Universidad Autónoma Metropolitana

Unidad Azcapotzalco

Evaluación de función de aptitud para modelos de memorias asociativas evolutivas

Reporte de Proyecto Terminal
Licenciatura en Ingeniería en Computación

Presenta
Magdalena Pineda León

Matrícula
205304176

Asesores
Dr. Juan Villegas Cortez
Dr. Carlos Avilés Cruz

México D.F. — Diciembre 2013

RESUMEN

Una Memoria Asociativa (MA) es un caso particular de una Red Neuronal Artificial (RNA), cuyo principal propósito es realizar una rápida asociación entre patrones, a diferencia de la asociación más compleja realizada por los modelos clásicos de RNAs. En 2009 se inició el desarrollo de un nuevo modelo de MA llamado Memoria Asociativa Evolutiva (MAE). Estas MAEs se desarrollaron con ayuda de las técnicas de los Algoritmos Bioinspirados: Programación Genética (PG). La PG se basa en principios neodarwinianos de la evolución biológica, que permite la generación automática de programas de cómputo por medio de la selección natural, y es usada para hallar soluciones a problemas complejos del mundo real. Al igual que en la selección natural los algoritmos bioinspirados tienen una manera para medir la eficacia y eficiencia de los individuos generados, lo que nos conduce a encontrar la solución mas adecuada al problema dado, esta medida se conoce como Fitness o Aptitud. La única Función Fitness probada para la generación de las MAEs es la función Arco Coseno. En este proyecto, se han implemetado tres nuevas funciones de Aptitud, se presentan los resultados experimentales obtenidos al generar las MAEs y aplicandolas al problema de Reconocimiento de Patrones, (trabajando con patrones en valores reales). Además de mostrar el desarrollo de una sencilla aplicación que permite una fácil manipulación de algunos parámetros (como la función fitness) necesarios para la obtencion de las MAEs.

Índice general

Índice de figuras

Índice de tablas

Simbología

$\mathbb{R}$:= Números reales.

$\mathbb{Z}$:= Números enteros.

$\mathbb{N}$:= Números naturales.

$\mathbb{B}^n = \{0, 1\}$:= Subconjunto de números enteros.

M := Matriz de memoria asociativa.

m_{ij} := Componente i, j-ésimo de la memoria asociativa M

μ_{ij} := Matriz de asociación entre dos vectores, construida por medio de un operador Op^k.

$Op^k(X_i, Y_j^T)$:= Operador evolucionado de asociación entre los vectores X_i y Y_j^T.

M_k := Matriz de la Memoria Asociativa, formada a partir de $\sum \mu_{ij}$.

$\bar{X}$:= Conjunto de n vectores, acomodados en una matriz de dimensión $[n \times m]$, con cada vector tal como:

$X \in \bar{X}$, con X de dimensión $[1 \times m]$, tal que $X = \{x_1, x_2, \ldots, x_m\}$

$\widetilde{X}$:= Versión con ruido del patrón X.

$\widetilde{\bar{X}}$:= Conjunto de patrones X alterados con ruido.

$\widehat{Y}$:= Patrón recuperado-aproximado a Y.

Capítulo 1

Introducción

1.1. Objetivo General

Probar el desempeño de las memorias asociativas evolutivas (MAE) en tres escenarios con patrones en valores reales, usando tres tipos de funciones de aptitud, a fin de dar un criterio de uso de función de aptitud vs. complejidad de patrones.

1.2. Objetivos Específicos

- Analizar, diseñar e implementar la generación de memorias asociativas mediante programación genética.
- Implementar tres funciones de aptitud para el modelo coevolutivo.
- Probar en valores reales las funciones de aptitud implementadas.
- Proporcionar un criterio de uso de la función de aptitud vs. Complejidad de patrones.
- Diseñar y crear una interfaz gráfica para el código coevolutivo.
- Comparar los resultados de las pruebas realizadas.

1.3. Justificación

Las Memorias Asociativas (MA) con Programación Genética (PG) planteadas en [40] son un modelo de Red Neuronal Artificial (RNA) muy específico, pero desde la pers-

pectiva de un modelo coevolutivo que divide el proceso de búsqueda de una solución en dos poblaciones, una para el proceso de "asociación" y otra para el proceso de "recuperación". Actualmente el código reportado en [5] trabaja con una sola función de aptitud, y se ha probado para patrones en valores binarios y reales. Estas MA con PG funcionan bien con patrones en valores binarios, pero no se han probado con diversos tipos de patrones reales y con más de una función de aptitud. Las MA son importantes porque han demostrado su eficiencia en la asociación de patrones al usar funciones elementales (e.g. suma, resta, mínimo, máximo, multiplicación a nivel escalar), en comparación de las Redes Neuronales Artificiales (RNA) cuya complejidad involucra el uso de funciones trascendentales y trigonométricas, así como el uso de la derivada. Construir MA no es una tarea fácil, normalmente en su concepción básica funcionan para un solo tipo de patrones y tienden rápidamente a saturarse, i.e. a dejar de ser útiles para asociar cantidades de patrones superiores a a diez, o bien no tienen un buen comportamiento bajo patrones con ruido en algunos casos. Las MA con PG son evolucionadas acorde al tipo de patrones y la cantidad de los mismos, teniendo con esto MAs a la medida necesaria, logrando con esto tener asociaciones sencillas para diversos fines o aplicaciones (filtros de ruido en patrones, diccionarios de palabras, etc.). Con esto, probar tres funciones de aptitud en la creación de MA con PG es importante porque se podrán obtener las MA evolucionadas con base a usar mas de una función de aptitud, mejorando con ello el proceso evolutivo y obteniendo mejores resultados.

Capítulo 2

Marco Teórico

2.1. Memorias Asociativas

Una memoria asociativa M es un dispositivo que permite asociar patrones de entrada con patrones de salida. Esto es, al presentarle a una memoria asociativa M un patrón de entrada X, ésta responde con el correspondiente patrón (vector) de salida Y. Podría decirse que una memoria asociativa es una Red Neuronal Artificial (RNA) de una sola capa.

Una MA es un tipo especial de RNA que permite la recuperación de un patrón de salida dado un patrón de entrada, aún habiendo experimentado una alteración a través de un tipo de ruido (aditivo, sustractivo o mixto) en el patrón. En las últimas décadas se han desarrollado ya diversos modelos de MA como los descritos en [2, 30, 26, 12, 39, 3, 29, 27, 31, 4]. Algúnos modelos trabajan bien sólo con un tipo de ruido (aditivo o sustractivo), pero no funcionan con ambos tipos de ruido, a excepción de los modelos propuestos en [27, 39] que son robustos al ruido mixto, y en especial el trabajo de [39] que desarrolla la aplicación de MA a patrones con valores reales en imágenes en color.

La asociación entre un patrón de entrada X y uno de salida Y se denota como (X^k, Y^k), donde k es el índice de la asociación correspondiente. La memoria asociativa M es representada por una matriz cuyos componentes m_{ij} se puede considerar como las conexiones "sinapsis" de una RNA. El operador M es generado mediante un conjunto finito de asociaciones conocidas a priori. Este conjunto conocido de asociaciones es denotado en la literatura como el conjunto fundamental de asociaciones y se representa como: $\{(X^k, Y^k)|k = 1, ..., p\}$, con p el número de asociaciones.

El proceso de construir dicha matriz (M) es llamado *aprendizaje* o *entrenamiento*,

mientras que el obtener un patrón de salida cuando un patrón de entrada es presentado a la memoria es llamado *recuperación*.

Si $X^k = Y^k \ \forall k = 1, ..., p$ entonces M es autoasociativa $X \overset{M}{\rightarrow} X$, de otra forma es heteroasociativa $X \overset{M}{\rightarrow} Y$. Se define como $\widetilde{X}$ una versión con ruido del patrón X . Si una MA M es alimentada (propagada) con un patrón distorsionado de $\widetilde{X}^k$ y como salida se obtiene exactamente Y^k , se dice que la recuperación es correcta.

Al hablar de las entradas m_{ij} de la MA M, éstas están conformadas por operaciones muy simples como sumas, restas, multiplicaciones, máximos, mínimos y sub asociaciones simples, buscando establecer una asociación local entre los pares asociados, mientras que la matriz final M conforma la asociación global.

2.2. Introducción al Cómputo Evolutivo

El llevar a las computadoras la famosa teoría de Darwin (publicada en *El origen de las especies basada en la selección natural*) consiste en tratar de imitar, con programas de cómputo, la capacidad de una población de organismos vivos para adaptarse a su medio ambiente por medio de mecanismos de selección y reproducción. En los últimos cuarenta años varios métodos estocásticos de optimización se han basado en este principio. *Darwinismo Artificial* o *algoritmos evolutivos* es el nombre común para estas técnicas, quizás las más comúnmente escuchadas sean *algoritmos genéticos, estrategias evolutivas* o *programación genética*.

Los componentes comunes en estas técnicas son las *poblaciones*, que representan puntos muestra de un espacio de búsqueda, y que evolucionan bajo la acción de operadores estocásticos. La evolución usualmente se organiza en *generaciones* y esto asemeja de una forma simple la genética natural. El motor de esta evolución está dada por:

1. *Una selección*, asociada a una medida de la calidad de un individuo respecto al problema a resolver,

2. *Operadores genéticos*, usualmente *mutación* y *cruza* o *recombinación*, que producen una nueva generación de individuos.

La eficiencia de un algoritmo evolutivo depende fuertemente de los parámetros establecidos: las poblaciones sucesivas (generaciones) tienen que converger hacia lo que se desea, esto es, hacia el óptimo global de una función de desempeño. Una gran parte de la búsqueda teórica en algoritmos evolutivos está dedicada al delicado problema de la

convergencia, así también al tratar de focalizar qué tipo de problemas es fácil o difícil para un algoritmo evolutivo. La respuesta, en teoría es que converge, sin embargo otras cuestiones prácticas que ponen en evidencia, como la rapidez de convergencia, permanecen abiertas. Se puede ver que el interés en las técnicas evolutivas están suficientemente bien fundadas teóricamente, y de esto que justificadamente en los últimos cuarenta años se han podido aplicar con éxito en diferentes desarrollos experimentales.

Hay que agregar que las técnicas evolutivas son de optimización estocástica de orden cero, esto es, que no son necesarias las propiedades de continuidad y deribabilidad. La única información requerida es el valor de la función a optimizar en los puntos muestra (algunas veces hasta una aproximación puede ser usada); es así que estos métodos son particularmente adaptables para funciones irregulares, raras, complejas o con condiciones malas. Sin embargo su tiempo de cómputo puede ser muy grande.

Las técnicas evolutivas se suelen recomendar cuando los métodos clásicos y sencillos fallan (para espacios de búsqueda muy grandes, variables mixtas, cuando se tienen varios óptimos locales, o cuando las funciones son muy irregulares). En otros problemas tales como los dinámicos o interactivos también pueden retomarse desde los algoritmos evolutivos, y finalmente estos métodos también pueden combinarse (hacerlos híbridos) con los métodos clásicos de optimización (e.g. gradiente descendente, búsqueda tabú)[15].

Más allá de lo atractivo en simplicidad que pueda parecer un proceso evolutivo, la construcción y eficiencia de un algoritmo evolutivo es una tarea difícil, dado que un proceso estocástico evolutivo es muy sensible a los parámetros y estructura del algoritmo. La elaboración de un algoritmo evolutivo eficiente se basa en un buen conocimiento del problema a resolver, así como del buen entendimiento de los mecanismos de evolución. El decidirse a atacar la solución de un problema, sin esto último, sería como estar tratando con un problema de "caja negra"; y esto no se recomienda.

Se ha tenido mucho éxito en la implementación de estas técnicas en la industria [10], en diversos y variados campos como lo es el dominio del análisis de imágenes [14, 16, 7, 34, 33, 38, 11, 18, 35, 17, 19, 36, 23, 24, 20, 21] y la visión robótica [37].

2.3. Conceptos básicos de la evolución artificial

Los algoritmos evolutivos han tomado prestado (de manera muy simplificada) algunos principios de la genética natural. Es así que se habla de *individuos* que representan soluciones o puntos de un espacio de búsqueda, llamado *ambiente*. En este ambiente,

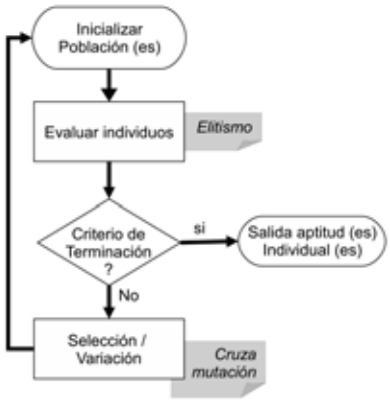

Figura 2.1: Organigrama de un algoritmo evolutivo simple.

hallar un máximo de una *función de aptitud* o *función de evaluación* es lo que está en juego, su búsqueda.

Los individuos se representan usualmente como códigos (reales, binarios, de tamaño variable o fijo, simple o complejo), son *cromosomas* o *genomas*, esto es, *genotipos*. Las soluciones correspondientes (i.e., los vectores del espacio de búsqueda) son *fenotipos*. Un algoritmo evolutivo involucra su población de tal manera que vuelve a sus individuos más y más *adaptados* al ambiente. En otras palabras, la función de aptitud es *maximizada*.

En la siguiente sección se describe lo más básico de un algoritmo evolutivo "canónico". Las aplicaciones en la vida real son por supuesto mucho más complejas, con el principal problema de adaptar, o más aún de crear, operadores que correspondan al problema que se tiene en mano.

2.4. El ciclo de la evolución

El primer elemento evolutivo es un ciclo de generaciones de poblaciones de individuos, donde cada individuo corresponde a una solución potencial del problema a tratar de resolver. Véase la Figura 2.1.

La inicialización usualmente es aleatoria (algunas veces otras estrategias también son usadas, particularmente en espacios de búsqueda complejos o multidimensionales). Las soluciones iniciales (obtenidas, e.g., mediante una técnica clásica de optimización)

también puede integrarse en la población inicial. Si el contenido de la población inicial, en teoría, no tiene importancia (la distribución límite para tales procesos estocásticos es siempre la misma), se ha visto experimentalmente que la inicialización tiene un gran influencia sobre la variación de los resultados y la velocidad de convergencia. Es muy eficiente algunas veces la inserción de información "a priori" acerca del problema en la etapa de inicialización.

En la selección se decide cuales individuos de la actual población se reproducirán. Esto se basa en la noción de "calidad" de un individuo, y esto se encaja en la *función de aptitud*.

El principal parámetro de selección es la *presión selectiva*, usualmente definida como el cociente de la probabilidad de seleccionar al mejor individuo sobre la probabilidad de seleccionar un individuo promedio. La presión selectiva tiene una fuerte influencia en la *diversidad genética* de la población, y consecuentemente sobre la eficiencia del algoritmo completo. Por ejemplo, una presión selectiva excesiva puede llegar a producir una rápida concentración de la población en la vecindad de sus mejores individuos, con el riesgo de una convergencia prematura hacia un óptimo local.

La selección más simple es la *selección proporcional*, implementada con un tiro aleatorio sesgado, donde la probabilidad de selección de un individuo es directamente proporcional a su valor de aptitud:

$$P(i) = \frac{aptitud(i)}{\sum_{k=1}^{TamPob} aptitud(k)} \tag{2.1}$$

Este esquema no permite controlar la presión selectiva. Otros esquemas de selección, y más eficientes, son:

1. *Escalado*, que linealmente escala la función de aptitud en cada generación tratando de obtener un máximo en la aptitud que es C veces la aptitud promedio de la población actual. La C mide la presión selectiva, usualmente fija entre 1.2 y 2 [10].

2. *Ranking*, que le asigna a cada individuo una probabilidad que es proporcional a una lista ordenada de aptitud.

3. *Torneo*, que selecciona aleatoriamente T individuos de la población (independientemente de sus valores de aptitud) y elige el mejor. La presión selectiva se relaciona con el tamaño T de la ruleta.

La **reproducción** genera una nueva descendencia. En el esquema canónico "à la Goldberg" [10], 2 padres producen 2 hijos, un número de padres igual al número de individuos en la nueva población es seleccionado. Por supuesto también se pueden programar otros esquemas (2 padres producen 1 hijo, n padres producen p hijos, etc.).

Los dos principales *operadores de variación* son la cruza, o recombinación, que combina genes de los padres, y la mutación, que delicadamente perturba al genoma. Estos operadores son aplicados aleatoriamente, basándose en dos parámetros: probabilidad de cruza p_c y probabilidad de mutación p_m.

Intuitivamente, la selección y cruza tienden a concentrar la población cerca de individuos "buenos" (explotación de la información). Por el lado contrario, la mutación limita la atracción de los mejores individuos con el fin de dejar a la población explorar otras áreas del espacio de búsqueda.

La **evaluación** calcula (o estima) la calidad de los nuevos individuos. Este operador es el único que usa la función a ser optimizada. Ninguna hipótesis se hace sobre esta función, excepto para el hecho de que debe ser usada para definir una probabilidad o al menos una calificación para cada solución.

El **reemplazo** controla la composición de la generación $n + 1$. El elitismo se recomienda para las tareas de optimización con el fin de lograr contener a los mejores individuos de una población (generación) a la siguiente. Las estrategias suelen transmitir directamente un porcentaje dado de los mejores individuos en la siguiente población (e.g., la brecha generacional).

En el caso de implementaciones en paralelo, algunas veces es más práctico usar otros esquemas en lugar del basado en generaciones: el esquema del *estado constante* agrega directamente cada nuevo individuo en la población en turno vía un operador de reemplazo (selección inversa) que reemplaza los individuos malos de la población actual por nuevos.

El **criterio de paro** del proceso evolutivo en el momento adecuado es crucial en términos prácticos, pero si no se tiene la mínima información disponible, o ninguna, acerca del valor óptimo buscado es muy difícil conocer cuando detener el proceso. Una estrategia usual consiste en detener la evolución después de un número fijo de generaciones, o cuando ocurra un estancamiento. También es posible probar la dispersión de la población. Un buen control que se puede utilizar como criterio de paro que obviamente influenciará la eficiencia del algoritmo, y es un importante parámetro a considerar en la evolución (tamaño de la población, probabilidades de cruza y mutación, presión selectiva, porcentaje de reemplazo, etc.).

Un algoritmo evolutivo (AE) es parcialmente un algoritmo de búsqueda ciega, cuyo componente ciego o aleatorio tiene que ser afinado o delimitado de forma inteligente, como una función de qué es lo que se conoce "a priori" acerca del problema a ser resuelto, ya que demasiada aleatoriedad es tiempo consumido, y puede en una pequeñísima proporción hacer que el proceso se estanque en un óptimo local.

2.5. Representaciones y operadores

Los operadores genéticos dependen directamente de la elección de la representación, la cual, por ejemplo, hace la diferencia entre algoritmos genéticos, estrategias de evolución, programación genética, y evolución gramática. En la siguiente sección se muestran las representaciones más usadas, los operadores y los esquemas de selección y reemplazo. En la literatura se pueden hallar otros esquemas para espacios de búsquedas no estándar o espacios de grafos.

2.5.1. Representación discreta

Los algoritmos genéticos se basan en el uso de representaciones binarias de soluciones, extendidas más tarde a representaciones discretas.

Cada individuo de la población se representa por una cadena de longitud fija, con los caracteres (genes) elegidos de un alfabeto finito. Esta representación es obviamente apropiada para problemas de combinatoria discreta, y los problemas de tipo continuo pueden modelarse con esta representación gracias al muestreo del espacio de búsqueda. En este caso la precisión del muestreo (relativo a la longitud del cromosoma) es un parámetro importante del método.

Los operadores de cruza más clásicos usados en tareas de optimización se muestran en la Figura 2.2. La *cruza en un punto* elige aleatoriamente un posición del cromosoma y entonces intercambia partes de cadena alrededor del punto. La *cruza a dos puntos* también intercambia porciones de los cromosomas, pero selecciona dos puntos para el intercambio. Finalmente, la *cruza uniforme* es una generalización multipunto del previo: cada gen de una nueva generación es aleatoriamente elegido de entre los genes de los padres en la posición dada. Existen otros operadores de cruza especializados, como en el problema del agente de ventas viajero o en problemas de distribución de horarios, que considera la estructura específica del gen codificado.

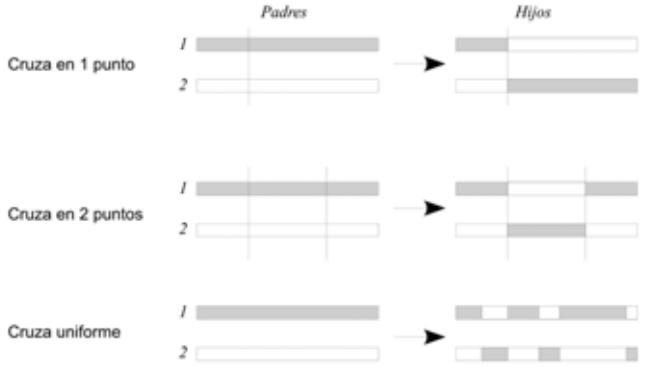

Figura 2.2: *Ejemplos de cruzas binarias.*

Figura 2.3: Ejemplo de mutación binaria.

La mutación binaria clásica intercambia cada bit del cromosoma con una probabilidad p_m (véase la Figura 2.3). La probabilidad de mutación p_m es usualmente muy baja y es constante a lo largo del proceso evolutivo, pero en algúnos esquemas existentes, esta probabilidad de mutación decrece a lo largo de las generaciones.

2.5.2. Representación continua

La representación continua o representación real se relaciona históricamente con estrategias evolutivas. Este alcance realiza una búsqueda en $\mathbb{R}^n$ o en una parte de él. Los operadores genéticos asociados son extensiones al espacio continuo de los operadores discretos, o bien directamente operadores continuos.

La *cruza discreta* es una mezcla de genes reales de un cromosoma, sin cambio en su contenido. Los operadores binarios de cruza previos (un punto, dos puntos, uniforme) pueden ser adaptados de forma directa.

El beneficio de una representación continua es mejor aprovechada con operadores especializados, es decir, una *cruza continua* que mezcle íntimamente los componentes

de los vectores padres para producir nuevos individuos. La cruza *barycentrica*, también llamada *aritmética*, produce una descendencia x' de una pareja (x, y) de $\mathbb{R}^n$ gracias al tiro uniformemente aleatorio de una constante α en [0,1] (o $[-\epsilon, 1+\epsilon]$ para la cruza $BLX-\epsilon$) tal que:

$$\forall i \in 1, \ldots, n, \quad x'_i = \alpha x_i + (1-\alpha) y_i \tag{2.2}$$

La constante α puede ser escogida para todas las coordenadas de x', o de forma independiente para cada coordenada.

La generalización para la cruza de más de 2 padres, o de toda la población (cruza "global") se muestra con más detalle en [28].

Varios operadores de mutación se han propuesto para la representación real. La más clásica es la *mutación Gaussiana*, que agrega ruido Gaussiano a los componentes del individuo. Se requiere de un parámetro adicional, σ, la desviación estándar del ruido, tal que se aplica como:

$$\forall i \in 1, \ldots, n, \; x'_i = x_i + N(0, \sigma) \tag{2.3}$$

La afinación de σ es relativamente compleja (demasiado pequeña alentará la evolución, demasiado larga afectará negativamente la convergencia del algoritmo). Se han probado varias estrategias para hacer que σ tenga variaciones a lo largo del proceso evolutivo: σ como una función del tiempo o del valor de la aptitud, como una función de la dirección de búsqueda, o auto adaptables (i.e., considerando a σ un parámetro adicional, que vaya evolucionando con el algoritmo). Otros estudios se han enfocado en el uso de ruido no Gaussiano.

2.5.3. Representación de árboles y Programación Genética

La Programación Genética (PG) corresponde con una representación de estructuras de longitud variable como árboles. La PG fue diseñada a manejarse con programación en LISP [13], con el propósito de crear programas capaces de resolver problemas para los cuales no hubiesen sido explicitamente programados. La riqueza y versatilidad de la representación de tamaño variable en forma de árbol (véase la Figura 2.4 en el Capítulo 3), es parte del origen del éxito de la PG. Muchos problemas de optimización o control pueden ser formulados como un problema de inducción de programas. Recientemente en el dominio de la visión por computadora, la PG ha demostrado alcanzar resultados

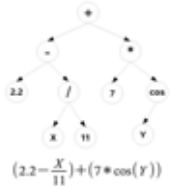

Figura 2.4: Ejemplo de una función representada como una estructura de árbol.

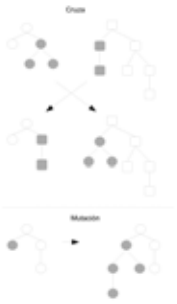

Figura 2.5: Ejemplos de los operadores de cruza y mutación en representación de árboles.

humanamente competitivos [34].

Un algoritmo de PG explora un espacio de búsqueda de programas recursivos hechos de elementos de un conjunto de funciones, un conjunto de variables (datos, constantes)[1]. Los individuos de la población son programas que cuando son ejecutados producen una solución para el problema dado.

Las cruzas son un intercambio de subárboles. Las mutaciones son más complejas, y diversas mutaciones se tienen que usar, produciendo diferentes tipos de perturbaciones sobre la estructura del genoma: supresión/adición de un nodo, modificación del contenido de un nodo, mutación de las constantes (valores continuos), y la mutación de variables discretas (véase la Figura 2.5).

Las aplicaciones de la PG son numerosas, por ejemplo, en control óptimo, en planeación de trayectorias y movimientos de robots, o en regresión simbólica (la búsqueda de una fórmula matemática que aproxime un conjunto finito de puntos), por mencionar algunas [13].

[1]Un problema actual en la PG es el llamado *bloat*, esto es la saturación del espacio de memoria dado el desproporcionado crecimiento del tamaño de los árboles a lo largo de la evolución. Una buena forma de evitar este efecto es limitar el tamaño del genoma.

2.6. Coevolución

La hipótesis básica de aplicación de un algoritmo evolutivo (AE) a un problema es ver a los individuos generados como posibles soluciones del problema, pero dado el éxito de la aplicación de los AE a cada vez más problemas, estos se han vuelto más complicados o complejos acorde a los nuevos campos que se van abarcando, de esta forma la noción implícita de *modularidad* se ha ido introduciendo en la medida en que se acoplan las soluciones, i.e., se han separado en partes a los problemas en un principio enunciados como monolíticos, y a este problema se le ajusta una población de posibles soluciones a evolucionar; el enfoque se da en separar entonces poblaciones para cada parte del problema que se reconozca como separable y armar una solución a partir de las partes evolucionadas; de esta forma las posibilidades de hallar soluciones óptimas se incrementa, así también la diversidad de datos que éstas arrojan. Este paradigma de romper el problema en partes, conocido también como el *paradigma coevolutivo*, se muestra en la Figura 2.6. Problemas de ejemplos de gran complejidad los tenemos en el dominio de la robótica, donde las tareas a llevar a cabo pueden parecer simples a primera vista, pero éstas se descomponen en sub-tareas o módulos que se tienen que interconectar para poder lograr realizarce. El cómo descomponer de manera "natural" una tarea en varias, analizar cómo se interconectan, en qué medida puede ser independiente cada parte, su sub-comportamiento, es una tarea difícil y depende fuertemente del entorno del campo de aplicación [22], todo esto lleva a conceptualizar una coadaptación de las partes, y esto es un requisito crítico en el proceso evolutivo natural, mismo que inspira su implantación en los modelos tradicionales de AE.

Existen dos principales razones por las cuales los AE tradicionales no son totalmente adecuados para solucionar este tipo de problemas. Primero, la población de individuos a evolucionar por estos AE tradicionales tienen una tendencia fuerte a converger como respuesta a un gran número de ensayos o pruebas en una área o región del espacio de búsqueda a partir de una medida de aptitud promedio. Esta propiedad de convergencia fuerte evita la preservación a largo plazo de la diversidad de los subcomponentes dado que cualquiera de los individuos más fuertes o aptos será eliminado. Segundo, los individuos evolucionados a partir de los AE tradicionales representa tipicamente soluciones completas y son evaluados de forma aislada. Considerando que la interacción entre poblaciones no es modelada, se tiene que no hay presión para que una coadaptación ocurra con el modelo tradicional de AE. Más aún, complicado es el que se debe tener una idea pre-concebida de cuántas subpoblaciones se deben tener, qué partes modelar, y cuales

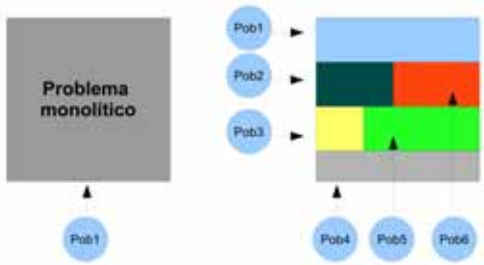

Figura 2.6: El paradigma coevolutivo: Problema monolítico (una sola población, una sola parte) vs. problema en varias partes (varias poblaciones, una para cada parte).

no. La dificultad que surge de aquí es de qué forma se pueda dar la separación de éstas partes de forma no supervisada, de forma razonable para la naturaleza del problema a considerar. Un análisis detallado de los avances en el desarrollo de la co-evolución en los AE puede consultarse en [22], y de cómo su bioinspiración proviene también de la teoria de juegos puede verse en [8].

Hablando de historia, Nils Aall Barricelli fue uno de los pioneros del cómputo evolutivo, al conceptualizar lo que ahora llamamos *vida artificial* en un estudio donde realizó la simulación de una población de números en un arreglo matricial [2]. En el arreglo, cada número se movía de acuerdo a una regla específica para cada uno. A éstas las llamó *reglas de migración*. Barricelli hizo diversas observaciones acerca de los patrones que emergieron a partir de las reglas simples, a estos los calificó como "organismos". Estos organismos se definieron de forma *independiente* para el caso en que pudieran reproducirse sin requerir de otro organismo proveniente de un patrón diferente, el cual definió como "otra especie". De forma análoga un organismo que no era independiente se definió como *dependiente*, ya que requiere de otros organimos para su reproducción, por lo que se le describió también como un *parásito*. Barricelli notó nuevos patrones a partir de la recombinación de patrones con las operaciones que definió en su experimento. Posteriormente experimentó una versión de dos dimensiones de su experimento obteniendo resultados similares al trabajo del *Juego de la vida de Conway* publicado en [6], y es entonces que llega a la simulación de un juego co-evolucionado.

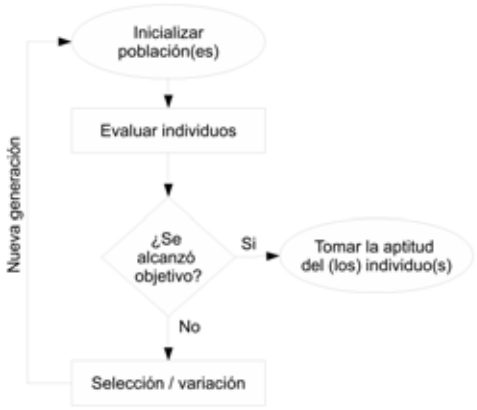

Figura 2.7: Diagrama de flujo de la coevolución convencional.

En la Figura 2.7 se muestra la forma convencional de operación de los algoritmos coevolutivos. Se puede ver que la diferencia aparece en la evaluación de los individuos, el cálculo de su *aptitud*, y en ello es donde recae la forma de operar de una u otra forma en que se aplica esta teoría. En general el paradigma de la coevolución trata de flexibilizar los AE para hallar mejores soluciones a los probelmas, al de preservar la diversidad en los individuos obteniendo con ello una rápida convergencia [32]. Por otro lado, se tiene que el éxito de la aplicación de la coevolución depende de la apropiada descomposición del problema en su espacio de búsqueda, el cual pudiera conocerse *a priori* y/o éste puede cambiar de forma dinámica con el tiempo, con lo que se tiene la introducción de mecanismos diversos de elitismo y diversidad cuyos diseños son particulares para cada problema.

Los algoritmos coevolutivos se pueden clasificar en dos tipos:

- Coevolución competitiva. El modelo de coevolución competitiva es inspirada por las interacciones depredador-presa o anfitrión-parásito, donde la presa (o el anfitrión) implementa las soliciones potenciales para la optimización del problema, mientras el depredador (o el parásito) implementa casos particulares de aptitud. Con esta idea base se tienen dos sub poblaciones, donde la aptitud de una impacta en el desempeño de la otra, a fín de que una especie sobreviva a la otra.

- Coevolución cooperativa. El modelo de coevolución cooperativa es inspirado por

la relación ecológica de simbiosis para el caso en que varias especies conviven juntas en una relación beneficiosa mutuamente. La idea básica de la coevolución cooperativa es *divide y vencerás*, i.e., dividir un sistema grande en módulos, evolucionando los módulos por separado y entonces combinarlos conjuntamente para tener al sistema completo, de tal forma de proporcionar soluciones por partes a sistemas o problemas complejos.

En [32] se muestra con mayor detalle estos dos tipos de coevolución, así como de sistemas de coevolución híbridos, donde se aplica para algunos problemas de optimización multiobjetivo.

2.7. Fitness

Fitness(*Aptitud*) es la fuerza impulsora de la selección natural Darwiniana y asimismo, de los algoritmos genéticos convencionales y la programación genética.

En la naturaleza el fitness de un individuo es la probabilidad de sobrevivir a la edad de reproducción y reproducirse. Esta medida puede ser ponderada la considerar el número de descendientes. En el mundo artificial de algoritmos matemáticos se mide la aptitud de alguna manera y después se utiliza esta medida para controlar la aplicación de la operación que modifica la estructura de nuestra población artificial.

El fitness se puede medir de muchas maneras distintas, algunas explicitas y algunas implícitas.

El método mas común para medir el fitness es crear una medida de fitness explicita para cada individuo de la población.

Este enfoque es usado en la mayoría de las aplicaciones de algoritmos genéticos convencionales. A cada individuo en la población se le asigna una valor fitness escalar mediante algún procedimiento de evaluación explícito bien definido.

El fitness también se puede calcular de una manera co-evolutiva como cuando la condición física de un juego de estrategia de juego se determina tocando esa estrategia contra toda una población (o muestreo) de las estrategias opuestas. El hecho de que existen individuos y sobrevivir en la población y reproducirse con éxito puede ser indicativo de su estado físico (como es el caso en la naturaleza). Esta definición implícita de aptitud se utiliza a menudo en la investigación de la vida artificial.

2.8. Reconocimiento de Patrones

Un *patrn* es una descripción estructural o cuantitativa de un objeto o de alguna otra entidad de interés en una imagen. En general un patrón esta formado por uno o mas descriptores. En otras palabras, un patrón es una disposición de descriptores, o características. Una clase de patrones se representan por $\omega_1, \omega_2, ..., \omega_M$, donde M es el número de clases. El reconocimiento de patrones mediante una máquina supone la utilización de técnicas que permitan asignar los patrones a sus respectivas clases, automáticamente y con la menor intervención humana posible.

Los patrones que se trabajan en este proyecto son problemas de clasificación bien conocidos, las bases de datos se obtuvieron de [1] y son:

- **Base de Datos de la Planta Iris**. Esta es quizás la mejor base de datos conocida que se encuentran en la literatura de reconocimiento de patrones. El conjunto de datos contiene 3 clases de 50 casos cada uno, donde cada clase se refiere a un tipo de planta iris. Cada registro muestra 4 atributos de la planta Iris. Una clase es linealmente separable de las otras 2, y las ultimas 2 no son linealmente separables una de otra. Los datos son de tipo Reales.

- **Base de Datos de Diabetes Indios Pima**. Esta base de datos cuenta con información de mujeres indígenas proveniente de la tribu Pima de Arizona, igual o mayores a 21 años de edad. A los indios Pima se les conoce por ser genéticamente predispuestos a padecer diabetes. La base de datos está compuesta por 768 instancias, recibiendo como entrada un conjunto de 8 variables o atributos. Con tipo de datos Enteros y Reales.

- **Base de Datos de Vinos**. Estos datos son el resultado de un análisis químico de los vinos cultivados en una misma región de Italia, pero se derivan de tres cultivares diferentes. El análisis determina las cantidades de 13 componentes (atributos) que se encuentran en cada uno de los tres tipos de vinos. El conjunto de datos contiene 3 clases: Clase 1 con 59 instancias, Clase 2 con 71 instancias y Clase 3 con 48 instancias. Los datos son de tipo Entero y Reales.

2.9. Interfaz de Usuario

La **interfaz de usuario** es el medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto

entre el usuario y el equipo. Normalmente suelen ser fáciles de entender y fáciles de accionar.

2.9.1. Interfaz Gráfica de Usuario

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una computadora.

Habitualmente las acciones se realizan mediante manipulación directa, para facilitar la interacción del usuario con la computadora.

En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

Capítulo 3

Metodología y desarrollo

3.1. Memorias Asociativas Evolutivas

La Memoria Asociativa Evolutiva (MAE) se desarrolla a partir de una perspectiva de "divide y venceras" e inspiradas en el páradigma *coevolutivo* en el que se ve el problema en varias partes y se plantea hallar una solución para cada una de ellas, y que la solución de una parte involucre a la otra.

Considerando los dos procesos básicos de toda MA, el de asociación y el de recuperación, para las MAEs tenemos cada uno de estos procesos en evolución cooperando para una solución común, una coevolución cooperativa, considerando tres aspectos fundamentales: la evaluación de los individuos, la selección o variación en la población, y las aptitudes en cada uno de los procesos evolucionados.

El paradigma coevolutivo inspirado en aspectos biológicos plantea dos aspectos en el proceso de evolución por partes en su interacción, el aspecto de competencia (super vivencia de una sola especie en juego depredador-presa), y el de cooperación (cada una de las especies aporta una parte de la solución) [9]. Con esto en mente, ahora se tienen dos espacios de búsqueda, uno para cada etapa, y se tiene el modelo en tres pasos:

- Primero, se definen manualmente los operadores para la etapa de asociación.
- Segundo, se define el espacio de recuperación al considerar la matriz de asociación formada con cada una de las reglas-operadores de asocición evolucionados en el primer paso.
- Tercero, se implementa el aspecto co-evolutivo que nos resuelve el problema al usar ambos espacios, conectándolos a partir de evolucionar reglas-operadores para la

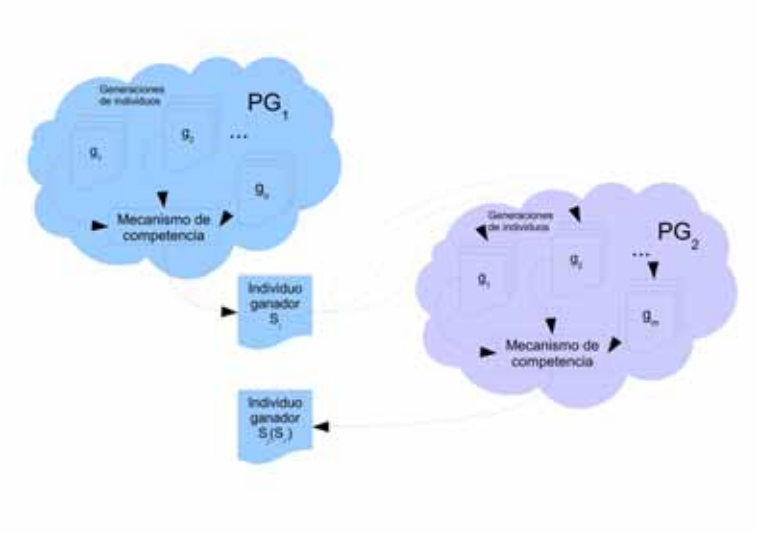

Figura 3.1: Diagrama del modelo Coevolutivo propuesto para el desarrollo de MA mediante PG.

etapa de recuperación.

En la Figura 3.1 se muestra el diagrama de aplicación de esta idea en tres pasos. En la figura se muestra que las soluciones, las nuevas MA, provienen de ambos espacios de búsqueda, y de la interacción de ambos procesos evolutivos, i.e., la influencia del primer proceso evolutivo en el segundo proceso. La función de aptitud del proceso global pretende ser representativa.

Los componentes de este modelo se definen a continuación (véase la Figura 3.2):

- Op_k^a. El operador evolucionado para la asociación de patrones. Este corresponde al genotipo codificado. La matriz de asociación local μ_i se construye al aplicar este operador sobre las componenes de cada par de vectores en la asociación local $\{Y_j, X_j^T\}$

- M_k. La matriz de asociación general, se construye sumando todas las matrices de asociaciones locales µi, proporcionando la acumulación del conocimiento de asociación local (por inspiración del principio del perceptrón).

- T_a. El *Conjunto Terminal* para la etapa de asociación:

$$T_a = \{X_j, Yj\}$$

  Estos nodos son entradas de sus respectivos vectores-patrones $\{X\}$ y $\{Y\}$; esto es para tener codificada la asociación local como una correlación de entrada.

- F_a. El *conjunto de funciones* para la etapa de asociación:

$$F_a = \{+, -, min, max, times\}$$

- Op_p^r. El operador evolucionado para la recuperación de los patrones. Este operador involucra al vector de entrada Xj , así como a la matriz de asociación M generada por el operador evolucionado del primer proceso.

- T_r. El *Conjunto Terminal* para la etapa de recuperación:

$$T_r = \{v, Ren_1, Ren_2, ..., Ren_3, M_k\}$$

  con $v \in X$ como el vector de entrada, y Ren_i como el vector-renglón i-ésimo que pertenece a M_k, tal como se muestra en la Figura 3.3.

- F_r. El *conjunto de funciones* para la etapa de recuperación:

$$F_a = \{+, -, min, max, mytimesm\}$$

  donde *mytimesm* se define como el operador de multiplicación entre las componentes de los vectores como: $mytimesm(X, Y) = [x_1 * y_1, x_2 * y_2, ..., x_n * y_n]$, y cuando se considera en la operación la matriz de asociación en turno, M_k, se define con el orden de operación como: $mytimesm(X, M_k) = X * M_k$; es así que se satisface la condición de dimensiones entre los vectores y la matriz para la operación.

- $\hat{Y}$. Finalmente, el patrón aproximado resultante de Y_j, de la aplicación de la regla-operador evolucionado Op_p^r en la etapa de recuperación.

3.1.1. Asignación de la función de aptitud

El proceso evolutivo demanda la asignación de un valor de *aptitud* para los individuos de cada etapa.

La función de aptitud se aplica para evaluar cada individuo generado Op_k^a de la etapa de asociación entre el conjunto de patrones origen X y el conjunto de patrones destino Y, ambos definen el conjunto fundamental. Una evaluación de aptitud se realiza en cada parte del proceso de la siguiente forma:

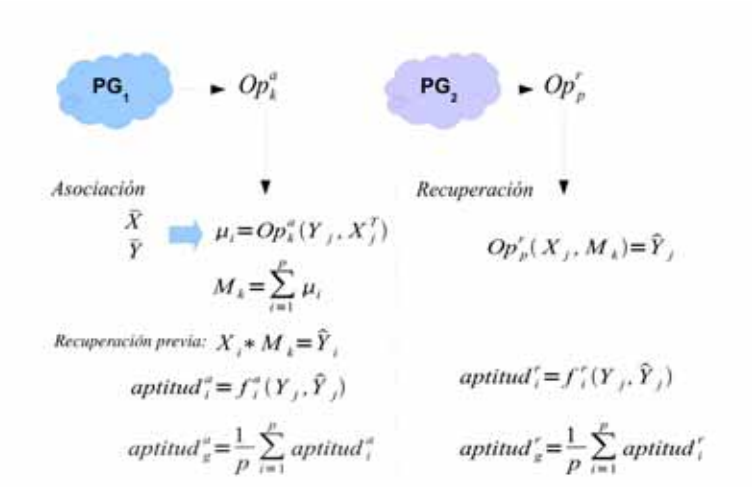

Figura 3.2: Modelo Coevolutivo propuesto para el desarrollo de MA mediante PG.

- Primero, se aplica la asociación entre X y Y.
- Segundo, se realiza una recuperación simple mediante el operador de multiplicación entre la matriz de asociación general generada en esta primera etapa M_k, y el vector de entrada X, con la idea de tener un estimador de aptitud local para la asociación. Los patrones recuperados en esta etapa son $\hat{\hat{Y}}$.
- Tercero, se calcula la aptitud entre los patrones Y y $\hat{\hat{Y}}$ para todas las asociaciones locales $aptitud_i^a$, para luego calcular la aptitud como el promedio global de la asociación $aptitud_g^a$. Este ultimo valor es la aptitud asignada a Op_k^a.

Este proceso se repite en la etapa de recuperación:

- Primero, se calcula la recuperación del patrón asociado a cada vector de entrada X, al usar el operador ganador del proceso evolutivo de recuperación Op_p^r.
- Segundo, se calcula la aptitud para este operador-regla, la aptitud de recuperación local $aptitud_i^r$, para cada par de asociación, asociado a una matriz de asociación M_k generada con cada operador ganador del proceso de asociación Op_p^r.
- Tercero, se realiza el cálculo de la aptitud global promedio, $aptitud_g^r$, como se muestra en la parte inferior derecha de la Figura 3.2. Este valor de aptitud se asigna al operador de recuperación Op_p^r.

$$M_t = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1n} \\ m_{21} & m_{22} & \dots & m_{2n} \\ \vdots & & & \vdots \\ m_{m1} & m_{m2} & \dots & m_{mn} \end{bmatrix} \begin{matrix} \blacktriangleright Ren_1 \\ \blacktriangleright Ren_2 \\ \\ \blacktriangleright Ren_m \end{matrix}$$

Figura 3.3: Matriz de asociación Mt construida en la etapa de asociación, descompuesta en renglones, cadsa uno forma del conjunto terminal para la etapa de recuperación.

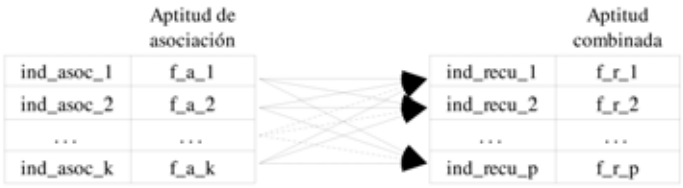

Figura 3.4: Modelo co-evolutivo propuesto de evaluación de la aptitud coevolutiva para el desarrollo de Ma mediante PG.

Finalmente se conforma la cooperación de aptitudes de la siguiente forma (véase la Figura 3.4): Se forman dos tablas, en la primera tabla se disponen los índices de los operadores de asociación Op_k^a, con su respectivo valor de aptidud f_k^a; analogamente en la segunda tabla se disponen los operadores de recuperación Op_p^r con sus valores de aptitud, con un valor de aptitud relacionado con cada uno de los índices de los operadores de asociación, esto es, el operador de recuperación prueba su efectividad conjunta con cada uno de los operadores de asociación, y se toma en "competencia" al que obtenga el valor de fitness más alto con el operador de recuperación en turno, este valor de aptitud ganador se da por la más alta cooperación entre los operadores, y es el que se da en la columna "aptitud combinada" de la tabla.

3.1.2. Funciones de aptitud implementadas

Al iniciarse el desarrollo de las memorias asociativas evolutivas, se consideró como primera funcion de aptitud la propuesta de [25, 41] que difine un cociente para medir kla coincidencia de pixeles de una imagen con otra, f como se muestra en las ecuaciones XXXX y YYYY, que nos da una medida de *similitud* ($0 \leqslant f \leqslant 1$) como el coeficiente de correlación normalizado entre la imagen origen (f) y la imagen destino (g), para nuestro caso lo consideramos entre el vector-patrón obtenido ($\hat{Y}$) y el que seamos aproximar (Y)

$$f = \frac{Y \cdot \hat{Y}}{\sqrt{Y \cdot Y}\sqrt{\hat{Y} \cdot \hat{Y}}} \tag{3.1}$$

con Y y $\hat{Y}$ patrones de dimensión de $1xN$, y el producto $Y \cdot \hat{Y}$ definido como se muestra

$$Y \cdot \hat{Y} = \frac{1}{N} \sum_{j=1}^{N} Y(1,j) \cdot \hat{Y}(1,j) \tag{3.2}$$

El valor óptimo $f = 1$ se tiene cuando todos los píxeles son comunes en ambas imagenes (o matrices), el peor valor $f = 0$ se tiene cuando no se tiene ni un solo valor de píxel en común.

3.2. Interfaz de Usuario

Este proyecto se desarrollo con los siguientes recursos:

- Hardware
  - Computadora Workstation con procesador Intel Xeon de 64 bits, 4GB de RAM y Sistema Operativo Linux Mandriva con kernel de 64 bits.
- Software
  - MathWorks Matlab ver. 6.0 de 64 bits
  - Toolbox GPLab ver. 3.0
  - Bases de datos de la UCI-Machine Learning Repository
  - GraphViz. Software para visualizacion de grafos

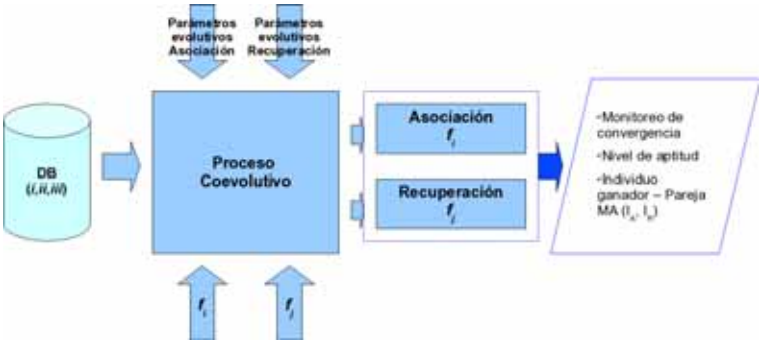

Figura 3.5: Sistema de integración de la metodología de las MA mediante PG.

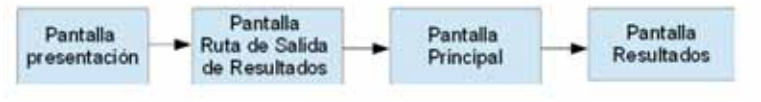

Figura 3.6: Secuencia de Pantallas.

Antes de este proyecto ya se contaba con un programa para la generación de las memorias asociativas evolutivas. En este programa se introducian los datos de manera manual modificando el codigo segun los requerimientos.

Despues de analizar este codigo se logro separar en bloques para poder implementarle a este una interfaz grafica para poder introducir los parametros y jugar con ellos durante la fase de pruebas.

En figuras 3.5 y 3.6 se muestra una diagrama del sistema general

La interfaz gráfica se compone de:

- Pantalla de Presentación.- Muestra los datos de proyecto y el manual de usuario
- Pantalla de Ruta de Resultados.- Es una pantalla para seleccionar la ruta donde se almacenaran los datos de salida de la interfaz
- Pantalla Principal.- Es la parte central de la interfaz, donde se introducen los datos necesarios para la ejecución del programa. Cada que se termina de ejecutar un proceso se puede iniciar uno nuevo.
- Pantalla de Resultados.- Muestra las parejas de individuos ganadores resultantes de la ejecucion del proceso.

Capítulo 4

Resultados experimentales y Conclusiones

4.1. Pruebas

4.1.1. Caso 1. Patrones Binarios

Como primeras Pruebas se hicieron pruebas de las 4 funciones de aptitud en una base con patrones binarios. Los patrones utilizados corresponden a las imagenes en blanco y negro de los numeros naturales. En las Figuras 4.1, 4.2 y 4.3 se muestran los resultados de estas pruebas. Podemos observar que se obtuvo fitness=1.

	Función Arco Coseno	Función Error Cuadrático Medio	Función Suma de las Diferencias Absolutas	Función Métrica Canberra
Caso Autoasociativo	Fitness Global=1	Fitness Global=1	Fitness Global=1	Fitness Global=1

Figura 4.1: Tabla con el fitness global obtenido para las pruebas con cada una de las funciones implementadas.

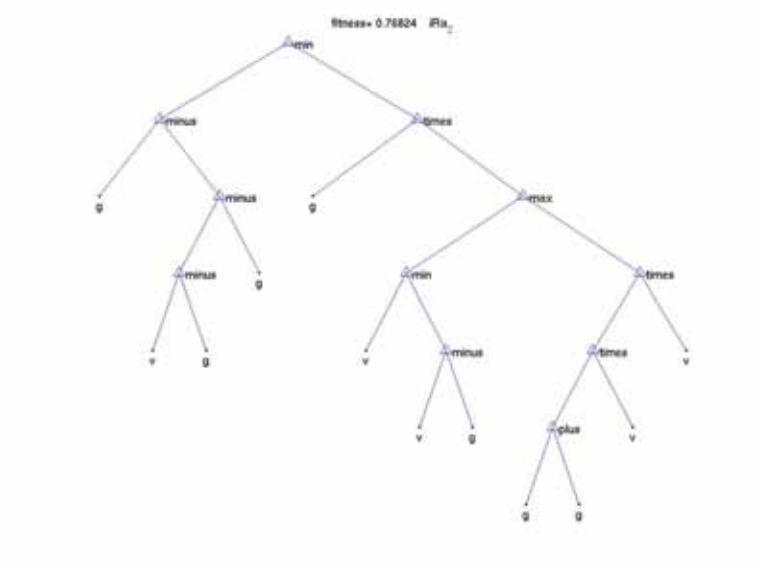

Figura 4.2: Individuo obtenido para asociacion en una de las pruebas.

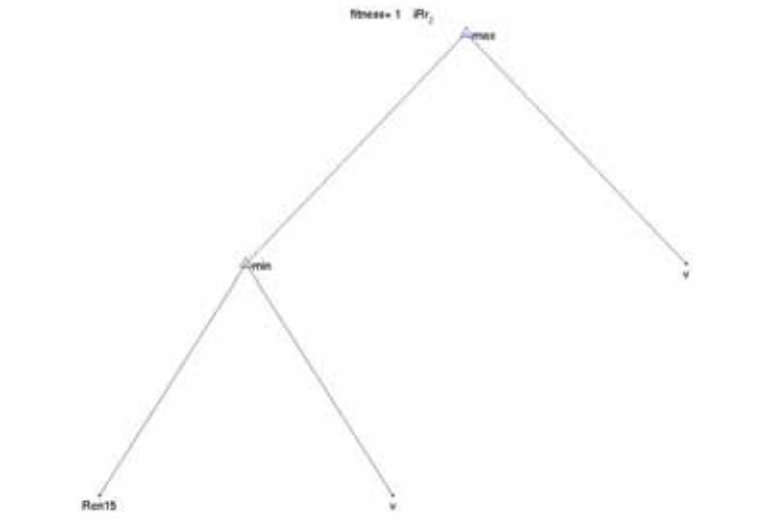

Figura 4.3: Individuo para recuperacion, este es la pareja ganadora junto con la figura anterior.

Bibliografía

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[2] N. Aall Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68, 1954.

[3] R. Barrón. *Memorias asociativas y redes neuronales morfológicas para la recuperación de patrones*. PhD thesis, CIC-IPN, 2006.

[4] R. Barrón and H. Sossa. Extended $\alpha\beta$ associative memories. *Revista Mexicana de Física*, 1(53):10–20, 2007.

[5] Juan Villegas Cortez. *Síntesis Automática de Memorias Asociativas mediante Programación Genética*. PhD thesis, Centro de Investigación en Computación del Instituto Politécnico Nacional (CIC-IPN), Av. Juan de Dios Bátiz, Esq. Miguel Othón de Mendizábal, Col. Nueva Industrial Vallejo. Delegación Gustavo A. Madero, C.P 07738 México D.F., Dec. 2009.

[6] A.K. Dewdney. Computer recreations: The game of life acquires some successors in three dimensions. *Sci. Amer.*, 256(2):16–24, Feb. 1987.

[7] E. Dunn, G. Olague, and E. Lutton. Parisian camera placement for vision metrology. *Pattern Recogn. Lett.*, 27(11):1209–1219, 2006.

[8] S. G. Ficici and J. B. Pollack. Game theory and the simple coevolutionary algorithm: Some preliminary results on fitness sharing. *GECCO 2001 Workshop on Coevolution: Turning Adaptative Algorithms upon Themselves*, 2001.

[9] Chi-Keong Goh and Kay Chen Tan. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 13(1):103–127, February 2009.

[10] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[11] B. Hernández, G. Olague, R. Hammoud, L. Trujillo, and E. Romero. Visual learning of texture descriptors for facial expression recognition in thermal imagery. *Comput. Vis. Image Underst.*, 106:258–269, 2007.

[12] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[13] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[14] Y. Landrin-Schweitzer, P. Collet, and E. Lutton. Interactive gp for data retrieval in medical databases. *Genetic Programming, LNCS 2610. Springer Berlin / Heidelberg*, (2610):93 – 106, Jan. 2003.

[15] D. G. Luenberger. *Programación lineal y no lineal*. Addison-Wesley Iberoamericana, 1989.

[16] G. Olague and C. Puente. Honeybees as an intelligent based approach for 3d reconstruction. *International Conference on Pattern Recognition. August 20-24, 2006. Hong Kong, China.*, Aug. 2006.

[17] G. Olague, E. Romero, L. Trujillo, and B. Bhanu. Multiclass object recognition based on texture linear genetic programming. In *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, pages 291–300, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] C. B. Pérez and G. Olague. Unsupervised evolutionary segmentation algorithm based on texture analysis. In *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, pages 407–414, Berlin, Heidelberg, 2007. Springer-Verlag.

[19] C. B. Perez and G. Olague. Learning invariant region descriptor operators with genetic programming and the f-measure. *International Conference on Pattern Recognition. ICPR.*, 2008.

[20] C. B. Pérez and G. Olague. Evolutionary learning of local descriptor operators for object recognition. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1051–1058, New York, NY, USA, 2009. ACM.

[21] C. B. Pérez and G. Olague. Evolving local descriptor operators through genetic programming. In *EvoWorkshops '09: Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, pages 414–419, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] Potter, A. Mitchell, and Kenneth A. De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation.*, 8(1):1–29, 2000.

[23] C. Puente, G. Olague, S. V. Smith, S. Bullock, M. A. Gonzalez, and A. Hinojosa. Genetic programming methodology that synthesize vegetation indices for the estimation of soil cover. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1593–1600, New York, NY, USA, 2009. ACM.

[24] C. Puente, G. Olague, S. V. Smith, S. H. Bullock, M. A. González-Botello, and A. Hinojosa-Corona. A novel gp approach to synthesize vegetation indices for soil erosion assessment. In *EvoWorkshops '09: Proceedings of the EvoWorkshops 2009 on Applications of Evolutionary Computing*, pages 375–384, Berlin, Heidelberg, 2009. Springer-Verlag.

[25] M. Quintana, R. Poli, and E. Claridge. Morphological algorithm design for binary images using genetic programming. *Genetic Programming and Evolvable Machines*, 7, Issue 1:81–102, 2006.

[26] G. X. Ritter, P. Sussner, and J. L. Díaz de León S. Morphological associative memories. *IEEE Transactions on Neural Networks*, 9(2):281–293, 1998.

[27] G. X. Ritter, G. Urcid, et al. Reconstruction of patterns from noisy inputs using morphological associative memories. *International Journal of Mathematical Imaging and Vision*, 19(2):95–111, 2003.

[28] Hans-Paul Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 1995.

[29] H. Sossa. New associative memories to recall real-valued patterns. *LNCS 3287. Springer Verlag.*, pages 195–202, 2004.

[30] K Steinbuch. Die lernmatrix. *Biological Cybernetics*, 1(1):36–45, Jan. 1961.

[31] P. Sussner. Generalizing operations of binary auto-associative morphological memories using fuzzy set theory. *Journal of mathematical imaging and vision*, 19 (2):81–93, 2003.

[32] K.C. Tan, Y.J. Yang, and T.H. Lee. A distributed cooperative coevolutionary algorithm for multiobjective optimization. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 4, pages 2513–2520 Vol.4, Dec. 2003.

[33] L. Trujillo and G. Olague. Synthesis of interest point detectors through genetic programming. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 887–894, New York, NY, USA, 2006. ACM.

[34] L. Trujillo and G. Olague. Using evolution to learn how to perform interest point detection. In *ICPR (1)*, pages 211–214, 2006.

[35] L. Trujillo and G. Olague. Scale invariance for evolved interest operators. In *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, pages 423–430, Berlin, Heidelberg, Springer-Verlag, 2007.

[36] L. Trujillo and G. Olague. Automated design of image operators that detect interest points. *Evolutionary Computation. MIT Press.*, 16(4):483–507., 2008.

[37] L. Trujillo, G. Olague, F. Fernández, and E. Lutton. Behavior-based speciation for evolutionary robotics. *Genetic and Evolutionary Computation Conference. July 12-16, 2008. Atlanta, GA, USA.*, 2008.

[38] L. Trujillo, G. Olague, P. Legrand, and E. Lutton. Regularity based descriptor computed from local image oscillations. *Opt. Express*, 15(10):6140–6145, 2007.

[39] R. A. Vázquez and H. Sossa. A new model of associative memories network. *Third International Workshop on Artificial Networks and Intelligent Information Processing (ANNIP 2007). Angers, France*, pages 9–12, 2007.

[40] J. Villegas-Cortez, J. H. Sossa, C. Avilés-Cruz, and G. Olague. Evolutionary associative memories through genetic programming. *Revista mexicana de física*, 57:110–116, 2011.

[41] K. Yamamoto, H. Yamada, and I. Yoda. Automatic acquisition of hierarquical mathematical morphology procedures by genetic algorithms. *Image and Vision Computing*, 17 (10):749–760, 1999.

Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencia Básicas e Ingeniería

Ingeniería en Computación

“Manual de usuario de la aplicación ”

Alumna: Pineda León Magdalena

Matricula: 205304176

Trimestre 13-O

Asesores:

Dr. Juan Villegas Cortez
No. económico: 23417

Dr. Carlos Avilés Cruz
No. económico: 24935

Índice de contenido

Introducción

Este manual explica la instalación y ejecución de esta aplicación desarrollada para el manejo de las Memorias Asociativas Evolutivas (MAE).

Una Memoria Asociativa (MA) es un tipo especial de red neuronal artificial (RNA), diseñada para recuperar patrones a partir de patrones de entrada, por medio de operaciones simples y estructuras de calculo reducidas. En una MA la información es almacenada por medio de un proceso de aprendizaje en el que el patrón de entrada X es asociado a un patrón de salida Y. Este proceso de aprendizaje conforma la memoria asociativa, la memoria asociativa es representada por una matriz cuyos componentes m_{ij} pueden considerarse como las conexiones-sinapsis de una RNA simple.

Si asociamos patrones de entrada X con patrones de salida X decimos que la memoria es autoasociativa, ya que los patrones de entrada y salida son los mismos. En caso contrario la memoria es heteroasociativa.

Las Memorias Asociativas Evolutivas (MAE), se auxilian de la Programación Genética y el Paradigma Coevolutivo que dividen el proceso en 2 subprocesos mas sencillos, uno para la Asociación que permite encontrar la mejor regla de asociación para el aprendizaje u otro para obtener una regla de recuperación a partir de la regla de asociación. Cada subproceso inicia creando una población aleatoria de reglas, estas se evalúan mediante una función de aptitud, si ninguno de estos individuos resulta ser una buena solución al problema, se seleccionan algunos individuos mediante algún criterio de selección y se evolucionan mediante cruza o mutación y crean una nueva generación de individuos que nuevamente se evalúan mediante la función de aptitud si ninguno resulta ser una solución buena se repite el proceso hasta que se encuentra una solución adecuada al problema, en este caso una regla se asociación o recuperación.

Esta aplicación permite seleccionar los parámetros necesarios para el manejo de las Memorias Asociativas Evolutivas(MAE), como son: el tipo de memoria Autoasociativa/Heteroasociativa, el tipo de patrones Binarios/Reales, seleccionar los archivos en los que se encuentran almacenados los patrones con los que se desea trabajar, y los parámetros necesarios para cada subproceso evolutivo (Asociación y Recuperación), el número de generaciones, número de individuos por generación y la función de aptitud, estos parámetros no necesariamente deben ser iguales para cada subproceso, de ahí lo interesante de la aplicación ya que al poder variar estos parámetros se pueden analizar los resultados obtenidos principalmente sobre la efectividad de cada función de aptitud (Función Fitness).

Requisitos e instalación del Sistema

Requisitos

1. Para ejecutar esta aplicación, debe tener instalado el software Matlab® de MathWorks, de preferencia la versión 7, ya que en esta versión fue desarrollada la aplicación.
2. Descargar GPLab Toolbox (A Genetic Programming Toolbox for MATLAB by Sara Silva), descomprimir el archivo descargado.
3. Crear los archivos con los patrones a utilizar en la aplicación. Estos archivos pueden ser generados en cualquier Editor de Textos y salvados con extensión .txt, el formato de estos archivos debe ser tipo CSV (del inglés *comma-separated values*), que son un tipo de archivo sencillo y abierto que sirven para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma) y las filas por saltos de línea. Para esta aplicación los datos de las columnas serán separados por un espacio en blanco (en vez de coma o punto y coma). Ejemplo:

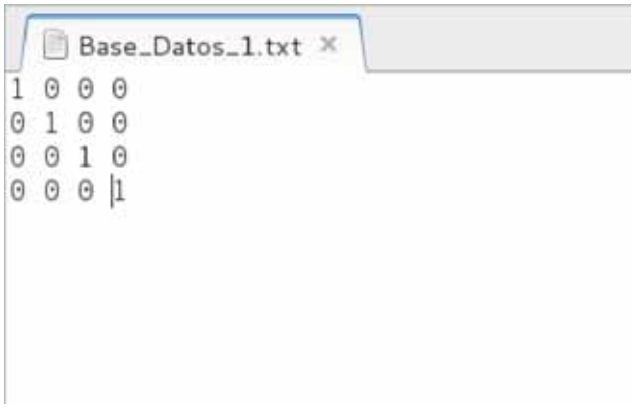

Figura 1: Base de Datos con Patrones Binarios

Instalación

1. El primer paso es crear una carpeta especial para la aplicación, puede ser en la carpeta "Mis documentos/" (si se esta trabajando en entorno Windows) o en "home/user/Documentos/" en el ambiente Linux. En esta carpeta se deben colocaran los archivos necesarios para la ejecución de la aplicación. En la Figura 2, puede observarse un ejemplo de la organización de las carpetas, en un ambiente Linux:

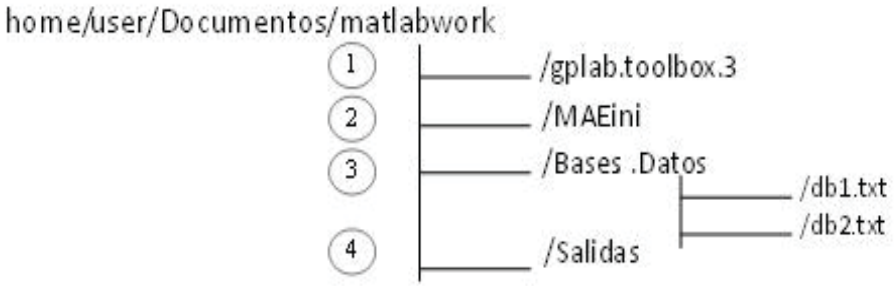

Figura 2: Ejemplo de la organización de las carpetas necesarias para la aplicación

En el ejemplo de la Figura 2, se creo una carpeta llamada “matlabwork/”, dentro de esta carpeta se colocaron las siguientes carpetas:

1. gplab.toolbox.3, esta carpeta se obtiene de la descompresión del archivo mencionado en el punto 2 de los requisitos.
2. MAEini, es la carpeta que contiene los archivos propios de la aplicación
3. Bases.Datos, en esta carpeta se colocan las bases de datos creadas con el formato mencionado en el punto 3 de los requisitos. Guardarlos en esta carpeta es solo una recomendación, las bases de datos pueden guardarse en cualquier otra ubicación.
4. Salidas, la última carpeta se crea manualmente y es vacía en un inicio para guardar los resultados que arroja la ejecución de esta aplicación, a manera de ejemplo en la Figura 2 se llamó “Salidas/” pero puede asignarle otro nombre e incluso otra ubicación.

2. Una vez reunidos los archivos y carpetas necesarios, procedemos a cargarlos en la ruta de trabajo de Matlab, para esto seguimos los siguientes pasos:
   1. Ejecutar Matlab.
   2. Posicionarse en el menu: File → Set Path... (Figura 3), se desplegará una ventana como la de la Figura 4, presionar el botón Add whit Subfolders..., y en la ventana de navegación (Figura 5) buscar la carpeta que creo en el paso anterior y OK, en la Figura 6 se observa como se agrega la carpeta seleccionada y sus subcarpetas, y finalmente Salvar.

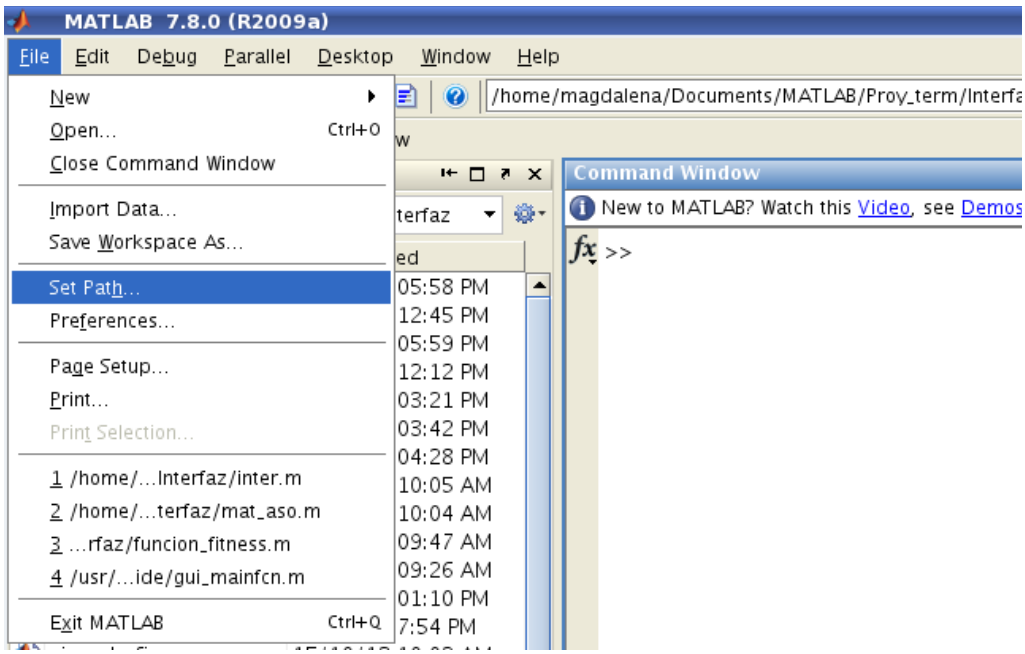

Figura 3: Ubicación del menú Set Path

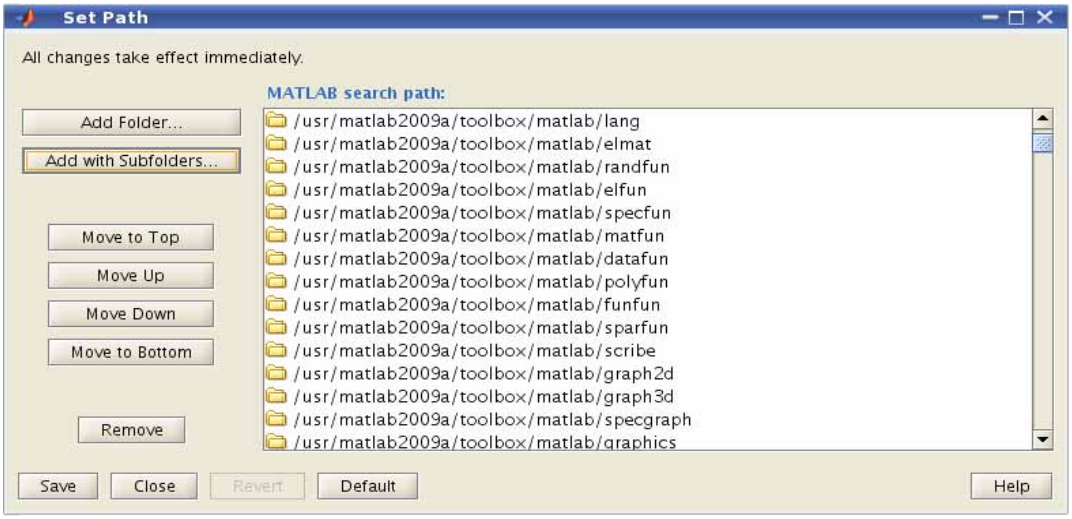

Figura 4: Ventana Set Path

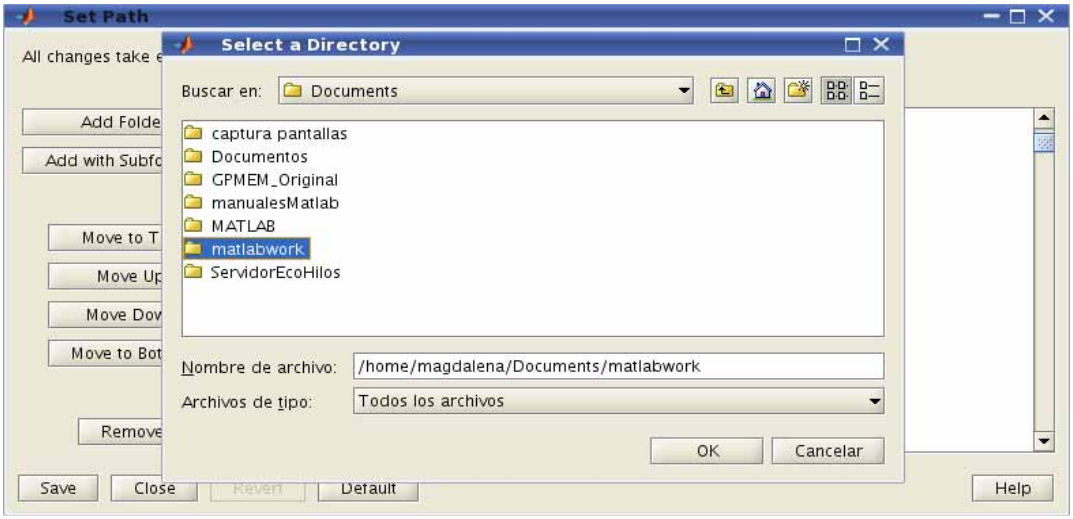

Figura 5: Seleccionar carpeta creada en el punto 1 de la instalación

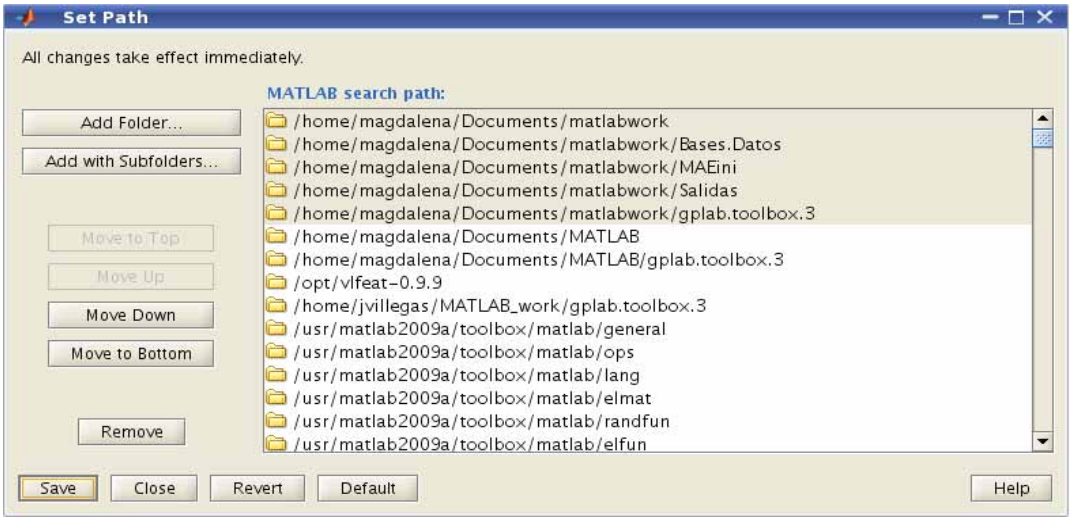

Figura 6: Se agregó al path la carpeta matlabwork y sus subcarpetas

Descripción de la aplicación (Ejecución)

Pantalla Inicio

Para iniciar la ejecución de esta aplicación, teclee en el Command Window de Matlab, el nombre de la aplicación como se muestra en la Figura 7:

Figura 7: Ejecutando la aplicación

Al momento se desplegará la pantalla de presentación de la aplicación. Figura 8:

Figura 8: Pantalla de inicio

Como puede observarse, la ventana (Figura 8) tiene 2 botones:

1. Botón Salir. Si se presiona este botón, aparecerá una ventana de notificación preguntando si realmente desea salir de la aplicación (Figura 9), si elige la opción Si, se detendrá la ejecución de la aplicación, caso contrario, la aplicación seguirá en ejecución.

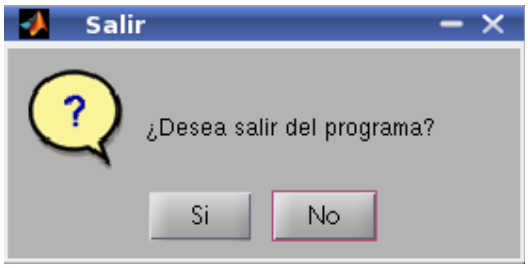

Figura 9: Confirmar salida de la aplicación

2. Botón Continuar. Si se presiona este botón se desplegara la siguiente ventana, para seleccionar la ruta de almacenamiento de los Resultados (Figura 10). A continuación se explica esta ventana.

Pantalla Ruta Resultados

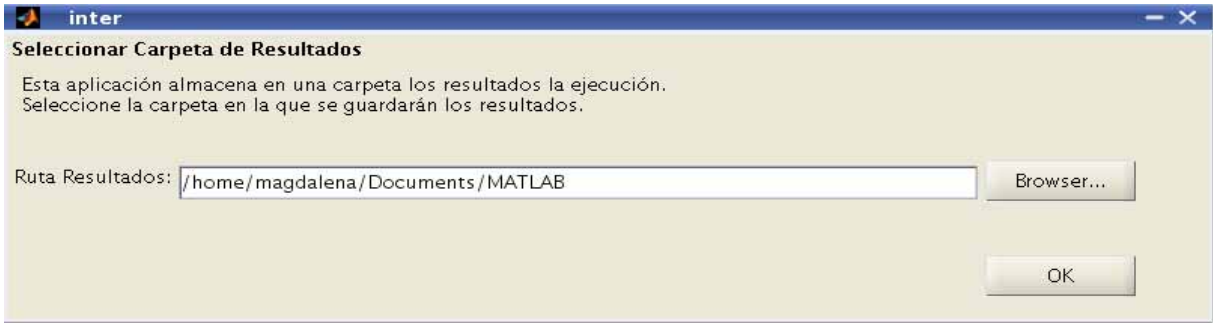

Figura 10: Ruta Resultados

Es necesario que en esta ventana seleccione la ruta para guardar los resultados de la ejecución, esta ruta puede ser modificada mas adelante en el modulo central de la aplicación.

En esta ventana se muestra por default la ruta de usuario, si es que existe, sino se puede elegir la ruta deseada en la ventana de navegación que se despliega al presionar el botón *Browser.* Figura 11

Se recomienda seleccionar la carpeta Salidas, creada en el apartado de instalación, o crear una carpeta donde se desee al momento de navegar por las carpetas de la maquina.

Mientras no seleccione una ruta para los resultados el Botón OK permanecerá deshabilitado y no podrá continuar.

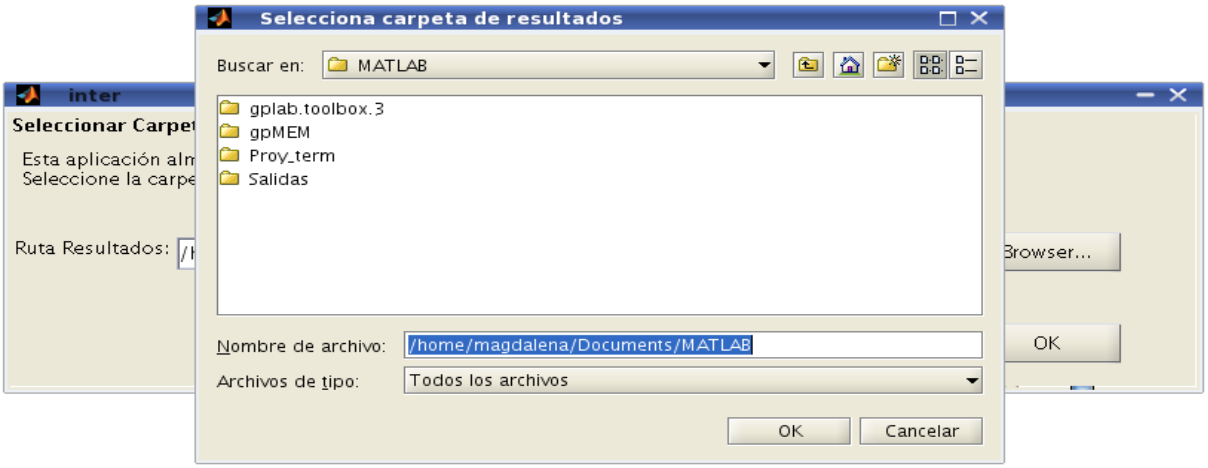

Figura 11: Seleccionando una ruta

Una vez seleccionada una ruta, presione el botón OK que le mostrara la pantalla central de la aplicación.

Pantalla Principal

En la Figura 12 se observa la primera presentación de la Pantalla principal, en la pantalla se observa la primera instrucción “Selecciona tipo de proceso”, esto porque, como se mencionó en un principio con esta aplicación se puede trabajar con 2 tipos de memoria asociativa, Autoasociativa o Heteroasociativa.

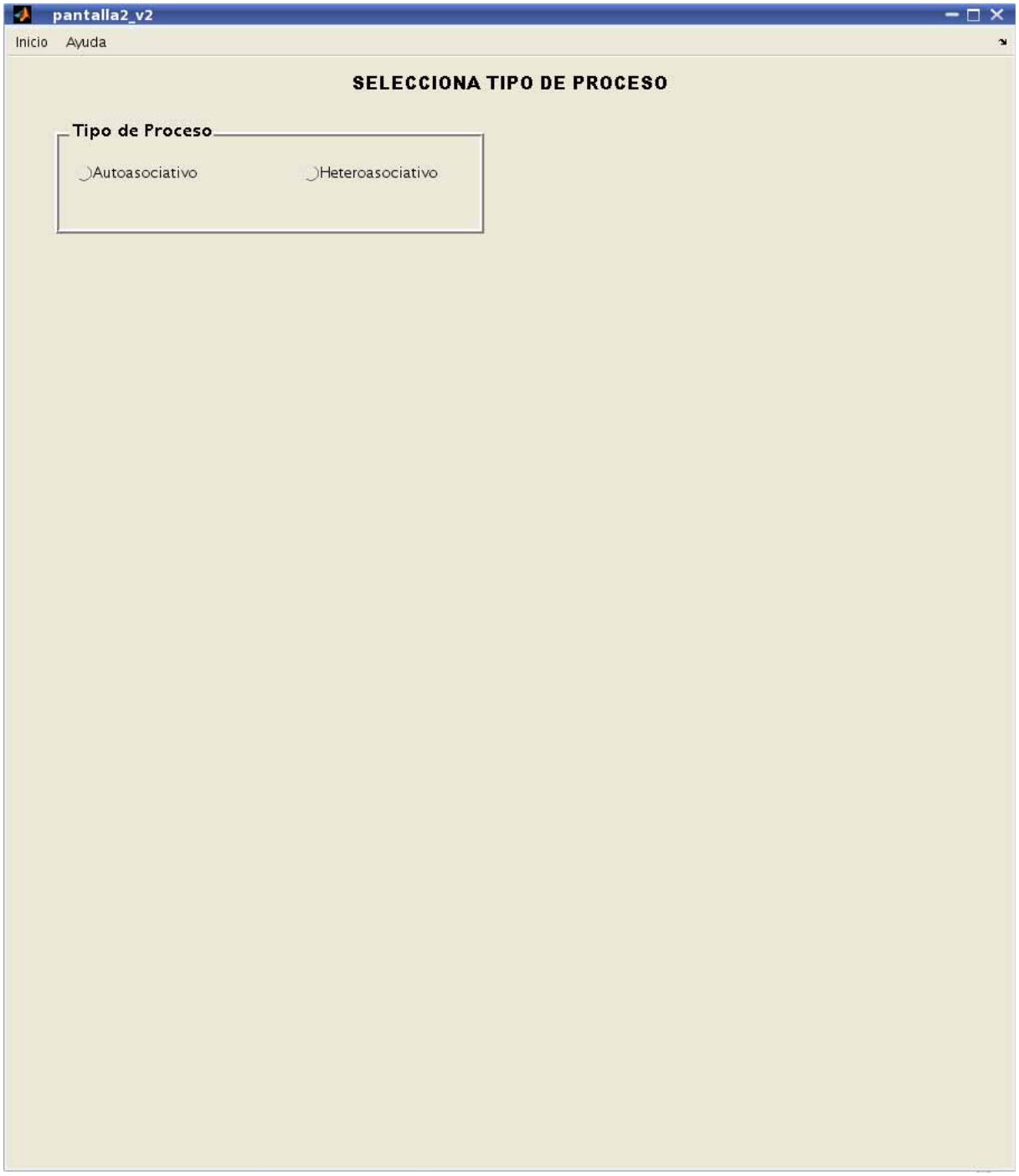

Figura 12: Primera presentación de la pantalla principal

Después de seleccionar el tipo de memoria con el que se desea trabajar se mostraran los campos necesarios para trabajar con el tipo de memoria seleccionado.

Suponiendo que se elige el tipo Autoasociativo, en la Figura 13 se observa la nueva presentación de esta pantalla, con todos los campos necesarios para trabajar con una memoria asociativa evolutiva, de manera Autoasociativa.

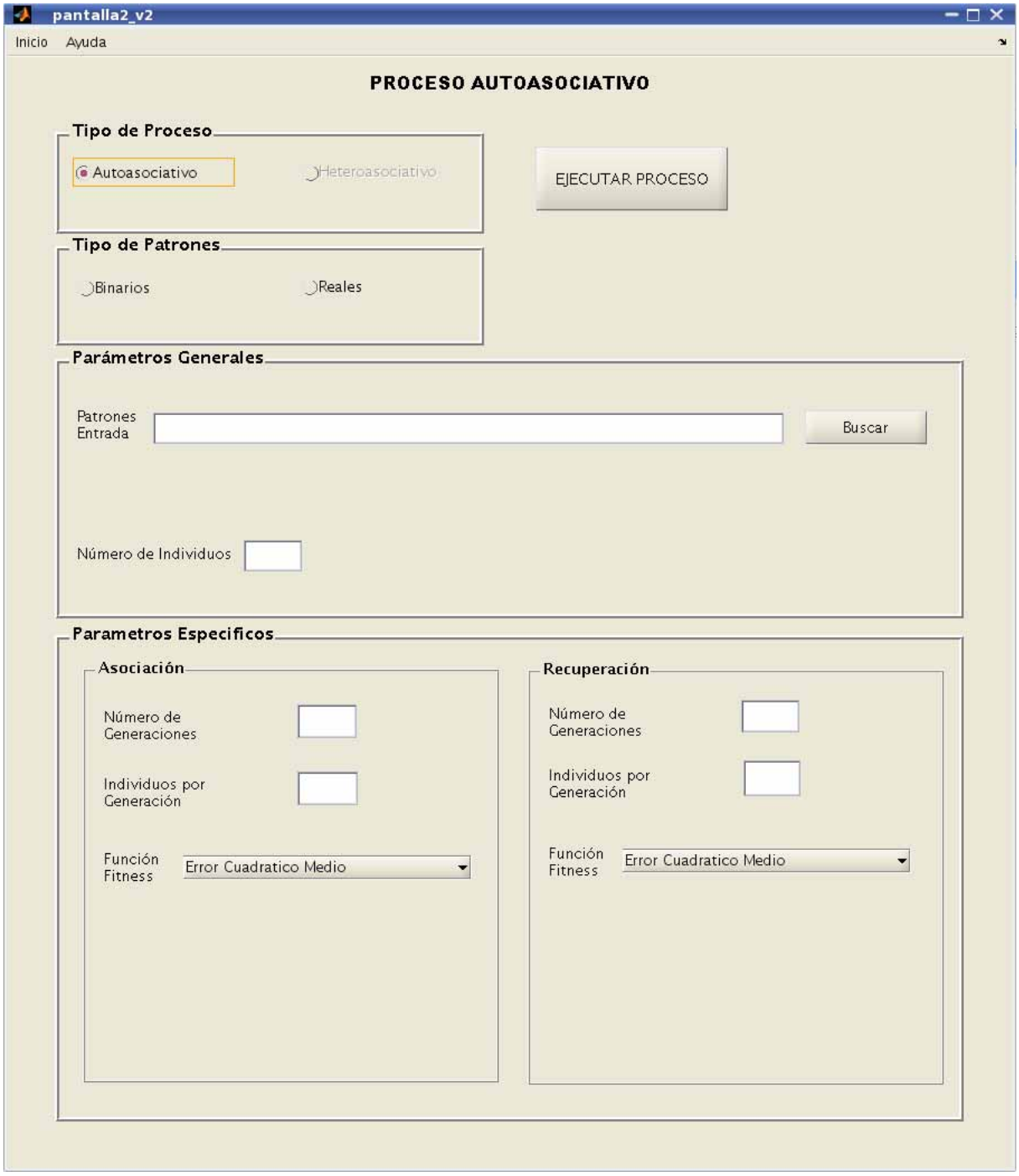

Figura 13: Pantalla principal completa para el proceso Autoasociativo

Ejecución

Para iniciar la ejecución introduzca los datos necesarios (Figura 14):

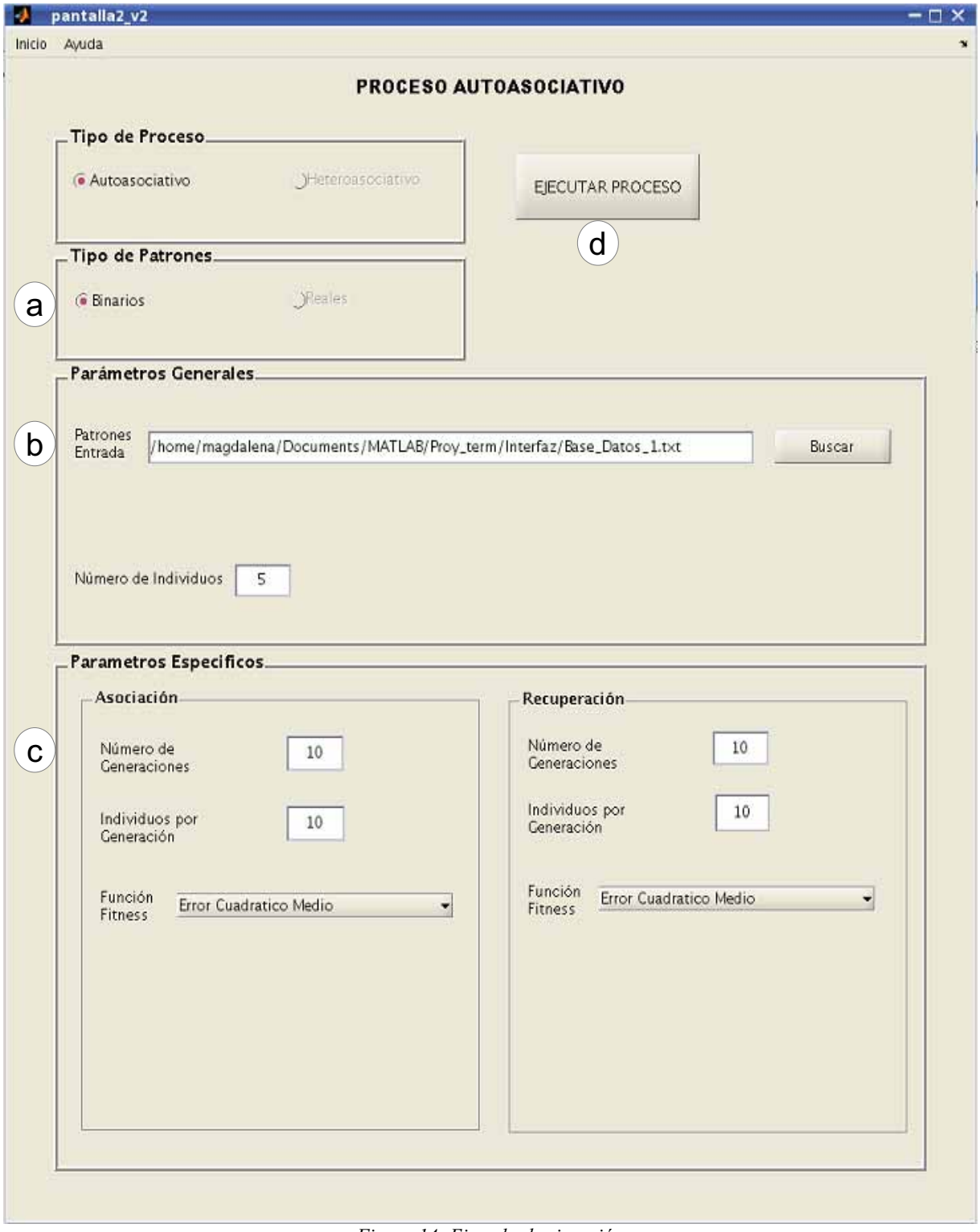

Figura 14: Ejemplo de ejecución

a) **Tipo de Patrones**.- Es necesario e importante seleccionar el tipo de patrones con el que se va trabajar, ya que el tipo de patrones binarios se convierten a notación bipolar (Figura 15), esto porque la memorias asociativas utilizan operaciones como la multiplicación y la división, entonces si utilizamos la notación binaria al multiplicar por cero, todo el resultado se va a cero, y si dividimos por cero el programa nos podría continuar porque la división por cero no esta definida. Por lo tanto internamente lo que sucede en el programa es lo siguiente:

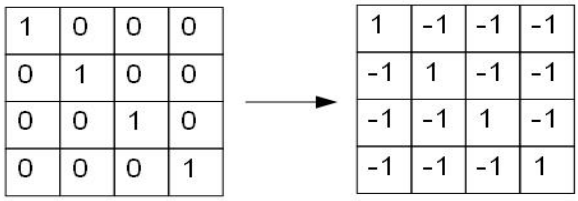

Figura 15: Cambia de notación binaria a notación bipolar

b) **Parámetros Generales**.- En este panel se debe seleccionar el archivo que contiene los patrones, el botón Buscar abre una ventana para seleccionar dicho archivo. Para el caso Autoasociativo solo se necesita 1 archivo, ya que el los patrones de Entrada y Salida son los mismos. En el campo Numero de Individuos se escribe el numero reglas de Asociación y Recuperación que se desea obtener.

c) **Parámetros Específicos.-** este panel se divide en 2, ya que se pueden introducir valores distintos para cada subproceso:

   - *Asociación*.- En este panel se deben introducir el Numero de Generaciones y el Número de Individuos para cada generación, ademas de seleccionar la función fitness que evaluará a los individuos, en esta aplicación los mejores individuos serán calificados con fitness = 1.
   - *Recuperación*.- Al igual que en el punto anterior se deben introducir el Numero de Generaciones y el Numero de Individuos por generación, ademas de seleccionar la función de aptitud para este subproceso.

d) **Ejecutar Proceso**.- Ya que se han introducido los datos necesarios, presione el botón "EJECUTAR PROCESO" y espere a que la ejecución termine.

Al terminarse la ejecución se mostrara un mensaje de notificación "La ejecución ha terminado" y se mostrará en la pantalla principal un nuevo botón llamado "RESULTADOS" (Figura 16). Este botón muestra una nueva pantalla que se detalla mas adelante.

Como ejemplo en la Figura 14 se introdujeron los siguientes datos:

- Tipo de Proceso: Asociativo
- Tipo de Patrones: Binario
- Patrones de Entrada/Salida: Los mostrados en la Figura 1 (Base_Datos_1.txt)
- Numero de individuos: 5
- Para el subproceso de asociación y recuperación se escogieron 10 Generaciones de 10 Individuos por Generación, utilizando como Función Fitness el Error Cuadrático Medio para ambos subprocesos.

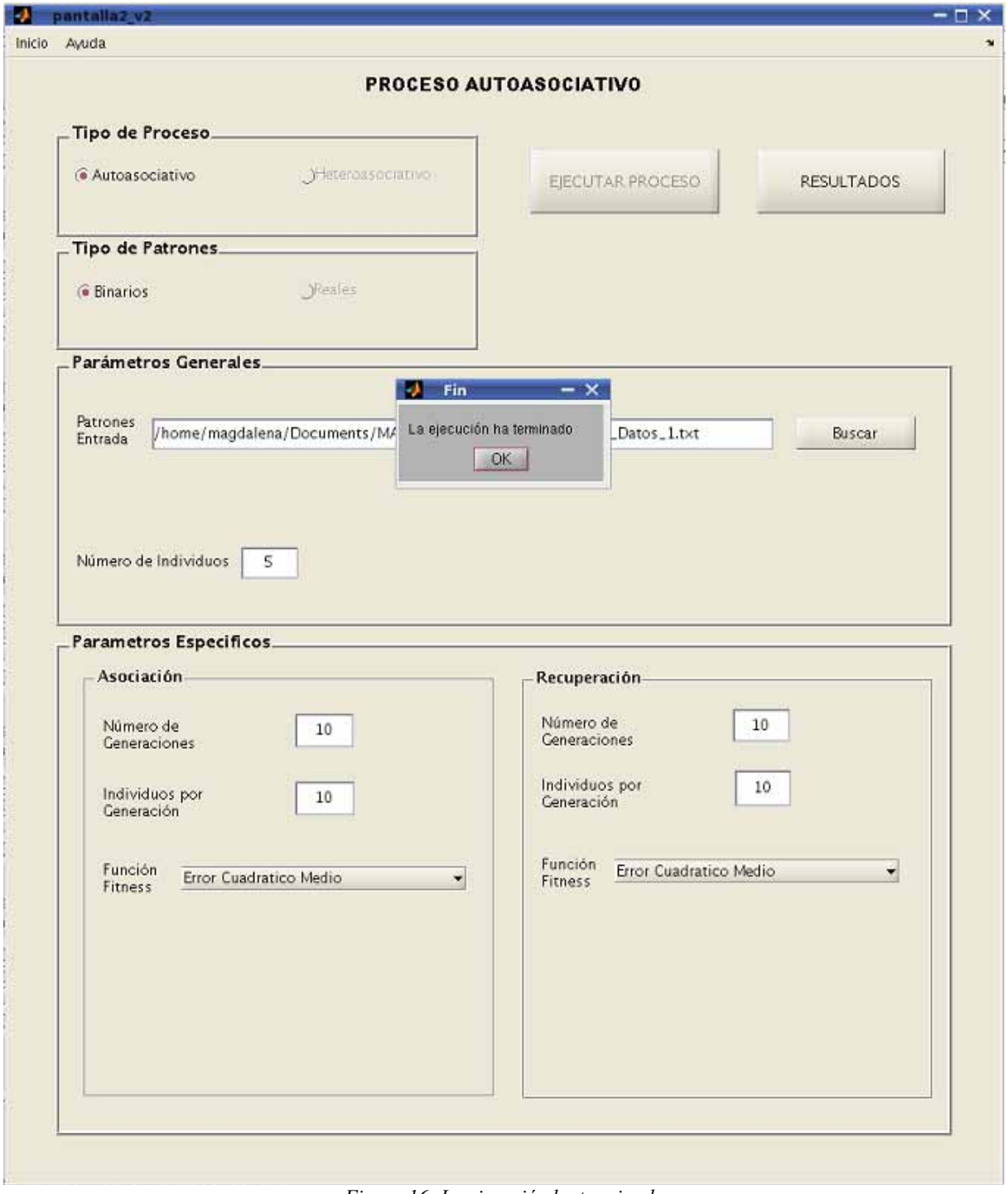

Figura 16: La ejecución ha terminado

En caso de que se hubiese seleccionado el tipo Heteroasociativo, la pantalla tendría una variante en el panel de "Parámetros Generales", ya que para trabajar con una memoria heteroasociativa es necesario seleccionar 2 bases de datos diferentes, Patrones de Entrada y Patrones de Salida, respectivamente (Figura 17).

Figura 17: Panel de Parámetros Generales para una memorias asociativa de tipo Heteroasociativa

Menú

En esta pantalla principal se observa en la parte superior izquierda un pequeño menú:

- Inicio.- Contiene las opciones:
  - Nuevo Proceso (Ctrl+N): Al terminar el proceso y después de observar los resultados obtenidos se puede iniciar un nuevo proceso, con esta opción la pantalla principal se limpia y se ve como en la Figura 12 para iniciar nuevamente. O también puede utilizarse en caso de que se haya seleccionado un Tipo de Proceso (Autoasociativo o Heteroasociativo) y se desee cambiar.
  - Guardar en (Ctrl+G): Se utiliza para cambiar la ruta donde se almacenan los resultados, si es que se desea una ruta diferente cada vez que se ejecute un nuevo proceso. Es recomendable elegir una nueva ruta cada que se inicie un nuevo proceso ya que los resultados de un nuevo proceso sustituirían los archivos generados por el proceso anterior.
  - Salir (Ctrl+S): Termina con la ejecución de la aplicación.
- Ayuda.- Contiene la opción:
  - Manual Usuario (Ctrl+M): Muestra este manual de usuario

Puede observarse que cada menú tiene definidas teclas de acceso directo, para facilitar su uso.

Pantalla Resultados

Esta pantalla muestra las parejas ganadoras, en conjunto una regla de Asociación con una de Recuperación y el fitness global de la pareja. Figura 18 y Figura 19. Estas parejas resultan del subproceso de recuperación donde cada regla de recuperación prueba su efectividad conjunta con cada una de las reglas de Asociación resultantes en el subproceso de asociación, así se elige la de mas alto valor fitness.

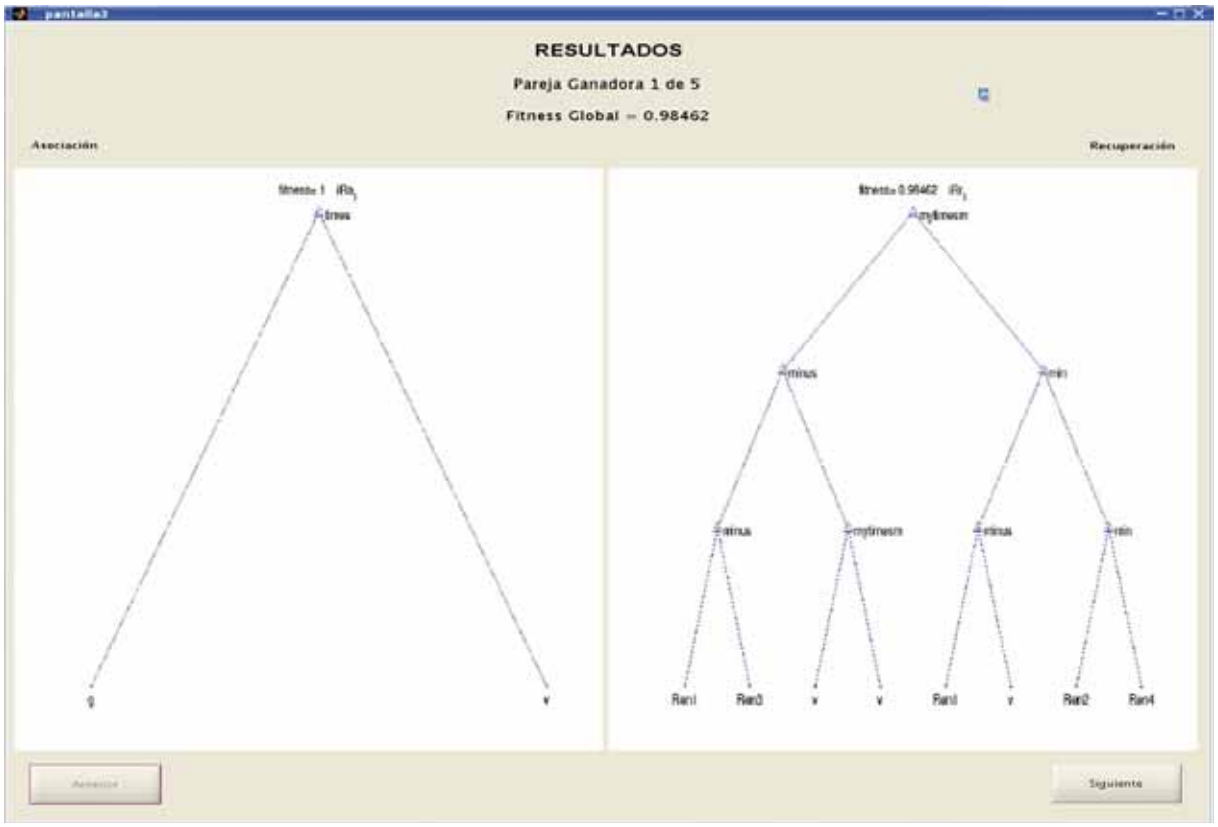

Figura 18: Pantalla Resultados que muestra la primera pareja ganadora del proceso

Por medio de los botones inferiores puede desplazarse hacia adelante o hacia atrás para observar todas las parejas resultantes del proceso.

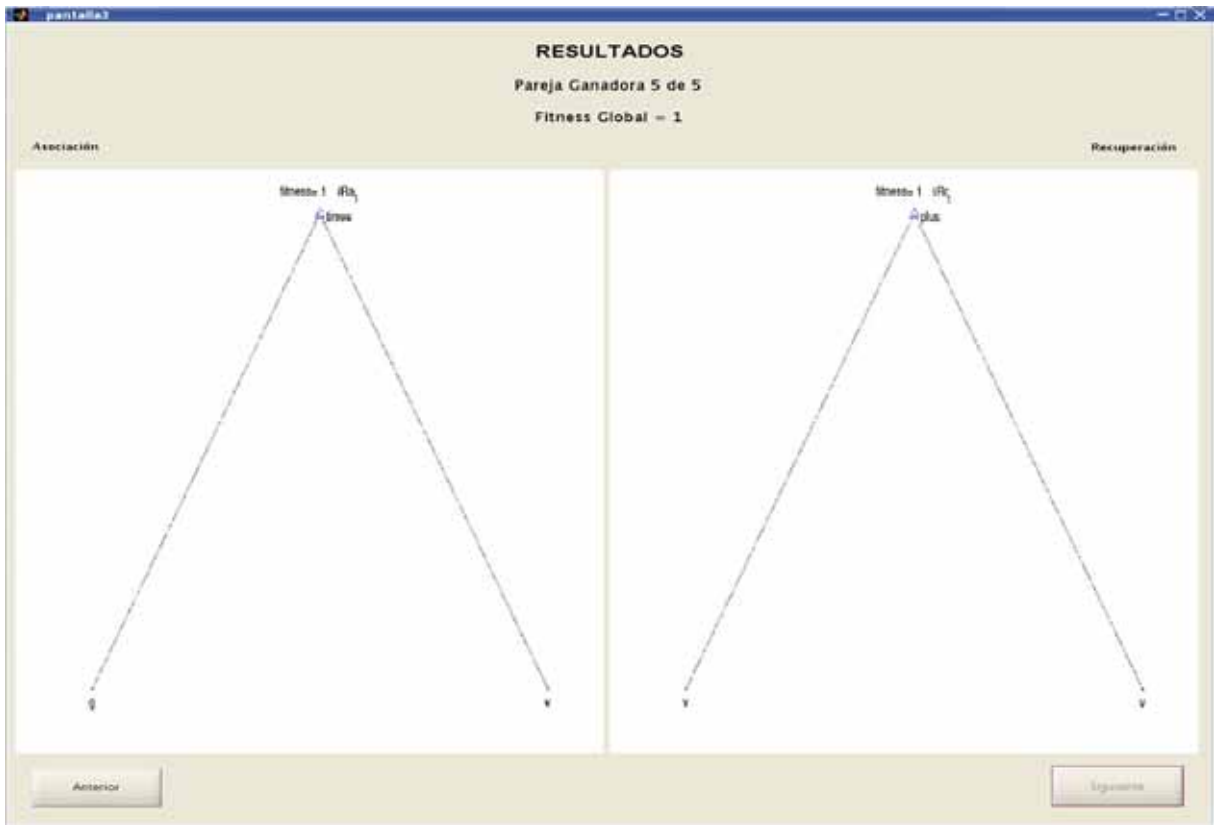

Figura 19: Pantalla de resultados que muestra la ultima pareja ganadora del proceso

Carpeta Salidas

Figura 20: Cargando en Matlab los archivos guardados en la carpeta Salidas (archivos resultados de la ejecución de la aplicación)

La carpeta Salidas creada en el apartado de la instalación es utilizada por la aplicación para guardar los resultados generados durante la ejecución de la aplicación. Los archivos que la aplicación genera son:

- Imágenes de las reglas de Asociación y de las reglas de Recuperación en formato JPEG. Representación de árbol
- Las reglas ganadoras como variables de matlab con extensión .mat. Estas contienen el cada individuo ganador y sis características, como la cadena representada como árbol en el punto anterior, profundidad del árbol, numero de nodos, valor del fitness... etc.
- La tabla que muestran los indices de las parejas ganadoras y el fitness global de la pareja.
- La matriz de asociación de todo el proceso.

Para analizar los archivos .mat seleccione la carpeta Salidas y se mostraran los datos guardados, y con ayuda del Workspace se puede ver lo que contiene cada archivo con extensión .mat