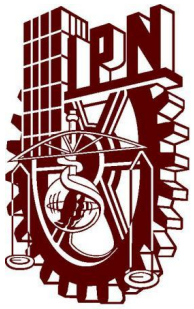



**INSTITUTO POLITÉCNICO NACIONAL**

**CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN**

Laboratorio de Inteligencia Artificial
Laboratorio de Simulación y Modelado


# Aprendizaje de fenómenos con representación de estados en 2D

# TESIS

QUE PARA OBTENER EL GRADO DE:

**MAESTRO EN CIENCIAS DE LA COMPUTACIÓN**

P R E S E N T A:


**Ing. Héctor Daniel Moreno Leyva**

Directores de tesis:
**Dr. Salvador Godoy Calderón**
**M. C. Germán Téllez Castillo**


México, CDMX Octubre 2020

# Resumen

Los autómatas celulares han sido una herramienta importante para el modelado de sistemas físicos, químicos y biológicos, debido a que tienen la capacidad de modelar sistemas dinámicos de manera discreta tanto en el tiempo como en el espacio. Sin embargo, esta herramienta tiene un uso limitado por la complejidad que existe en la generación de las reglas de evolución que modela el sistema completamente. Se ha demostrado que los algoritmos de aprendizaje simbólicos y sub-simbólicos han resultado de gran utilidad para generar modelos interpretables a partir de un conjunto de datos. Por esta razón, en el presente proyecto de investigación se estudió cómo usar estas herramientas de modelado, diseñando e implementando un algoritmo de aprendizaje automático que permita generar reglas a partir de un conjunto de datos representativos obtenidos de cierto fenómeno, sin perder la semántica que se encuentra embebida en los datos de los que se está aprendiendo.

# Abstract

Cellular automata have been an important tool for the modeling of physical, chemical and biological systems, because of their capacity to model dynamic systems discretely in space and time. However, the use of this tool is limited by the complexity that exists in the selection of the right evolving transition rules to completely model the system. It has been demonstrated that symbolic and sub-symbolic learning algorithms have been very useful to generate interpretative models from a data set. For this reason, in order to facilitate the use of these tools, we are proposing a model that generates a set of rules from a representative data set of a certain phenomenon, without loosing the semantics of what does the data represent.

# Thanks

I deeply appreciate the support received from CIC-IPN ...

To my family ...

# Índice general

# Índice de figuras

# Índice de cuadros

# Glosario

- $E$.- conjunto de ejemplos.
- $E^+$.- subconjunto de ejemplos positivos para la clase que se esta queriendo aprender.
- $E^-$.- subconjunto de ejemplos negativos para la clase que se esta queriendo aprender.
- Patrones de bioconvección.- son patrones que se forman ante la estimulacion luminosa sobre un fluido con microorganismos.
- Aprendizaje simbólico.- es un proceso que busca formular un modelo mediante la manipulación de símbolos, cuya semántica de operandos y operadores está definida; éstas características hacen a éste proceso replicable y rastreable.
- Aprendizaje sub-simbólico.- es un proceso que busca formular un modelo, mediante la manipulación de símbolos, cuya semántica de operandos está definida pero, al contrario del aprendizaje simbólico, los operadores pueden tener elementos aleatorios y esto es causa de que el proceso no sea replicable ni rastreable.

# Capítulo 1

# Introducción

Desde hace algunos siglos, los seres humanos se han beneficiado del modelado y la simulación de sistemas dinámicos que permiten estudiar y entender algunos fenómenos de la naturaleza. En la actualidad, existe una amplia gama de métodos de modelado y simulación, uno de ellos es el método de los autómatas celulares.

Los autómatas celulares son sistemas dinámicos en los cuales el espacio y el tiempo son discretos. Los estados de las células de una lattice regular, son actualizados con base en reglas de evolución de interacción local. Las reglas de evolución local son obtenidas con base en la experiencia, experimentos, o análisis de la información del problema de tal forma que las restricciones del problema se cumplan.

Hoy en día, este problema se está abordando desde distintos enfoques, sin embargo, muchos de ellos se centran en crear métodos específicos para el área de estudio en el que se está realizando la modelación del fenómeno. Como ejemplo tenemos la búsqueda de reglas para modelar el crecimiento urbano (Naghibi and Delavar, 2016), o la búsqueda de reglas para modelar patrones de bioconvección en algas unicelulares (Kawaharada et al., 2016). Existen otros trabajos de propósito más general que emplean algoritmos evolutivos para la generación de reglas, donde el comportamiento específico no es conocido, es decir, se trata de llegar de un estado inicial a un estado final sin tomar en cuenta lo que suceda en los estados intermedios.

El trabajo realizado en Kawaharada et al. (2016) es el más apegado a los

objetivos de esta tesis, no obstante, la diferencia principal entre los trabajos es que a nosotros nos interesa tomar en cuenta la información del fenómeno de principio a fin, pasando por cada uno de los estados intermedios, para poder crear un conjunto de reglas que repliquen un fenómeno con la mayor precisión posible. Con esto nos referimos a que el autómata celular debe evolucionar por todos los estados de los cuales tenemos información.

## 1.1. Definiciones

En esta sección definiremos lo que es un autómata celular ya que, en el presente trabajo, se utilizan como herramienta de modelación para realizar la simulación y predicción, de el conjunto de datos que se obtuvo para realizar la experimentación. Así mismo, se define que es el aprendizaje automático, con el fin de dar al lector los conceptos clave de esta área de estudio, la cual es relevante para este trabajo de tesis y, por ultimo, se describen los algoritmos genéticos brevemente, debido a que esta es la clase de algoritmos a la que pertenece nuestra propuesta de resolución del problema planteado.

### 1.1.1. Autómata celular

**Definición 1.1.1** Un autómata celular es una 5-tupla $(L, d, S, H, f)$, donde:

- $L$ es una matriz de dimensión $d$
- $d \, \epsilon \, N \cup \{0\}$ y es la dimensión del autómata
- $S$ es un conjunto finito de elementos llamados estados y es denotado por:
  $S = \{s_k : k\epsilon\{0, \ldots, |S| - 1\}\}$, donde $|S|$ es la cardinalidad del conjunto de estados $S$
- $H$ es un subconjunto finito de $Z^d$ llamado vecindad y es denotado por $\{v_j : x_{1,j}, \ldots, x_{d,j} : j\epsilon 1, \ldots, |H|\}$, 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

**Definición 1.1.2** Se dice que un autómata celular 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 1.1.3** Se dice que un autómata celular posee fronteras periódicas, si los extremos derecho e izquierdo son adyacentes el uno del otro.

### 1.1.2. Aprendizaje automático

El área de aprendizaje automático es una sub-área de la Inteligencia Artificial (IA), que se basa en distintos enfoques para mejorar el desempeño de un programa computacional sobre una tarea, utilizando la experiencia que se tiene sobre esa tarea. El aprendizaje puede ser de distintos tipos, por ejemplo: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado.

**Aprendizaje supervisado**: Consiste en el uso de un conjunto de entrenamiento para mejorar el desempeño en el programa computacional utilizando como entrada pares $(x, y)$ y la tarea es encontrar una función $f$ que al ingresar $x$ produzca $y$ como salida.

**Aprendizaje no supervisado**: Consiste en el uso de un conjunto de entrenamiento donde se conocen valores de entrada $x$, pero estos datos no están etiquetados. La tarea entonces es encontrar una función $f$, tal que al ingresar $x$ como entrada produzca como salida $y$ de tal forma que $y$ sea igual para todas las entradas $x$ que compartan cierta medida de similitud.

**Aprendizaje reforzado**: En este tipo de aprendizaje no tenemos conocimiento previo sobre la tarea, sino que se va obteniendo experiencia sobre la tarea conforme se va realizando. En este tipo de aprendizaje, el programa computacional se va ajustando con respecto a la métrica de desempeño obtenida de la tarea en cada iteración.

### 1.1.3. Algoritmos Genéticos

Los Algoritmos Genéticos (AG) son un conjunto de algoritmos de búsqueda que se basan en las mecánicas de la selección natural. Combinan la supervivencia del más apto con el intercambio de información para formar algoritmos de búsqueda con el novedoso instinto de búsqueda humana. A

causa de éstas características, se optó por el diseño de un algoritmo genético para realizar la búsqueda de las reglas que gobiernen un fenómeno –en este caso, los datos utilizados de los autómatas celulares con los que se realizaron los experimentos–, que es el objetivo de esta tesis.

**Algoritmo Genético Multi-poblacional**

Los Algoritmos Genéticos (AG) son una clase especial de los algoritmos genéticos, ya que estos cuentan con una población compuesta por subpoblaciones, las cuales cada una de ellas evoluciona independientemente la una de la otra. Este tipo de algoritmos pueden contar o no, con un mecanismo de migración entre las subpoblaciones.

## 1.2. Planteamiento del problema

Dada una secuencia de estados, representados cada uno de ellos como una lattice bidimensional, encontrar un conjunto de reglas de la forma IF *Ant* THEN *Cons*, donde *Ant* es una cláusula en forma normal conjuntiva (FNC) y *Cons* es un valor dentro del dominio del espacio de estados del autómata, tales que, al ingresarlas a un autómata celular, éste sea capaz de reproducir los estados de entrada.

## 1.3. Objetivos

### 1.3.1. Objetivo general

Diseñar un algoritmo nuevo de aprendizaje sub-simbólico con el fin de aprender reglas que gobiernan un fenómeno para las cuales se tienen rejillas de estados en 2 dimensiones como entrada.

### 1.3.2. Objetivos específicos

- Revisar y seleccionar algoritmos de aprendizaje de reglas, que sirvan de base para resolver el problema planteado.
- Diseñar un algoritmo genético multi-poblacional de aprendizaje.

- Diseñar procedimientos específicos para la simplificación del conjunto de reglas.
- Diseñar el modelo de autómata celular que sea capaz de reproducir los estados obtenidos a partir del conjunto de datos obtenido de la simulación de los autómatas seleccionados (Mite, Brain, Evoloops y Byl), cuando es alimentado con las reglas aprendidas.
- Usar los errores de aprendizaje y generalización para evaluar los resultados de cada experimento.

## 1.4. Justificación

La búsqueda de las reglas que son utilizadas para que un autómata celular pueda reproducir un fenómeno es una tarea compleja. Esto se debe a que el espacio de búsqueda crece exponencialmente, en la medida que aumenta el número de estados y el tamaño de la vecindad del autómata. Es por esta razón que se requiere tomar en consideración información adicional, que permita ejecutar una búsqueda eficiente en este espacio.

### 1.4.1. Beneficios esperados

Esta tesis genera un algoritmo genético multi-poblacional distribuido, que soluciona el problema planteado.

### 1.4.2. Alcances y límites

Este trabajo de tesis, se diseña e implementa un algoritmo que aprende un conjunto de reglas de la forma IF *Ant* THEN *Cons*, donde *Ant* es una cláusula en forma normal conjuntiva (FNC) y *Cons* es valor en el dominio de los estados del autómata. Para esta tarea, se utilizan los estados obtenidos de la evolución de autómatas celulares; esto se debe a que la ardua tarea de obtener estados de la forma requerida de fenómenos reales es una tarea compleja.

# Capítulo 2

# Marco teórico

## 2.1. Autómata celular

Los autómatas celulares son sistemas dinámicos y discretos en espacio, estado y tiempo. Propuestos por los matemáticos John von Neumann y Stanislaw Marcin Ulam como un sistema discreto para crear un modelo reduccionista de la auto-replicación, creando así en 1950 el primer autómata celular como método para calcular el movimiento de un líquido (Biaynicki-Birula and Biaynicka-Birula, 2012).

En 1970, John Horton Coway inventó un autómata celular llamado "Game of life" (Gardner, 1970), el cual es un autómata celular de dos dimensiones y dos estados, y que se hizo popular a partir de su publicación (octubre de 1970) en la columna: Mathematical Games de Martin Gardner en la revista Scientific American.

En 1984, Stephen Wolfram clasifica a los autómatas celulares lineales en cuatro clases (Wolfram, 1984). Y en 1985 conjeturó que la regla 110 de un autómata celular elemental era equivalente a una máquina de Turing, cosa que demostró Matthew Cook en el 2004 (Cook, 2004).

Como podemos ver, a través de los años con el incremento del poder de cómputo y del trabajo que se ha ido realizando alrededor de los autómatas celulares, también ha ido en aumento el interés por utilizarlos para modelar sistemas naturales y artificiales.

Los autómatas celulares pueden ser empleados como una alternativa a las ecuaciones diferenciales para modelar sistemas físicos (Toffoli, 1984), y como un modelo de cómputo paralelo y distribuido (Hillis, 1984).

Los autómatas celulares han sido aplicados exitosamente en diferentes áreas del conocimiento, tales como: la simulación de tránsito (Nagel and Schreckenberg, 1992; Simon and Nagel, 1998), la dinámica de fluidos (Margolus et al., 1986) y la formación de patrones (Tamayo and Hartman, 1987; Boerlijst, 1992). Asimismo, se ha tenido éxito en las conexiones con los lenguajes formales (Nordahl, 1989; Culik II et al., 1990) y, como se mencionó anteriormente, en el modelado y simulación de diversos sistemas físicos (Vichniac, 1984; Manneville et al., 2012) y biológicos (Ermentrout and Edelstein-Keshet, 1993).

Como podemos observar, los autómatas celulares tienen un gran alcance en la resolución de diversos problemas.

Los autómatas celulares clásicos poseen las siguientes cinco características (Ilachinski, 2001):

- **Una lattice de células discretas**: El sistema consiste de una estructura llamada lattice la cual puede ser de dimensionalidad $d$, donde $d \epsilon N \cup \{0\}$
- **Homogeneidad**: Todas las células son equivalentes.
- **Estados discretos**: Cada célula toma un estado de un conjunto finito de estados discretos.
- **Interacciones locales**: Cada célula interactúa 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.

## 2.2. Algoritmos genéticos

Los algoritmos genéticos fueron desarrollados por John Holland, sus colegas y estudiantes en la Universidad de Michigan. Los objetivos de esta investigación fueron los que se enlistan a continuación (**?**).

1. Abstraer y explicar rigurosamente los procesos adaptativos de los sistemas naturales.

2. Diseñar sistemas artificiales de software que retuvieran los mecanismos importantes de los sistemas naturales.

El tema central de la investigación sobre los algoritmos genéticos ha sido la robustez, el balance entre la eficiencia y la eficacia necesaria para sobrevivir en distintos ambientes. Los algoritmos genéticos han probado teórica y experimentalmente que proveen una búsqueda robusta en espacios complejos.

La implementación de un algoritmo genético consta de la definición de:

- La representación de las soluciones.
- La función de aptitud.
- Los operadores de selección, mutación y cruza.
- El tamaño de la población.
- Los valores de probabilidad con la que se aplican los operadores genéticos.

El pseudocódigo del Algoritmo 1, ilustra el funcionamiento de un algoritmo genético simple.

A continuación, se explican de manera general los requerimientos de los algoritmos genéticos y posteriormente, en la sección 2.3 y en la sección 4, se precisan detalles para cada algoritmo genético implementado en este trabajo de investigación.

```
1 Inicializar la población con candidatos aleatorios de la solución;
2 Evaluar cada candidato;
3 mientras La condicion de paro no se satisfaga hacer
4 |   Paso 1: Seleccionar a los padres;
5 |   Paso 2: Recombinar los pares de padres;
6 |   Paso 3: Mutar a los hijos;
7 |   Paso 4: Evaluar a los nuevos candidatos;
8 |   Paso 5: Seleccionar a los individuos para la siguiente generación;
9 fin
```

**Algoritmo 1:** Pseudocódigo de un algoritmo genético simple.

### 2.2.1. Representación

Este paso consiste en crear un vínculo entre el contexto original del problema y espacio de solución del problema, donde la evolución se lleva a cabo. Esto se realiza definiendo cómo es que las soluciones serán especificadas y guardadas para que puedan ser manipuladas por una computadora.

Los objetos que forman posibles soluciones en el contexto original, son llamados fenotipos; mientras que su codificación en el espacio de solución se denominan genotipos.

### 2.2.2. Función de aptitud

El rol de la función de aptitud es representar los requerimientos a los que se debe adaptar la población. Desde la perspectiva de resolución de problemas, representa la tarea a ser resuelta en el contexto evolutivo.

Técnicamente es una función o procedimiento que le asigna una medida cualitativa a los genotipos.

### 2.2.3. Población

La población forma la unidad básica de evolución, los individuos son objetos estáticos que no cambian o se adaptan, es la población la que lo hace. La diversidad de la población es la que indica las diferentes soluciones

que hay presentes.

### 2.2.4. Mecanismo de selección de padres

Este mecanismo tiene como objetivo distinguir a los mejores individuos para que sean los padres de la siguiente generación, con base en su calidad. Este mecanismo, junto con el mecanismo de selección de sobrevivientes, son los responsables de generar mejoras en la calidad de las soluciones.

### 2.2.5. Recombinación

Este operador es un operador variacional que recombina la información de dos padres en uno o más genotipos descendientes. La recombinación es una operación estocástica, esto significa que cómo se eligen los genes de los padres y cómo se combinan, depende del azar.

El principio detrás de la recombinación es sencillo — al aparear a dos individuos con diferentes pero deseables atributos, se puede obtener una descendencia que combine ambas características.

### 2.2.6. Mutación

Este operador variacional unario, cuando es aplicado a un genotipo, resulta en un mutante ligeramente modificado. El operador de mutación siempre es un operador aleatorio y es el causante de ingresar ”sangre fresca” a la población.

### 2.2.7. Mecanismo de selección de sobrevivientes

Este mecanismo toma la decisión de qué individuos serán los elegidos para pasar a la siguiente población; esta decisión se basa frecuentemente en sus valores de aptitud. Al contrario del mecanismo de selección de padres, que es aleatorio, este mecanismo es frecuentemente determinista.

## 2.3. GA-Nuggets

El GA-Nuggets (*Genetic Algorithm Nuggets*), fue diseñado por **?** para el modelado de dependencias en dos versiones: la primera versión es de una

población centralizada de individuos, donde diferentes individuos pueden representar reglas de predicción para diferentes atributos objetivo. La segunda versión mantiene una población distribuida de muchas sub-poblaciones, donde cada una de ellas evoluciona de manera independiente, pero con la posibilidad de que algunos individuos puedan migrar entre las sub-poblaciones. En esta segunda versión, cada población tiene asociado un atributo objetivo a predecir, es por esto que todos los individuos de cada sub-población representan reglas para predecir el mismo atributo objetivo.

### 2.3.1. Representación de los individuos

Cada individuo representa una regla de predicción candidata de la forma IF $Ant$ THEN $Cons$, donde $Ant$ es el antecedente de la regla y $Cons$ es el consecuente.

El antecedente $Ant$ consiste en una conjunción de condiciones, donde cada condición es un par atributo-valor de la forma $A_i = V_{ij}$, donde $A_i$ es el i-ésimo atributo y $V_{ij}$ es el j-ésimo valor del atributo $A_i$.

Es importante mencionar que el algoritmo solo maneja valores categóricos, por lo que es necesario discretizar los valores.

El consecuente $Cons$ consiste en un solo par atributo-valor de la forma $G_k = V_{kl}$, donde $G_k$ es el k-ésimo atributo objetivo y $V_{kl}$ es el l-ésimo valor del atributo $G_k$. El conjunto de atributos objetivos son seleccionados por el usuario entre todos los atributos del conjunto de datos y el resto de los atributos se utiliza como atributos predictivos.

Un individuo es codificado en una cadena de longitud fija conteniendo $z$ genes, donde $z$ es el número de atributos considerando los atributos predictivos y objetivo.

De los valores de los atributos codificados en el genoma, solo un subconjunto de los valores de los atributos sera decodificado. Para realizar esto, se utiliza un valor que no se encuentre entre los posibles valores de los atributos, por ejemplo $-1$, para indicar que ese gen no sera decodificado en un antecedente para la regla; gracias a esto, a pesar de que la longitud de la cadena sea fija, podemos tener un antecedente de longitud variable.

Una vez que se generó el antecedente de la regla, el algoritmo selecciona el atributo objetivo que incremente la aptitud del individuo.

### 2.3.2. Función de aptitud

La función de aptitud cuyo dominio es entre 0 y 1, consiste de dos partes: la primera mide el grado de interés de la regla, y la segunda mide su certeza de predicción. El grado de interés de la regla se define en dos términos, el primero se refiere al antecedente y el segundo al consecuente de la regla.

El grado de interés de la regla se evalúa con una métrica de teoría de la información, donde el grado de interés para el antecedente de la regla está dado por:

$$AntInt = 1 - \left( \frac{\sum_{i=1}^{n} InfoGain(A_i)/n}{log_2(|dom(G_k)|)} \right), \tag{2.1}$$

donde $n$ es el número de atributos que ocurren en el antecedente de la regla y $|dom(G_k)|$ es la cardinalidad del dominio (es decir, el número de valores posibles) del atributo de destino $G_k$ que se produce en el consecuente. El término *log* se incluye en la fórmula 2.1 para normalizar el valor de $AntInt$, de modo que esta medida tenga un valor entre 0 y 1. El $InfoGain$ viene dado por:

$$InfoGain(A_i) = Info(G_k) - Info(G_k|\, A_i), \tag{2.2}$$

donde,

$$Info(G_i) = -\sum_{i=1}^{mk} \left( Pr(V_{kl}) log_2(Pr(V_{kl})) \right), \tag{2.3}$$

y

$$Info(G_k|\, A_i) = \sum_{i=1}^{n_i} \left( Pr(V_{ij}) \left( -\sum_{j=1}^{mk} Pr\left(V_{kl}|V_{ij}\right) log_2\left(Pr\left(V_{kl}|V_{ij}\right)\right) \right) \right). \tag{2.4}$$

Donde $m_k$ es el número de posibles valores que puede tomar el atributo objetivo $G_k$, $n_i$ es el número de posibles valores para el atributo $A_i$, $Pr(X)$ denota la probabilidad de $X$ y $Pr(X|Y)$ denota la probabilidad de $X$ dado $Y$.

La métrica que proporciona $AntInt$ puede ser justificada porque, en general, dado un atributo predictor $A_i$, cuya ganancia de información sea alta con respecto a $G_k$, nos lleva a considerar que $A_i$ es un buen predictor de $G_k$ cuando se considera individualmente, ignorando sus interacciones con otros atributos predictivos.

Por lo anterior, si se considera que el usuario ya conoce los atributos que son buenos predictores del atributo objetivo $G_k$, estos atributos podrían no parecerle interesantes al usuario.

Adicionalmente, a los atributos que tienen baja ganancia de información y que aparecen en el antecedente de la regla, el usuario podría considerarlos como irrelevantes por sí solos, sin embargo, cuando interaccionan con los otros atributos podrían generar que éstos aumenten su relevancia, lo que los haría interesantes para el usuario.

La computación del grado de interés del consecuente de la regla está basada en la idea de que mientras más raro sea el valor de un atributo objetivo con respecto a un valor común del atributo, más interesante será para el usuario. En otras palabras, mientras mas grande sea la frecuencia relativa en el conjunto de entrenamiento del valor predicho en el consecuente, menos interesante será para el usuario.

Puntualmente, la regla para predecir el grado de interés del consecuente de la regla es:

$$ConsInt = (1 - Pr(G_{kl}))^{1/\beta}, \tag{2.5}$$

donde $Pr(G_{kl})$ es la frecuencia relativa del valor del atributo objetivo $G_{kl}$, y $\beta$ es un parámetro especificado por el usuario.

La segunda parte de la función de aptitud mide la exactitud predictiva de la regla, y está dada por:

$$PredAcc = \frac{|A\&C| - 1/2}{|A|}, \tag{2.6}$$

donde $|A\&C|$ es el número de ejemplos que satisfacen tanto el antecedente como el consecuente de la regla, y $|A|$ es el número de ejemplos que satisfacen solo el antecedente. El término $1/2$ es para penalizar las reglas que cubren pocos ejemplos de entrenamiento.

Finalmente, la función de aptitud sería:

$$Fitness = \frac{w_1(AntInt + ConsInt)/2 + w_2 PredAcc}{w_1 + w_2}, \tag{2.7}$$

cuyo dominio estaria entre 0 y 1, y donde $w_1$ y $w_2$ son pesos definidos por el usuario, con valores positivos mayores a 0.

### 2.3.3. Método de selección y operadores genéticos

GA-Nuggets utiliza el bien conocido "selección por torneo" con un tamaño de torneo de 2, y utiliza una cruza uniforme extendida con un procedimiento de reparación. En la cruza uniforme existe una probabilidad de aplicar cruza a dos individuos y otra probabilidad para intercambiar cada gen. Después de que se realiza la cruza, el algoritmo verifica si se generó un individuo inválido. Si es así, se inicia un procedimiento de reparación para generar individuos validos. El operador de mutación transforma de forma aleatoria el valor de un atributo en otro, dentro del mismo dominio del atributo.

Además de la cruza y la mutación, se agregan otros dos operadores: insertar-condición y eliminar-condición, que controlan el tamaño de las reglas que están siendo evolucionadas, de la siguiente forma: mientras mayor sea el número de condiciones en el antecedente de la regla actual, menor será la probabilidad de aplicar el operador insertar-condición; y este operador no se aplicará si el antecedente ya tiene el máximo número de condiciones especificado por el usuario. En cambio, la probabilidad de aplicar el operador eliminar-condición será mayor mientras menor sea el número de condiciones en el antecedente de la regla; y este operador no se aplicará si el antecedente solo tiene una condición.

## 2.4. Enfoque OCAT (One Clause At a Time)

El enfoque "Una Cláusula A la Vez" u OCAT, por sus siglas en inglés (One Clause At a Time) es un enfoque que trata de resolver dos debilidades con las técnicas de aprendizaje bayesianas y conexionistas como lo son las redes neuronales, estas debilidades son las siguientes:

1. Las formas en que estos métodos funcionan y producen recomendaciones pueden no ser atractivos por los expertos en el dominio.
2. El conjunto de entrenamiento puede no contener suficientes ejemplos como para que garanticen resultados estadisticamente significantes. Como resultado estos métodos pueden no ser fidedignos para aplicaciones reales.

Este enfoque está basado en conceptos de lógica matemática y optimización discreta. El enfoque es de naturaleza *avara*, en el sentido de que busca aceptar a todos los ejemplos positivos y rechazar tantos ejemplos negativos como sea posible. Esto se realiza sucesivamente para cada una de las cláusulas que genera, hasta que ya no tenga más ejemplos negativos para rechazar.

El enfoque OCAT busca generar un conjunto de cláusulas que pueden estar en forma normal conjuntiva (FNC) o forma normal disyuntiva (FND). Es conocido (Peysakh, 1987) que cualquier función Booleana puede ser transformada en forma FNC o FND.

Algunas suposiciones que este enfoque toma son las siguientes:

- Se tienen observaciones que describen el comportamiento del sistema a aprender.
- Que un conjunto de $n$ atributos de estas observaciones describen totalmente el sistema.
- Cada una de las observaciones pertenece a una y solo una de las K clases.
- Que las observaciones están libres de ruido.
- También asume que la clase a la que pertenece la observación es la correcta.

El siguiente pseudocódigo ejemplifica el funcionamiento general del enfoque OCAT.

```
1 i = 0; C = ∅;{Inicializaciones};
2 mientras E⁻ ≠ ∅ hacer
3 |   Paso 1: i ← i + 1;
4 |   Paso 2: Encontrar una cláusula c_i que acepte todos los miembros
  |    de E⁺ mientras rechaza tantos miembros sea posible de E⁻;
5 |   Paso 3: SeaE⁻(c_i) el conjunto de miembros de E⁻ que son
  |    rechazados por c_i;
6 |   Paso 4: Sea C ← C ∧ c_i;
7 |   Paso 5: Sea E⁻ ← E⁻ − E⁻(c_i);
8 fin
```

**Algoritmo 2:** Pseudocódigo de enfoque OCAT para generar cláusulas en forma normal conjuntiva.

### 2.4.1. Heurística RA1

El enfoque heurístico, denominado RA1 por *Randomized Algorithm 1*, fue propuesto en (Deshpande and Triantaphyllou, 1998) y es un algoritmo que selecciona de forma aleatoria dentro los mejores atributos candidatos, tal cual se muestra en el algoritmo 3. De esta manera se evita el estancamiento del algoritmo en un mínimo local.

**Definiciones:**

- $C$ es el conjunto de atributos en la clausula actual.
- $A_k$ es un atributo dado que $A_k \in A$, donde $A$ es el conjunto de todos los atributos $A_1, \ldots, A_k$
- $POS(A_k)$ el numero total de ejemplos positivos en $E^+$ que serán aceptados si el atributo $A_k$ es incluido en la clausula actual.
- $NEG(A_k)$ el numero total de ejemplos negativos en $E^-$ que serán aceptados si el atributo $A_k$ es incluido en la clausula actual.
- $l$ el tamaño de la lista de candidatos.

- $ITRS$ el numero de veces que el procedimiento de formación de clausulas es repetido.

Hacer por $ITRS$ numero de iteraciones;
**mientras** $E^- \neq \emptyset$ **hacer**
  $C = \emptyset$ (inicialización); **mientras** $E^+ \neq \emptyset$ **hacer**
    **Paso 1:** Clasificar en orden descendente todos los atributos $a_i \in a$ (donde $a_i$ es $A_i$ o $\neg A_i$) de acuerdo a su valor $POS(a_i)/NEG(a_i)$. Si $NEG(a_i) = 0$, entonces $POS(a_i)/NEG(_i) = 1000$ (i.e., un valor arbitrariamente alto);
    **Paso 2:** Formar una lista de candidatos de los atributos que tienen los $l$ valores más altos $POS(a_i)/NEG(a_i)$;
    **Paso 3:** Elegir al azar un atributo $a_k$ de la lista de candidatos;
    **Paso 4:** Sea $C \leftarrow C \vee a_k$ el conjunto de atributos en la cláusula actual;
    **Paso 5:** Sea $E^+(a_k)$ el conjunto de miembros de $E^+$ aceptados cuando $a_k$ se incluye en la cláusula FNC actual;
    **Paso 6:** Sea $E+ \leftarrow E^+ - E^+(a_k)$;
    **Paso 7:** Sea $a \leftarrow a - a_k$;
    **Paso 8:** Calcular los nuevos valores $POS(a_i)$ para todos $a_i \in a$;
  **fin**

```
10
11  |  Paso 9: Sea E⁻(C) el conjunto de miembros de E⁻ que son
    |   rechazados por C;
12  |  Paso 10: Sea E⁻ ← E⁻ − E⁻(C);
13  |  Paso 11: Reiniciar E⁺;
14 fin
15 elegir el sistema Booleano final que tenga el menor número de
     cláusulas, de los sistemas ITRS anteriores.
```

**Algoritmo 3:** La heurística RA1 (Deshpande and Triantaphyllou, 1998)

## 2.5. Algoritmo Quine–McCluskey

Este algoritmo es un método utilizado para la simplificación de funciones Booleanas y fue desarrollado por Willard V.Quine 1955 y extendido por Edward J. McCluskey en 1956. Este método involucra dos pasos:

1. Encontrar los implicantes primos de la función.
2. Usar esos implicantes primos para encontrar los implicantes primos esenciales de la función, así como los otros implicantes primos necesarios para realizar la cobertura de la función.

El pseudocódigo a continuación define este algoritmo, que es empleado en la metodología de experimentación para minimizar el conjunto de cláusulas que se obtienen a partir de los

```
1 inicio
2  |  Organizar los términos mínimos en orden ascendente;
3  |  Formar como máximo n + 1 grupos en función del número de
   |   unos presentes en sus representaciones binarias[1];
```

[1]Para $n$ variables Booleanas en una función Booleana, o para $n$ bits en su equivalente binario de términos mínimos.

```
14
15    inicio
16        Obtener los implicantes primos:
17        mientras no se tengan todos los implicantes primos hacer
18            comparar los términos mínimos presentes en grupos
               sucesivos
              si existe un cambio en la posición de un solo bit
              entonces
19                Tomar ese par de dos términos mínimos;
20                Colocar el símbolo "_" en la posición del bit de cambio;
21                Mantener el resto de los bits tal como están;
22            fin
23        fin
24    fin
25    inicio
26        Formular la tabla de primos implicantes, que consiste en un
           conjunto de filas y columnas:
          inicio
27            Los implicantes primos se colocan en sentido de las filas;
28            Los términos mínimos se colocan en sentido de las
               columnas;
29            Colocar un ″1″ en las celdas correspondientes a los
               términos mínimos que se cubren en cada implicante
               primo;
30        fin
31    fin
32    inicio
33        Encontrar los implicantes primos esenciales:
          para todo término mínimo de la función Booleana hacer
34            observar cada columna de la tabla de primos implicantes
               si el término mínimo está cubierto solo por un implicante
               primo entonces
35                este término es un implicante esencial y formará parte
                   de la función Booleana simplificada
36            fin
37            eliminar la fila de cada implicante primo esencial y las
               columnas correspondientes a los términos mínimos que
               están cubiertos en ese implicante primo esencial
38        fin
39    fin
40 fin
```

**Algoritmo 4:** Método Tabular Quine-McCluskey: para simplificar funciones Booleanas.

# Capítulo 3

# Estado del Arte

De los trabajos que existen en la actualidad, podemos mencionar tres que resultan de especial interés para este trabajo de investigación. Las tres metodologías de nuestro interés son las siguientes:

- Búsqueda de reglas locales para un autómata celular mediante el análisis estadístico de las observaciones de un fenómeno.
- Configuración de reglas locales para un autómata celular mediante el empleo de algoritmos evolutivos.
- Búsqueda de reglas locales para un autómata celular mediante el empleo de algoritmos evolutivos para satisfacer un patrón específico.

Estos son algunos de los enfoques para emplear los autómatas celulares como herramienta para modelar fenómenos, que si bien tienen relación con este trabajo de tesis en el aspecto de buscar configurar o encontrar reglas de evolución para un autómata celular, no se conectan directamente con él debido a que emplean otros enfoques como las tres metodologías que se mencionaron previamente.

## 3.1. Análisis estadístico para el descubrimiento de reglas

Uno de los enfoques como se ve en (Kawaharada et al., 2016), se centra en descubrir reglas de evolución para un autómata celular unidimensional,

empleando métodos de análisis estadísticos para determinar dichas reglas, a partir de los datos observados de los patrones de bioconvección que son generados como resultado de la respuesta a la estimulación luminosa de un tipo de alga unicelular llamada *Euglena gracilis*.

El experimento consistió en iluminar por la parte de abajo de un contenedor anular conteniendo una suspensión de E. gracilis con una fuerte intensidad de luz. Los individuos, al ser alcanzados por la luz, sufren de fototaxis negativa, lo cual provoca que se acumulen cerca de la superficie.

Como la densidad del E. gracilis es mas pesada que el agua, las regiones ricas en esta alga se precipitan al fondo para conducir el flujo local. Estas interacciones entre los individuos y el flujo eventualmente forma patrones de bioconvección.

El proceso se realizó de la siguiente forma:

1. Recopilación de imágenes de los patrones de bioconvección.
2. Transformación de las imágenes recopiladas a escala de grises.
3. Uso de filtros para la eliminación del ruido.
4. Discretización de las imágenes en escala de grises.
5. Se predetermina el número de estados para el sitio $k$ y se discretiza la información observada acorde a esto.
6. Basado en el número de vecinos establecido $m$ se calcula la frecuencia con la que aparecen los estados por cada combinación de posibles estados vecinos.
7. Con base en esta frecuencia se determinan las reglas locales para el Autómata Celular (AC).

Como se puede observar en los pasos *6* y *7* del proceso anterior, la determinación de las reglas locales del AC se realiza por medio de la frecuencia en la que aparecen los estados y resulta en reglas locales especializadas que definen su comportamiento; además, es importante mencionar que éste comportamiento está establecido de acuerdo a los datos que se utilizaron para obtener las reglas.

El algoritmo que se propone en este trabajo de tesis logra realizar una búsqueda más robusta en el espacio de reglas, para obtener un conjunto de reglas con un comportamiento generalizado que se encuentra establecido en los datos.

## 3.2. Configuración de reglas para un AC con algoritmos evolutivos

En Naghibi and Delavar (2016) se muestra un método para descubrir las reglas de transición de un autómata celular de lattice bidimensional que modele el crecimiento urbano empleando un algoritmo de optimización de colonia de abejas artificiales. Para ello, se utilizan como datos de entrada información multitemporal sensada remotamente de el área urbana de la ciudad de Urmia, Irán.

El autómata celular en este trabajo se define con la siguiente ecuación:

$$P_{ij}^{t} = S_{ij}^{t} \times \Omega_{ij}^{t} \times Con \times e_r, \tag{3.1}$$

donde:

- $P_{ij}^{t}$ es el potencial de desarrollo de una celda $ij$.
- $S_{ij}^{t}$ es la idoneidad de la celda $ij$ para cambiar basada en factores relevantes en el tiempo $t$.
- $\Omega_{ij}^{t}$ es el efecto que tiene la densidad de desarrollo del vecindario.
- $Con$ es una condición funcional que se vuelve verdadero cuando se encuentra la idoneidad de la celda.
- $e_r$ es un término para la perturbación estocástica por errores.

El potencial de desarrollo se compara con un valor umbral para determinar si una célula no urbanizada va a poder ser transformada en una célula urbanizada en el tiempo $t + 1$. Entonces, los valores a encontrar son los valores de umbral para los cuales se produce la transición de una célula no urbanizada a una urbanizada.

Como requerimiento para la evaluación del algoritmo, se emplean otras dos técnicas además de la colonia de abejas artificiales, las cuales son:

- Particle Swarm Optimization (PSO)

- Logistic Regression

La figura 3.1 muestra el procedimiento que se siguió para la calibración de las reglas del autómata celular, utilizando un algoritmo de optimización de colonia de abejas artificiales.

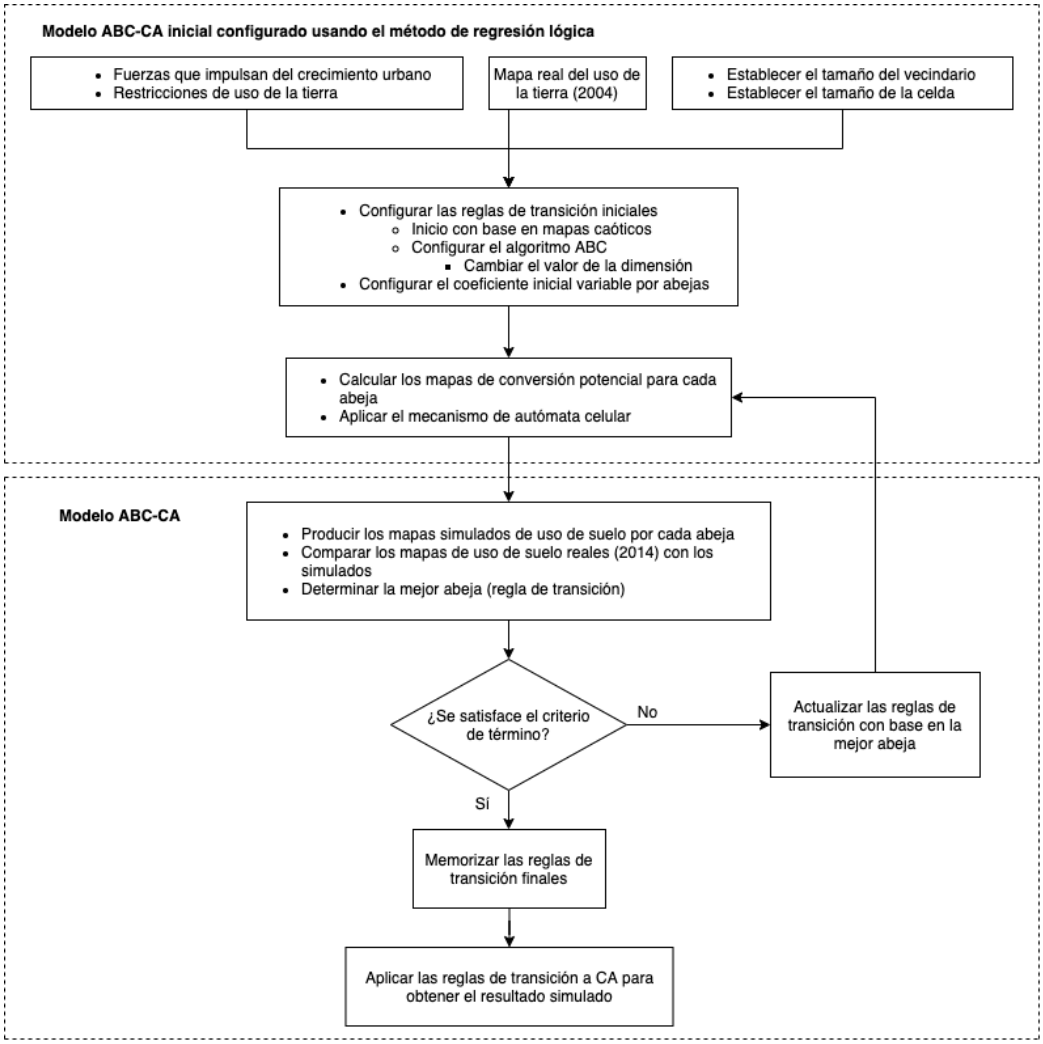



Figura 3.1: Modelo para la obtención de reglas del autómata celular empleando un algoritmo de optimización de colonia de abejas artificiales. Diagrama traducido de Naghibi and Delavar (2016).

Como vemos en la figura 3.2 y en la figura 3.1, son similares los procesos si no tomamos en cuenta las particularidades de cada algoritmo, un proceso de simulación, uno de evaluación, y por ultimo un proceso de actualización, son las partes que identifican un proceso general de aprendizaje.

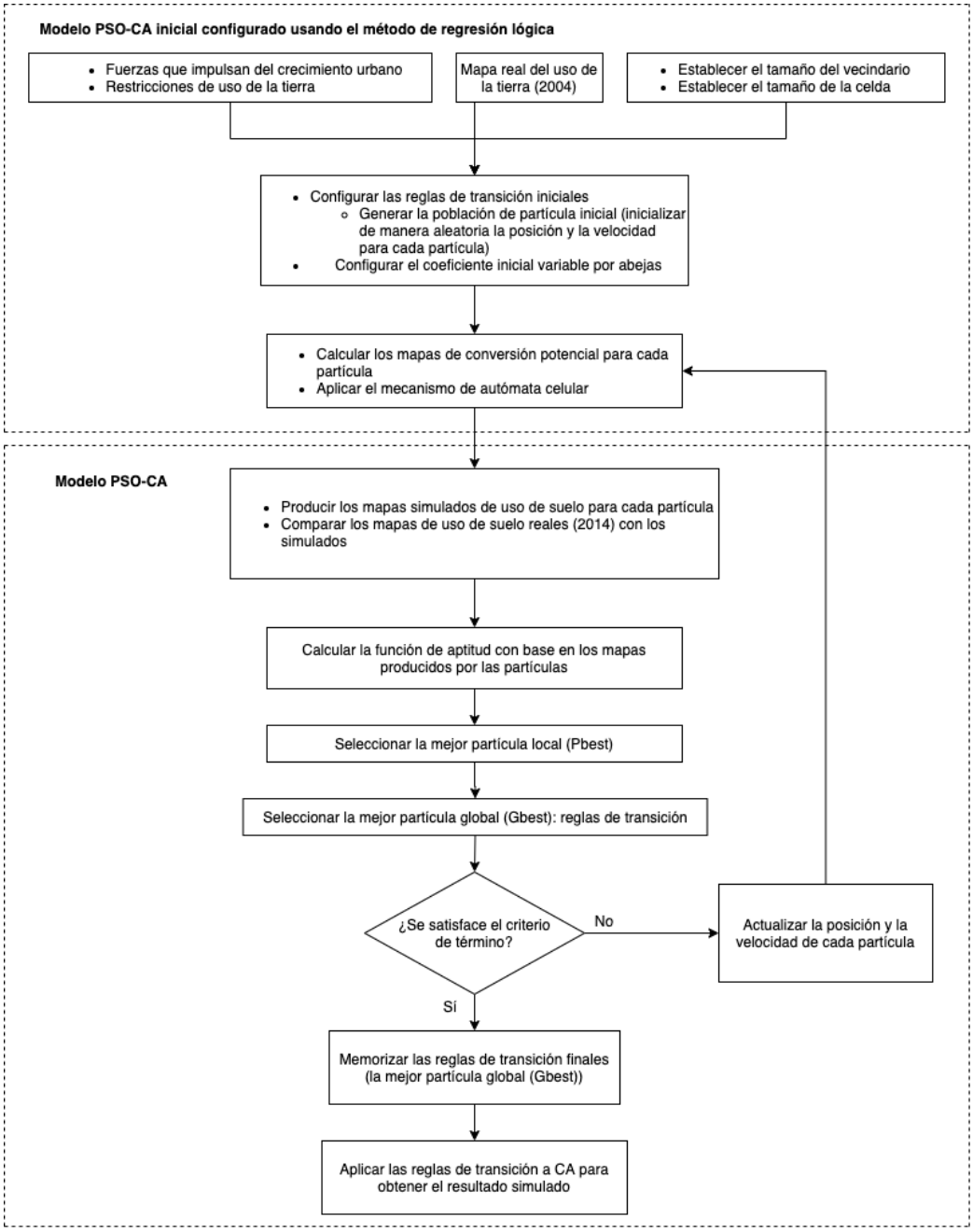



Figura 3.2: Modelo para la obtención de reglas del autómata celular empleando un algoritmo de optimización de enjambre de partículas. Diagrama traducido de Naghibi and Delavar (2016).

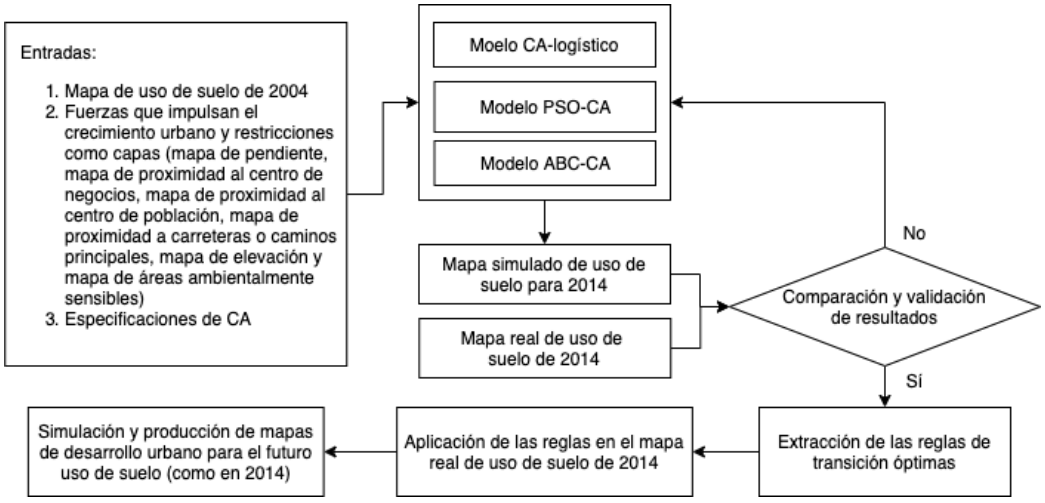



Figura 3.3: Modelo de evaluación de los algoritmos. Diagrama traducido de Naghibi and Delavar (2016).

En la figura 3.3 vemos el proceso que se siguió para realizar la evaluación de los 3 algoritmos, simulando cada uno por separado, después comparando el resultado de la simulación con la realidad y entonces obteniendo el porcentaje de exactitud de la simulación.

| **Validation Method** | **Prediction Approach** | | |
|---|---|---|---|
| | **CA-Logistic** | **PSO-CA** | **ABC-CA** |
| Overall accuracy (%) | 82.8 | 87.5 | 89 |
| Figure of merit (%) | 30 | 32.6 | 35.7 |

| | | | |
|---|---|---|---|
| False alarms (%) | 15.1 | 7.7 | 6.2 |
| Misses (%) | 2.1 | 4.8 | 4.8 |
| Allocation disagreement (%) | 17.2 | 12.5 | 11 |
| Correctly predicted unchanged cells (%) | 75.4 | 81.4 | 82.9 |
| Protection of agricultural areas from urbanization (%) | 62.2 | 68.6 | 74.1 |
| Areas of the simulated gain of the urban lands in 2004-2014 (the actual gain area of the city is 2500 hectares) (hectares) | 4960 | 3155 | 2824 |
| TOC (closeness to maximum boundary) | low | medium | high |

Cuadro 3.1: Resultados de la evaluación de los algoritmos. Tabla obtenida de Naghibi and Delavar (2016).

Como podemos ver en la tabla de resultados 3.1, el algoritmo de colonia de abejas fue capaz de obtener un mejor rendimiento en comparación con los otros métodos. Esto ayudará a obtener un mejor despeño en la estimación correcta del crecimiento urbano. Sin embargo, el método aquí planteado sigue teniendo ciertos obstáculos como lo es encontrar una correcta definición de la ecuación que determine al autómata celular, por ejemplo la ecuación 3.1 que se usa en este trabajo.

## 3.3. Búsqueda de reglas de evolución para replicar estructuras

El aporte principal de Bidlo (2016) es la creación de un método para la obtención de un conjunto de reglas que puedan replicar un comportamiento específico. Sin embargo, es importante resaltar que no se toma en cuenta conocimiento previo del fenómeno que se quiere replicar. Lo que se utiliza es

un algoritmo evolutivo, cuya función de aptitud es dependiente del resultado al que se quiere llegar. La codificación de las reglas viene dada de la siguiente forma:

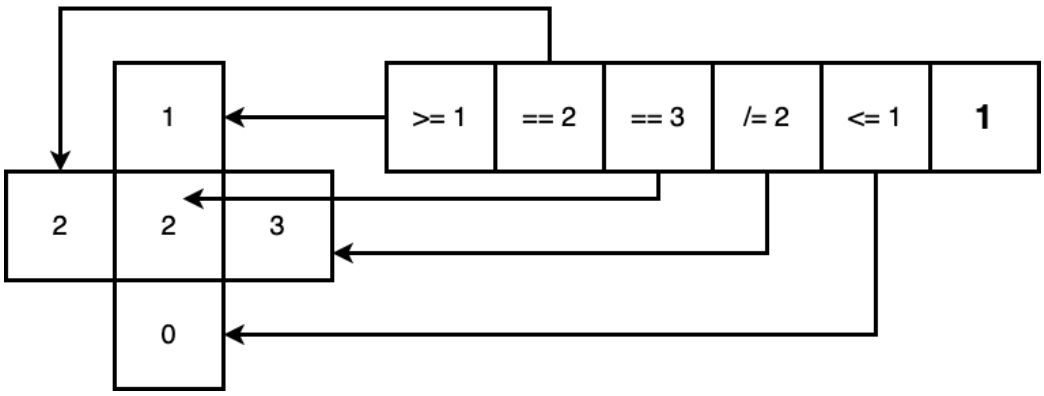



Figura 3.4: Ejemplo de la codificación de las reglas de evolución del autómata con un vecindario de tipo Moore. Figura obtenida de Bidlo (2016).

De este artículo (Bidlo, 2016) encontramos especialmente útil la forma en que se codificaron las reglas, esto sirvió como inspiración, para el diseño de la representación de las reglas en el algoritmo propuesto en este trabajo.

# Capítulo 4

# Propuesta de solución: LRDEA (Local Rule Discovery Evolutive Algorithm)

La propuesta de solución para el problema, se basa en un algoritmo genético distribuido cuyo funcionamiento combina el principio del algoritmo RA1 —que equivale a ir generando cláusulas que cubran a los ejemplos positivos y rechace a los ejemplos negativos—, y el principio de un algoritmo genético —que implica tener recombinación entre un conjunto de individuos para explorar el espacio de búsqueda—. El algoritmo propuesto maneja una subpoblación por cada uno de los valores en el espacio de estado del autómata celular.

## 4.1. Representación

Al igual que el algoritmo GA-Nuggets, la codificación para cada individuo representa una regla de predicción candidata de la forma IF *Ant* THEN *Cons*, donde *Ant* es el antecedente de la regla y *Cons* es el consecuente. El antecedente *Ant* consiste en una conjunción de condiciones, donde cada condición es un par vecino-valor de la forma $n_i = V_i$, donde $n_i$ es el i-ésimo vecino y $V_i$ es un conjunto de los posibles valores que puede tener $n_i$ de la forma $\{v_0, \ldots, v_n\}$ donde $v$ es un valor dentro del dominio del vecino $n_i$.

El algoritmo solo maneja valores categóricos, por lo que es necesario dis-

cretizar los valores. El consecuente *Cons* consiste en un solo valor en el dominio que puede tomar la celda que se esta evaluando $c$. Se utiliza el valor -1 para indicar que el par vecino-valor no va a formar parte de la regla.

$$\begin{array}{ccc} n_1 & n_2 & n_3 \\ n_4 & c & n_5 \quad \rightarrow \text{Cons} \\ n_6 & n_7 & n_8 \end{array}$$

Cuadro 4.1: Representación del vecindario.

Finalmente, conviene mostrar una representación de la codificación de un vecindario como el propuesto en el Cuadro 4.1.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $n_1 = V_1$ | $n_2 = V_2$ | $n_3 = V_3$ | $n_4 = V_4$ | $c = V_c$ | $n_5 = V_5$ | $n_6 = V_6$ | $n_7 = V_7$ | $n_8 = V_8$ | Cons |

Cuadro 4.2: Representación de la codificación.

## 4.2. Función de aptitud

La función de aptitud propuesta se define por la siguiente ecuación:

$$Fitness(C, E^+) = \sum_{i=1}^{|E^+|} \left( \frac{1}{|C|} \sum_{j=1}^{|C|} Eq(E_{ij}^+, C_j) \right) \tag{4.1}$$

$$Eq(x, y) = \begin{cases} 1 & S\ x = y \\ 0 & \text{de otra manera} \end{cases} \tag{4.2}$$

donde:

- $C$ es el cromosoma.
- $|C|$ es la cardinalidad del cromosoma.
- $E^+$ es un subconjunto de los ejemplos que pertenece a la clase para la cual se esta evaluando la aptitud del cromosoma.

- $|E^+|$ es la cardinalidad de $E^+$.
- $E^+_{i,j}$ es un valor en la posición j del elemento en la posicion i de $E^+$.
- $C_j$ es el valor del cromosoma en la posición j.

## 4.3. Mecanismo de selección de padres

Para el mecanismo de selección de padres se eligió la *selección por torneo*, cuyo pseudocódigo se muestra en el algoritmo 5:

**entrada:** k (la cantidad de individuos participantes en el torneo)
1 **Paso 1:** Escoger k individuos de la población aleatoriamente;
2 **Paso 2:** Escoger el individuo más apto del torneo con probabilidad p;
3 **Paso 3:** Escoger el segundo individuo más apto con probabilidad $p(1-p)$;
4 **Paso 4:** Escoger el k-esimo individuo más apto con probabilidad $p(1-p)^k$;

**Algoritmo 5:** Pseudocódigo de selección por torneo.

## 4.4. Cruza

Para el proceso de cruce se seleccionó el método de *cruce en un punto* y se lleva a cabo como se muestra en el algoritmo 6.

**entrada:** padre1, padre2
1 **Paso 1:** Escoger de forma aleatoria el punto p donde p es menor a la cardinalidad de padre1 y padre2;
2 **Paso 2:** Partir el cromosoma del padre1 y el padre2 en el punto p;
3 **Paso 3:** Recombinar la primera mitad del padre1 con la segunda mitad del padre2 para generar al hijo1;
4 **Paso 4:** Recombinar la segunda mitad del padre1 con la primera mitad del padre2 para generar al hijo2;

**Algoritmo 6:** Pseudocódigo de cruce en un punto.

## 4.5. Mutación

El operador de mutación consiste en dos partes, la primera es la probabilidad de realizar la mutación o no, esta probabilidad es un parámetro definido por el usuario, y la segunda es la selección del gen que se va a mutar, para la selección primero se decide con base en el numero de atributos con valor si se va agregar o se va a remover un valor, si se va a agregar se ordenan los genes de menor a mayor numero de valores, y entonces se selecciona de forma aleatoria proporcional al numero de valores, se se va a eliminar se ordenan de mayor a menor y se selecciona de forma aleatoria proporcional al numero de valores.

Finalmente, el gen mutado puede adquirir un nuevo valor en el conjunto de valores o puede removerse uno de estos valores.

## 4.6. Mecanismo de selección de sobrevivientes

Para el mecanismo de selección de sobrevivientes es necesario ordenar los individuos de cada subpoblación y se dividen en dos; se procede a tomar los primeros $n$ elementos de cada una de las mitades, y se descarta el resto.

# Capítulo 5

# Experimentación

En esta sección se detalla cuál fue la metodología empleada para llevar a cabo los experimentos y se detalla el algoritmo propuesto como solución. En la figura 5.1 a continuación, se puede ver el panorama general de dicha metodología experimental.

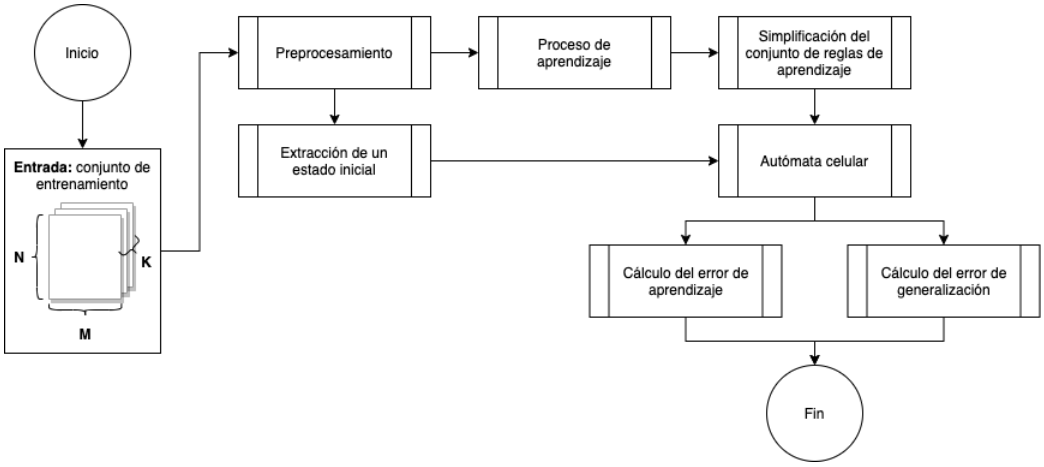


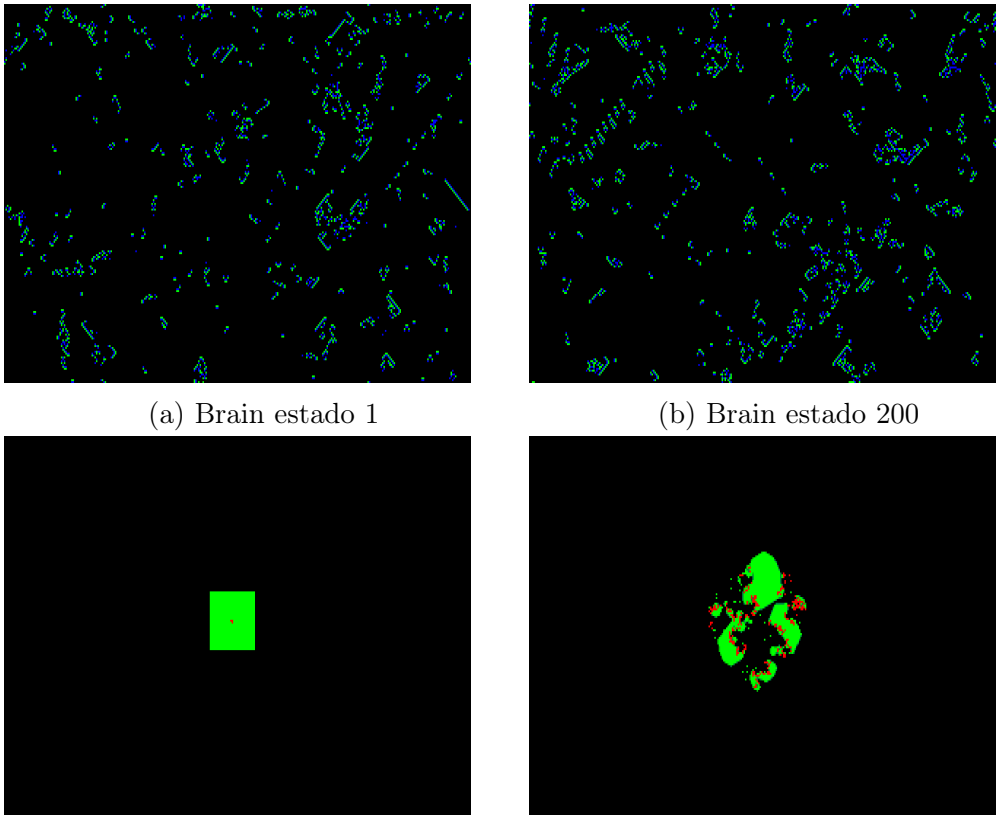

Figura 5.1: Diagrama general de la metodología.

## 5.1. Conjunto de datos

Para realizar los experimentos, se optó por integrar un conjunto de datos que consiste en los estados de la evolución de cuatro diferentes autómatas

celulares bidimensionales que fueron obtenidos de Rucker and Walker (2020).

Para cada uno de los autómatas se obtuvieron 200 imágenes de 50x50 pixeles en escala RGB, que se ejemplifican en las imágenes de la Figura 5.2 a continuación.



(a) Brain estado 1

(b) Brain estado 200

(c) Mite estado 1

(d) Mite estado 200

Figura 5.2: Ejemplo del conjunto de datos

A continuación, se mencionan los autómatas celulares en los cuales se implementaron este tipo de imágenes.

### 5.1.1. Brain

Es un autómata celular bidimensional, desarrollado por Brian Silverman, que modela la evolución de conjuntos de neuronas y sus interacciones. En

otras palabras, el AC Brain modela cierto nivel de actividad cerebral en escenarios simples y consiste en lo siguiente:

- **Vecindario:** es de tipo Moore con 8 vecinos.
- **Espacio de estados:** 3 estados.
  - 0 que representa el estado apagado
  - 1 que representa el estado encendido
  - 2 que representa el estado muriendo.
- **Función de evolución:**
  - 1 si se encuentra apagado y si dos o mas vecinos se encuentran encendidos.
  - 2 si en el estado anterior estaba encendido.
  - 0 si en el estado anterior estaba muriendo.

### 5.1.2. Byl

El AC Byl (Byl, 1989) es un autómata celular bidimensional que fue desarrollado por Jhon Byl en 1989 con la capacidad de autoreplicarse. Este AC consiste en lo siguiente:

- **Vecindario:** es de tipo Von Neumann con 5 vecinos, en el cuadro 5.2 podemos ver como se codifica el vecindario.
- **Espacio de estados:** consiste de 6 estados del 0 al 5.
- **Función de evolución:** se muestra en el cuadro 5.1.

| CTRBL → I | | CTRBL → I | | CTRBL → I | | CTRBL → I | | CTRBL → I | |
|---|---|---|---|---|---|---|---|---|---|
| 00003 | 1 | 10003 | 3 | 20215 | 5 | 31215 | 1 | 40242 | 4 |
| 00012 | 2 | 10004 | 0 | 20235 | 3 | 31223 | 1 | 40252 | 0 |
| 00013 | 1 | 10033 | 0 | 20252 | 5 | 31233 | 1 | 40325 | 5 |
| 00015 | 2 | 10043 | 1 | 2 - - - - | 2 | 31235 | 5 | 4 - - - - | 3 |
| 00025 | 5 | 10321 | 3 | 30001 | 0 | 31432 | 1 | 50022 | 5 |
| 00031 | 5 | 11253 | 1 | 30003 | 0 | 31452 | 5 | 50032 | 5 |
| 00032 | 3 | 12453 | 3 | 30011 | 0 | 3 - - - - | 3 | 50212 | 4 |
| 00042 | 2 | 1 - - - - | 4 | 30012 | 1 | | | 50222 | 0 |
| 0 - - - - | 0 | 20000 | 0 | 30121 | 1 | 40003 | 0 | 50322 | 0 |
| | | 20015 | 5 | 30123 | 1 | 40043 | 0 | 5 - - - - | 2 |
| 10000 | 0 | 20022 | 0 | 31122 | 1 | 40212 | 0 | | |
| 10001 | 0 | 20202 | 0 | 31123 | 1 | 40232 | 0 | | |

Cuadro 5.1: Función de evolución de Byl CA (Byl, 1989).

| | | |
|---|---|---|
| | T | |
| L | C | R → I |
| | B | |

Cuadro 5.2: Codificación del vecindario.

### 5.1.3. Evoloops

El AC Evoloops (Sayama, 1998) es un autómata celular bidimensional, desarrollado por Hiroki Sayama para autoreplicarse, consiste en lo siguiente:

- **Vecindario:** es de tipo Von Neumann con 5 vecinos, en el cuadro 5.2 podemos ver como se codifica el vecindario.
- **Espacio de estados:** consiste de 8 estados del 0 al 7.

- **Función de evolución:** se muestra en el cuadro 5.3.

| CTRBL → I | | CTRBL → I | | CTRBL → I | | CTRBL → I | | CTRBL → I | | CTRBL → I | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 00001 | 2 | 10202 | 1 | 11272 | 7 | 20172 | 2 | 21322 | 2 | 40125 | 0 |
| 00004 | 3 | 10211 | 1 | 11273 | 5 | 20202 | 2 | 21422 | 2 | 40162 | 0 |
| 00012 | 2 | 10212 | 1 | 11322 | 1 | 20203 | 2 | 21622 | 2 | 40212 | 0 |
| 00015 | 2 | 10213 | 1 | 11332 | 1 | 20205 | 2 | 21722 | 2 | 40215 | 0 |
| 00021 | 2 | 10221 | 1 | 11542 | 4 | 20206 | 5 | 22224 | 2 | 40222 | 1 |
| 00024 | 2 | 10224 | 4 | 11572 | 7 | 20207 | 3 | 22227 | 2 | 40232 | 1 |
| 00042 | 2 | 10227 | 7 | 11624 | 4 | 20212 | 2 | 22234 | 2 | 40262 | 6 |
| 00045 | 2 | 10232 | 4 | 11627 | 7 | 20215 | 2 | 22237 | 2 | 40312 | 0 |
| 00075 | 2 | 10241 | 4 | 12224 | 4 | 20221 | 2 | 22243 | 2 | 40322 | 1 |
| 00102 | 2 | 10242 | 4 | 12227 | 7 | 20222 | 2 | 22244 | 2 | 50002 | 5 |
| 00214 | 1 | 10243 | 4 | 12243 | 4 | 20223 | 2 | 22273 | 2 | 50012 | 5 |
| 00217 | 1 | 10251 | 1 | 12273 | 7 | 20232 | 3 | 22277 | 2 | 50021 | 5 |
| 00232 | 2 | 10252 | 7 | 12324 | 4 | 20242 | 2 | 22324 | 3 | 50023 | 2 |
| 01122 | 1 | 10254 | 3 | 12327 | 7 | 20245 | 2 | 22327 | 3 | 50024 | 5 |
| 01212 | 1 | 10257 | 7 | 12426 | 6 | 20252 | 5 | 30001 | 3 | 50027 | 5 |
| 01232 | 1 | 10271 | 7 | 12433 | 3 | 20262 | 0 | 30002 | 2 | 50042 | 5 |
| 01242 | 1 | 10272 | 7 | 12627 | 6 | 20265 | 0 | 30003 | 2 | 50072 | 5 |
| 01245 | 1 | 10273 | 5 | 20001 | 2 | 20272 | 2 | 30004 | 3 | 50202 | 2 |
| 01252 | 6 | 10512 | 1 | 20002 | 2 | 20275 | 2 | 30007 | 4 | 50205 | 2 |
| 01262 | 6 | 10542 | 4 | 20004 | 2 | 20312 | 2 | 30012 | 3 | 50212 | 5 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 01272 | 1 | 10572 | 7 | 20005 | 2 | 20322 | 2 | 30032 | 2 | 50215 | 2 |
| 01275 | 1 | 10621 | 1 | 20006 | 0 | 20342 | 2 | 30042 | 1 | 50242 | 5 |
| 01342 | 1 | 10624 | 4 | 20007 | 1 | 20345 | 2 | 30102 | 1 | 50272 | 5 |
| 01372 | 1 | 10627 | 7 | 20012 | 2 | 20372 | 2 | 30125 | 0 | 50312 | 0 |
| 01422 | 1 | 11112 | 1 | 20015 | 2 | 20412 | 2 | 30212 | 3 | 60202 | 2 |
| 01425 | 1 | 11122 | 1 | 20021 | 2 | 20422 | 2 | 30242 | 3 | 60212 | 2 |
| 01432 | 1 | 11124 | 4 | 20022 | 2 | 20442 | 2 | 30252 | 1 | 60222 | 0 |
| 01435 | 1 | 11125 | 1 | 20023 | 2 | 20512 | 2 | 30272 | 3 | 60242 | 2 |
| 01442 | 1 | 11127 | 7 | 20024 | 2 | 20542 | 5 | 30332 | 1 | 60272 | 2 |
| 01462 | 1 | 11162 | 1 | 20026 | 0 | 20572 | 5 | 31212 | 3 | 61222 | 0 |
| 01722 | 1 | 11212 | 1 | 20027 | 2 | 20612 | 5 | 31242 | 3 | 62224 | 0 |
| 01725 | 1 | 11213 | 1 | 20032 | 4 | 20621 | 2 | 31252 | 1 | 62227 | 0 |
| 01756 | 1 | 11215 | 1 | 20042 | 3 | 20642 | 5 | 31272 | 3 | 70102 | 0 |
| 01762 | 1 | 11222 | 1 | 20045 | 2 | 20672 | 5 | 32424 | 3 | 70112 | 0 |
| 01772 | 1 | 11224 | 4 | 20054 | 5 | 20712 | 2 | 32425 | 1 | 70122 | 0 |
| 10001 | 1 | 11227 | 7 | 20057 | 5 | 20722 | 2 | 32427 | 3 | 70125 | 0 |
| 10012 | 1 | 11232 | 1 | 20062 | 0 | 20772 | 2 | 32527 | 1 | 70162 | 0 |
| 10021 | 1 | 11242 | 4 | 20072 | 2 | 21122 | 2 | 32727 | 3 | 70212 | 0 |
| 10024 | 4 | 11243 | 4 | 20075 | 2 | 21222 | 2 | 40000 | 1 | 70215 | 0 |
| 10027 | 7 | 11252 | 7 | 20102 | 2 | 21223 | 2 | 40002 | 1 | 70222 | 1 |
| 10121 | 1 | 11254 | 3 | 20112 | 2 | 21224 | 2 | 40102 | 0 | 70232 | 0 |
| 10124 | 4 | 11257 | 7 | 20122 | 2 | 21227 | 2 | 40112 | 0 | 70262 | 6 |
| 10127 | 7 | 11262 | 6 | 20142 | 2 | 21232 | 3 | 40122 | 0 | 70312 | 0 |

Cuadro 5.3: Función de evolución de Evoloops CA (Sayama, 1998).

### 5.1.4. Mite

El AC Mite es un autómata celular bidimensional, desarrollado por Dan Drake en 1990, que modela el sistema presa-depredador y consiste en lo siguiente:

- **Vecindario:** es de tipo Moore con 8 vecinos.
- **Espacio de estados:** consiste de 3 estados del 0 al 2.
- **Función de evolución:**
  - mover la posición del predador de manera aleatoria dentro de su vecindario local.

- si no hay presa en su posición el predador muere.

- si hay suficientes presas en su posición se incrementa el número de predadores.

- si hay dos presas juntas se incrementa el número de presas.

## 5.2. Preprocesamiento

Una vez adquiridas las secuencias de imágenes de cada autómata celular, se continúa con los procesos de discretización y binarización que se explican en las subsecciones siguientes.

En términos generales, la discretización consta en pasar las imágenes de los canales RGB a valores discretos, mientras que el proceso de binarización solo se realiza para los datos que se van a ingresar al algoritmo RA1. De cualquier manera, se mencionan a continuación algunas características específicas de cada proceso.

### 5.2.1. Discretización

La implementación de este procedimiento se realizó en el lenguaje Python y se ejemplifica con el siguiente pseudocódigo.

```
   entrada: La ruta de la carpeta donde se encuentran las imágenes
   salida  : Un archivo con formato .pkl que contiene las imágenes
             procesadas.
 1 diccionario = {} ; contador = 0; imágenes=[ ] ;
    /* inicializaciones */
 2 para cada Imagen en carpeta hacer
 3 |   Paso 1:Imagen' = Transformar imagen a escala de grises;
   |   /* Se considera a la imagen como una matriz de pixeles
   |      */
 4 |   nuevaImagen = [ ];
 5 |   para cada fila en Imagen hacer
   |   |   /* una fila es un arreglo de pixeles              */
 6 |   |   nuevaFila = [ ] ;
 7 |   |   para cada pixel en fila hacer
   |   |   |   /* un pixel corresponde a un valor entre 0 y 255
   |   |   |      despues de la transformación               */
 8 |   |   |   si pixel no esta en diccionario entonces
 9 |   |   |   |   Paso 2: diccionarion[pixel] = contador;
10 |   |   |   |   Paso 3: contador = contador + 1;
11 |   |   |   fin
12 |   |   |   Paso 4: nuevaFila.agregar(diccionario[pixel]);
13 |   |   fin
14 |   |   Paso 5: nuevaImagen.agregar(nuevaFila);
15 |   fin
16 |   Paso 6: imagenes.agregar(nuevaImagen);
17 fin
18 Paso 7: Guardar imágenes en formato .pkl;
```

**Algoritmo 7:** Pseudocódigo para la discretización de las imágenes.

### 5.2.2. Binarización

Este proceso se realiza solamente para los datos que ingresarán al algoritmo RA1 debido a que este algoritmo solo es capaz de aprender de datos categóricos. El siguiente pseudocódigo ejemplifica este proceso.

```
   entrada: MEstado: matriz de estado
   salida  : El estado binarizado
 1 nuevoEstado = [];
 2 para cada fila en MEstado hacer
 3 |   nuevaFila = [];
 4 |   para cada celda en fila hacer
 5 |   |   Paso 1: Encontrar el dominio de la celda;
 6 |   |   Paso 2: dominio = ordenarAscendente (dominio);
 7 |   |   para cada valor en dominio hacer
 8 |   |   |   si celda <valor entonces
 9 |   |   |   |   Paso 3a: nuevaFila.agregar(1);
10 |   |   |   en otro caso
11 |   |   |   |   Paso 3b: nuevaFila.agregar(0);
12 |   |   |   fin
13 |   |   fin
14 |   fin
15 |   Paso 4: nuevoEstado.agregar(nuevaFila);
16 fin
```

**Algoritmo 8:** Pseudocódigo para la binarización de las imágenes.

## 5.3. Algoritmos de aprendizaje

Los algoritmos de aprendizaje utilizados en este trabajo de investigación son los siguientes:

- RA1
- GA-Nuggets
- LRDEA el algoritmo diseñado en este trabajo.

Cabe mencionar que, debido a que no se encontraron las implementaciones de los algoritmos RA1 y GA-Nuggets, fue necesario realizarlas en el lenguaje de programación Python.

## 5.4. Simplificación de reglas

El proceso de simplificación de reglas se divide en dos partes:

1. remover redundancia en las cláusulas.

2. emplear el algoritmo Quine–McCluskey para minimizar las cláusulas.

El primer método de simplificación sirve para eliminar términos que no aportan aportan nueva información, esto es por que otros términos contienen a estos. En el siguiente ejemplo podemos ver como es este proceso.

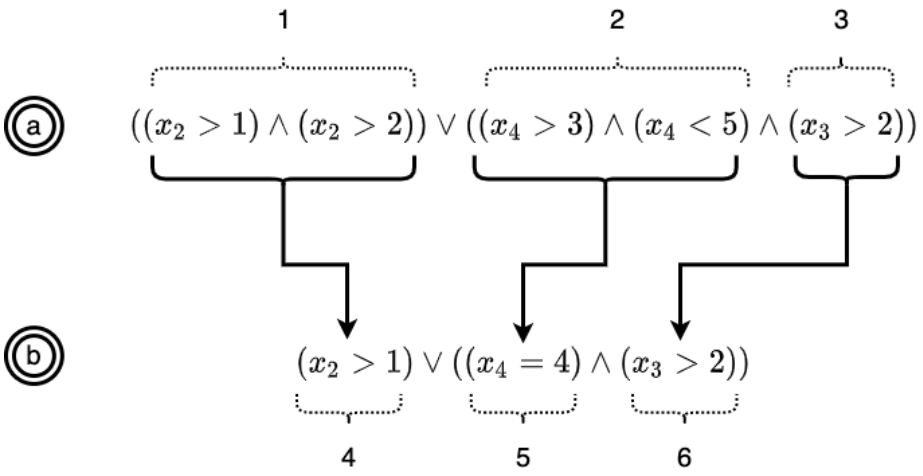



Figura 5.3: Ejemplo del proceso de simplificación 1.

En el diagrama de la figura 5.3 vemos cómo, en la cláusula *a*, los términos 1 y 2 pueden ser reducidos a los términos 4 y 5 de la cláusula *b*, respectivamente. Asimismo, podemos ver cómo el término 3 de *a* no se puede reducir, por lo que pasa a *b* directamente.

Para la simplificación 2 de las reglas se utiliza el algoritmo desarrollado por Willard V.Quine 1955 y extendido Edward J. McCluskey en 1956. Este mecanismo solo se emplea para la simplificación de las reglas generadas por el algoritmo RA1.

## 5.5. Evaluación

En el proceso de evaluación se emplea la validación *camina hacia adelante* o *Walk-Forward*, que se ejemplifica en el Algoritmo 9.

```
   /* el rango va del 80% del total de datos al 100%      */
1  para cada i en rango(160,200) hacer
       /* tomamos un subconjunto del conjunto de datos     */
2      Paso 1: entrenamiento = datos[0:(i-1)];
       /* tomamos un elemento adicional del conjunto de datos
          */
3      Paso 2: testing = datos[i] ;
       /* aprendemos del conjunto de entrenamiento         */
4      Paso 3: aprender(entrenamiento);
5      Paso 4: vExactitud = exactitud(entrenamiento[i-1]);
6      Paso 5: tExactitud = exactitud(testing);
7  fin
```

**Algoritmo 9:** Pseudocódigo de validación de caminata hacia adelante (Walk-Forward).

La toma de los subconjuntos de datos se representa en la figura 5.4, donde podemos observar la dinámica de ”caminar” hacia adelante.

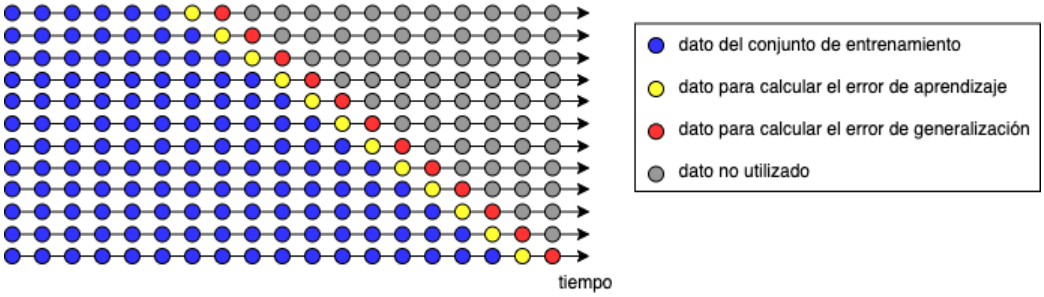



Figura 5.4: Diagrama de evaluación camina hacia adelante (Walk-Forward) obtenida de Hyndman and Athanasopoulos (2018).

Por último, como métrica de desempeño del autómata propuesto, se emplea la *exactitud* (ecuación 5.1) para calcular el error de aprendizaje y el error de generalización.

$$exactitud = \frac{VP + VN}{VP + FP + VN + FN} \tag{5.1}$$

donde:

- $VP$ : verdaderos positivos.
- $VN$ : verdaderos negativos.
- $FP$ : falsos positivos.
- $FN$ : falsos negativos.

Esta técnica de evaluación, es comúnmente utilizada para la evaluación de series de tiempo, debido a permite evaluar cómo se comportaría el sistema de forma general en ventanas de tiempo. Adicionalmente, en específico para este trabajo, esta técnica evalúa cómo se comportarían las reglas aprendidas por el algoritmo LRDEA, una vez que se ingresen al autómata celular y se realice la simulación del fenómeno por un periodo de tiempo.

# Capítulo 6

# Resultados y análisis

Como se mencionó anteriormente, se implementaron tres algoritmos diferentes (RA1, LRDEA y GA-Nuggets) y se realizó la experimentación utilizando los datos obtenidos de tres autómatas celulares diferentes (AC Brain, AC Byl, AC Evoloops y AC Mite). Para cada uno de los experimentos realizados con los autómatas celulares, se utilizaron los siguientes parámetros:

- **Vecindario:** de tipo Moore con 8 vecinos.
- **Parámetros de GA-Nuggets:**
  - $w1 = 1$
  - $w2 = 2$
  - $\beta = 2$
  - $pMutacion = 0{,}05$
  - $noHijos = 2$
  - $poblacion = 100$
  - $noIteraciones = 100$
  - $maximoAntecedente = 50$
  - $minimoAntecedente = 3$
- **Parámetros de LRDEA:**
  - $noHijos = 2$

- $poblacion = 100$
- $pMutacion = 0{,}05$
- $noIteraciones = 100$

- **Parámetros de RA1:**
  - $l = 5$

El algoritmo RA1, al contrario del algoritmo GA-Nuggets y el algoritmo LRDEA, es un algoritmo cuyo funcionamiento depende solamente de una variable aleatoria. Esto se debe a que, como es un algoritmo *avaro*, existe la posibilidad de que se quede estancado en un óptimo local; para evitar esto, es necesaria la mencionada variable aleatoria.

Finalmente, se puede notar en los parámetros que se enlistan arriba, que se requieren configurar menos parámetros para los algoritmos RA1 y LRDEA en comparación con el algoritmo GA-Nuggets.

## 6.1. Brain

En las figuras 6.1, 6.2 y 6.3 se muestran los resultados de exactitud de la implementación de cada uno de los tres algoritmos para los datos obtenidos del AC Brain.

Estos resultados de exactitud o *accuracy* se muestran a partir del paso de simulación o *Simultion step* número 160, debido a que los estados previos (1-159) se utilizan para el proceso de entrenamiento, y el estado 160 se utiliza para calcular el error de aprendizaje, exactitud de entrenamiento o *train accuracy*.

El otro dato que se visualiza en las gráficas, denominado error de generalización, exactitud de prueba o *test accuracy*, se calcula con un estado que no se le ha mostrado al algoritmo en el entrenamiento, en este caso, se calcula la exactitud de prueba con el estado siguiente (161).

De manera general, para calcular la exactitud de entrenamiento del estado $n$, se toma como conjunto de entrenamiento los *n-1* estados previos y

finalmente se compara el estado $n$ con el estado $n$ que se predijo. Mientras que para calcular la exactitud de prueba del estado $n$, se toma el estado $n+1$ (no mostrado previamente).

Según el diagrama de la figura 5.4, el conjunto de entrenamiento (en azul) serían los estados previos a $n$, el dato para calcular la exactitud de entrenamiento $n$ sería el amarillo y el dato $n+1$, utilizado para calcular la exactitud de prueba, sería el dato rojo.

Como se puede observar en la figura 6.1, el algoritmo RA1 tuvo un desempeño superior a los otros algoritmos con un promedio de exactitud de 89 % dentro del conjunto de entrenamiento y de 85 % fuera del conjunto de entrenamiento, a comparación del algoritmo LRDEA (figura 6.2) que obtuvo un promedio de 88 % y 81 % respectivamente, y el algoritmo GA-Nuggets con 88 % y 79 % (figura 6.3).

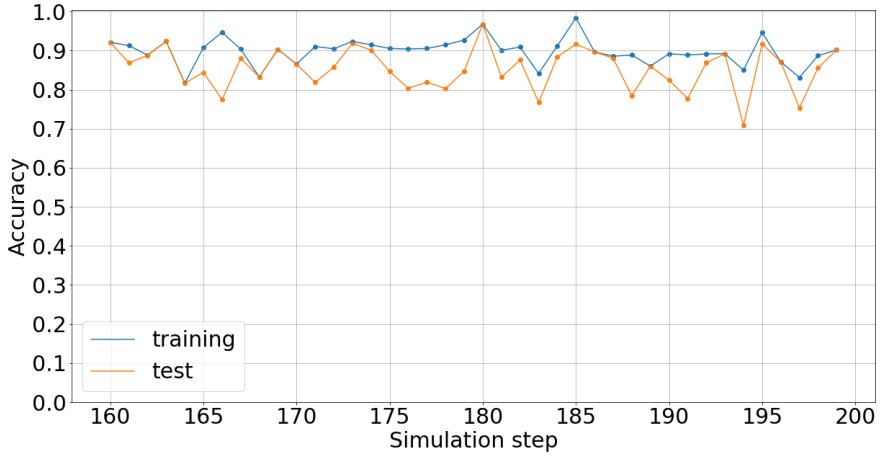



Figura 6.1: Exactitud de el algoritmo RA1 sobre el conjunto de datos del AC Brain.

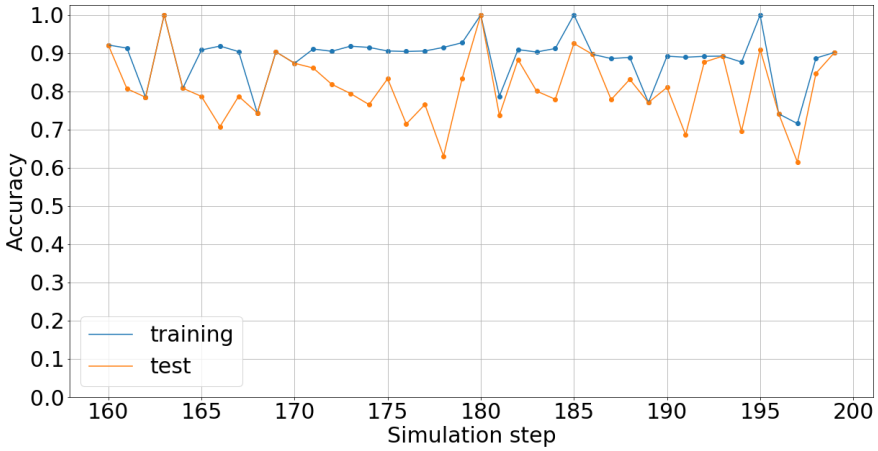



Figura 6.2: Exactitud de el algoritmo LRDEA sobre el conjunto de datos del AC Brain.

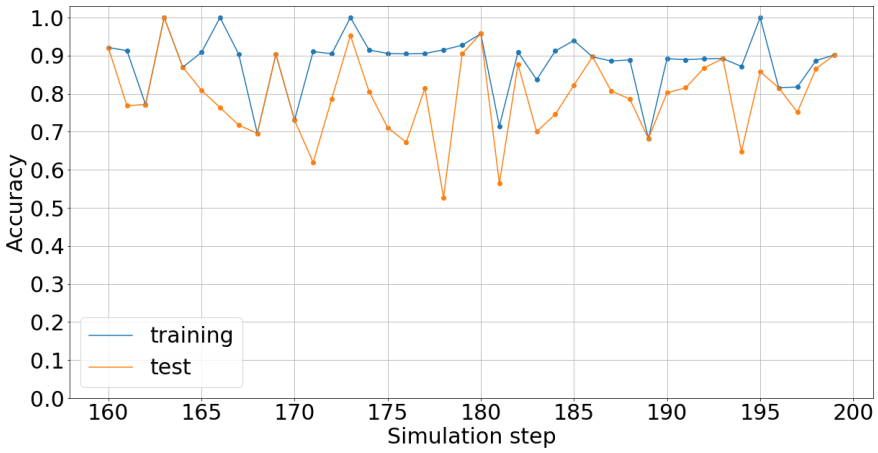



Figura 6.3: Exactitud de el algoritmo GA-Nuggets sobre el conjunto de datos del AC Brain.

## 6.2. Byl

De manera similar a la presentación de los resultados anteriores, en las figuras 6.4, 6.5 y 6.6 se muestran los resultados de exactitud de los tres algoritmos implementados en los datos del AC Byl.

Como se logra apreciar en las figuras a continuación, en ciertos pasos de la simulación se logró obtener una exactitud del 100 %, esto es debido a que ciertos estados tienen un comportamiento más trivial que los otros, lo que resulta más fácil de aprender para los algoritmos.

De la misma forma que los resultados para los datos del AC Brain, el algoritmo que obtuvo mejor valor de exactitud fue el RA1 con un promedio de 90 % en el conjunto de entrenamiento y un 85 % fuera del conjunto de entrenamiento.

A su vez, el algoritmo LRDEA obtuvo un promedio de 91 % y 84 % , lo que nos indica que el algoritmo tiene más variación con respecto al RA1. Por último, el algoritmo GA-Nuggets obtuvo 89 % y 77 % con mayor variación y menor valor de exactitud con respecto a los otros dos algoritmos.

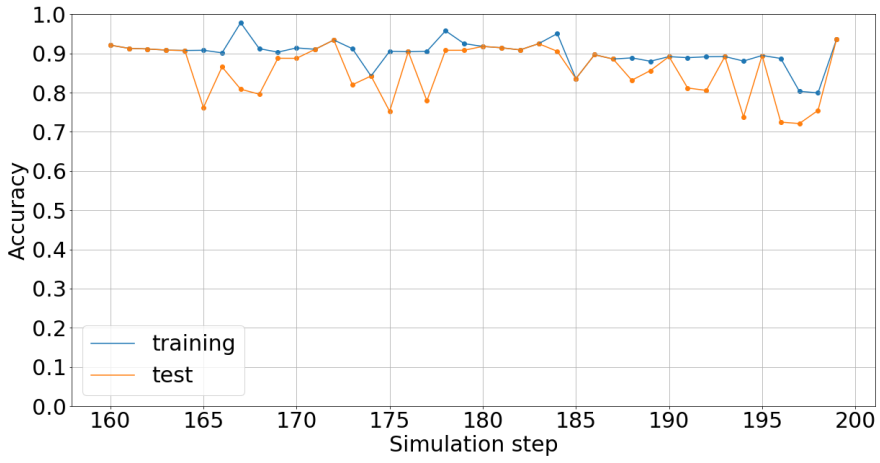



Figura 6.4: Exactitud de el algoritmo RA1 sobre el conjunto de datos del AC Byl.

Es interesante notar cómo en la figura 6.5 y la figura 6.6 se ve un comportamiento similar, a comparación con la figura 6.4. Esto puede deberse a las dinámicas que rigen a los algoritmos genéticos, se podría esperar que el algoritmo RA1 sea menos propenso a variaciones porque depende en menor medida de la probabilidad.

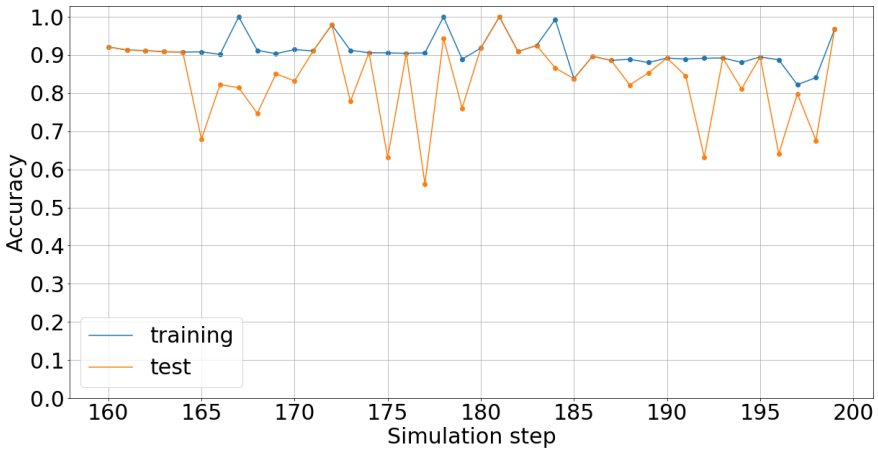



Figura 6.5: Exactitud de el algoritmo LRDEA sobre el conjunto de datos del AC Byl.

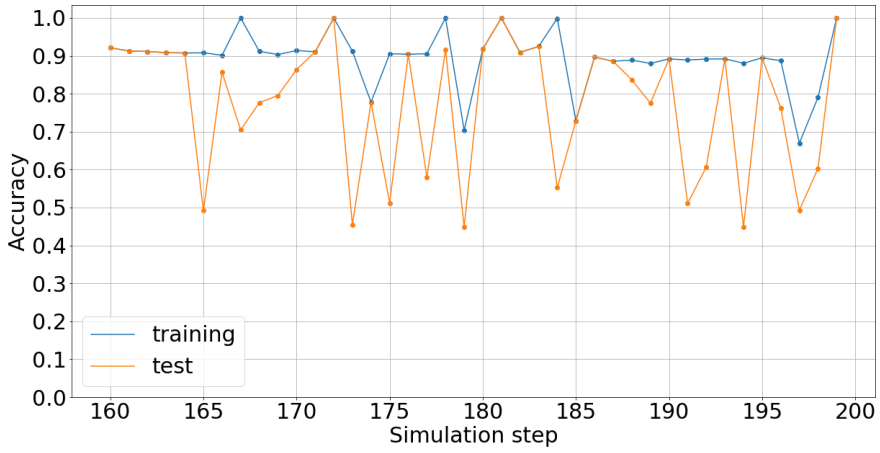



Figura 6.6: Exactitud de el algoritmo GA-Nuggets sobre el conjunto de datos del AC Byl.

## 6.3. Evoloops

En este experimento, al igual que con el AC Byl, el algoritmo RA1 obtuvo un desempeño superior al algoritmo LRDEA Y GA-nuggets, con un promedio de 90 % de exactitud dentro del entrenamiento y 86 % fuera de entrenamiento. En segundo lugar, se encuentra el algoritmo LRDEA con 89 % y 82 % y, por último, el algoritmo GA-Nuggets con 88 % y 78 %.

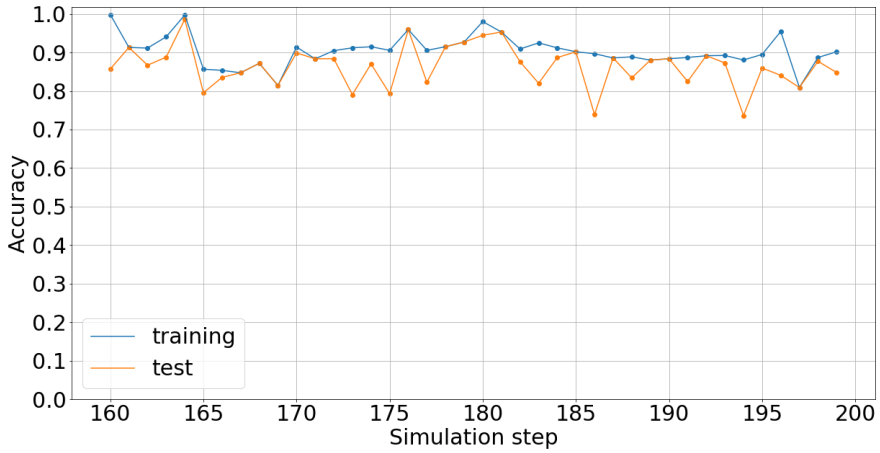



Figura 6.7: Exactitud de el algoritmo RA1 sobre el conjunto de datos del AC Evoloops.

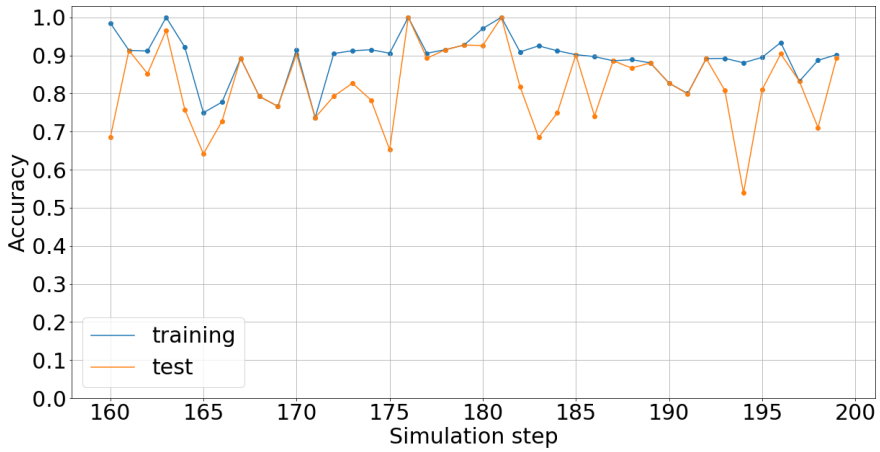



Figura 6.8: Exactitud de el algoritmo LRDEA sobre el conjunto de datos del AC Evoloops.

En la siguiente gráfica se puede observar cómo el GA-Nuggets puede tener mucha variación en su exactitud, en comparación a los otros dos algoritmos.

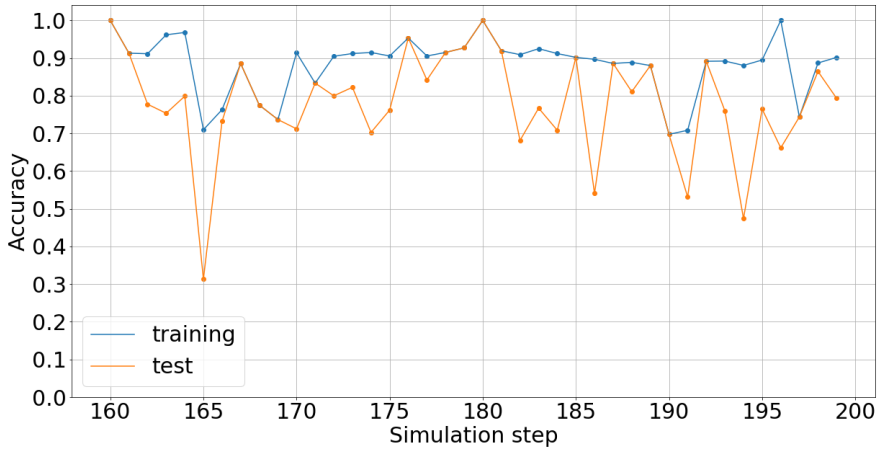



Figura 6.9: Exactitud de el algoritmo GA-Nuggets sobre el conjunto de datos del AC Evoloops.

## 6.4. Mite

Este experimento fue en el cual se obtuvieron los resultados más bajos, sin embargo, aun así podemos ver que la propuesta (LRDEA) sigue siendo capaz de mantener resultados en promedio superiores a los resultados del algoritmo GA-Nuggets.

Los promedios de exactitud dentro y fuera de entrenamiento para los datos del AC Mite, son los siguientes: RA1 con 89 % y 84 %, LRDEA con 89 % y 81 %, y por último GA-Nuggets con 88 % y 75 %, respectivamente.

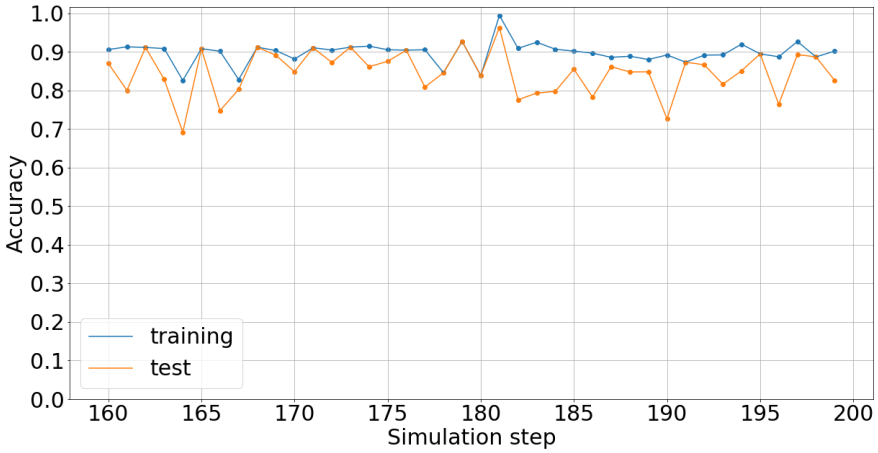



Figura 6.10: Exactitud de el algoritmo RA1 sobre el conjunto de datos del AC Mite.

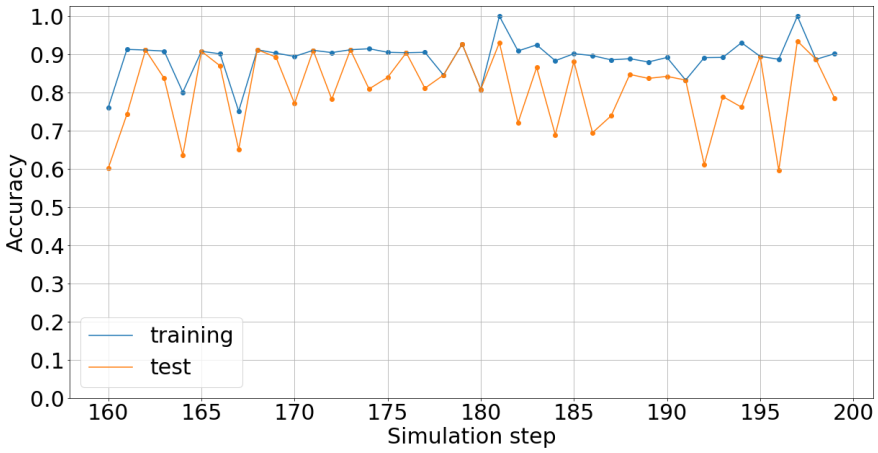



Figura 6.11: Exactitud de el algoritmo LRDEA sobre el conjunto de datos del AC Mite.

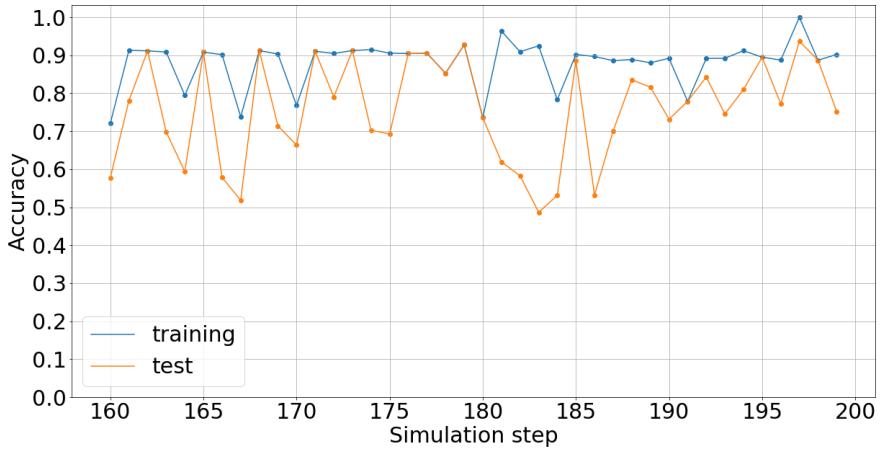



Figura 6.12: Exactitud de el algoritmo GA-Nuggets sobre el conjunto de datos del AC Mite.

En la siguiente tabla se resumen los resultados obtenido por los algoritmos en cada uno de los experimentos.

| | **RA1** | | **LRDEA** | | **GA-Nuggets** | |
|---|---|---|---|---|---|---|
| | **Entrenamiento** | **Generalización** | **Entrenamiento** | **Generalización** | **Entrenamiento** | **Generalización** |
| Brain | 0.89 | 0.85 | 0.88 | 0.81 | 0.88 | 0.79 |
| Byl | 0.90 | 0.85 | 0.91 | 0.84 | 0.89 | 0.77 |
| Evoloops | 0.90 | 0.86 | 0.89 | 0.82 | 0.88 | 0.78 |
| Mite | 0.89 | 0.84 | 0.89 | 0.81 | 0.88 | 0.75 |

Cuadro 6.1: Resultados de la evaluación de los algoritmos.

# Capítulo 7

# Conclusiones y trabajo futuro

Con la realización de este trabajo de investigación, se diseñó e implementó el nuevo algoritmo de aprendizaje sub-simbólico LRDEA (Local Rule Discovery Evolutive Algorithm), que es capaz de aprender un conjunto de reglas de la forma IF *Ant* THEN *Cons*, y estas reglas son capaces de reproducir ciertos fenómenos aprendidos.

De igual manera, se seleccionaron e implementaron los algoritmos de aprendizaje de reglas (RA1 y GA-Nuggets) que fueron parcialmente la base del algoritmo propuesto (LRDEA), esto se debe a que cada algoritmo tiene un funcionamiento diferente y, consigo, un comportamiento también diferente.

Se diseñó un modelo de autómata celular con el fin de evaluar los conjuntos de reglas de aprendizaje que se obtuvieron con los algoritmos RA1, GA-Nuggets y LRDEA.

Adicionalmente, se diseñaron procedimientos específicos para la simplificación de los conjuntos de reglas, tal es el caso del proceso que se ejemplifica en la figura 5.3. Este método elimina la redundancia de las reglas, mientras que el algoritmo Quine-McCluskey colabora en la simplificación de reglas al minimizar las cláusulas generadas por el algoritmo RA1.

Se seleccionaron también los autómatas celulares que reproducen ciertos fenómenos (simulación de la actividad cerebral, el sistema presa-depredador y la autoreplicación) con el fin de obtener datos bidimensionales para ingre-

sarlos a los algoritmos de aprendizaje de reglas y poder compararlos entre sí.

Además, se implementó el algoritmo de evaluación camina hacia adelante, con el cual se lograron obtener los errores de aprendizaje y generalización para cada experimento realizado.

Con base en los resultados de los experimentos realizados, se puede concluir que el algoritmo LRDEA es capaz de aprender un fenómeno a partir de un cierto conjunto de datos proporcionado. Este aprendizaje se realizó con un porcentaje de exactitud que sobrepasa al algoritmo GA-Nuggets en todos los casos. Sin embargo, como se puede observar en las figuras 6.2, 6.5, 6.8 y 6.11, es evidente que todavía es posible mejorar la propuesta.

Este espacio de mejora se observa principalmente en que el algoritmo LRDEA puede llegar a tener variaciones muy grandes entre la exactitud dentro del conjunto del entrenamiento y fuera de entrenamiento, en comparación con el algoritmo RA1, que se caracteriza por tener un comportamiento más estable, al menos en los experimentos realizados.

Otra conclusión que es importante resaltar es que los algoritmos genéticos mutli-poblacionales, como el LRDEA, realizan búsquedas robustas en espacios complejos y, como en este caso se realizó una búsqueda de reglas de aprendizaje a partir de un conjunto de lattices bidimensionales, esta complejidad se incrementa. A pesar de esto, el algoritmo propuesto realiza la búsqueda de reglas de una manera eficiente, lo cual posiciona al LRDEA como un algoritmo genético competitivo en esta tarea, al menos en exactitud, en comparación con el GA-Nuggets y el RA1.

## 7.1. Trabajo a futuro

Como trabajo a futuro próximo, se propone realizar la experimentación utilizando otros conjuntos de datos, como por ejemplo: imágenes aéreas de áreas urbanas para el aprendizaje de reglas que simulen el crecimiento de población u otros conjuntos de datos cuya representación de estados sea un conjunto de lattices bidimensionales.

De igual manera, se propone investigar una función de aptitud diferente,

con la cual sea posible a reducir la variación entre la exactitud dentro y fuera de entrenamiento.

Finalmente, se propone la implementación de otras métricas de evaluación que incluyan tomar en cuenta el tiempo de cómputo que cada algoritmo tomó para llevar a cabo estos u otros experimentos, o bien, diversos métodos de evaluación estadística.

# Acrónimos, abreviaturas y siglas

En éste apartado se listan los acrónimos, abreviaturas y siglas usadas en éste texto. Se incluye una descripción breve, la respectiva traducción en los casos que así lo requieren entre paréntesis. y la página donde aparece citada por primera vez a fin de que el lector pueda conocer el contexto en el que fue enunciada.

ABC Artificial Bees Colony, página 27

AC Autómata celular, página 5

AG Algoritmo Genetico, página 5

FN Falsos Negativos, página 41

FNC Forma Normal Conjuntiva, página 5

FND Forma Normal Disyuntiva, página 19

FP Falsos Positivos, página 41

GA-Nuggets Genetic Algorithm- Nuggets, página 19

IA Inteligencia Artificial, página 5

LRDEA Local Rule Discovery Evolutive Algorithm, página 41

OCAT One Clause At a Time, página 19

PSO Particle Swarm Optimization, página 27

RA1 Randomized Algorithm 1, página 19

RGB Red Green Blue, página 41

VN Verdaderos Negativos, página 41

VP Verdaderos Positivos, página 41

# Bibliografía


Biaynicki-Birula, I. and Biaynicka-Birula, I. (2012). *Modeling reality: how computers mirror life*. Oxford University Press.

Bidlo, M. (2016). On routine evolution of complex cellular automata. *IEEE Transactions on Evolutionary Computation*, 20(5):742–754.

Boerlijst, M. (1992). k hogeweg, p.(1992). self-structuring and selection: Spiral waves as a substrate for prebiotic evolution. *Artificial Life II. Addison Wesley, Reading, Mass*, page 255.

Byl, J. (1989). Self-reproduction in small cellular automata. *Physica D: Nonlinear Phenomena*, 34(1):295 – 299.

Cook, M. (2004). Universality in elementary cellular automata. *Complex systems*, 15(1):1–40.

Culik II, K., Hurd, L. P., and Yu, S. (1990). Computation theoretic aspects of cellular automata. *Physica D: Nonlinear Phenomena*, 45(1-3):357–378.

Deshpande, A. and Triantaphyllou, E. (1998). A greedy randomized adaptive search procedure (grasp) for inferring logical clauses from examples in polynomial time and some extensions. *Mathematical and Computer Modelling*, 27(1):75 – 99.

Ermentrout, G. B. and Edelstein-Keshet, L. (1993). Cellular automata approaches to biological modeling. *Journal of theoretical Biology*, 160(1):97–133.

Gardner, M. (1970). Mathematical games. *Scientific American*, 222(6):132–140.

Hillis, W. D. (1984). The connection machine: A computer architecture based on cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):213–228.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Ilachinski, A. (2001). *Cellular automata: a discrete universe*. World Scientific Publishing Company.

Kawaharada, A., Shoji, E., Nishimori, H., Awazu, A., Izumi, S., and Iima, M. (2016). Cellular automata automatically constructed from a bioconvection pattern. In *Recent Advances in Natural Computing*, pages 15–25. Springer.

Manneville, P., Boccara, N., Vichniac, G. Y., and Bidaux, R. (2012). *Cellular Automata and Modeling of Complex Physical Systems: Proceedings of the Winter School, Les Houches, France, February 21–28, 1989*, volume 46. Springer Science & Business Media.

Margolus, N., Toffoli, T., and Vichniac, G. (1986). Cellular-automata supercomputers for fluid-dynamics modeling. *Physical Review Letters*, 56(16):1694.

Nagel, K. and Schreckenberg, M. (1992). A cellular automaton model for freeway traffic. *Journal de physique I*, 2(12):2221–2229.

Naghibi, F. and Delavar, M. (2016). Discovery of transition rules for cellular automata using artificial bee colony and particle swarm optimization algorithms in urban growth modeling. *ISPRS International Journal of Geo-Information*, 5(12):241.

Nordahl, M. G. (1989). Formal languages and finite cellular automata. *Complex Systems*, 3(1).

Peysakh, J. (1987). *Fast Algorithm to Convert Boolean Expression Into a DNF*. IBM Corporation, Thomas J. Watson Research Center.

Rucker, R. and Walker, J. (2020).

Sayama, H. (1998). Constructing evolutionary systems on a simple deterministic cellular automata space.

Simon, P. and Nagel, K. (1998). Simplified cellular automaton model for city traffic. *Physical Review E*, 58(2):1286.

Tamayo, P. and Hartman, H. (1987). Cellular automata, reaction-diffusion systems, and the origin of life. In *ALIFE*, pages 105–124.

Toffoli, T. (1984). Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D: Nonlinear Phenomena*, 10(1-2):117–127.

Vichniac, G. Y. (1984). Simulating physics with cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):96–116.

Wolfram, S. (1984). Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1):1 – 35.