Aprendizaje de Juegos mediante Cúmulos de Partículas con Tamaño de Población Variable

Victoria Leza
Facultad de Informática. Universidad Nacional de La Plata
La Plata, Argentina, 1900
victorialeza@gmail.com

Laura Lanzarini
III-LIDI (Instituto de Investigación en Informática LIDI)
Facultad de Informática. Universidad Nacional de La Plata
La Plata, Argentina, 1900
laural@lidi.info.unlp.edu.ar

Abstract

Game learning has encouraged the development of several Artificial Intelligence strategies. Even though there exist different approaches, the adaptation to the information environment is an expected characteristic when we are to solve a complex problem, since it is unnecessary to count with the codification of any type of initial knowledge.

In this area, neural networks have provided excellent results though with a generally high computational cost. There exists previous work in which population heuristics have been used to reduce the training times.

In particular, the use of evolving strategies based on fixed-size particle swarms has rendered successful results. This paper proposes to modify the size of the population during the adaptation process, adding and deleting individuals in function of their skills to solve the problem.

The proposed method has been applied in a neural networks training to play the game TicTaeToe. The results obtained have been compared to those of a strategy based on particle swarms with fixed-size population. Finally, conclusions and some future lines of work are presented

Keywords: Swarm Intelligence, Particle Swarm Optimization, Neural Networks,

Resumen

El aprendizaje de juegos ha motivado el desarrollo de numerosas estrategias de Inteligencia Artificial. Si bien existen diferentes enfoques, la adaptación al entorno de información es una característica deseable cuando se busca resolver un problema complejo ya que hace innecesaria la codificación de cualquier tipo de conocimiento inicial.

En esta área, las redes neuronales han brindado excelentes resultados pero generalmente con un costo computacional elevado. Existen trabajos previos donde se han utilizado heurísticas poblacionales, para reducir el tiempo de entrenamiento.

En particular, el uso de estrategias evolutivas basadas en cúmulos de partículas de tamaño fijo ha producido resultados satisfactorios. Este artículo propone modificar el tamaño de la población durante el proceso de adaptación, agregando y eliminando individuos en función de su aptitud para resolver el problema planteado.

El método propuesto ha sido aplicado en el entrenamiento de una red neuronal para jugar el juego de TaTeTi. Los resultados obtenidos han sido comparados con los de una estrategia basada en cúmulos de partículas con tamaño de población fija. Finalmente se exponen las conclusiones así como algunas líneas de trabajo futuras.

Palabras Claves: Optimización mediante Cúmulos de Partículas (PSO), Redes Neuronales

1. Introducción

Desde los principios de la Inteligencia Artificial, los investigadores han intentado desarrollar estrategias que les permitan a las computadoras aprender de una manera automática. En general se busca crear algoritmos capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. De esta manera, a semejanza del aprendizaje de los seres humanos, para aprender no se requiere conocer la solución específica del problema sino que la misma es extraída de la información disponible. Si bien estas estrategias poseen una amplia gama de aplicaciones, el aprendizaje de distintos tipos de juegos ha sido motivo de inspiración continua y aun lo sigue siendo.

Muchos algoritmos de juegos tradicionales basan su funcionamiento en un árbol de jugadas. Esta estructura les permite seleccionar el próximo movimiento a realizar. Sin embargo, más allá de las complicaciones que implica la construcción del árbol y los esfuerzos realizados tanto para reducir su tamaño como para mejorar las estrategias sobre él aplicadas, resultan inapropiados cuando el espacio de búsqueda es amplio y complejo.

Como alternativa a este tipo de soluciones se propone el uso de estrategias inteligentes que permitan desarrollar agentes capaces de aprender un juego basados únicamente en sus reglas y teniendo conocimiento del estado actual del mismo.

En esta dirección existen soluciones basadas en Redes Neuronales que han producido excelentes resultados pero generalmente con un costo computacional elevado. Existen trabajos previos donde se han utilizado heurísticas poblacionales, para reducir el tiempo de entrenamiento. En particular, la optimización mediante cúmulos de partículas (PSO - Particle Swarm Optimization) ha producido resultados satisfactorios en una amplia gama de problemas de optimización, incluyendo el entrenamiento de redes neuronales y la minimización de funciones. En todos los casos, el tamaño de la población utilizada permanece fijo durante el proceso adaptativo.

Este artículo se propone diseñar un agente basado en una red neuronal entrenada mediante cúmulos de partículas con tamaño de población variable donde los individuos serán incorporados y eliminados en función de su aptitud para resolver el problema planteado. Esto se realiza a través del concepto de edad que permite determinar el tiempo de vida de cada elemento de la población. Si bien existen distintas formas para estimar el tiempo que cada individuo permanecerá dentro de la población, se utilizará el definido en [Lan01] ya que ha demostrado ser capaz de considerar adecuadamente la aptitud de los individuos.

Con la intención de no poblar excesivamente un mismo lugar del espacio de soluciones, se utilizará una técnica de nichos de construcción dinámica definida en [Bir06]. En ella, se realiza un análisis estadístico de la población para definir la distancia adecuada entre dos individuos que pertenecen al mismo nicho.

Este artículo se encuentra organizado de la siguiente forma: …

2. Algoritmos basados en Cúmulos de Partículas

Un algoritmo basado en Cúmulos de Partículas, también llamado Particle Swarm Optimization (PSO), es una técnica heurística poblacional donde cada individuo representa una posible solución del problema y realiza su adaptación teniendo en cuenta tres factores: su conocimiento sobre el entorno (su valor de fitness), su conocimiento histórico o experiencias anteriores (su memoria), el conocimiento histórico o experiencias anteriores de los individuos situados en su vecindario [Ken95]. Su objetivo es evolucionar su comportamiento de manera de asemejarse a los individuos con más éxito dentro de su entorno. En este tipo de técnica, cada individuo permanece en continuo

movimiento dentro del espacio de búsqueda y nunca muere. Por su parte, la población puede verse como un sistema multiagente donde cada individuo o partícula se mueven dentro del espacio de búsqueda guardando y eventualmente comunicando, la mejor solución que han encontrado.

Existen distintas versiones de PSO; las más conocidas son *gBest PSO* que utiliza como criterio de vecindad a la población completa y *lBest PSO* que, por el contrario, utiliza un tamaño de vecindad pequeño [Ken95][Shi99]. El tamaño de la vecindad influye en la velocidad de convergencia del algoritmo así como es la diversidad de los individuos de la población. A mayor tamaño de vecindad, la convergencia del algoritmo es más rápida pero la diversidad de individuos es menor.

Cada partícula p_i está compuesta por tres vectores y dos valores de fitness:

- El vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ almacena la posición actual de la partícula en el espacio de búsqueda.
- El vector $pBest_i = (p_{i1}, p_{i2}, \ldots, p_{in})$ almacena la posición de la mejor solución encontrada por la partícula hasta el momento.
- El vector velocidad $v_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ almacena el gradiente (dirección) según el cual se moverá la partícula.
- El valor de fitness $fitness\_x_i$ almacena el valor de aptitud de la solución actual (vector x_i).
- El valor de fitness $fitness\_pBest_i$ almacena el valor de aptitud de la mejor solución local encontrada hasta el momento (vector $pBest_i$)

La posición de una partícula se actualiza se la siguiente forma

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (1)$$

Como se explicó anteriormente, el vector velocidad se modifica teniendo en cuenta su experiencia y la de su entorno. La expresión es la siguiente:

$$v_{ij}(t+1) = w \,.\, v_i(t) + \varphi_1 \,.\, rand_1 \,.\, (pBest_i - x_i(t)) + \varphi_2 \,.\, rand_2 \,.\, (g_i - x_i(t)) \quad (2)$$

donde w representa el factor de inercia [Shi98], φ_1 y φ_2 son constantes de aceleración, *rand1* y *rand2* son valores aleatorios pertenecientes al intervalo (0,1) y g_i representa la posición de la partícula con el mejor *pBest_fitness* del entorno de p_i (*lBest* o *localbest*) o de todo el cúmulo (*gBest* o *globalbest*). Los valores de w, φ_1 y φ_2 son importantes para asegurar la convergencia del algoritmo. Para más detalles sobre la elección de estos valores puede consultar [Cle02] y [Ber02].

La figura 1 contiene el algoritmo PSO básico

```
S ← InicializarCumulo()
while no se alcance la condición de terminación do
    for i = 1 to size(S)  do
       evaluar la partícula xi del cúmulo S
       if fitness(xi) es mejor que fitness(pBesti) then
             pBesti ← xi; fitness(pBesti) ← fitness(xi)
        end if
    end for
    for i = 1 to size(S)  do
        Elegir gi según el criterio de vecindad utilizado
        vi ← w . vi + φ1 . rand1 . (pBesti – xi) + φ2 . rand2 . (gi – xi)
        xi ← xi + vi
    end for
end while
Salida : la major solución encontrada
```

Figura 1

3. Cúmulos de Partículas con tamaño de población variable

El algoritmo descripto en la sección anterior trabaja sobre una población de tamaño fijo. En este artículo se propone utilizar un cúmulo de tamaño variable buscando mejorar la relación de compromiso existente entre la velocidad de convergencia y la diversidad de la población.

La variación en el tamaño de la población se logra permitiendo que las partículas se reproduzcan y mueran durante el proceso de adaptación. Esto requiere la definición de dos criterios: uno que permita saber si un individuo existente puede generar nuevos descendientes y otro que indique como quitar elementos de la población.

3.1. Criterio de reproducción

La reproducción busca incrementar la velocidad de convergencia incorporando individuos en las zonas menos pobladas. Debe ser aplicada con moderación a fin de no aumentar excesivamente la cantidad de individuos dentro del espacio de búsqueda. Por lo tanto, es preciso analizar la distribución de las partículas dentro de dicho espacio para luego determinar si la reproducción puede llevarse a cabo o no.

Para calcular la probabilidad de reproducción de cada una de las n partículas de la población S, se analizará su entorno tomando un radio de distancia r calculado de la siguiente forma [Bir06]:

$$d_i = \min\{\| x_i - x_j \| \; ; \forall j \; x_i, x_j \in S \; \wedge \; x_i \neq x_j\} \qquad \text{con } i = 1..n \tag{3}$$

$$r = \frac{\sum_{i=1}^{n} d_i}{n} \tag{4}$$

El valor de r a utilizar es el mismo para todas las partículas.

La probabilidad de reproducción de la i-ésima partícula se calcula de la siguiente forma [Fer05]:

$$Prob_i = 1 - (CantVecinosExistentes_i \; / \; MAX\_CANTIDAD) \quad \text{con } i = 1..n \tag{5}$$

siendo *CantVecinosExistentes* la cantidad de partículas encontradas dentro del entorno del i-ésimo individuo de la población S y *MAX_CANTIDAD* es un parámetro del algoritmo que representa la máxima cantidad de individuos permitida.

Para aquellos casos en que la medida de distancia de (3) no resulte apropiada para la representación del espacio de búsqueda seleccionada, pueden verse otras alternativas en [Ert02].

Cada individuo de la población, según su probabilidad de reproducción, podrá generar un único descendiente aplicando mutación uniforme sobre los pesos de la red neuronal y copiando en el nuevo individuo su vector velocidad y su conocimiento social.

3.2. Cálculo del tiempo de vida

La permanencia de los individuos dentro de la población depende de su tiempo de vida. Dicho valor se expresa en cantidad de iteraciones, transcurridas las cuales, la partícula es eliminada. Este valor tiene una estrecha relación con la aptitud de la partícula y permite que los mejores individuos permanezcan en la población por mayor tiempo, influenciando el comportamiento del resto.

Para estimar el tiempo de vida de cada individuo de la población de utilizó el método de asignación por clases definido en [Lan01] ya que ha demostrado poseer la capacidad de brindar buenos resultados con una cantidad de individuos muy inferior a la empleada por los métodos convencionales.

En [Lan01] los individuos de la población son agrupados según su valor de aptitud en *k* clases utilizando un método de clustering competitivo del tipo *winner-take-all*, por ejemplo k-medias.

Sobre el resultado de este agrupamiento puede aplicarse uno de los siguientes métodos:

a) Asignación de tiempo de vida fijo por clase

Se divide el máximo tiempo de tiempo de vida a asignar por la cantidad de clases, *k*. Esto permite saber el rango de tiempo que le corresponde a cada clase. Dentro de una misma clase, sus individuos recibirán un tiempo de vida proporcional a la clase a la que pertenecen y a la cantidad de individuos que se encuentran en su misma clase, de la siguiente forma:

```
AnchoClase := MAX_LT / k
TVClasesAnt := (ClaseMasCercana – 1)* AnchoClase
TVClaseActual  := AnchoClase
Desplazamiento  = ( fitness[i] – Clase[ClaseMasCercana].MinFit ) /
             abs(Clase[ClaseMasCercana].MaxFit – Clase[ClaseMasCercana].MinFit))
TiempoDeVida[i] := trunc(TVClasesAnt + WidthClass * Desplazamiento)
```

donde

- *AnchoClase* es el rango del tiempo de vida asignado a cada clasen (dada por MAX_LT / Número de clases)
- *ClaseMasCercana* es el número de clase a la que pertenece cada individuo.
- *TVClaseActual* es el rango de tiempo de vida de la clase a la pertenece el individuo.
- *TVClasesAnt* es el rango de tiempo de vida asignado a las clases anteriores a *ClaseMasCercana.*
- Clase[*ClaseMasCercana*].MinFit y Clase[*ClaseMasCercana*].MaxFit son los valores de aptitud mínimo y máximo de la clase a la que pertenece el individuo en consideración.
- *Fitness[i]* es el valor de aptitud del i-ésimo individuo de la población.

b) Asignación de tiempo de vida proporcional a la cantidad de individuos de cada clase

Cada clase recibe un rango de tiempo de vida proporcional a la cantidad de elementos que contiene. Es decir, que los individuos pertenecientes a las clases numerosas podrán tener un rango de tiempo de vida más amplio. El cálculo es el siguiente:

```
TotalAnt := 0
for i:=1 to ClaseMasCercana-1 do TotalAnt := TotalAnt + Clase[i].Cantidad
TVAnterior := MAX_LT * TotalAnt / TotalIndiv
TVClaseActual  := MAX_LT * Clase[ClaseMasCercana].Cantidad/TotalIndiv
Desplazamiento  = ( fitness[i] – Clase[ClaseMasCercana].MinFit ) /
             abs(Clase[ClaseMasCercana].MaxFit – Clase[ClaseMasCercana].MinFit))
TiempoDeVida[i] := trunc(TVAnterior + WidthClass * Desplazamiento)
```

donde

- *Clases[ClaseMasCercana].Cantidad* representa la cantidad toal de individuos de la clase más cercana.
- *TotalIndiv* es la cantidad total de individuos de la población.

Estas dos formas de calcular los tiempos de vida de los individuos deben combinarse a fin de lograr una asignación correcta. Se propone aplicar la asignación b) durante un cierto porcentaje de la cantidad de generaciones máximas del algoritmo y en las restantes utilizar a). Esto se debe a que los agrupamientos iniciales se realizan sobre individuos que aun no están lo suficientemente adaptados y por consiguiente dan lugar a agrupamientos de tamaños muy disímiles. Si sobre estos

agrupamientos se aplicara directamente la distribución indicada en a) muchos individuos recibirían tiempos de vida similares llevando al algoritmo a incrementar innecesariamente la cantidad de individuos de la población.

3.3. Algoritmo propuesto

El algoritmo comienza con una población de N individuos generados al azar dentro del espacio de búsqueda y calcula para cada uno de ellos su fitness y tiempo de vida correspondientes.

Una vez medida la aptitud de todos los individuos de la población se calcula el radio según la ecuación (4). Esta medida será utilizada para estimar la cantidad de vecinos de cada individuo y tendrá una fuerte incidencia en la probabilidad de reproducción de cada uno de ellos ya que se utilizará la ecuación (5).

La inercia utilizada para actualizar los vectores velocidad es ajustada durante el algoritmo de la siguiente forma [Mei02]:

$$w = w_{start} - \frac{(w_{start} - w_{end})}{ITERACIONES\_TOTALES}.IteracionActual \quad (6)$$

donde w_{start} es el valor inicial de w y w_{end} es el valor final.

El uso de un peso de inercia variable facilita la adaptación de la población. Un valor de w alto al comienzo de la evolución le permite a las partículas realizar movimientos grandes ubicándose en distintas posiciones del espacio de búsqueda. A medida que avanza el número de iteraciones, el valor de w se reduce permitiéndoles realizar un ajuste más fino.

El cálculo del valor de fitness depende del tipo de problema a resolver y será explicado con detalle en la sección de resultados.

En cada iteración y utilizando la probabilidad de reproducción de cada individuo, existe la posibilidad de que se incorporen nuevos. Si esto ocurre, se define una nueva red neuronal copiando la red del padre (individuo original) con una mínima mutación uniforme y manteniendo el mismo vector velocidad.

Finalmente, el algoritmo termina cuando se cumple una de las siguientes condiciones:

- Se alcanzó la cantidad máximas de iteraciones indicadas inicialmente
- El mejor individuo no se ha modificado durante el 20% de las iteraciones totales
- La población se ha quedado sin individuos.

La figura 2 contiene un pseudocódigo del algoritmo descripto.

4. Resultados

El algoritmo propuesto en este trabajo fue utilizado para entrenar una población de redes neuronales para jugar el juego de TaTeTi.

Es decir que cada individuo es una red feedforward totalmente interconectada con 9 entradas, una para cada posición del tablero e igual número de salidas. La función de transferencia utilizada en la capa oculta y en la capa de salida fue la sigmoidal, con la distinción que las salidas fueron redondeadas a modo de que pudiera interpretarse como una variable booleana. Se utilizó una codificación especial para los valores de entrada para permitirle a la red interpretar cuáles son sus posiciones ocupadas, cuáles las libres y cuáles las ocupadas por su oponente en el tablero, estando cada uno representado con el valor 1, 0 y -1, respectivamente. Sus pesos fueron adaptados mediante

el proceso evolutivo. Las mediciones realizadas acerca de la cantidad de neuronas de la capa oculta demuestran que los mejores resultados se obtienen utilizando entre 5 y 7 unidades.

```
// --- Población Inicial ---
S ← {}
for i = 1 to N  do
   x_i ← Red Neuronal inicializada al azar con pesos ∈ (-1,1).
   v_i ← inicializar al azar entre el rango mínimo y máximo especificado.
   evaluar el fitness de la partícula x_i
   pBest_i ← x_i
   fitness(pBest_i) ← fitness(x_i)
   S ← S ∪ { (x_i, v_i, pBest_i, fitness(x_i), fitness(pBest_i)) }
end
Calcular el Tiempo de Vida de cada partícula según 3.2.b)

w ← INERCIA_MAXIMA
while no se alcance la condición de terminación do
    Calcular el radio r según la ecuación (4)
    for i = 1 to size(S)  do
       Elegir LBest_i,la partícula con mejor fitness del entorno de x_i
       v_i ← w . v_i + φ1 . rand_1 . (pBest_i - x_i) + φ2 . rand_2 . (LBest_i - x_i)
       x_i ← x_i + v_i
       Prob_i ← Calcular la probabilidad de reproducción de x_i
       //-- Reproducción --
       if rand < Prob_i then
           Obtener x'_i aplicando mutación uniforme sobre x_i
           Calcular el fitness de x'_i
           S ← S ∪ { (x'_i, v_i, pBest_i, fitness(x'_i), fitness(pBest_i)) }
            if (IteracionActual >  al 20% de las ITERACIONES_TOTALES))
              Calcular el Tiempo de Vida de x'_i según 3.2.a)
            Else
              Calcular el Tiempo de Vida de x'_i según 3.2.b)
        end if

        Disminuir en 1 el tiempo de vida de x_i
        Si Tiempo_de_vida(x_i)=0 then
               S ← S - { (x_i, v_i, pBest_i, fitness(x_i), fitness(pBest_i)) }
    end for
    w ← modificar dinámicamente la inercia
end while
```

Figura 2

Para evaluar el fitness de cada partícula se la hace jugar 7 veces contra otros individuos de la población elegidos al azar. En cada una de estas competencias también se selecciona al azar la partícula que comienza el juego. Cada red recibirá como resultado de cada enfrentamiento un puntaje que se irá acumulando a lo largo de las 7 pruebas. Los valores asignados son: 1 si gana el partido, -2 si pierde y 0 si empata. Aquellas situaciones en las que el juego no se haya resuelto en un número máximo de jugadas indicado a priori, serán consideradas equivalentes a haber perdido, es decir, que el puntaje asignado será -2. Esto último se relaciona con la aplicación de este algoritmo a otro tipo de juego de tablero distintos del TaTeTi. En este caso particular el límite de jugadas impuesto para este caso es el doble del tamaño del tablero, es decir, 18.

Los errores cometidos por la red durante el entrenamiento son considerados de manera aislada. Se considera error a aquellas situaciones en las que la red no brinda una jugada válida ya sea porque sugiere jugar en una posición ocupada o bien porque no indica ninguna salida. En estos casos se suman al total de errores de la res los valores 0.025 y 0.5 respectivamente.

En resumen, la función de fitness utilizada es la siguiente

$$1 / (1 + (\text{CantDeJuegosJugados} + \text{ValorDelContadorDePuntaje}) + \text{ErroresDeLaRed}) \qquad (7)$$

donde *CantDeJuegosJugados* vale 7, *ValorDelContadorDePuntaje* es el resultado acumulado de los 7 juegos efectuados y *ErroresDeLaRed* es el valor acumulado de los errores producidos durante los 7 enfrentamientos.

Se realizaron distintas pruebas utilizando una cantidad máxima de 300 iteraciones porque en ninguno de los casos se observaron mejoras significativas en el fitness más allá de ese valor.

Los valores de aprendizaje cognitivo y social utilizados, φ_1 y φ_2 descriptos en la ecuación (2), fueron ambos en 0.05. Los valores de inercia entre 0.05 y 0. El uso de valores tan pequeños lentifica el movimiento de las partículas impidiendo que se dispersen y caigan en la reproducción excesiva.

La cantidad de vecinas máximas permitidas para la reproducción es 2. Las pruebas demuestran que, con hasta 3 vecinos el crecimiento se mantiene controlado. Sin embargo, un excesivo números de individuos no garantiza alcanzar el óptimo sino más bien, lentificar el algoritmo por lo que el número de vecinos permitidos fue 2.

La cantidad de clases utilizadas para calcular el tiempo de vida de los individuos fue 4. Se realizaron pruebas con valores entre 2 y 10 confirmando que un valor alto para el número de clases, mejora la distinción del fitness entre los individuos pero incrementa sensiblemente el tamaño de la población. Por el otro lado, si la cantidad de clases es muy baja, la población puede llegar a desaparecer. Un valor de 4 clases es adecuado para el problema que se busca resolver.

En la figura 3 pueden observarse los resultados alcanzados con un rango de velocidades mínimo de -0.05 y máximo de 0.05 y una probabilidad de mutación igual a 0, para evitar el movimiento excesivo y, por consiguiente, la superpoblación como se ha mencionado. Como puede verse, las soluciones basada en cúmulos de partículas (PSO) con tamaño de población variable permiten obtener redes con un fitness más alto que *gBest PSO* y *lBest PSO*. También se observa que PSO variable logra valores de fitness equivalentes a sus pares de población fija en un número de generaciones mucho menor.

Las figuras 4 y 5 muestran la evolución de la diversidad de la población para las soluciones basada en cúmulos de partículas (PSO) con tamaño de población variable mostrando que están lejos de la convergencia prematura.

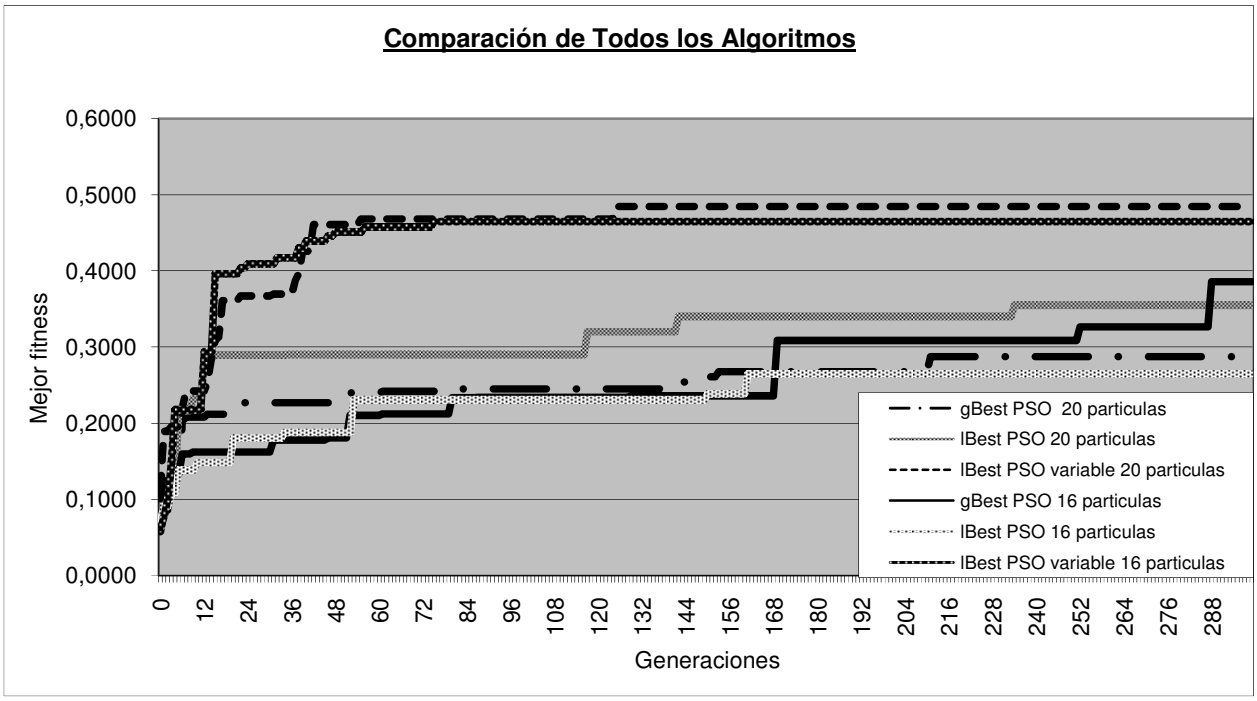

Figura 3

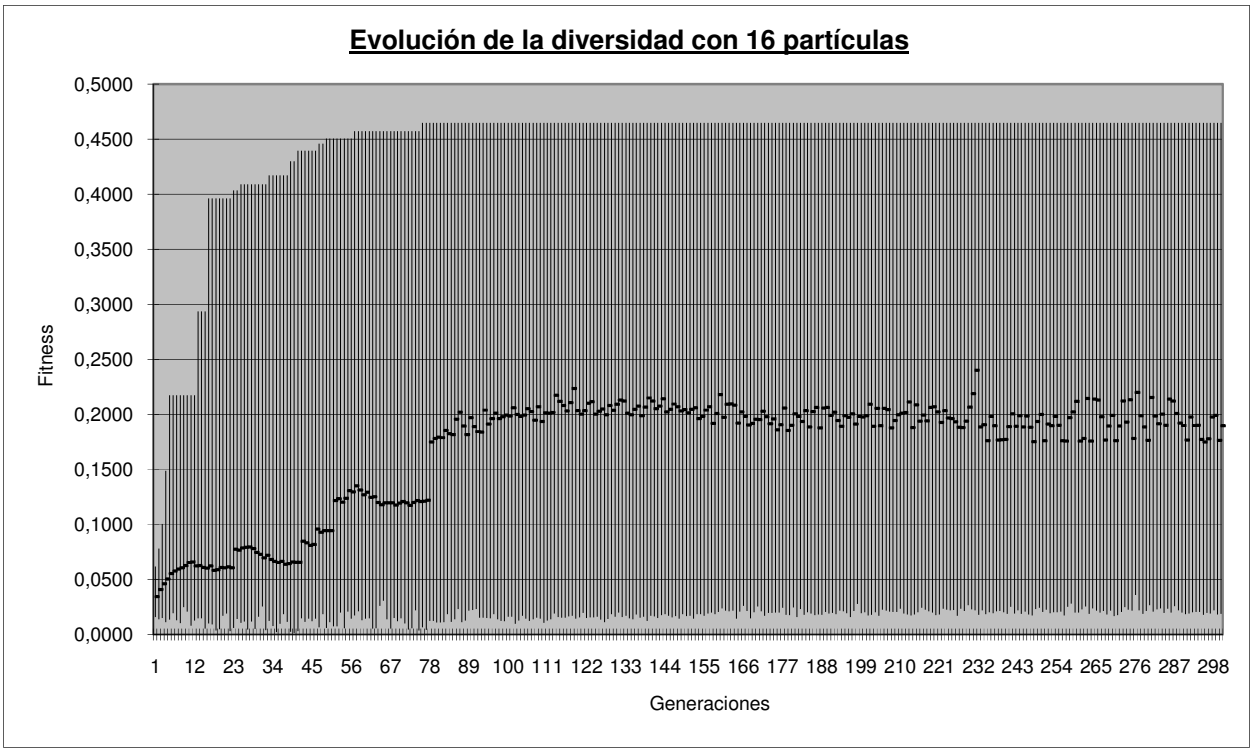

Figura 4

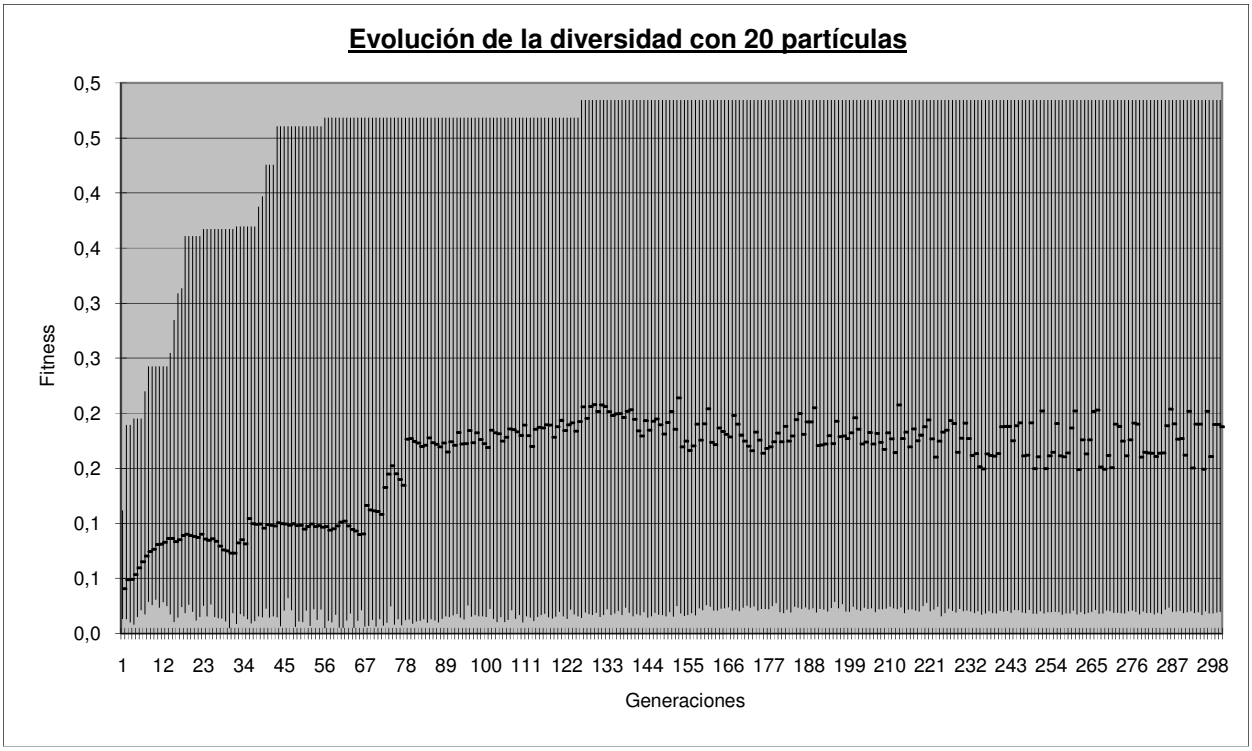

Figura 5

5. Conclusiones

Se ha presentado una estrategia basada en cúmulos de partículas de tamaño variable que permite entrenar redes neuronales. El mecanismo utilizado para incorporar nuevos individuos así como la forma de calcular el tiempo de vida preserva la diversidad de la población sin generar una cantidad excesiva de individuos.

Su aplicación en la resolución del juego de TaTeTi ha sido comparada con *gBest PSO* y *lBest PSO*.

Los resultados obtenidos muestran claramente que la estrategia propuesta en este artículo permite obtener redes con un mejor valor de aptitud que las producidas por ambas versiones de PSO con tamaño de población fijo. Es importante destacar que el algoritmo aquí propuesto genera redes neuronales con un valor de fitness equivalente a sus pares de población fija en una cantidad de generaciones muy inferior.

Aun resta evaluar el comportamiento de la estrategia propuesta en la resolución de algunas funciones complejas clásicas como las empleadas en [Lan00]. Actualmente se está aplicando este algoritmo en el aprendizaje de juegos de tablero más complejos.

6. Bibliografía

[Bir06] Bird S. and Li X. Adaptively Choosing Niching Parameters in a PSO. *Proceeding of Genetic and Evolutionary Computation Conference 2006 (GECCO'06)*, eds. M. Keijzer, et al., p.3 - 9, ACM Press. 2006.

[Lan00] Lanzarini L.,Sanz C. Naiouf M., Romero F. Mixed alternative in the assignment by classes vs. conventional methods for calculation of individuals lifetime in GAVaPS. *Proceedings of the 22nd International Conference on Information Technology Interfaces, 2000. ITI 2000*. pp. 383- 389. ISBN: 953-96769-1-6. June 2000

[Ken95] Kenedy J. and Eberhart R. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks*. Vol IV, pp.1942-1948. Australia 1995

[Shi99] Shi Y., Eberhart R. An empirical study of particle swarm optimization. *Proceeding on IEEE Congress Evolutionary Computation*. pp.1945-1949. Washington DC, 1999.

[Shi98] Shi Y., Eberhart R. Parameter Selection in Particle Swarm Optimization. *Proceedings of the 7th International Conference on Evolutionary Programming*. pp. 591-600. Springer Verlag 1998. ISBN 3-540-64891-7

[Ber02] Van den Bergh F. An analysis of particle swarm optimizers. Ph.D. dissertation. Department Computer Science. University Pretoria. South Africa. 2002

[Cle02] Clerc M., Kennedy J. The particle swarm – explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*. Vol 6,nro. 1, pp. 58-73. Feb.2002

[Ert02] Ertöz L., Steinbach M. and Kumar V. A new shared nearest neighbor clustering algorithm and its applications. Proc. *Workshop on Clustering High Dimensional Data and its Applications*. Arlington, VA, USA, 2002, pp. 105--115.

[Fer05] Fernandes C., Ramos V., Rosa A. Varying the Population Size of Artificial Foraging Swarm on Time Varying Landscapes. Artificial Neural Networks: Biological Inspirations, Proc. ICANN´05: 15th Int. Conf., Warsaw, Poland, LNCS series, Vol. 3696, Part I, Springer-Verlag, pp. 311-316, 2005.

[Mei02] Meissner M., Schmuker M., Schneider G. Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural networks training. BMC Bioinformatics 2006; 7: 125. Published online 2006 March 10. DOI: 10.1186/1471-2105-7-125.