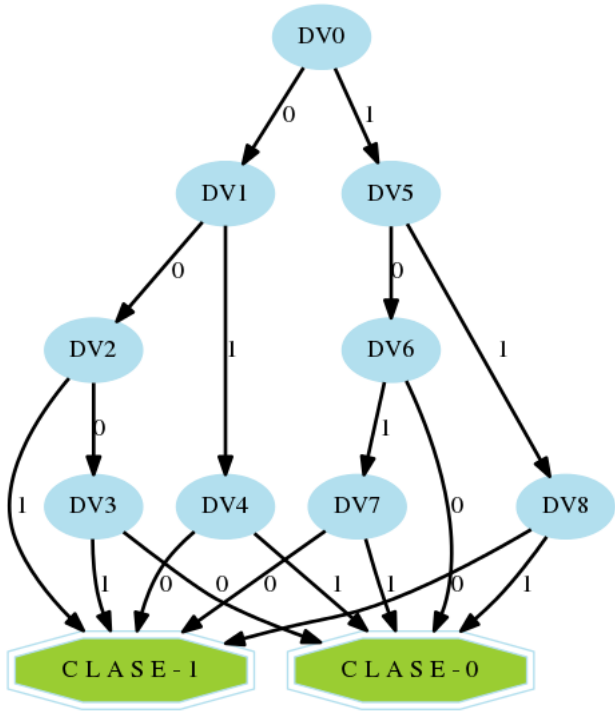


Figura 5.7: Clasificador MCJJ2 para el conjunto de patrones Dígitos 0 y 7.

Donde: **DV0** = (1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0
1 0 1 1 0 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 1 0 1 1
1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1
1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1
0 1 1 0 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 0
1 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1
1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 1
0 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1
1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1
0 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0
1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1
1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0
1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0
1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0
1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 1
1 1 1 0 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1
0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0

1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 0
1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 0 1 1
1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0
1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1
0 1 1 1 0 0),

DV1 = (0 0 0

0 0 0

0 0 0

0 0 0 0 0 1 1 1

5.3.2. Escenario de simulación 2

En este escenario de simulación se prueba el comportamiento del sistema MCJJ2 con varios tamaños de la muestra de supervisión. Estas simulaciones se realizarán con las bases de datos 3 y 4. Se realizaron en total seis simulaciones con el 20 %, 40 % y 60 % del total de patrones para la muestra de supervisión y el resto de patrones para la muestra de control, en todos los casos los patrones de la muestra de supervisión se seleccionaron de manera pseudo-aleatoria. En la tabla 5.17 se muestran los resultados obtenidos de estas simulaciones.

Tabla 5.17: Resultados del MCJJ2 con varios tamaños de muestra de supervisión.

BASE DE DATOS	**MUESTRA DE SUPERVISIÓN**	**Tiempo (segundos)**	**Precisión (%)**	
			Supervisión	**Control**
Splice	20 %	292	100	87.45
	40 %	561	100	91.01
	60 %	627	100	99.34
Skin	20 %	1863	100	99.50
	40 %	1360	100	99.80
	60 %	1952	100	99.74

En la figura 5.8 se muestra el clasificador MCJJ2 correspondiente al conjunto de patrones Splice que se obtuvo de la simulación con el 60 % de la muestra de supervisión.

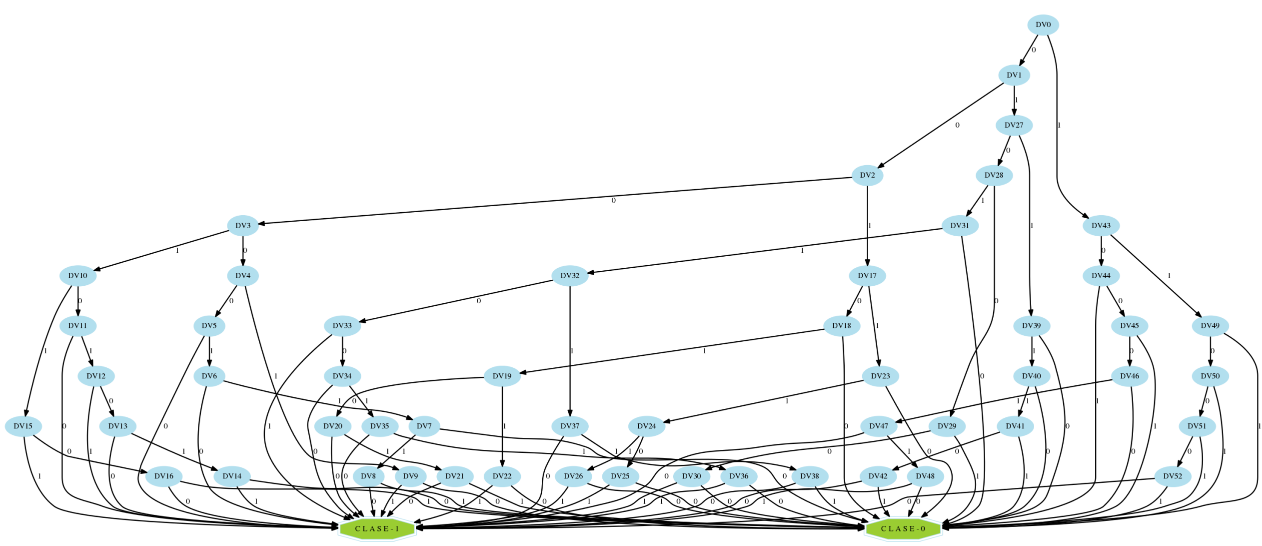


Figura 5.8: Clasificador MCJJ2 correspondiente a Splice.

Donde: **DV0** = (0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0),

DV1 = (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0),
DV2 = (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0),
DV3 = (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0),
DV4 = (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0),

⋮

DV51 = (1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1
1 0 1 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0
1 0 1 1 0 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1
0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0
1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1),
DV52 = (0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0
1 1 0 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 1
0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1
0 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1
1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1)

5.3.3. Escenario de simulación 3

En este escenario de simulación se prueba el clasificador MCJJ2, con una muestra de supervisión conformada por los patrones más representativos, menos representativos y una muestra aleatoria, los resultados de estas simulaciones se muestran en la tabla 5.18.

Los resultados de la tabla 5.18 se realizaron con la base de datos Dígitos y se eligieron los números 1 y 5. La muestra de supervisión contiene 50 patrones y la muestra de control contiene 245 patrones.

Para obtener los patrones más representativos y menos representativos se uso la distancia de *hamming* (definición 2.3.1).

Tabla 5.18: Simulaciones del clasificador MCJJ2 con diferentes muestras de supervisión

Muestra de supervisión	**Supervisión**				**Control**			
	Precisión (%)		**Recall (%)**		**Precisión (%)**		**Recall (%)**	
	C-0	**C-1**	**C-0**	**C-1**	**C-0**	**C-1**	**C-0**	**C-1**
Más representativos	100	100	100	100	88.571	90.612	90.416	88.8
Menos representativos	100	100	100	100	86.938	55.102	65.944	80.838
Aleatorio	100	100	100	100	77.959	82.857	81.974	78.988

5.4. Comparación de los clasificadores DS-DV y MCJJ2 contra otros clasificadores

En esta sección se comparan los DS-DV y MCJJ2 contra otros clasificadores, las comparaciones se realizaron con la base de datos MONK y Skin. En la tabla 5.19 se muestran los resultados de otros clasificadores sobre la base de datos MONK [17].

Tabla 5.19: Resultados de diferentes clasificadores probados con MONK

Clasificador	**Precisión (%)**		
	MONK1	**MONK2**	**MONK3**
MCJJ2	100	96.29	90.74
DS-DV	72.22	82.87	82.87
mFOIL	100	69.2	100
ID3	98.6	67.9	94.4
ID5R	79.7	69.2	95.2
AQ17-DCI	100	100	94.2
PRISM	86.3	72.7	90.3
Backpropagation	100	100	93.1
ECOBWEB lead prediction	71.8	67.4	68.2
Cascade Correlation	100	100	97.2

En la tabla 5.20 se muestra la comparación realizada con un clasificador *Fuzzy decision tree* de propósito específico [18].

Tabla 5.20: Resultados de diferentes clasificadores probados con Skin

Clasificador	**Precisión (%)**	
	Clase Skin	**Clase Nonskin**
MCJJ2	99.28	99.63
DS-DV	71.34	89.72
Fuzzy decision tree	98.09	92.51

Capítulo 6

CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

Se diseñaron e implementaron dos clasificadores de patrones basados en los conceptos de autómatas celulares de multiples sumideros (ACMS) y algoritmo genético más la característica de elitismo (AGT), ya que el ACMS posee la propiedad de que los patrones que pertenecen a un mismo sumidero tengan una distancia *hamming* menor que cualquier otro AC lineal, y el algoritmo genético AGT mostró los mejores resultados sobre el AGS por su característica de elitismo. El clasificador DS-DV en la etapa de entrenamiento utiliza el sistema AGT-DSDV para crear una población de DS's y DV's (los cuales representan dos ACMS) que puedan realizar la clasificación, al final se selecciona al mejor individuo de esta población. Mientras que el clasificador MCJJ2 crea un DV para cada nodo del árbol binario con el sistema AGT-ADV. Aun queda por probar más variantes del AG, ya que no se garantiza que el AGT sea la mejor opción de todas.

Se diseñó el algoritmo genético AGT-DSDV para la etapa de entrenamiento del clasificador de dos etapas DS-DV.

Se crearon los algoritmos necesarios para el AGT del clasificador DS-DV y se diseñó un nuevo clasificador llamado MCJJ2.

Se diseñó el algoritmo genético AGT-ADV que obtiene un vector de dependencia para cada nodo del árbol de decisiones del clasificador MCJJ2.

Se diseñó un algoritmo CREA-MCJJ2 para la creación de un árbol de decisiones basado en vectores de dependencia.

6.2. Logros alcanzados y limitaciones

En este documento se han presentado los algoritmos diseñados, implementados y simulados, los cuales están basados en los conceptos de AC, AG y árboles de decisión, también se ha presentado una metodología que nos permite resolver el problema planteado en esta tesis.

Los clasificadores MCJJ2 y DS-DV, tienen varias características: el trabajar con operaciones AND y XOR lo que hace eficiente el computo del proceso. La muestra de supervisión sólo requiere el conocimiento del dominio de cada atributo. Ambos clasificadores están programados en paralelo, lo que reduce el tiempo de ejecución y además se pueden procesar cientos de miles de patrones. Con respecto al número de patrones en la muestra de supervisión, durante las simulaciones realizadas en esta tesis se pudo observar que basta con tener un 20 % del total de patrones para la muestra de supervisión.

El clasificador DS-DV tiene una alta precisión, durante las simulaciones realizadas se observa que el porcentaje de precisión que es alcanzado en la muestra de supervisión se ve reflejado de igual manera en la muestra de control, esto es, si en la muestra de supervisión se obtiene el 90 % de precisión, en la muestra de control se espera que la precisión sea de alrededor de un 90 %. Pero es difícil llegar a una precisión del 100 % en la muestra de supervisión ya que el espacio de búsqueda es muy grande.

El clasificador MCJJ2 en la parte de entrenamiento alcanza un 100 % de precisión y lo hace más rápido que el clasificador DS-DV.
El clasificador MCJJ2 puede resolver la tarea de clasificar patrones con una alta precisión y depende muy poco del número de patrones de la muestra de supervisión, en las pruebas realizadas se muestra que con un 20 % del total de patrones para la muestra de supervisión, es suficiente para el entrenamiento del clasificador y lograr obtener una precisión muy buena de más del 90 %.
El clasificador MCJJ2 tiene una complejidad lineal, lo que lo hace excelente para clasificar grandes cantidades de patrones.

Ambos clasificadores tanto el DS-DV como el MCJJ2 son clasificadores binarios, es decir que sólo clasifican un conjunto de patrones en dos clases diferentes. En caso de querer clasificar más de dos clases habría que crear varios clasificadores para ir dividiendo los patrones hasta que cada patrón pertenezca a su propia clase.

Con respecto a los experimentos realizados, estos muestran que el clasificador diseñado resuelve el problema propuesto con una precisión mayor al 90 %, alcanzando el 100 % en algunos casos.

Finalmente, los dos clasificadores diseñados en esta tesis fueron comparados con los clasificadores mostrados en las tablas 5.19 y 5.20. Como se observa en estas tablas el clasificador MCJJ2 tiene una mejor precisión que los clasificadores *ID3, ID5R PRISM, ECOBWEB lead prediction* y *Fuzzy decisión tree* que es un clasificador de propósito especifico, mientras que el clasificador DS-DV muestra una mejor precisión que el algoritmo *ECOBWEB lead prediction*.

6.3. Aportaciones

Dentro de las contribuciones que se consiguieron están:

- El diseño de los clasificadores MCJJ2 y DS-DV están basados en AG y AC, y el cómputo pesado (la evaluación de un individuo) del proceso de ambos clasificadores es ejecutado en paralelo.
- Caracterización de los parámetros de los AG tanto del clasificador DS-DV como del MCJJ2.
- Se diseñaron funciones de evaluación eficientes para el AGT-DSDV.
- Se diseño una metodología que nos permite, en función del conjunto de patrones de la muestra de supervisión, determinar la función de evaluación que debemos de utilizar.
- Se diseño una función que implementa una intersección dado que la función que tiene commond lisp no es eficiente para nuestro caso.
- Los clasificadores MCJJ2 y DS-DV fueron desarrollados en *COMMOND LISP* que es un compilador moderno, multiparadigma y de alto rendimiento, el cual nos permite ejecutar los algoritmos de los clasificadores MCJJ2 y DS-DV con una alta eficiencia y rapidez.

6.4. Trabajo futuro

De este trabajo existen diferentes opciones que permitirían reforzarlo o mejorar alguna de sus cualidades, por ejemplo:

- Diseñar algoritmos para generalizar el clasificador MCJJ2, para que pueda clasificar patrones en más de dos clases.
- Explorar otras formas de crear el árbol de decisión, con el objetivo de incrementar la precisión.
- Paralelizar todas las partes del sistema MCJJ2 que son paralelizables, por ejemplo el cruzamiento o la creación de nodos.
- Experimentar con otras variantes de AG para reducir el tiempo de búsqueda del DV para cada nodo.

- ⋆ Experimentar con otras variantes de AG para reducir el tiempo de búsqueda del sistema DS-DV.
- ⋆ Experimentar con otras formas de cruzar dos individuos.
- ⋆ Experimentar con otra forma de selección de dos individuos.
- ⋆ Experimentar con nuevas funciones de evaluación para el AG del clasificador MCJJ2.
- ⋆ Experimentar un clasificador híbrido que combine el clasificador DS-DV con el clasificador MCJJ2.
- ⋆ El sistema MCJJ2 contiene funciones las cuales pueden ser mejoradas para permitir una mayor robustez del mismo, por ejemplo hacerlo más eficiente computacionalmente.

Bibliografía

[1] A. Ilachinski "Cellular Automata A Discrete Universe", World Scientific Publishing. 2001.

[2] P. Pal Chaudhuri, D. R. Chowdhury, S. Nandi, S. Chattopadhyay "Additive Cellular Automata Theory and Applications",Volumen 1 , IEEE Computer Society. 1997

[3] Á. Kuri Morales, J. Galaviz Casas "Algoritmos genéticos", Fondo de cultura económica. 2002.

[4] Z. Michalewicz "Genetic Algorithms + Data Structures = Evolution Programs", Springer. 1999.

[5] S. Mitra, T. Acharya "Data Mining Multimedia, Soft Computing, and Bioinformatics", Wiley Interscience. 2003.

[6] P. Maji, Ch. Shaw, "Theory and Application of Cellular Automata For Pattern Classification", Fundamenta Informaticae, 58, 2003, 321–354.

[7] N. Ganguly, P. Maji, S. Dhar, B. K. Sikdar, y P. Pal Chaudhuri, "Evolving Cellular Automata as Pattern Classifier", 5th Internatinal Conference on Cellular Automata for Research and Industry , ACRI 2002 Geneva, Switzerland, 2002.

[8] R. Ranjan Maiti, A. Mallya, A. Mukherjee, N. Ganguly, "Understanding the correlation of the properties of human movement patterns", Advances in Complex Systems, 17(6), 2014.

[9] N. Ganguly, B. K. Sikdar, P. Pal Chaudhuri, "Exploring Cycle Structures of Additive Cellular Automata", Fundamenta Informaticae, 87(2), 2008, 137-154.

[10] O. Babaoglu, G. Canright Telenor R and D, A. Deutsch *et. al*, "ACM Transactions on Autonomous and Adaptive Systems (TAAS)", 1(1), 2006, 26-66.

[11] S. Chattopadhyay, S. Adhikari, S. Sengupta, M. Pal, "Highly Regular, Modular, and Cascadable Design of Cellular Automata-Based Pattern Classifier", IEEE Transaction on VLSI Systems, 8(6), December 2000, 724–735.

[12] K. Paul, A. Roy, P.K. Nandi, B.N. Roy, M. Deb Purkayastha, S. Chattopadhyay, P. Pal Chaudhuri, "Theory and application of multiple attractor cellular automata for fault diagnosis", Test Symposium, 1998. ATS '98. Proceedings. Seventh Asian , 2(4), 1998, 388-392.

[13] B.K. Sikdar, N. Ganguly, A. Karmakar, S.S. Chowdhury, P. Pal Chaudhuri, "Multiple attractor cellular automata for hierarchical diagnosis of VLSI circuits", Test Symposium, 2001. Proceedings. 10th Asian, 2001, 385-390.

[14] B.K. Sikdar, N. Ganguly, P. Majumder, P. Pal Chauhuri, "Design of multiple attractor GF(2p) cellular automata for diagnosis of VLSI circuits", VLSI Design, 2001. Fourteenth International Conference on, 2001, 454-459.

[15] F.V. Jensen, "An Introduction to Bayesian Networks", UCL Press, London (1996)

[16] T. Fawcett, "Data mining with cellular automata", ACM SIGKDD Explorations Newsletter, 10(1), 2008, 32-39.

[17] S.B. Thrun, J. Bala, E. Bloedom *et. al*, "The MONK's Problems - A Performance Comparasion of Different Learning Algorithms", Computer Science, 1991.

[18] R.B. Bhatt, G. Sharma, A. Dhall, S. Chaudhury, "Efficient Skin Region Segmentation Using Low Complexity Fuzzy Decision Tree Model", India Conference (INDICON), 2009 Annual IEEE, 2009, 1,4, 18-20.

[19] Niloy Ganguly, "Cellular Automata Evolution: Theory and Applications in Pattern Recognition and Classification". Phd, Department of Computer Science and Technology Bengal Engineering College, 2003.

[20] I. Csiszar, "Maxent, mathematics, and information theory. In: Hanson, K.M., Silver, R.N. (eds.) Maximum Entropy and Bayesian Methods",CKluwer, Norwell, MA (1996), 35-50.

[21] K.S. Fu, "Syntactic Pattern Recognition and Applications"Prentice-Hall, Englewood Cliffs. 1982.

[22] L.I. Perlovsky, "Conundrum of combinatorial complexity", IEEE Trans. Pattern Anal. Mach. Intell. 20, 1998, 666–670.

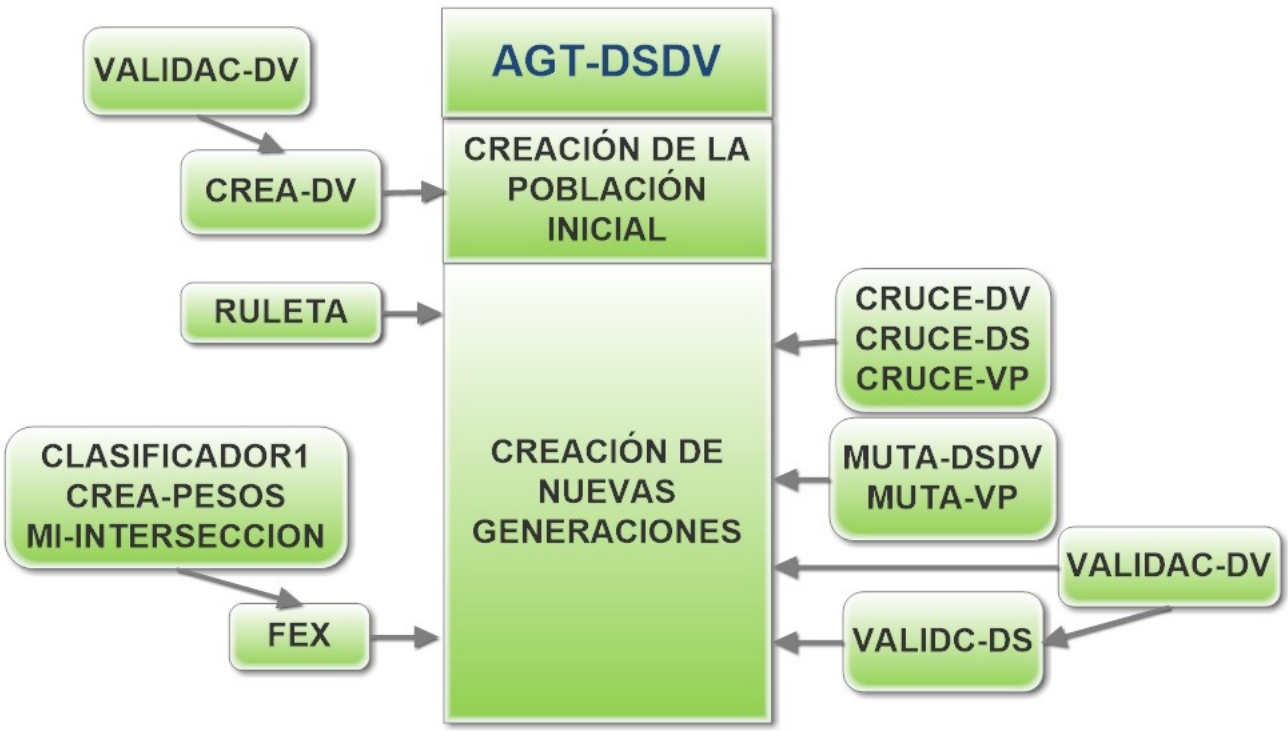
[23] J. Shawe-Taylor, T. De Bie, N. Cristianini, "Data mining, data fusion and information management", IEE Proceedings - Intelligent Transport System, 153(3), 2006, 221-229.

[24] UC Irvine Machine Learning Repository. https://archive.ics.uci.edu/ml/index.html. Consultada: diciembre 2014.

Apéndice A

Diagramas

A.1. Diagrama del sistema AGT-DSDV



A.2. Diagrama del sistema CREA-MCJJ2