La Programación Evolutiva en la Identificación de Sistemas Dinámicos a Eventos Discretos

Jose Aguilar, *Member IEEE*

Resumen— **El presente trabajo evalúa el uso de la programación evolutiva en problemas de automatización industrial. Es bien sabido, que técnicas de la computación evolutiva como los algoritmos genéticos han proporcionado resultados satisfactorios al enfrentar problemas de control automático, en especial de tipo continuo, mientras que el estudio de dinámicas discretas ha estado un poco relegado. En este trabajo se implementa una técnica de identificación basada en la programación evolutiva para Sistemas Dinámicos a Eventos Discretos (SDED), usando autómatas de estados finitos (Máquinas de Mealy). Así, se aborda la identificación de sistemas, pero cuya dinámica esta regida por eventos discretos. Además, se construyo un sistema computacional basado en la programación evolutiva para diseñar autómatas de estados finitos.**

Palabras clave— **Programación evolutiva, identificación de sistemas, sistemas dinámicos a eventos discretos.**

I. Introducción

Los sistemas diseñados por el hombre presentan dinámicas altamente no lineales, que en muchos casos no pueden ser modelados por dinámicas continuas, por lo cual se recurre a los **sistemas dinámicos a eventos discretos** (SDED). Dichos sistemas son manejados a través de transiciones. Es conveniente acotar que cuando se modelan dinámicas continuas estas están apegadas a leyes físicas. En el caso de las dinámicas a eventos discretos los modelos se formulan de acuerdo a la experiencia del "ingeniero", y frecuentemente es necesario tomar en cuenta algunas consideraciones y/o restricciones para obtener un modelo a eventos basado en algún punto de vista.

Un **sistema a eventos discretos** (SED) describe el comportamiento mediante la enumeración (u ocurrencia) de eventos desde un estado inicial, o mediante la historia de los cambios de estados ocurridos a partir de un estado de arranque. Existen varias maneras de estudiar el comportamiento de los SDED. Nosotros particularmente, nos remitiremos a trabajar con **máquinas secuenciales**.

Por otro lado, la identificación de sistemas trata de aproximar el comportamiento entrada-salida de una planta desconocida usando un modelo apropiado. En caso de dinámicas continuas el comportamiento de una planta desconocida puede ser modelado como una función no lineal de la señal de entrada, la señal de entrada previa y la salida previa de la planta. La finalidad es seleccionar un modelo que pueda reproducir exactamente el comportamiento de la planta.

Algunas técnicas inteligentes como las **redes neuronales**, la **lógica difusa** y la **programación evolutiva** han sido utilizadas para crear modelos de plantas continuas, y así resolver una gran cantidad de problemas de identificación. Pero hasta el momento las técnicas inteligentes aplicadas al problema de identificación de dinámicas discretas son muy limitadas, se podría decir que hasta escasas, por lo cual el propósito de este trabajo es introducir una técnica de la Computación Evolutiva para ser aplicada al estudio de sistemas cuyas dinámicas se rigen por eventos discretos (SDED).

Así, el propósito de este trabajo es estudiar la identificación de sistemas a eventos discretos desde un nuevo enfoque. El enfoque a usar es la **programación evolutiva**, que tiene sus bases en la Computación Evolutiva, y no es más que un método de búsqueda y optimización. El planteamiento es hacer evolucionar un conjunto de individuos, cada uno representando un modelo de SED, que a su vez es una posible solución del sistema que se desea identificar. Así, a través de un número finito de generaciones estos sistemas se deberían ir ajustando al entorno, para resultar en un conjunto de máquinas de estados finitos que se lograrán acoplar a una secuencia de eventos, eventos obtenidos del sistema real y que sirvan para reproducir el comportamiento general del modelo. Finalmente, se desarrolla una herramienta basada en nuestra propuesta.

Este trabajo esta organizado de la siguiente manera: se hace una introducción al marco teórico que sustenta este trabajo; después se presenta la propuesta y una herramienta computacional basada en ella que permite diseñar autómatas de estados finitos; a continuación se presentan algunas de las pruebas a las cuales fue sometida la propuesta/herramienta; finalmente se exponen las conclusiones.

II. Aspectos teóricos

A. Sistemas a eventos discretos

Un SED es un sistema cuyo espacio de estados es un conjunto discreto el cual es manejado por eventos, tal que su evolución de estados depende exclusivamente de la ocurrencia asíncrona de eventos discretos en el tiempo [18]. Un evento resulta un concepto primitivo, el cual puede ser pensado como una ocurrencia instantánea que causa una transición de un estado a otro.

Los SED tienen un ancho rango de aplicaciones, en sistemas de comunicaciones, sistemas de tráfico, el computador digital, circuitos secuenciales, etc.

Este trabajo fue financiado en parte por el CDCHT-ULA (Proyecto I-820-05-02-AA), y el Fonacit (Proyecto 2005000170).

J. Aguilar trabaja en CEMISID. Departamento de Computación, Facultad de Ingeniería, Universidad de los Andes, Mérida, 5101, Venezuela (e-mail: aguilar@ula.ve).

Si consideramos un SED como un generador de un lenguaje formal, se puede introducir la característica de Control al tomar en cuenta ciertos eventos o transiciones que pueden ser inhabilitados por un controlador externo; por lo cual se puede afirmar que un SED Controlado es influenciado en su evolución por la prohibición de la ocurrencia de eventos claves en ciertos momentos. Pero antes de poder controlar cualquier sistema debe existir primero un modelo.

No siempre es posible obtener ese modelo analizando el proceso, puede ser matemáticamente muy difícil, o no todos los parámetros del modelo pueden ser conocidos. Entonces, se deben llevar a cabo técnicas de identificación. Finalmente, los SDED son modelos naturales para describir el comportamiento de los sistemas de supervisión en procesos de producción.

B. Máquinas de estados finitos (MEF)

Una MEF es un transductor el cual opera sobre un conjunto finito de símbolos de entrada (alfabeto), posee un número finito de estados internos, y produce símbolos de salida desde un alfabeto finito [2, 4, 12, 13, 14, 15]. Así, una MEF es una lógica matemática. Es esencialmente un programa de computadora: representa una secuencia de instrucciones a ser ejecutadas, donde cada instrucción depende del estado actual de la máquina y del actual estimulo.

Las posibles entradas al sistema son una secuencia de símbolos seleccionadas desde un conjunto finito I de símbolos de entrada, y las salidas resultantes son secuencias de símbolos escogidas desde un conjunto finito Z de símbolos de salida. Cualquier "caja negra" que produce un símbolo de salida cuando un símbolo de entrada es aplicado y que satisface las propiedades anteriormente mencionadas es llamada máquina secuencial o transductor finito.

La selección de un conjunto de estados para representar una máquina dada no es un proceso único, esto no es una limitación muy seria, ya que el principal objetivo es describir el comportamiento general entrada-salida de la máquina en vez de su construcción.

1) Representación de estados finitos

Una máquina secuencial en la cual el conjunto de estados Q contiene un número finito de elementos es llamada MEF. Más formalmente, una MEF es una 5-tupla:

$$M = (Q, I, Z, s, o)$$

Donde: Q es el conjunto de estados; I es el conjunto de símbolos de Entrada; Z el conjunto de símbolos de salida; s(QxI)->I es la función de estado siguiente; o(QxI)->Q es la función de salida siguiente. El comportamiento de una MEF es descrito como una secuencia de eventos que ocurren en instantes discretos t=1, 2, 3, etc.

Cualquier 5-tupla de conjuntos y funciones que satisfaga la definición anterior será interpretada como una descripción matemática de la MEF, es decir, si es proporcionado un símbolo de entrada x estando en un estado q, el símbolo de salida estará dado por o(q, x), y la transición al siguiente estado estará dada por s(q, x). Sólo la información contenida en el estado actual describirá como actuará la máquina para un estimulo dado.

Así que una MEF no es más que un transductor que puede ser estimulado por un alfabeto finito de símbolos de entrada, que puede responder con un alfabeto finito de símbolos de salida y que posee un número finito de diferentes estados internos. El correspondiente par símbolo entrada-salida y transición al estado siguiente para cada símbolo de entrada, tomado sobre cada estado, específica el comportamiento de cualquier MEF, dado un estado inicial.

Existen tres técnicas comúnmente usadas para representar las propiedades analíticas de una máquina: las tablas de transición, los diagramas de transición y las matrices de transición. A continuación se muestra un ejemplo de una matriz de transición para una máquina con tres estados, cuyo conjunto de entradas es {0,1} y el conjunto de salidas es {α, β, δ}.

Tabla 1. Matriz de Transición

Edo. Actual/ Edo. Siguiente	A	B	C
A	1/β	0/β	
B		0/δ	1/α
C	1/δ	0/β	

C. Identificación de sistemas

El problema de Identificación se refiere a tener una caja negra, donde sólo podemos observar el cambio de la salida debido al cambio de la entrada [5, 6, 7, 8]. El objetivo es encontrar una descripción "simple" del comportamiento que se observa. Si no tenemos alguna idea de lo que podría estar en la caja, la tarea de identificación es más difícil, por ejemplo, nos costaría mas interpretar los datos obtenidos. En situaciones reales, lo que se suele hacer es probar con un número finito de experimentos e incrementar cada vez más la complejidad del modelo, hasta lograr explicar o justificar los datos resultantes.

Otro problema es tratar de encontrar el modelo "más simple" de la caja, ya que no se conoce la medida de "simplicidad" objetivamente, por ejemplo, es difícil cuando se están considerando dos representaciones del mismo comportamiento, una descripción es larga, pero con una estructura simple, la otra es pequeña pero tiene una estructura compleja, entonces ¿Cuál es la más simple?.

Generalmente, para llevar a cabo la identificación de una MEF se establece un conjunto de condiciones que deben ser satisfechas para poder determinar las propiedades de la máquina desconocida. Primero se asume que todo el conjunto de símbolos de entrada está definido, si no se posee esta información no se puede definir completamente las funciones de transición (salida y estado siguiente).

La condición final que se debe cumplir para identificar una MEF es que ésta debe ser fuertemente conectada. El problema de una máquina que no es fuertemente conectada radica básicamente en el hecho de que no es posible obtener una secuencia (o experimento) que evolucione a través de los distintos estados de una máquina. Dos ejemplos de máquinas no fuertemente conectadas son cuando la máquina posee estados especiales (recurrentes, etc.), o cuando la MEF no es globalmente alcanzable (posee uno o más estados inalcanzables). Esto hace que la máquina no es globalmente controlable. En algunos trabajos que presentan métodos para

identificar máquinas, además de las restricciones anteriores, también restringen la identificación a máquinas de estados (son aquellas que no poseen estados equivalentes).

La identificación de MEF se ha llevado a cabo a través de experimentos de naturaleza adaptativos, los cuales requieren que el ingeniero (y/o identificador) haga uso de su conocimiento de máquinas secuenciales y SED para seleccionar la apropiada secuencia de entrada en cada etapa de la identificación. La idea básica es aplicar una secuencia y observar la respuesta de la máquina, de esa información se formulan los diagramas de estado parciales. Luego se aplica otra secuencia de entrada y se usa la respuesta para extender o eliminar los previamente formados diagramas de estado parciales, y así se continúa el proceso hasta que se obtenga al menos un diagrama de estado completamente definido que describa la máquina que se está investigando.

La selección de una secuencia de prueba en cualquier punto del experimento depende del ingeniero (y/o identificador) y su interpretación de los resultados previos. Sin embargo, muchas veces se escoge una secuencia de longitud *n* igual al número máximo posible de estados. También como se asumió que la máquina no presenta estados equivalentes, para dos estados cualesquiera q1 y q2 existen secuencias de entrada que generan distintas secuencias de salida.

Los procedimientos de identificación anteriormente mencionados son los que generalmente se llevan a cabo en el momento de enfrentarse a problemas de identificación de dinámicas discretas. En la literatura que aborda este tema se hace referencia a diversas técnicas de identificación de máquinas de estados finitos, pero en el fondo todas parten del mismo planteamiento y presentan las mismas restricciones. Finalmente, cuando la salida de la máquina es independiente de la entrada, se trata de una máquina de Moore, de lo contrario se asume que la máquina se comporta como el modelo de Mealy.

C. Computación Evolutiva

Bajo el término de Computación Evolutiva se engloba a un amplio conjunto de técnicas de resolución de problemas complejos basados en la emulación de los procesos naturales de evolución [1, 3, 9, 10]. Es decir, la Computación Evolutiva abarca los modelos computacionales que usan como elemento clave algún mecanismo de la teoría de la evolución para su diseño e implementación.

El principal aporte de la Computación Evolutiva a la metodología de resolución de problemas consiste en el uso de mecanismos de selección de soluciones potenciales y de construcción de nuevos candidatos por recombinación de características de otros ya presentes, de modo parecido a como ocurre en la evolución de los organismos naturales.

A modo más específico, la Computación Evolutiva es un enfoque alternativo para abordar problemas complejos de búsqueda y aprendizaje a través de modelos computacionales de procesos evolutivos. Las implementaciones concretas de tales modelos se conocen como **algoritmos evolutivos**. El propósito de los algoritmos evolutivos es guiar una búsqueda estocástica haciendo evolucionar un conjunto de estructuras y seleccionando de modo iterativo las más aptas. Un algoritmo evolutivo se ejecuta sobre una población de individuos, que representan a un conjunto de candidatos a soluciones de un problema, dichos individuos son sometidos a una serie de transformaciones con las que se actualiza la búsqueda y después a un proceso de selección que favorece a los mejores individuos. Cada ciclo de transformación-selección constituye una generación. Se espera del algoritmo evolutivo que tras cierto número de generaciones (iteraciones) el mejor individuo esté razonablemente próximo a la solución buscada.

Los **operadores evolutivos** son los que realizan los cambios de la población durante la ejecución de un algoritmo evolutivo. Clásicamente, existen dos operadores genéticos: mutación (Es un operador unario que simula el proceso evolutivo que ocurre en los individuos cuando cambian su estructura genética.) y cruce (Es un operador normalmente binario, que permite representar el procesos de apareamiento natural usando operaciones sencillas. toman diversos componentes de distintos individuos para generar con ellos nuevos individuos, de tal manera que hereden características de sus padres). Sin embargo, han sido propuestos otros operadores que se acoplan a problemas particulares, por ejemplo: dominación, segregación, traslocación, inversión, entre otros.

Los pasos de un algoritmo evolutivo son los siguientes:

- Generación de una población inicial, generalmente en forma aleatoria.
- Evaluación de los individuos
- Selección de algunos individuos de la población, a través de algún mecanismo.
- Modificación de los genes de los progenitores seleccionados usando los operadores genéticos.
- Generación de una nueva población mediante algún mecanismo de reemplazo.
- Verificación del criterio de parada del algoritmo, y se regresa al paso 3, si es el caso.

Las principales técnicas de la Computación Evolutiva son [1]: los **algoritmos genéticos** propuestos por Holland, las **estrategias evolutivas** propuestas por Rochenberg y Schwefel, la **programación genética** propuesta por Koza, y la **programación evolutiva** propuesta por Fogel.

1) Programación evolutiva (EP)

La **programación evolutiva** fue originalmente concebida por Lawrence J. Fogel en 1960 [1, 9, 10, 11] y consiste en un mecanismo que hace evolucionar un conjunto de máquinas de estados finitos. Esta técnica desarrolla un conjunto de soluciones las cuales muestran un comportamiento óptimo con respecto a un ambiente y a una función deseada.

A continuación se hace una explicación exhaustiva de como la programación evolutiva hace evolucionar las máquinas de estado finito.

- Población inicial (P(t)): Se comienza con una población de máquinas de estados finitos que representan soluciones al problema en cuestión.
- Evaluación: Cada máquina es evaluada por medio de una función específica, que calcula la capacidad del individuo para resolver el problema.

- Mecanismo de selección: La selección de las máquinas que se convertirán en padres de la próxima generación se hace de forma secuencial, es decir, todos los individuos son seleccionados ya que cada máquina genera un descendiente a través de un proceso de mutación que se aplica sobre ella.
- P'(t) = mutación de P(t): cada miembro de la población es alterado a través de un proceso de mutación, el cruce no es utilizado. Clásicamente existen cinco tipos de mutaciones aleatorias:
  - Cambiar un símbolo de salida.
  - Cambiar la transición de un estado a otro.
  - Agregar un estado.
  - Eliminar un estado.
  - Cambiar el estado inicial.
- P(t+1) = supervivientes de P(t) y P'(t): se escogen los individuos que van a sobrevivir para la siguiente generación. Normalmente se toman las k mejores máquinas de la población total. Generalmente el tamaño de la población permanece constante, pero no hay restricción en este caso.
- Criterio de parada: El proceso termina cuando la solución alcanza una calidad predeterminada, cuando ha sido obtenido un número específico de iteraciones (generaciones), o algún otro criterio de parada.

III. La Programación Evolutiva en la Identificación de Sistemas Dinámicos

En esta sección se presenta el sistema que permite obtener modelos a eventos discretos.

A. Diseño de la estructura del genotipo

El **genotipo** en la programación evolutiva debe representar una MEF, además debe ser capaz de generar una secuencia de salida debido a una de entrada (debe poseer una función de salida), dicho requisito es indispensable por el hecho de que se desea hacer un proceso de identificación. A continuación se presentan dos modelos de máquinas de estados finitos (o dos formas de máquinas secuenciales): Mealy y Moore.

1) Modelo de Mealy

El modelo de Mealy no es más que un autómata generador de lenguaje representado por la siguiente 6-tupla:

$$M = (Q, I, Z, s, o, q(0))$$

Donde: Q es el conjunto de estados; I es el conjunto de símbolos de entrada; Z es el conjunto de símbolos de salida; s(QxI)->Q es la función de estado siguiente; o(QxI)->Z es la función de salida siguiente; q(0) es el estado inicial.

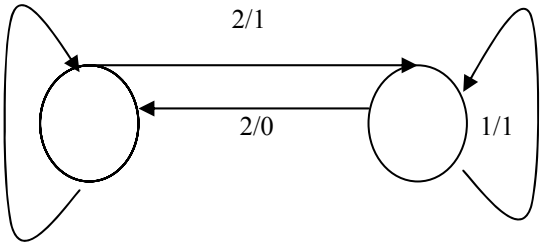

Figura 1. Diagrama de Transición de una Máquina de Mealy.

En esta estructura se debe notar que la salida depende de la entrada, esto quiere decir, que una máquina de Mealy sólo genera un símbolo de salida cuando es presentado un símbolo de entrada.

La semántica procedimental del modelo de Mealy es el siguiente, la máquina se encuentra en un estado q(0), posteriormente, cuando recibe un literal de entrada, entonces emite el símbolo de salida y transita al nuevo estado. El diagrama de transición y la tabla de transición de Mealy se observan en la figura 1 y en la tabla 2, respectivamente.

Tabla 2. Tabla de Transición de una Máquina de Mealy

Estados / Entradas	1	2
A	A/0	B/1
B	B/1	A/0

2) Modelo de Moore

Una máquina de Moore es similar a una de Mealy, salvo en que la respuesta sólo depende del estado actual de la máquina y es independiente de la entrada. Precisamente, una máquina de Moore es una estructura de la forma:

$$M = (Q, I, Z, s, o, q(0))$$

Donde: Q es el conjunto de estados; I es el conjunto de símbolos de entrada; Z es el conjunto de símbolos de salida; s(QxI)->Q es la función de estado siguiente; o(QxI)->Z es la función de salida siguiente; q(0) es el estado inicial.

El modelo de Moore opera de la siguiente manera, iniciando la máquina se encuentra en el estado q(0). Posteriormente, cuando recibe una literal de entrada, entonces transita al nuevo estado y después emite el símbolo de salida.

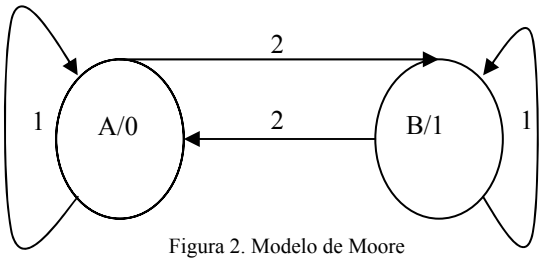

Figura 2. Modelo de Moore

El diagrama de transición y la tabla de transición de Moore se observan en la figura 2 y en la tabla 3, respectivamente.

Tabla 3. Tabla de Transición de una Máquina de Moore

Estados / Entradas	1	2	Salida
A	A	B	0
B	B	A	1

3) Equivalencias entre los dos modelos

Una MEF representada como un modelo de Mealy (Moore) tiene su equivalente representación en un modelo de Moore (Mealy), ya que ambas máquinas son completamente equivalentes (La comprobación de equivalencia entre las máquinas de Mealy y Moore se puede encontrar en [2, 12, 13, 14, 15]). A causa de esta equivalencia, existen procedimientos sencillos de conversión entre un modelo y otro.

La razón fundamental de selección de que tipo de estructura se utiliza para codificar el genotipo es por rendimiento computacional, una máquina de tipo Mealy generalmente

requiere de menos estados que una de Moore para reconocer una señal de entrada y generar la salida correspondiente. Como se requiere trabajar con poblaciones relativamente grandes, entonces, no se desea disminuir la velocidad de búsqueda y optimización del algoritmo al introducir individuos muy complejos.

B. Inicialización de las máquinas

Este proceso sólo se ejecuta una vez, y es al principio del algoritmo. En el sistema se implementaron dos vías de inicializar la población de máquinas de estados finitos:

- Completamente aleatoria, se asigna un estado inicial, después se escoge al azar un valor que será el número de estados que posee la máquina, que debe ser menor o igual al número máximo de posibles estados. Luego se comienza a llenar cada una de las celdas de la tabla de transiciones, la cantidad de columnas está previamente definida, ya que es el conjunto de símbolos de entrada. En cada intersección de estado con símbolo de transición se encuentra la celda que indica hacía que estado se transitará y cual será el correspondiente símbolo de salida, estos dos elementos también son generados aleatoriamente. Por último, se crea una vector que contendrá las etiquetas (por llamarlo de alguna manera) respectivas para identificar los estados, este vector no puede contener elementos repetidos (estados repetidos), y se va actualizando junto con la tabla de transiciones al momento que se efectúe una mutación sobre el individuo (sólo si la mutación es agregar o eliminar un estado).
- La otra manera es inicializarla desde una población ya existente.

C. Tipos de mutaciones

Se usan las cinco mutaciones enunciadas por Fogel, aleatoriamente se escoge que tipo de mutación se va a aplicar sobre cada individuo.

D. Función de costos

Posee dos atributos que son dos cadenas de caracteres, la secuencia de entrada y la secuencia de salida, que son compartidas para todos los individuos de la población. La capacidad de cada máquina se mide comparando la secuencia de salida que genera el individuo con la secuencia de salida de referencia, cuando es enfrentado a la secuencia de entrada. Este procedimiento se ejecuta de la siguiente manera: se aplica el primer símbolo de entrada al individuo que se encuentra en un estado inicial, si ese símbolo no es reconocido por el autómata se penaliza con un valor positivo, de lo contrario, si el símbolo es reconocido se compara el símbolo de salida que genera el autómata con el símbolo de salida de referencia, si los símbolos son diferentes el individuo es penalizado con una valor positivo, pero, sí los símbolos de salida coinciden la máquina es premiada con una valor negativo. Luego, se verifica hacia que estado transita el autómata y se repite el proceso para el siguiente símbolo de entrada-salida correspondiente hasta que se hayan alcanzado el final de las secuencias.

Como se pudo constatar en el procedimiento anteriormente relatado, la función de costos se esta minimizando, por lo cual aquellos individuos que posean funciones con valores más negativos serán los más aptos de la generación.

Además se incorporó en la función de costos la capacidad de recompensar a aquellos autómatas que poseen menos estados, esto es opcional y es con la finalidad de obtener máquinas de estados finitos que no sólo transducen una secuencia de entrada, sino que también sean simples (desde el punto de vista del tamaño).

IV. Sistema Computacional

A. Biblioteca de objetos evolutivos

La construcción del sistema se apoya en la biblioteca de objetos evolutivos *Envolvable Evolutionary Objects Library* [17], que es un toolbox en C++ para la Computación Evolutiva. La biblioteca esta desarrollada usando STL (*Standard Template Library*) y bajo el estándar ANSI C++.

Para hacer posible la implementación de la programación evolutiva usando esta biblioteca es necesario crear los objetos que son dependientes del algoritmo, tales son: el individuo (debe representar una MEF), la inicialización de cada máquina, los operadores genéticos que se van a aplicar sobre la población (en este caso sólo es la mutación), y la función de costo.

B. Interfaz gráfica de usuario y uso del sistema

Interfaz fue diseñada utilizando *Componentes Swing* de *Java Foundation Classes (JFC)*. En la figura 3 se muestra la interfaz diseñada. A continuación se procederá a hacer un relato de los parámetros más importantes que pueden ser introducidos, agrupados por categorías. Para el resto ver [16].

1) General

Esta categoría agrupa los siguientes parámetros:

- Semilla: Es un número de tipo entero largo sin signo, si este campo se deja en blanco la semilla para generar los números aleatorios será tomada de *time*(0).
- Tamaño de la población: Es el número de individuos con que se inicia el algoritmo.
- Tamaño del **torneo**: Es el número de individuos que son enfrentados.

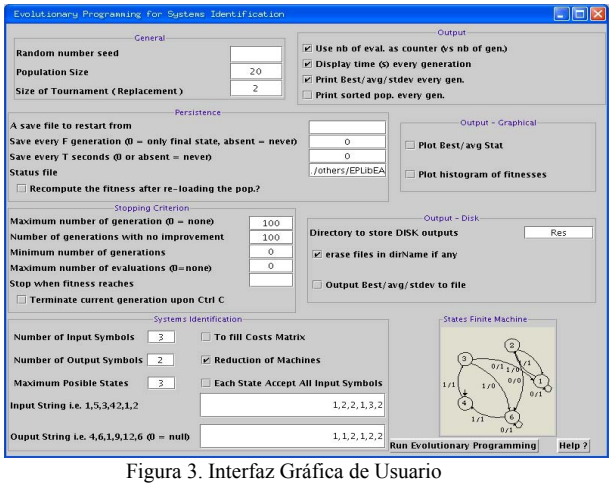

Figura 3. Interfaz Gráfica de Usuario

2) Salida por consola

Aquí se permite seleccionar que es lo que se va a mostrar en cada generación durante la ejecución del algoritmo,

- Número de evaluaciones: Permite mostrar en cada iteración la cantidad de evaluaciones que se han hecho sobre la población.
- Tiempo: Muestra en cada generación el tiempo que tarda en efectuar todas las operaciones.
- Imprimir parámetros estadísticos: Muestra en cada generación estadísticas como el valor de la función de costos del mejor individuo, el promedio de las funciones de toda la población y la desviación estándar.
- Población: muestra toda la población ordenada, de mejor a peor individuo, generación por generación.

3) Criterio de parada

En esta categoría se indica cuando parará la ejecución del algoritmo,

- Número máximo de generaciones: Es un número entero que indica la cantidad máxima de generaciones que se harán.
- Número de generaciones sin mejora: detiene la ejecución del algoritmo si se ha alcanzado un número de generaciones sin mejora alguna en los individuos.
- Número máximo de evaluaciones: El algoritmo parará su ejecución si se ha alcanzado ese número de evaluaciones. Debe ser un número entero, sí se introduce 0 queda inhabilitada esta opción.
- Objetivo alcanzado: En este campo se introduce el valor de la función de costos que se desea alcanzar, el algoritmo parará en cuanto haya al menos un individuo con ese valor.

4) Identificación de sistemas

En este grupo se encuentran los parámetros característicos del problema, es decir, los requisitos para llevar a cabo la identificación de dinámicas discretas:

- Número de símbolos de entrada: Es la cantidad de elementos que hay en el conjunto entrada, y es utilizado como valor de referencia para construir las tablas de transiciones.
- Número de símbolos de salida: la cantidad de elementos que existen en el conjunto de salida.
- Número máximo posible de estados (MAX): Es un número entero, el cual es tomado como referencia para construir los individuos, ninguna máquina tendrá más estados que los aquí señalados, pero sí puede tener menos estados.
- Llenar la matriz de costos: Al seleccionar esta opción permite incluir penalizaciones o gratificaciones a los individuos de acuerdo al símbolo de salida generado y el que debería generar.
- Realización de máquinas: Al seleccionar esta opción se premiarán las máquinas cuyo número de estados sea pequeño, mientras menos estados tenga la máquina, más negativa será su función de costos.
- Aceptar todas las transiciones: Esta opción se debe seleccionar si desea obtener máquinas en donde todas las transiciones están especificadas, de lo contrario habrá algunos eventos en los cuales la máquina no transitará a ningún estado siguiente (autómatas parcialmente especificados).
- Secuencia de entrada: Junto con la secuencia de salida se puede decir que son los parámetros más importantes para la identificación, no puede dejarse este campo vacío, sólo recibe símbolos enteros separados por coma, cuyo rango está entre 1 y la cantidad de símbolos que tenga el conjunto de entrada, por ejemplo, para el conjunto I={A,B,C} los posibles símbolos de entrada que se pueden introducir en la secuencia son 1=A,2=B,3=C.
- Secuencia de salida: Se deben cumplir las mismas especificaciones que para la secuencia de entrada, pero, además de los posibles símbolos de salida que contiene el conjunto se puede introducir el comodín cero (0), el cual se debe utilizar sólo si se desea especificar que para un símbolo de entrada cualquiera no existe una salida (salida ausente).

V. Experimentación

Se van a mostrar una serie de experimentos que se realizaron para verificar la eficacia de la programación evolutiva como técnica de identificación de dinámicas a eventos discretos. Para aplicar las pruebas se seleccionó MEF que abarcaran distintas características, de esta manera se puede estudiar los resultados obtenidos para cada tipo de autómata, y por ende emitir conclusiones. Los resultados obtenidos (o modelos) son comparados directamente con el autómata real.

Por cada experimento se mostraran dos figuras, el valor de la función objetivo de cada uno de los individuos de la ultima generación, y un histograma que dibuja como se han ido ajustando (mejorando o empeorando) los individuos a través de las generaciones. Además, se mostraran las figuras del modelo real y del modelo identificado. Finalmente, se dará la lista de algunos de los parámetros usados en la identificación, particularmente las secuencias de entrada-salida. Todos las MEF estudiadas en esta parte están propuestas en [4, 14, 18] y más resultados se consiguen en [16].

A. Máquinas completamente especificadas

Son aquellas donde cada estado de la máquina reconoce todos y cada uno de los eventos del conjunto de entrada, y además genera un símbolo de salida cuando se realiza una transición, ya sea a otro estado de la máquina o al mismo.

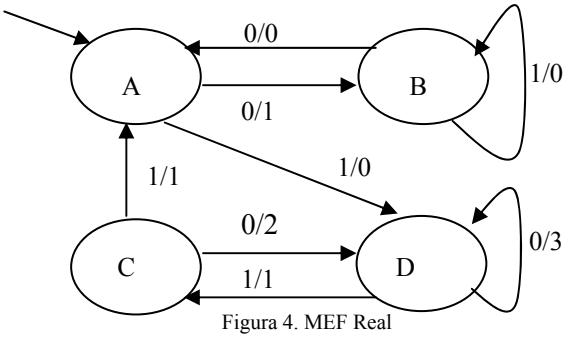

Figura 4. MEF Real

En la figura 4 se muestra un autómata que pertenece a esta categoría. Los parámetros más importantes usados en este experimento son: Tamaño de la población= 20; Número de generaciones = 695; Secuencia de entrada = 0, 0, 0, 1, 0, 1, 0, 0,1,0,0,1,1,0; Secuencia de salida = 1,0,1,0,0,0,3,3,1,2,3,1,1,1.

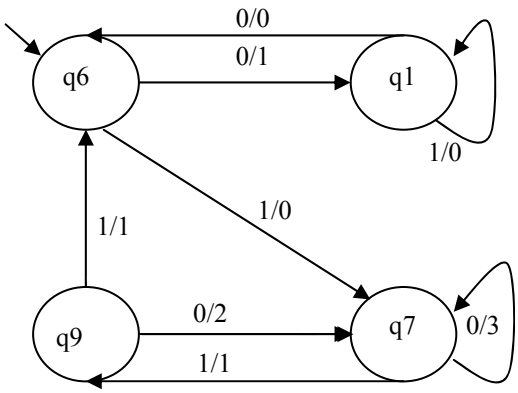

Figura 5. MEF Identificada (Experimento 1)

La MEF identificada se muestra en la Figura 5, es evidente que es igual al modelo real, no es necesario comprobar la aptitud del modelo generado ante una secuencia de validación. En todas las experiencias de este tipo la máquina identificada es completamente igual a la MEF real (ver [16] para mas detalles).

B. Máquinas parcialmente especificadas

Una máquina parcialmente especificada es aquella en la cual una o más transiciones son indefinidas o no especificadas, o la máquina reconoce el símbolo de entrada pero no genera una salida. La figura 6 es el primer ejemplo de máquinas parcialmente especificadas.

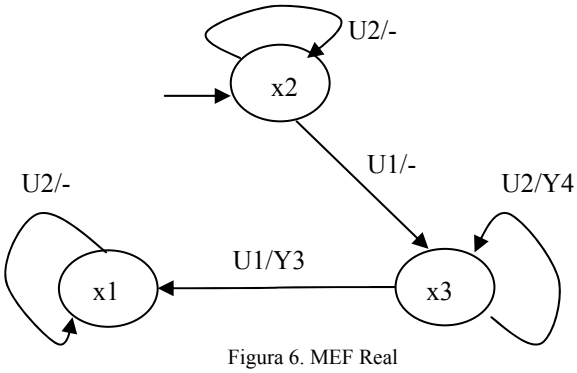

Figura 6. MEF Real

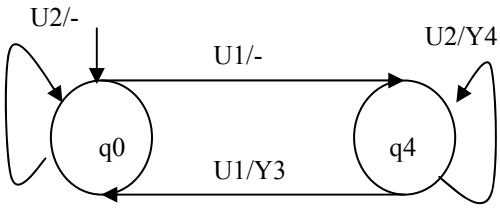

Figura 7. MEF Identificada

Algunos de los parámetros usados para su identificación fueron: Tamaño de la Población= 20; Número de Generaciones= 46, Secuencia de Entrada= u2, u1, u2, u1, u2; Secuencia de Salida=-,-, y4, y3,-.

A diferencia de la anterior prueba, la MEF resultante no es igual al modelo real (ver figura 7), por esa razón es conveniente examinar el comportamiento del modelo resultante ante una secuencia entrada-salida de validación. Se selecciono la secuencia: Secuencia de Entrada= u1, u1, u2; Secuencia de Salida = -, y3,-. La MEF identificada transduce de manera satisfactoria la secuencia de validación aplicada.

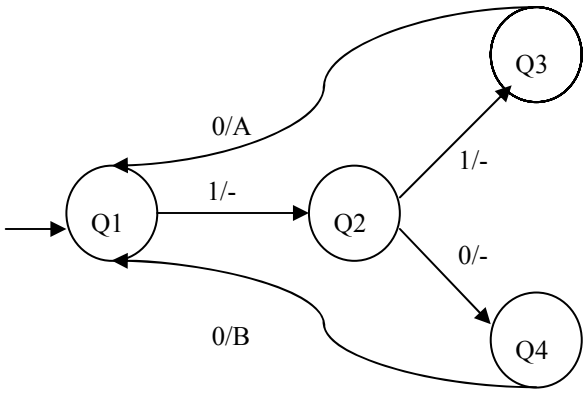

Figura 8. MEF Real

El ultimo experimento estudiado en esta sección (ver figura 8), al igual que el anterior, enseña una característica muy especial, transduce la secuencia de entrada en una de salida previamente especificada, pero requiriendo menos estados que el modelo real. Se ha encontrando así otra utilidad de la programación evolutiva, optimiza autómatas desde el punto de vista de estados.

Esta reducción o realización de MEF se comenta luego, por ahora, el experimento demuestra este hecho, pasando de un autómata de cuatro estados a uno con tan sólo tres, este último reconociendo la secuencia de validación aplicada (figura 9). Las MEF identificadas en las experiencias de esta parte introducen además una nueva categoría de máquinas, estas son aquellas que pueden transitar de un estado a otro con distintos eventos. Algunos de los parámetros usados en este experimento fueron: Secuencia de entrada= 1, 1, 0, 1, 0, 0, 1, 1 0; Secuencia de salida= -, -, A, -,-, B, -, A.

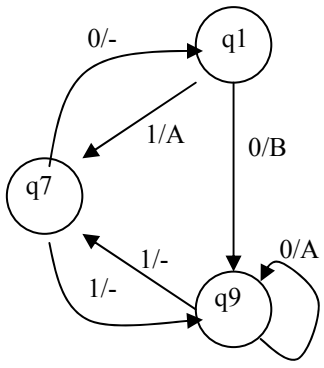

Figura 9. MEF Identificada

La secuencia de validación usada fue: Secuencia de Entrada=1, 0, 0, 1, 0, 0, 1, 1, 0; Secuencia de Salida=-,-, B,-, -, B,-,-, A. Ambas siguen la misma secuencia de salida.

C. Máquinas fuertemente conectadas

Una máquina con un conjunto Q de estados es fuertemente conectada sí para cualquier par de estados Qi y Qj ∈Q existe al menos una secuencia de entrada tal que el estado Qj pueda ser alcanzado por Qi [4]. Las máquinas fuertemente conectadas son interesantes (teórica y prácticamente) porque tienen la propiedad de que cualquier estado de la máquina puede ser alcanzado desde cualquier otro estado. En la teoría del Control Automático el concepto de "sistema controlable" es análogo a la noción de máquina fuertemente conectada.

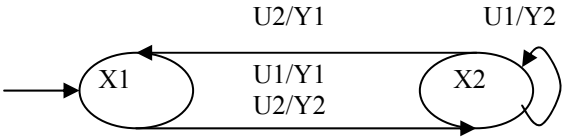

Figura 10. MEF real

El experimento que se muestra a continuación introduce máquinas de un tipo muy especial (figura 10), en estas la transición entre dos estados determinados puede no ser única, y además el símbolo de salida que genera el autómata cuando esta pasando de un estado a otro también puede no ser único, y va a depender de cual evento (de una lista de transiciones que reconoce) es el que esta ocasionando el cambio de estado.

Los parámetros más importantes usados en esta parte son: Tamaño de la población= 20; Número de generaciones= 777; Secuencia de entrada= u1, u1, u2, u2, u1, u1, u2, u1; Secuencia de salida= y1, y2, y1, y2, y2, y2, y1, y1.

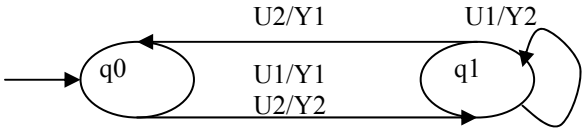

Figura 11. MFE identificada

La Figura 11 muestra que nuestro algoritmo identifica la misma máquina original (figura 10). Ahora bien, en este caso los tiempos de la programación evolutiva pueden llegar a ser más grande porque el número de enlaces en esa red es mayor. Esto último conlleva a un espacio de búsqueda mayor.

D. Realización de máquinas

Antes se había mencionado la posibilidad de que la herramienta diseñada usando la programación evolutiva era capaz de reducir el número de estados que un autómata necesita para definir un comportamiento entrada-salida. Esto es viable, ya que el algoritmo puede encontrar máquinas equivalentes con menos estados que el autómata en estudio.

Esta reducción no es fortuita, y generalmente se debe a la presencia de estados equivalentes en una MEF. Dos estados son equivalentes si tiene idénticas características entrada-salida, es decir, dos estado Qi y Qj son equivalentes sí y solo sí cada posible secuencia de entrada aplicada sobre la máquina produce la misma secuencia de salida cuando sus estados iniciales son tanto Qi como Qj [12].

Para este experimento no se generó secuencia de validación, sin embargo, al observar cada uno de los autómatas reales y los identificados, se puede constatar que no son iguales, entonces (ver figuras 12 y 13), ¿Por qué no es necesaria al menos una secuencia de validación?, la respuesta a esta pregunta es que la máquina real y la máquina identificada son máquinas equivalentes.

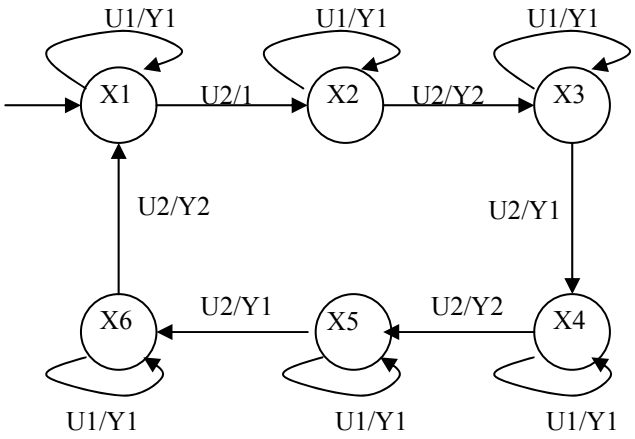

Figura 12. MEF Real

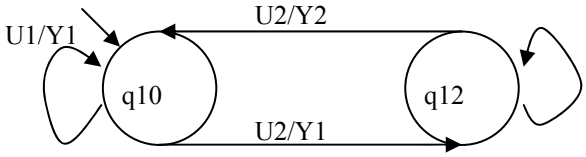

Figura 13. MEF Identificada

Los parámetros usados en este caso más importantes son: Tamaño de la población= 20 ; Número de generaciones= 139; Secuencia de entrada= u1, u2, u1, u2, u1, u2, u1, u2, u1, u2, u1, u2, u2, u2, u2, u2, u2, u2; Secuencia de salida= y1, y1, y1, y2, y1, y1, y1, y2, y1, y1, y1, y2, y1, y2, y1, y2, y1, y2.

Si se observa con atención el experimento, podrá encontrar los estados equivalentes en cada MEF. Por ejemplo, el autómata real posee estados como X1, X3 y X5 que son equivalentes entre sí, además los estados X2, X4 y X6 también lo son. Analizando la MEF generada, se comprueba que con sólo dos estados es suficiente para describir el comportamiento del autómata, los estados X1, X3 y X5 del modelo real, son representados por su equivalente q2 en el modelo identificado, lo mismo sucede para los estados X2, X4 y X6, con su equivalente q4.

E. Máquinas equivalentes

Dos máquinas son equivalentes si ambas traducen de idéntica manera a cualquier palabra de entrada. Los modelos de Mealy y Moore son ejemplos claros de máquinas equivalentes, ya que las máquinas de Moore son casos particulares de las máquinas de Mealy. Así, se tiene que toda máquina de Moore es equivalente a una de Mealy. El recíproco también se cumple.

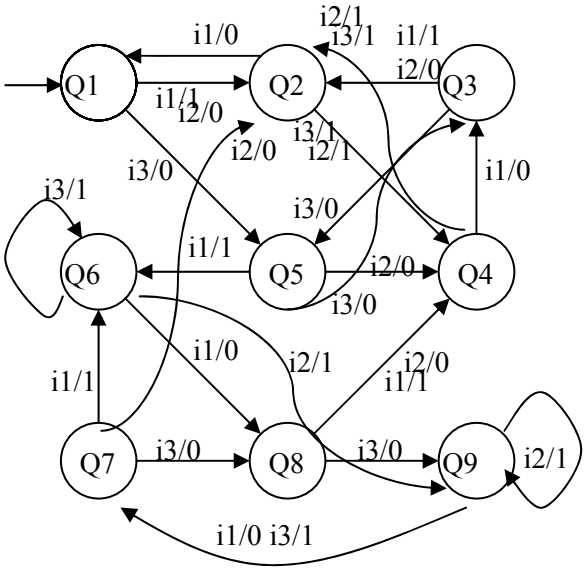

Figura 14. MEF Real

Con el último experimento no se introduce un tipo de máquina diferente a las ya estudiadas, sólo se desea conocer la respuesta de la herramienta al trabajar con modelos de sistemas dinámicos a eventos discretos mucho más complejos que los abordados antes, por esa razón se presenta la MEF de la figura 14, la cual presenta una interconexión de estados algo más elaborada. El autómata resultante (ver figura 15) para el experimento, fue validado según la secuencia siguiente: Secuencia de entrada= i3, i1, i2, i2, i3, i3, i1, i1, i3, i1, i1, i1, i1, i2, i1, i3, i3, i1, i2, i1, i2, i1, i3; Secuencia de salida = 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0.

Algunos de los parámetros usados en este experimento fueron: Número de generaciones= 4354; Secuencia de entrada= i1, i1, i3, i3, i1, i2, i2, i3, i1, i2, i3, i3, i3, i1, i3, i3 i3, i1, i1, i1, i1, i1, i3, i1, i2, i2, i1, i1, i1, i3, i3, i3, i2, i1, i3, i1, i2, i3, i2; Secuencia de salida= 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,0.

Para concluir, se debe destacar que la identificación de SDED no es hallar máquinas de estados finitos equivalentes, sólo se desean hallar modelos que se ajusten a las condiciones normales de operación del sistema, por consiguiente es que se ha estudiado el comportamiento que sigue el autómata identificado ante una cadena de validación, que es distinta a las secuencias utilizadas para la evolución de las máquinas de estados finitos.

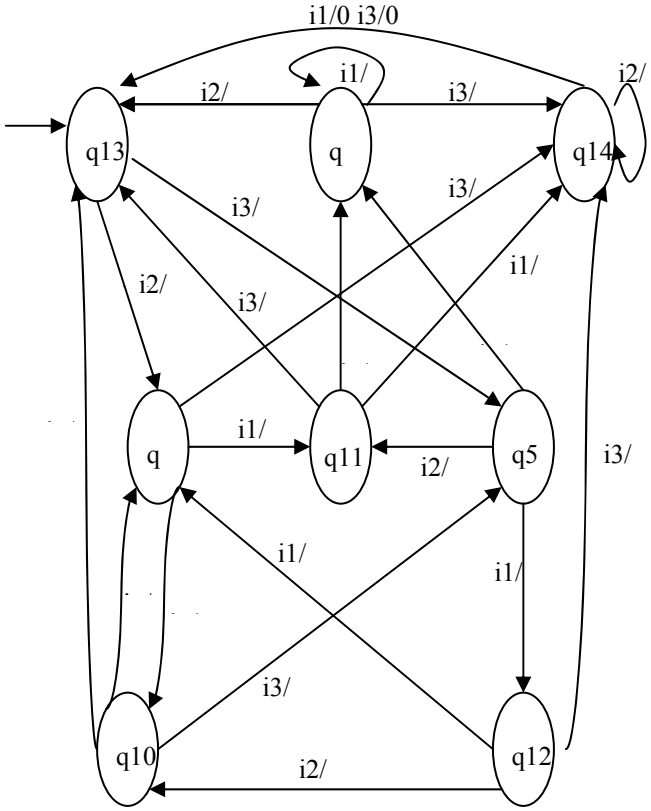

Figura 15. MEF Identificada

VI. CONCLUSIONES

Se ha analizado a lo largo de esta investigación la incidencia de la programación evolutiva para la resolución de problemas referentes a sistemas a eventos discretos, en particular se estudió la identificación de este tipo de dinámicas, obteniendo resultados satisfactorios y demostrando así la eficiencia que aporta un algoritmo evolutivo en la búsqueda de modelos que se adapten a las características presentadas.

Es evidente que la forma como la programación evolutiva genera posibles candidatos para identificar el sistema en estudio, es optimizando el comportamiento que cada autómata presenta hacia el entorno dinámicamente cambiante. Durante el desarrollo de este trabajo se fueron agregando características para generalizar el desempeño del algoritmo, por consiguiente, se abrió la posibilidad de analizar distintos tipos de comportamientos de los SDED.

Aunque la técnica de la programación evolutiva para la identificación de sistemas dinámicos a eventos discretos no está exenta de inconvenientes, estas complicaciones no son mayores que las que se presentan al identificar los sistemas por métodos convencionales, las cuales requieren un basto dominio de la teoría de máquinas secuenciales, entre otras cosas. Para concluir, la técnica introducida en este trabajo muestra una combinación de simplicidad, facilidad, rapidez y eficacia, que le permite competir con cualquier otra manera de analizar SDED.

VII. Referencias

[1] J. Aguilar, F. Rivas (Eds.). *Introducción a las Técnicas de Computación Inteligente,* MERITEC, Venezuela, 2001.

[2] I. Aleksander, F. Keith Hanna, *Automata Theory*: *An Engineering Approach*. Computer Systems Engineering Series. Crane, Russak & Company, New York, 1975.

[3] P. Angeline, D. Fogel, "An Evolutionary Program for the Identification of Dynamical Systems." in *Proceedings of SPIE (Volume 3077): Application and Science of Artificial Neural Networks III*, Bellingham, USA, 1997, pp. 409-417.

[4] T. Booth, *Sequential Machines and Automata Theory*. John Wiley and Sons, USA, 1967.

[5] M. Cerrada, J. Aguilar, A. Titli, E. Colina, "Dynamical Adaptive Fuzzy Systems: An Application on System Identification," in *Proceedings of the 15th Triennial World Congress of the International Federation of Automatic Control*, Barcelona, España, 2002 , pp. 1006-1011.

[6] M. Cerrada, J. Aguilar, A. Titli, E. Colina, "Identificación de un Sistema No-lineal Variante en el Tiempo usando Sistemas Difusos Adaptativos Dinámicos". *Revista Técnica de la Facultad de Ingeniería,* Universidad del Zulia, Vol. 25, N 3, pp. 171-180, 2002.

[7] M. Cerrada, J. Aguilar, "Genetic Programming-Based Approach for System Identification", in *Advances in Fuzzy Systems and Evolutionary Computation, Artificial Intelligence* (A Series of Reference Books and Textbooks) *(*Ed. N. Mastorakis*)*, World Scientific and Engineering Society Press, pp. 329-334, 2001.

[8] M. Cerrada, J. Aguilar, "Fuzzy Classifier System and Genetic Programming on System Identification Problems," in *Proceeding of the Genetic and Evolutionary Computation Conference* (Ed. L. Spector et al.), San Francisco, USA, 2001, pp. 1245-1251.

[9] D. Fogel, *Evolutionary Computation.* IEEE Press*,* USA, 1995.

[10] D. Fogel, *Handbook of Evolutionary Computation*, Oxford University, USA, 1997.

[11] J. Heitkotter, *D.* Beasley (Eds), *The Hitch Hiker's Guide to Evolutionary Computation: A list of frequently asked questions (FAQ)*, USENET: comp.ai.genetic, 1998.

[12] F. Hennie, *Finite-State Models For Logical Machines*. John Wiley and Sons*,* USA, 1968.

[13] R. Kain, *Automata Theory: Machines and Languages*. McGraw-Hill, USA, 1972.

[14] Z. Kohavi, *Switching and Finite Automata Theory*. McGraw-Hill, USA, 1970.

[15] R. Nelson *Introduction to Automata*. John Wiley and Sons*,* USA, 1968.

[16] A. Piña, J. Aguilar, J. Cardillo, "La Programación Evolutiva en Identificación de Sistemas", *Technical Report* 21-2004, CEMISID, Universidad de los Andes, 2004

[17] M. Keijzer, J. Merelo, G. Romero, M. Schoenauer, *Evolving Objects: A general purpose evolutionary computation library,* Artificial Evolution, Vol. 2310, pp. 231-244, 2001.

[18] A. Tornambe, *Discrete-Event System Theory: An Introduction*. World Scientific, Malasia, 1995.

.

VIII. Biografía

Jose Aguilar (M'1999) obtuvo una Maestría en Informática en 1991 en la Universidad Paul Sabatier-Toulouse-France, y el Doctorado en Ciencias Computacionales en 1995 en la Universidad Rene Descartes-Paris-France. Además, realizó un Postdoctorado en el Departamento de Ciencias de la Computación de la Universidad de Houston entre 1999 y 2000. Es profesor del Departamento de Computación de la Universidad de los Andes, Mérida-Venezuela, e investigador del Centro de Microcomputación y Sistemas Distribuidos (CEMISID) de la misma universidad. El Dr. Aguilar ha sido profesor/investigador visitante en varias universidades y laboratorios (Université Pierre et Marie Curie Paris-France, Laboratorie d'Automatique et Analyses de Systemes Toulouse-France, Universidad Complutense de Madrid-España, entre otras). Sus áreas de interés son los sistemas paralelos y distribuidos, computación inteligente, (redes neuronales artificiales, lógica difusa, sistemas multiagente, computación evolutiva etc.), optimización combinatoria, reconocimiento de patrones, sistemas de control y automatización industrial. Ha publicado más de 200 artículos y varios libros en las áreas de Sistemas Computacionales y Gestión en Ciencia y Tecnología, y editor de varias actas de conferencias y de libros. Ha formado parte de varios jurados de premios científicos; presidido varios simposios, talleres, etc.; y es revisor de revistas internacionales permanentemente. Además, ha recibido varios premios nacionales como internacionales.