Universidad Nacional Autónoma de México
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS
INTELIGENCIA ARTIFICIAL

La heterogeneidad temporal mejora los algoritmos de búsqueda

TESIS

QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
YOSHIO ISMAEL MARTÍNEZ ARÉVALO

Director de Tesis:
Dr. Carlos Gershenson García
IIMAS-UNAM

Ciudad Universitaria, Cd.Mx. Noviembre 2020

Resumen

El cómputo evolutivo está cobrando cada vez más importancia en el área de la inteligencia artificial, ya que nos permite atacar diferentes tipos de problemas sin requerir una gran capacidad computacional, y estos van desde encontrar una ruta corta en un pequeño grafo hasta entrenar una red neuronal profunda. Sin embargo, muchas veces estos algoritmos sólo encuentran una aproximación a la solución de un problema cuando éste es muy complejo y por tal motivo nos vemos en la necesidad de ajustar los parámetros iniciales para converger o tratar de tener una mejor aproximación. Con esto en mente, el presente trabajo muestra el efecto de la heterogeneidad temporal en la población de soluciones en estos algoritmos. El objetivo de esta investigación es tratar de acelerar el proceso de convergencia de los algoritmos con la cantidad mínima de cálculos y estudiar su comportamiento en diferentes dominios de búsqueda. La idea principal en esta investigación es mantener a los mejores individuos de la población con pocos o sin cambios y a los peores individuos cambiando rápidamente.
Sorprendentemente, la inclusión de este comportamiento en algoritmos de cómputo evolutivo redujo los tiempos de búsqueda al resolver problemas combinatorios, lo cual sugiere que es posible avanzar sobre el espacio de búsqueda evitando los óptimos locales de una manera eficiente, sin la necesidad de hacer una exhaustiva explotación de las soluciones.

Dedicatoria

Quiero agradecer enormemente a la Universidad Nacional Autónoma de México y al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas por la oportunidad de estudiar una maestría.
Agradezco especialmente al Dr. Carlos Gershenson, gracias por todo el apoyo y tiempo otorgado desde el inicio de la maestría hasta su fin, gracias a todo eso fue posible terminar este último trabajo.
Agradezco al CONACyT por el apoyo económico, fue de gran ayuda para la conclusión de mi estudios.

Finalmente agradezco a mi amada esposa Vanesa, por todo su apoyo y comprensión, le estaré por la eternidad siempre agradecido...

Índice

Índice de figuras

Índice de Tablas

1 Introducción

"Lo que sabemos es una gota de agua; lo que ignoramos es el océano."

Issac Newton.
1642−1727

Tanto en ciencia como en ingeniería hay muchos problemas que pueden resolverse con ayuda del cómputo evolutivo. También ya se ha visto que el uso de estos algoritmos como optimizadores de redes neuronales profundas para el aprendizaje por refuerzo profundo son una herramienta competitiva, al igual que otros optimizadores que si hacen uso del gradiente (Such y col., 2017). De manera similar, existe trabajo relacionado con otros tipos de problemas en donde se obtienen resultados aceptables, como el problema del vendedor viajero (Fu y col., 2018), regresión simbólica (Koza, 1990) y N-reinas (Farhan, Tareq y Awad, 2015; Bozikovic, Golub y Budin, 2003), entre muchos otros.

Sin embargo, en muchas ocasiones cuando el problema que intentamos resolver es NP-Completo, como el caso del problema del vendedor viajero, y pasamos de tener 100 ciudades a 7000, entonces se requiere un mayor esfuerzo computacional para poder converger a la solución, o en otro caso, obtener una aproximación aceptable de ella, por lo que esto implica que sea necesario modificar los parámetros iniciales, tales como el tamaño de la población, el número de generaciones, las probabilidades de cruza y de mutación, lo que tiene como consecuencia llevar a un mayor número de cálculos (hablando de algoritmos genéticos, esto ocurre principalmente en el operador cruza).

Existen otro tipo de algoritmos de búsqueda que usan otro tipo de heurísticas, por ejemplo, la optimización por enjambre de partículas (Particle swarm optimization PSO) (Kennedy y Eberhart, 1995), el cual funciona muy bien para encontrar mínimos o máximos globales. Este algoritmo también utiliza una población de individuos (en este caso partículas) que al aplicar ciertas fórmulas permite la convergencia a una solución. Debido a su eficiencia se ha usado en diversos problemas en donde se obtiene un muy buen rendimiento (Yu, Wang y Xi, 2008; Parsopoulos y Vrahatis, 2002; Hu, 2003; Mohandes, 2012).
Este capítulo presenta el planteamiento del problema, se describirán los objetivos, la metodología, la contribución del trabajo y finalmente su organización.

1.1 Planteamiento del problema

La ciencia es un área llena de desafíos donde se requiere encontrar soluciones de manera rápida y eficiente. Muchas veces, debido a la complejidad de un problema, no siempre es posible encontrar la solución óptima, por lo que es deseable encontrar una aproximación en el menor tiempo posible. Frecuentemente creemos que tenemos un algoritmo que resuelve un problema con ciertos tipos de parámetros, donde muchas veces hacemos un esfuerzo por encontrarlos, luego cambiamos el problema y también debemos cambiar los parámetros. Es decir, no hay generalización en los dominios cuando se trata de búsquedas computacionales, aunque se sabe que se necesita un equilibrio entre “exploración” y “explotación” (Blum y Roli, 2003).

Como ejemplo de algoritmos de búsqueda, tenemos los algoritmos genéticos (AG) y las redes neuronales (aunque esta últimas son diferentes a los algoritmos genéticos), que nos permiten encontrar soluciones a varios tipos de problemas, como problemas de predicción (Koza, 1990), diseño de sistemas de distribución de agua (Montesinos, Guzmán y Ayuso, 1997; Galeano y Narváez, 2003), diseño de topologías de circuitos impresos (Cohen, Aga y Weinberg, 2015), reconocimiento de imágenes (Cios y Shin, 1995), etc. Sin embargo, estos algoritmos, como cualquier otro, tienen limitaciones al aumentar el tamaño del problema. Esto implica que si queremos optimizar una función que tiene muchos máximos o mínimos locales (esto se conoce como multimodalidad), lo que sucederá es que podríamos estar obligados a aumentar el número de iteraciones lo que en algunos casos implicaría un mayor esfuerzo computacional. Los algoritmos genéticos nos permiten encontrar soluciones a diversos problemas y están inspirados en el proceso de evolución natural. Estos algoritmos evolucionan una población de individuos en cada generación y son evaluados con una función de adaptación que permite evaluar el progreso de la población (Goldberg, 1989).
Finalmente, como ya se comentó con anterioridad, los algoritmos genéticos también pueden ayudar a entrenar una red neuronal profunda para jugar juegos de Atari® o para clasificaciones (Kuri y col., 2003). Ésta es otra alternativa para entrenar redes neuronales sin tener que seguir el gradiente de una función y ha demostrado tener un muy buen rendimiento. En este trabajo de investigación se aborda la exploración de los efectos de la heterogeneidad temporal en los algoritmos de búsqueda para descubrir si de alguna manera tiene un efecto positivo o negativo en su rendimiento. Adicionalmente, se realiza una comparativa entre las técnicas de optimización que siguen el gradiente contra técnicas libres de gradiente. Para esto, entrenaremos una red neuronal profunda con la misma configuración del trabajo previo (Such y col., 2017), pero con optimizadores de pesos (para el dominio en particular, Videojuegos de Atari®) PSO y Evolución diferencial (Storn y Price, 1995). A este último aplicaremos heterogeneidad en la población.

1.2 Objetivos

1.2.1 Objetivo general

El objetivo general del proyecto es probar y comparar el rendimiento de los algoritmos de búsqueda cuando tenemos heterogeneidad temporal en ellos, suponiendo que la heterogeneidad proporciona robustez y adaptabilidad a los sistemas. Es decir, cuando los individuos más importantes de la población cambian más lentamente que los menos importantes (en algoritmos genéticos nos permite generar diversidad, ya que distintos individuos tienes diferentes probabilidades de cruza), lo que nos permite encontrar soluciones o aproximaciones a problemas en menos tiempo y, sobre todo, intentando minimizar la cantidad de instrucciones computacionales.

1.2.2 Objetivos específicos

1. Investigación del estado del arte de la heterogeneidad en sistemas.
2. Implementación del algoritmo genético básico.
3. Implementación de una red neuronal profunda para evolucionarla con optimización por enjambre de partículas y evolución diferencial.
4. Realizar pruebas comparando los algoritmos con y sin heterogeneidad temporal.

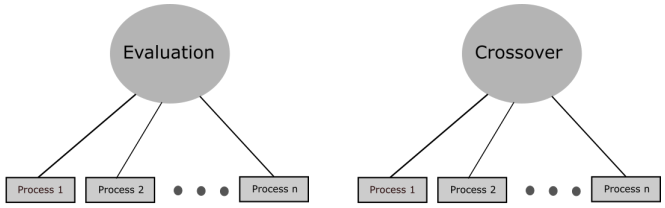
1.3 Metodología

1.3.1 N-reinas y Tsp

Para poder ver los resultados de la heterogeneidad temporal en los algoritmos de búsqueda, es necesario tener un algoritmo que gestione una población de soluciones candidatas que evolucionen con el tiempo, y en donde se permita la inclusión de la heterogeneidad temporal. Por lo tanto, se decidió hacer uso de los algoritmos genéticos con el objetivo de evolucionar poblaciones que tendrán como fin resolver problemas de matemática recreativa y combinatorios. La arquitectura del algoritmo con el que se trabajara tiene una estructura muy similar a la utilizada por (Bozikovic, Golub y Budin, 2003). Una vez que se ha generado la población inicial, se crean tantos procesos como individuos en la población. Cada proceso permite que un individuo sea evaluado independientemente y lo mismo sucede para la cruza(ver Figuras 1.1(a) y 1.1(b)). En la mutación se ha omitido hacer esto, ya que este operador genético no requiere tantas instrucciones como los otros dos procesos mencionados. La Figura 1.1 muestra la arquitectura. En el capítulo 3, "Experimentos", se describe a detalle cada uno de estos problemas.

1.3.2 Videojuegos de Atari®

Para el caso del entrenamiento de la red neuronal profunda que tiene como objetivo aprender a jugar juegos de Atari®, se usará la misma arquitectura de red neuronal del trabajo previo realizado por Uber IA labs (Such y col., 2017) y (Mnih y col., 2015). Sin embargo, aquí en lugar de usar un algoritmo genético o seguir el gradiente, se llevará a cabo la implementación de dos algoritmos distintos, con el objetivo de evolucionar los pesos de la red: optimización por enjambre de partículas (PSO) y evolución diferencial (ED). Para llevar a cabo las pruebas se hará uso del ambiente de trabajo OpenIA Gym (Brockman y col., 2016b), el cual proporciona los juegos de Atari®. De igual manera, se realizará la evaluación de las partículas y vectores de prueba en forma paralela usando la arquitectura de la Figura 1.1.



(a) Evaluación (b) Cruza

Figura 1.1 Arquitectura paralela simple de AG.

Aquí se muestra la arquitectura utilizada para evaluar los individuos del algoritmo genético utilizado en el presente trabajo. Se crea un proceso para cada individuo, en donde se evalúa para obtener su fitness y después es cruzado.

1.4 Contribución

Con la presente investigación se propone ver el efecto de la heterogeneidad temporal en los algoritmos de búsqueda, teniendo como hipótesis que **"la heterogeneidad proporciona robustez y adaptabilidad a los sistemas"**, donde en el contexto de algoritmos de búsqueda esto permite que se realicen una menor cantidad de instrucciones dando como resultado menores tiempos, ya que no todos los individuos están cambiando[1]. Dicho esto, identificamos las siguientes aportaciones:

- **Resolver fácilmente problemas de matemática recreativa y combinatorios:**
  La técnica que se propone tiene como objetivo acelerar los algoritmos genéticos de una manera sencilla, al reducir las instrucciones realizadas en el operador de cruza.

- **Alternativas para entrenar redes neuronales profundas:**
  En esta investigación se demuestra que, además de existir técnicas que siguen el gradiente para entrenar redes neuronales profundas para jugar juegos de Atari®, existen otras alternativas libres de gradiente como PSO y ED, con los

[1] En AG, el cambio de los individuos está en la cruza. En PSO y ED, está en las operaciones matemáticas realizadas en ellos.

cuales se obtienen resultados competitivos comparados con los obtenidos con técnicas que siguen el gradiente.

1.5 Organización de la tesis

Esta tesis es organizada en los siguientes capítulos:

- **Capítulo 2:** En este capítulo se presentan los antecedentes relacionados con los sistemas heterogéneos y su efecto en distintos campos de aplicación, también se explican los principales algoritmos con los que se trabajará.
- **Capítulo 4:** En este capítulo se describen detalladamente cómo se llevarán a cabo los experimentos para poder cumplir con los objetivos.
- **Capítulo 5:** En este capítulo se muestran los resultados que se obtienen de los experimentos realizados.
- **Capítulo 6:** Finalmente, en este capítulo se discuten los resultados, se presentan las conclusiones y se habla sobre el trabajo a futuro.

2 Trabajo relacionado

"Donde hay mucha luz hay una gran sombra."

Johann Wolfgang von Goethe.
1749-1832

En este capítulo se hablará sobre trabajo relacionado con la heterogeneidad en algunos sistemas y se presentan los algoritmos con los que se trabajará.
Primero se describe un trabajo previo de ranking de frecuencia de las palabras en los lenguajes. Después, se describe un trabajo relacionado en otro dominio donde se habla del tema de la sincronización y cooperación en los sistemas; para relacionar los tres temas entre sí.
Finalmente se presentan los diversos algoritmos con los que se trabajará y experimentará en este trabajo. Los cuales se conocen como "algoritmos de búsqueda", aunque todos son de paradigmas diferentes. Se inicia con los algoritmos genéticos, optimización por enjambre de partículas, evolución diferencial y finalmente se explican las redes neuronales que, aunque estos modelos son completamente distintos a los algoritmos anteriores, viéndolos desde un alto nivel, también son útiles para encontrar soluciones.

2.1 Diversidad en el ranking de lenguajes, deportes y juegos

En abril de 2015, se realizó un estudio llamado "Rank Diversity of Languajes: Genetic Behavior in Computational Linguistics" sobre el cambio en las frecuencias de las palabras utilizadas en un idioma a lo largo de los años. Se introdujo la medida de "diversidad de rango", que nos dice cómo cambian las palabras con el paso del tiempo (Cocho y col., 2015). Para esto, se utilizó la base de datos de Google Books, que contiene aproximadamente 4 % de todos los libros escritos entre 1800 y 2008, y se analizaron 6 idiomas diferentes: inglés, alemán, español, francés e italiano. Se encontró que existían palabras que conservaban su frecuencia de uso a lo largo del tiempo en todos los idiomas, se observó que las otras palabras cambiaron muy rapidamente su frecuencia de uso a lo largo de los años. A cada palabra se le asignó un rango inferior[1] que indica una frecuencia muy alta para esa palabra. Al aumentar el rango, existe una mayor variabilidad en la palabra que ocupa ese rango. Con esto se descubrió que se puede estimar el tamaño del "núcleo" de un idioma, es decir, el tamaño mínimo de las palabras para hablarlo y comprenderlo, que corresponden a las palabras más importantes de un idioma.

Posteriormente, en el 2016 se realizó otro estudio llamado "genetic temporal features of performance rankings in sport and games" donde se observó el mismo comportamiento a lo largo del tiempo, pero ahora en el desempeño de los jugadores y equipos en distintos deportes y juegos (Morales y col., 2016). Se realizó el mismo procedimiento, donde el puntaje de un atleta se clasificó por rango y se observó que los mejores atletas también cambiaron el rango más lentamente que los atletas no tan buenos a lo largo de los años (ver Figura 2.1[2]). Sin embargo, estos comportamientos también se encuentran en otros sistemas, como las empresas. Estas investigaciones mostraron que algunos sistemas exhiben un comportamiento en donde es notable cómo los elementos más importantes cambian más lentamente que los elementos menos relevantes que como antes se mencionaba, se tiene la hipótesis de que esta división de elementos les da a los sistemas robustez y adaptabilidad.

[1] En este estudio, la palabra más importante (tiene una mayor frecuencia) tiene el rango 0.

[2] Ilustración y descripción tomada del trabajo de (Morales y col., 2016)

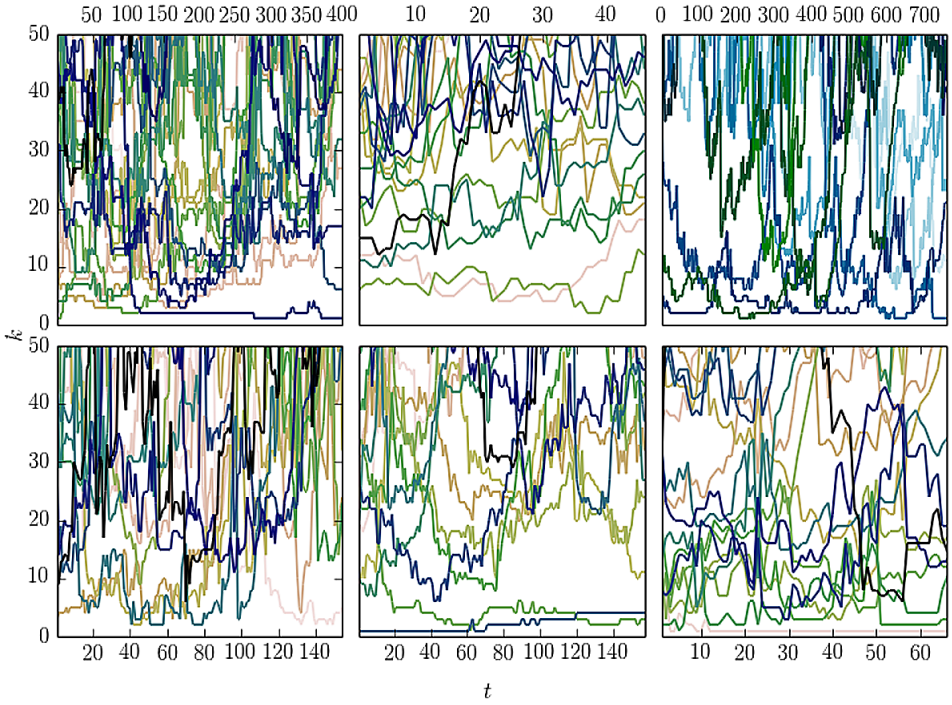


Figura 2.1 Evolución temporal de los rangos de jugadores y equipos.

Gráfico que muestra el cambio en el rango k a lo largo del tiempo t de todos los jugadores/equipos en cada deporte y juego considerado en este estudio (Morales y col., 2016): ATP, FIDE, OWGR, FIFA, FCWR y GPI (en el sentido de las agujas del reloj, comenzando desde la esquina superior izquierda). Para mayor claridad, solo se muestran algunos de los primeros 50 rangos (en todos los intervalos de tiempo disponibles). Tenga en cuenta que los jugadores/equipos en rangos más bajos tienden a cambiar menos que en los más altos, incluso cuando las clasificaciones entre actividades cambian a ritmos variables y la resolución de tiempo asociada se mueve de semanas a meses.

2.2 Sincronización y cooperación

2.2.1 Sincronización en redes complejas

En el campo de dinámica de redes ha existido mucho interés en la sincronización de redes de osciladores (Dörfler y Bullo, 2014; Dörfler, Chertkov y Bullo, 2013), donde algunos autores suponen que la heterogeneidad en redes facilita la sincronización (A. Motter, Zhou y Kurths, 2005); de igual manera ya está demostrada esa suposición (Nishikawa y A. Motter, 2010). En una investigación publicada por Yuanzhao

Zhang llamada “Asymmetry-induced synchronization in oscillator networks” se hace un análisis en sistemas heterogéneos y homogéneos (Zhang, Nishikawa y A. E. Motter, 2017) y con varios ejemplos muestran que dos redes con la misma estructura, una teniendo nodos iguales y otra distintos, logran sincronizar la red con heterogeneidad. La heterogeneidad estructural en redes de osciladores idénticos puede estabilizar estados sincrónicos inestables, rompiendo así la simetría de un sistema para estabilizar un estado simétrico. Del mismo modo, hay trabajos en los que se demuestra que esta heterogeneidad también promueve la cooperación en juegos de bien público (Francisco C. Santos y Pacheco, 2008; Perc y Szolnoki, 2008).

Este fenómeno de heterogeneidad ya se ha observado en sistemas complejos, como las redes de escala libre, en donde pocos elementos importantes tienen varias conexiones con otros elementos. Un ejemplo son las redes de aeropuertos en donde muchos tiene conexión con grandes ciudades y de manera inversa pocos llegan a ciudades no tan importantes. Otro ejemplo es la red de distribución eléctrica, donde grandes centros de abastecimiento abastecen a grandes centros y a la vez se abastecen pequeños centros. Todas estas redes tienen la propiedad de que las conexiones siguen una clara distribución de potencia.

Este análisis apunta a que en un sistema deben existir pocos elementos muy importantes (ya sea que tengan un mejor rendimiento, una cantidad mayor de recursos; ventas o ingresos por año[3]) y muchos no tan importantes. Entonces, es interesante simular este comportamiento en cómputo evolutivo para observar qué comportamiento tienen los algoritmos cuando se tiene el sistema con este rankeo en los individuos. Esta abstracción de ideas generará una versión de un algoritmo genético adaptativo (Lin, 2009; Mahmoodabadi y Nemati, 2016), que tendrá como objetivo evolucionar poblaciones para resolver distintos problemas. Es importante resaltar, que el algoritmo genético que aquí se presenta tiene similitud con el algoritmo CHC(Eshelman, 1991)

[3]Esto se menciona con respecto al sistema que forman las grandes empresas donde las mejores tienen más ventas y difícilmente con el tiempo cambian su rango.

2.3 Algoritmos genéticos

En esta sección presentamos los algoritmos genéticos, uno de los algoritmos que serán útiles para mostrar el efecto de la heterogeneidad temporal en una población de soluciones. Como se mencionó anteriormente, estos algoritmos están inspirados en el proceso de evolución de la naturaleza, es decir, evolucionan una población de individuos seleccionándolos y dando prioridad al más apto para luego aplicar acciones aleatorias, como mutaciones y recombinaciones, con el objetivo de mejorar el grupo de individuos seleccionados.

2.3.1 Algoritmo Genético

Un algoritmo genético es un algoritmo de búsqueda basado en los mecanismos de selección natural y genética natural. Estos algoritmos "evolucionan" una población de individuos, donde cada individuo de la población tiene una puntuación que indica qué tan bueno es para la población; este valor puede ser el error cuadrático medio u otra métrica de error. En resumen, los pasos a seguir son:

1. Generar población inicial.
2. Evaluar los individuos, obtener su fitness.
3. Seleccionar.
4. Cruzar.
5. Mutar.
6. Verificar si se cumple un criterio de terminación.

A continuación, se muestra el diagrama de flujo de un algoritmo genético para apreciar más claro el proceso.

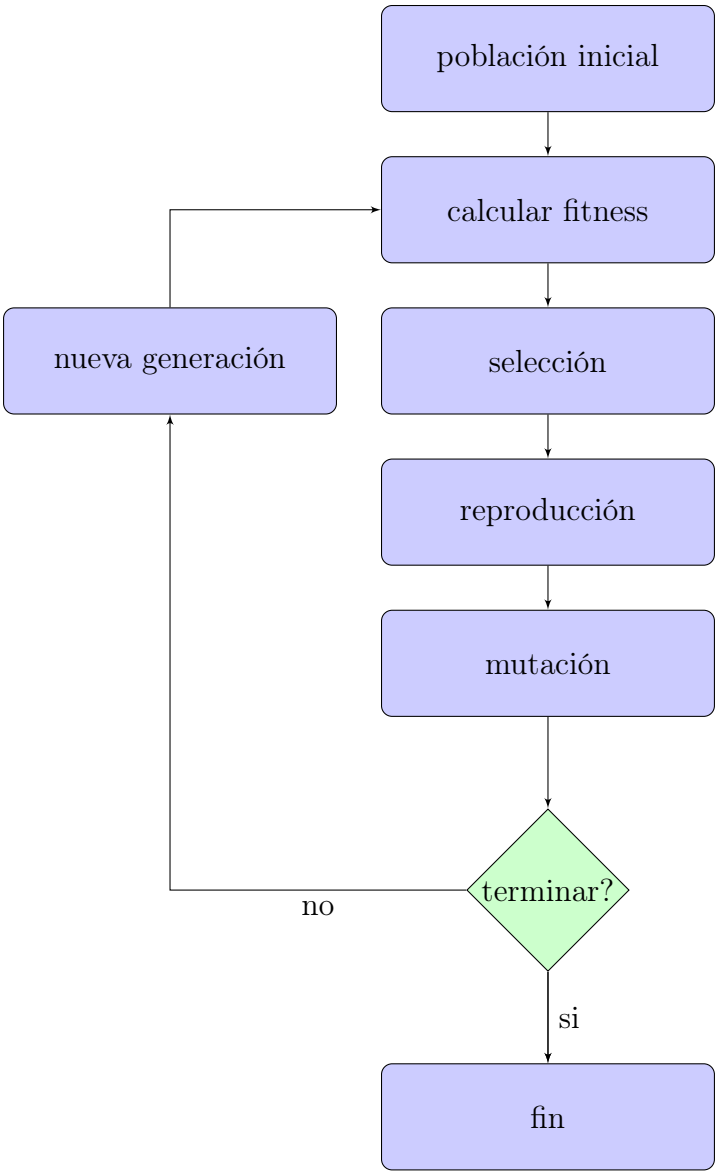


Diagrama de flujo de algoritmo genético.

2.3.2 Selección por Torneo

Esta técnica de selección consiste en seleccionar un conjunto de individuos $\mathbf{p}$ (que normalmente es 2) que "pelearán"[4] y el individuo con la mejor puntuación será seleccionado y será parte de la próxima generación. En este trabajo usaremos $\mathbf{p = 2}$. La Figura 2.2 muestra este proceso gráficamente.

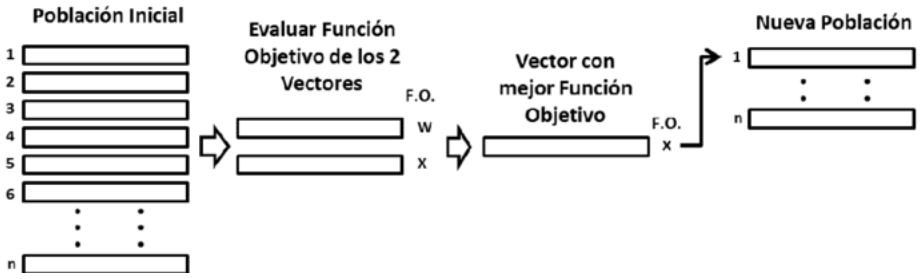


Figura 2.2 Selección por torneo

Proceso de selección por torneo, fuente (López-López, Tejada-Arango y López-Lezama, 2016)

Existen muchos más métodos de selección (Jebari, 2013), aunque es este trabajo únicamente se usará el de torneo.

2.3.3 Operador Cruza

Esta es la principal operación genética $\mathbf{\Psi}$, y es la que se encarga de la explotación de la población. Ésta opera con dos individuos de la población $(\mathbf{u_a}, \mathbf{u_b})$. Primero selecciona una posición de cruce aleatorio en el cuerpo de los individuos, y luego las porciones se combinan dando como resultado dos individuos cruzados, por ejemplo:

[4]Esta pelea consiste en una comparación entre el **fitness** de cada participante, el mejor es el ganador.

Supongamos que tenemos el individuo $\mathbf{u_a}$, como se muestra a continuación;

$$u_a = \{1, 0, 0, 0, 0, 1, 1\} \tag{2.1}$$

Y tenemos otro individuo $\mathbf{u_b}$;

$$u_b = \{1, 0, 0, 1, 1, 0, 0\} \tag{2.2}$$

Posteriormente seleccionamos una posición del individuo al azar, y en ese lugar procedemos a realizar la combinación genética, para este caso seleccionamos la posición con el número 4 comenzando de izquierda a derecha, con lo cual obtenemos los siguientes nuevos individuos en la población.

$$v_a = \{\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{1}, 0, 1, 1\} \tag{2.3}$$

$$v_b = \{\mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, 1, 0, 0\} \tag{2.4}$$

2.3.4 Operador Mutación

Este operador genético $\mathbf{\Phi}$ es el encargado de la búsqueda local y consiste en modificar aleatoriamente el material genético de un individuo seleccionado de la población $\mathbf{u_i}$, por ejemplo:

$$u_i = \{1, 0, 0, 1, 0, 1, 1\} \tag{2.5}$$

$$u_i = \{1, 0, 0, \mathbf{0}, 0, 1, 1\} \tag{2.6}$$

La probabilidad de que un individuo mute por lo regular suele ser muy baja, ya que una probabilidad muy alta podría ser muy destructiva para la población.

2.3.5 Fitness

El **fitness** o **aptitud** es el puntaje o valor que tiene cada uno de los $\mathbf{u_i}$ individuos en la población. Aquí se representan con un vector $\mathbf{Y}$ de longitud $\mathbf{x}$ (aquí indica la cantidad de individuos en la población), donde cada valor $\mathbf{y_i}$ es el fitness de un individuo. Este puede ser el error cuadrático medio que produce la evaluación de un individuo para regresión simbólica ó la cantidad de intersecciones que produce una configuración de un tablero de ajedrez con $\mathbf{n}$ reinas.

2.4 Optimización por Enjambre de Partículas

Este es un algoritmo que tiene relación tanto con la vida artificial como con los algoritmos genéticos (Kennedy y Eberhart, 1995). Es muy usado en la optimización de problemas no lineales, en donde se obtienen buenas aproximaciones a las funciones. El algoritmo trabaja con partículas[5], las cuales poseen, al igual que en la naturaleza, posición y velocidad que son representados como vectores. La estructura de los individuos aquí es un poco diferente, ya que se requiere tener para cada partícula un vector de posición w, de velocidad v y su respectivo fitness f. Sin embargo, el proceso de evolución de la población es mucho más sencillo que los algoritmos genéticos ya que aquí solo deben aplicarse dos sencillas fórmulas para evolucionar. Para nuestro caso usaremos PSO para evolucionar los pesos de la red neuronal profunda.
El algoritmo es muy sencillo de implementar y se muestra en el Algoritmo 1.

[5]En algoritmos genéticos es equivalente a los individuos.

Algoritmo 1: Algoritmo PSO

Result: regresa mejor partícula (élite)

inicializar partículas p; [a]

élite;

for $g = 1, 2, ..., generaciones$ **do**

  for $i = 1, 2, .., n$ **do**

    $p_i.f = F(p_i.w)$ [b]

    if $elite.f \leq p_i.w$ **then**

      $elite.f = (p_i.f, p_i.w)$

    end

  end

  actualizar velocidad; [c]

  actualizar posicion; [d]

end

[a] Para velocidad v se inicializó con 1's, posición w inicializó con números entre 0-1 y su fitness f en 0.

[b] Aquí F es la función que se encargar de evaluar el fitness de la partícula.

[c] Para $v_i(t+1) = v_i(t) + c_1 * r_1 * (w_i^{mejor} - w_i(t) + c_2 * r_2 * (elite.w - w_i(t))$.

[d] Para $w_i(t+1) = w_i(t) + v_i(t)$.

2.5 Evolución diferencial

Evolución diferencial es otro algoritmo iterativo (Storn y Price, 1995) que permite optimizar funciones no lineales sin necesidad de tener información inicial respecto al problema. ED, al igual que PSO, es muy fácil de implementar. Sin embargo, aquí sí existe la selección, cruza y mutación. A continuación, se muestra el algoritmo con las funciones más importantes[6] (Algoritmo 2). Así como en PSO, ED será usado para optimizar los pesos de la misma red neuronal con el mismo objetivo antes mencionado.

[6] Los detalles de su implementación se encuentran en el artículo (Storn y Price, 1995).

Algoritmo 2: Algoritmo evolución diferencial

Result: regresa *elite*

$inicializar - poblacion - aleatoriamente$;

for $g = 1, 2, ...generaciones$ **do**

  for $i = 1, 2, ..., n$ **do**

    selección de individuo

    mutación

    recombinación

    selección

  end

end

2.6 Redes Neuronales Artificiales

Las redes neuronales artificiales son modelos computacionales que tienen como inspiración las redes neuronales biológicas de los seres vivos. Como éstas, están compuestas de varias neuronas artificiales conectadas entre sí y, como consecuencia, surge una capacidad de "aprendizaje" que nos permite realizar operaciones muy complejas.

2.6.1 Neurona de McCulloch-Pitts.

Esta neurona fue la primera en ser presentada (Mcculloch y Pitts, 1943) y pretente modelar una neurona natural. Su representación gráfica la podemos observar en la Figura 2.3 y su Ecuación es la siguiente:

$$\Phi(x; w, \theta) = f(\sum_{i=1}^{n} x_i w_i + \theta) \tag{2.7}$$

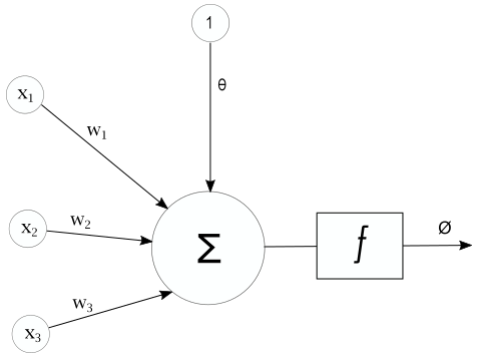


Figura 2.3 Neurona artificial

Representación gráfica de una neurona artificial con 3 entradas (x_1, x_2 y x_3) con sus respectivos pesos w_1, w_2 y w_3. La entrada del sesgo θ (que también es un peso) se mantiene como constante en 1 y f es una función de activación.

Existen distintas funciones de activación f, de las cuales se muestran las más comunes.

Funciones de activación más comunes

Función sigmoide:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

Función de unidad lineal rectificada ReLu:

$$f(x) = max(0, x) \tag{2.9}$$

Función softmax:

$$f(x)_j = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}, \quad \text{para } j = 1, ..., n \tag{2.10}$$

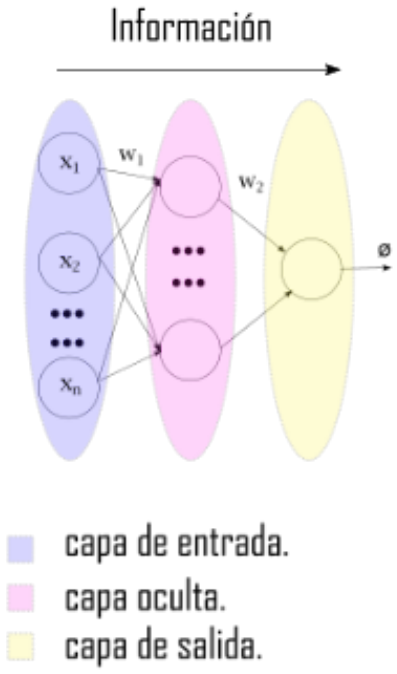


Figura 2.4 Perceptrón multicapa

Representación gráfica de un perceptrón multicapa con n entradas. Este perceptrón multicapa está constituido por 3 capas que podemos observar de distintos colores. Los pesos que conectan entre capas están representados por W_1 y W_2 también podemos apreciar que la capa de salida tiene una sola neurona.

2.6.2 Perceptrón Multicapa

El perceptrón multicapa es una red neuronal que está constituida por la interconexión de varias neuronas, las cuales forman capas, como se puede observar en la Figura 2.4 la información viaja por las capas de izquierda a derecha. El modelo calcula la salida como se muestra a continuación:

Cálculo de salida de neuronas

$$y_1^2 = f(\sum_{i=1}^{n} w_{1i}^1 x_i + \theta_1^1) \tag{2.11}$$

$$y_2^2 = f(\sum_{i=1}^{n} w_{2i}^1 x_i + \theta_2^1) \tag{2.12}$$

$$y_n^2 = f(\sum_{i=1}^{n} w_{ni}^1 x_i + \theta_n^1) \tag{2.13}$$

Calculamos las salidas para cada una de las n neuronas, el superíndice indica la capa (en este caso trabajamos con 3 capas, Figura 2.4):

Finalmente podemos calcular la salida:

Salida final

$$\phi_1^3(x) = f(\sum_{i=1}^{n} w_{1i}^2 y_i^2 + \theta_1^2) \tag{2.14}$$

Es posible crear redes con muchas más capas que la que es mostrada aquí, con muchas más entradas y múltiples salidas.

2.6.3 Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una clase de redes neuronales profundas que han tenido gran aceptación para resolver problemas que requieren una imagen de entrada. Las introdujo Fukushima en 1980 (Fukushima, 1980) y años después se mejoraron para el reconocimiendo de dígitos (Lecun y col., 1998). En general, estas redes están compuestas por tres capas distintas que puede ser apreciadas en la Figura 2.5.

En el presente trabajo no se hará uso de las capas de reducción, por lo tanto este tema será omitido.

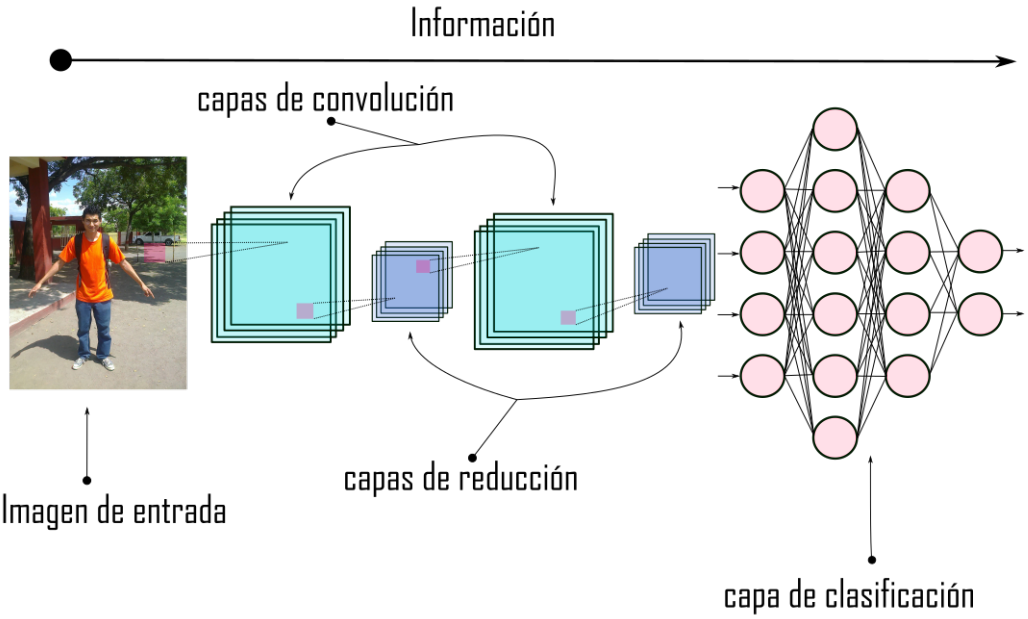


Figura 2.5 Estructura de red convolucional

Recibe como entrada una imagen que puede ser a color. Esta red está constituida por 3 tipos de capas; capa convolucional, capa de reducción y capa de clasificación. La información se mueve de izquierda a derecha.

Capa Convolucional

Como la Figura 2.5 lo muestra, esta capa recibe una imagen que puede ser a color[7] o en escala de grises. La imagen tiene una dimensión representada como $altura \times anchura \times c$. La imagen puede ser representada como una matriz de c dimensiones donde cada posición de matriz representa un píxel cuyo rango va de [0-255].
La Figura 2.6 muestra un ejemplo de cómo sería la representación de la letra b para poder ser procesada por la red convolucional.

[7]Una imagen a color tiene tres canales $c = 3$, un canal para cada color, rojo, verde y azul.

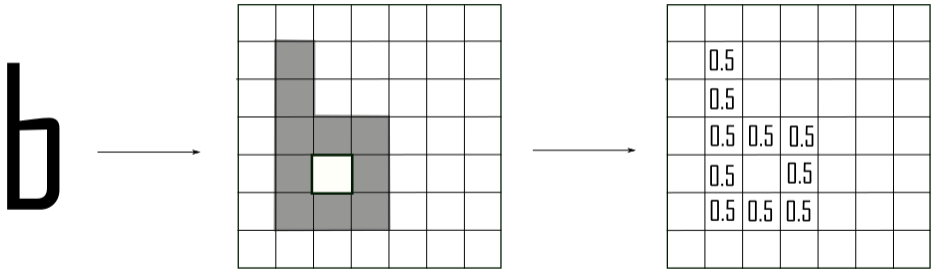


Figura 2.6 Entrada de datos para red convolucional

Representación de la letra b como una matriz de datos. Cada una de las casillas de la matriz representa un píxel normalizado.

En cada una de las capas convolucionales encontramos k kernels, que son pequeñas matrices que se encargan de recorrer la matriz de entrada y se obtiene como resultado una matriz para cada kernel. Las dimensiones de estos kernels son de $n \times n \times c$. Por lo general, los número o pesos que están contendidos en los kernes son inicializados de entre 0 y 1, después son ajustados con backpropagation, aunque en este caso se realiza con las técnicas antes mencionadas. La Figura 2.7 muestra el proceso que siguen los kernels.

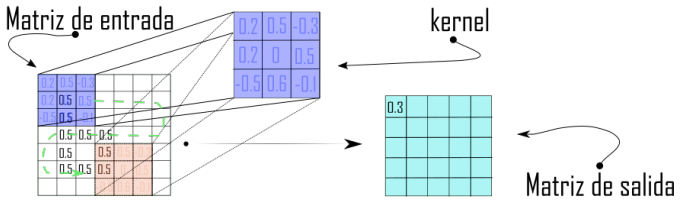


Figura 2.7 Funcionamiento del kernel.

Diagrama de recorrido de un kernel sobre la matriz de entrada. El movimiento es de izquierda a derecha y de arriba abajo como se observa en la línea punteada. Este proceso produce una matriz de salida que será la entrada para la capa posterior.

Capa completamente conectada

Esta última capa es la que se encarga de la clasificación, puede llegar a tener solo una salida o múltiples salidas[8]. Esta capa es simplemente un perceptrón multicapa como el que anteriormente se explicó. El principal objetivo de esta capa es el de clasificar y por lo general es la última capa de las redes convolucionales, como lo muestra la Figura 2.5.

[8]En el caso del presente trabajo las redes neuronales con las que se trabaja poseen múltiples salidas ya que cada juego tiene distintas acciones.

3 Experimentos

"El hombre sigue siendo la computadora más extraordinaria de todas."

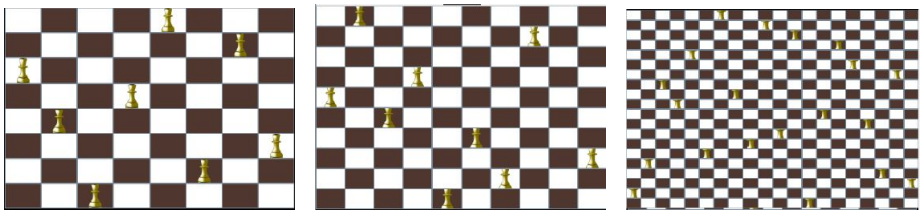
John F. Kenedy.
1917−1963

En este capítulo se detalla cómo se llevarán a cabo los experimentos para permitirnos ver el efecto de la heterogeneidad temporal en los algoritmos de búsqueda. Entendemos por heterogeneidad temporal a la diversidad de individuos que existen en una población, donde temporalmente estos individuos se encuentran en cierto rango y posteriormente se mueven a otro; aquí la heterogeneidad es dada por el puntaje, fitness o aptitud de cada individuo. Se comienza explicando el problema de las N-reinas[1], la representación de los individuos y los parámetros a utilizar; después TSP y finalmente el aprendizaje por refuerzo profundo.

[1]Cabe destacar que este problema ya ha sido tratado incluso con otras metodologías (Hu y col., 2003) sin embargo aquí solo exploramos el efecto que produce la heterogeneidad.

3.1 N-reinas

El problema de las N-reinas fue propuesto por Max Bezel en 1848. Su objetivo es colocar n reinas en un tablero (n x n) de tal forma que ninguna reina ataque a ninguna otra. Cada una de las columnas debe contener únicamente una reina, intentando que ésta no ataque a ninguna en sus columnas vecinas; lo mismo aplica a las filas del tablero.



(a) 8 reinas (b) 10 reinas (c) 20 reinas

Figura 3.1 Soluciones N-reinas

Para resolver este problema se define una población de individuos V, donde cada v_i representa a un individuo en la población. Para colocar una reina en el tablero con la configuración de un individuo, las columnas del tablero se toman como las posiciones de la matriz y las filas por el número contenido en dicha posición. A continuación, se tiene un individuo solución al problema de las 8 reinas y su colocación en el tablero se muestra en la Figura 3.1(a), donde también se muestran configuraciones válidas para 10 y 20 reinas(ver Figuras 3.1(b) y 3.1(c));

$$u_i = \{2, 4, 7, 3, 0, 6, 1, 5\} \tag{3.1}$$

La población inicial de individuos consiste en vectores generados aleatoriamente sin números repetidos por individuo. Aquí el objetivo es reducir el número de intersecciones entre reinas en el tablero (Algoritmo 3).

Algoritmo 3: Función objetivo de N-reinas

```
Result: regresa x
inicializar x en 0; (contador de intersecciones)
for g = 1, 2..., |v|^a do
    for i = 1, 2..., g do
        if v_i = v_g or abs(v_g − v_i) = abs(i − g) then
            x = x + 1;
        end
    end
end
```

[a] $|v|$ indica la longitud del individuos, es decir el número de reinas.

Para calcular la probabilidad de cruzar un par de individuos v_a y v_b realizamos lo siguiente:

- Se obtiene el valor de aptitud $\mathbf{\Phi}$ para cada individuo v_i de la población.
- Se obtiene el valor de aptitud τ del peor individuo de la población.
- Se decide si es posible cruzar[2] $\mathbf{\Psi}(v_a, v_b)$ utilizando la eq. 3.2 (codificada en el Algoritmo 4) para v_a y v_b, obteniéndose w_a y w_b

$$p(v_a) = 1 - \{\Phi(v_a)/\tau\} \tag{3.2}$$

Esta simple operación nos permite cruzar más lentamente a los individuos mejor adaptados, mientras que los peores se cruzan constantemente. La Tabla 3.1 muestra la configuración para N-reinas.

[2] $\mathbf{\Psi}$ en este trabajo representa el operador de cruza.

Algoritmo 4: Cruza en N-reinas y TSP

Result: regresa V

if $(1 - \{\Phi(v_a)/\tau\}) \leq random$ **then**
  $w_a = \Psi(v_a, v_b)^3$
end
if $(1 - \{\Phi(v_b)/\tau\}) \leq random$ **then**
  $w_b = \Psi(v_b, v_a)$
end

Reinas	Población	Cruza	Mutación	Generaciones	Corridas
20, 40, 60, 80, 100, 150	300	0.9	0.1	500	10
200, 250	300	0.9	0.1	1000	10
500	500	0.9	0.1	3000	50

Tabla 3.1 Configuración N-reinas.

El algoritmo genético se ejecutará la cantidad de corridas indicadas para cada una de las configuraciones establecidas en la Tabla 3.1, y después se obtendrá el promedio de instrucciones ejecutadas y el tiempo que se tardó cada configuración.

3.2 Problema del vendedor viajero

El problema del vendedor viajero se define formalmente de la siguiente manera (Sur Kolay, Banerjee y Murthy, 2003).

Este es un problema de combinación con el objetivo de encontrar la ruta con la longitud más corta (o el costo mínimo) en un gráfico no dirigido que representa ciudades o nodos a ser visitados. El vendedor viajero comienza en un nodo, visita todos los demás nodos consecutivamente solo una vez y finalmente regresa al punto de partida. En otras palabras, dadas n ciudades, llamadas $\{c_1, c_2, ..., c_n\}$ y permutaciones, $\{\sigma_1, \sigma_2, ..., \sigma n\}$, el objetivo es elegir σ_i de modo que la suma de todas las distancias euclidianas entre cada nodo y su sucesor se minimiza. El sucesor del último nodo en la permutación es el primero. La distancia euclidiana **d** entre dos ciudades con coordenadas (x_1, y_1) y (x_2, y_2) es calculado por:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.3}$$

Y la distancia mínima se calcula de la siguiente manera;

$$Minimizar\left(\sum_{n=1}^{N-1} d(c_n, c_{n+1})\right) + d(c_N, c_1) \tag{3.4}$$

Ahora se ejecutará el algoritmo genético para resolver el problema del vendedor viajero. De la misma manera que para el problema de las N-reinas, se generan vectores con números enteros, sin embargo, cada posición del vector corresponde a un punto de la Figura 3.2, y tenemos que pasar por todos ellos una sola vez.

Se tiene que encontrar la combinación de nodos que nos dan la ruta más corta. De la misma manera que con N-reinas se realizarán comparaciones entre cada una

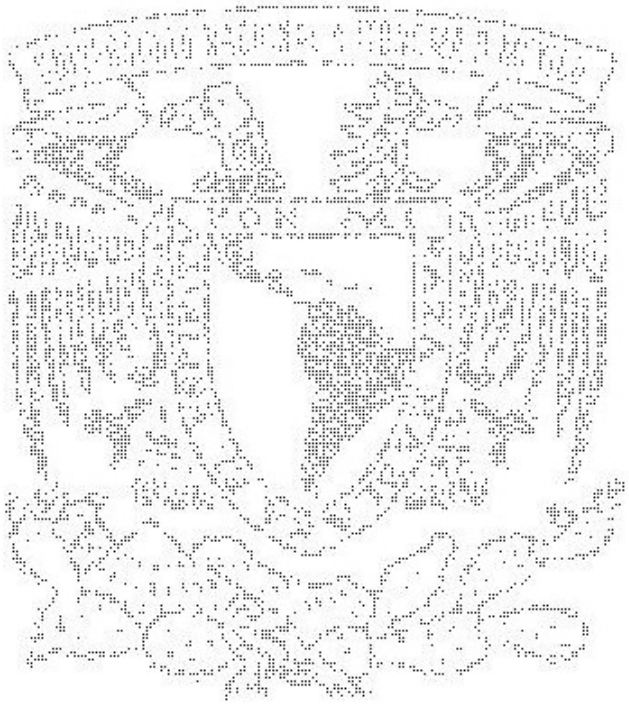


Figura 3.2 Imagen con 7,000 puntos.

de las ejecuciones, con la distancia obtenida, el tiempo de ejecución y el número de instrucciones realizadas. Para este caso, se ejecutará el algoritmo genético una vez con la configuración de la Tabla 3.2 y se graficarán los resultados.

Puntos	Población	Cruza	Mutación	Generaciones
7000	300	0.9	0.1	80000

Tabla 3.2 Configuración TSP.

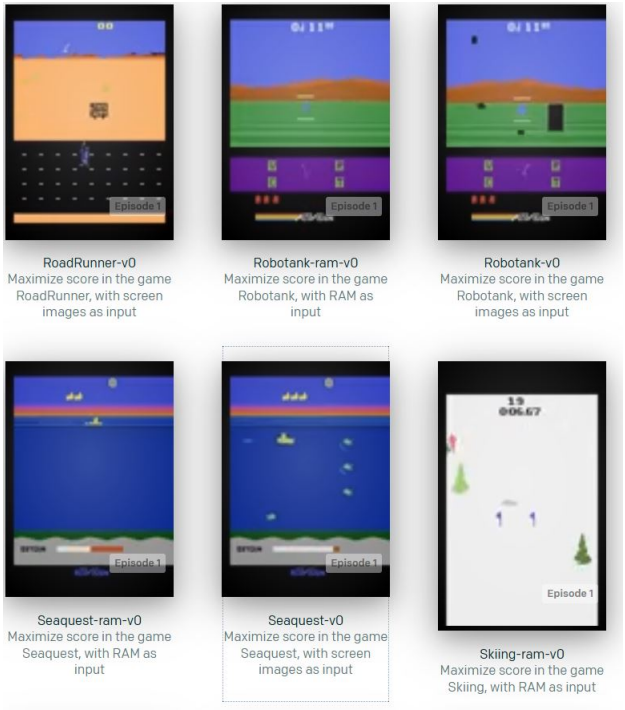


Figura 3.3 Ambiente proporcionado por OpenAI Gym

Ambiente de trabajo proporcionado por OpenAI Gym. Esta herramienta proporciona varios juegos de Atari®, Box2D, Robótica etcétera.

3.3 Juegos de Atari®

Recientemente se ha demostrado que, en algunos dominios, las redes neuronales optimizadas con el algoritmo de descenso de gradiente no funcionan del todo bien. El grupo de investigación de los laboratorios Uber IA publicó en 2017 un artículo llamado "Deep Neuroevolution: Genetic Algorithms are a competitive alternative to train deep neural networks for reinforcement learning" (Such y col., 2017), donde se prueba el rendimiento de una red neuronal profunda que fue entrenada para jugar juegos de Atari® con diferentes métodos (algoritmos genéticos, DQN , estrategias evolutivas, etc.). En el trabajo se propone utilizar algoritmos genéticos para entrenar redes neuronales con el objetivo de jugar juegos de Atari® y tareas de locomoción bípeda.

Sorprendentemente, obtuvieron buenos resultados en algunos juegos y peores en otros, concluyendo que, los algoritmos genéticos pueden considerarse como una herramienta más para el entrenamiento de redes neuronas profundas para el aprendizaje por refuerzo. Con esto nos damos cuenta de que es necesario contar con más herramientas, además de seguir el gradiente, ya que ninguna de éstas es general.

Dicho esto, en el presente trabajo se utilizará el Kit de herramientas de OpenAI Gym (Brockman y col., 2016a), el cual nos permite crear un ambiente para comparar algoritmos de aprendizaje por refuerzo (Figura 3.3). Para crear la red neuronal se hará uso de TensorFlow 2.0 (Martın Abadi y col., 2015) usando la misma arquitectura del artículo "Human-level control through deep reinforcement learning" (Mnih y col., 2015)(ver Figura 3.4) y para la optimización de los pesos de la red neuronal serán usados PSO y ED. Cabe destacar que solo en este último aplicaremos heterogeneidad en la población, y PSO solo será otro punto de comparativa entre algoritmos (las configuraciones se muestran en las Tablas 3.3 y 3.4).

Parámetro	Valor
Población inicial	100 partículas
R	0.9
C1	1.6
C2	1.7
Tamaño frame	64 x 64 píxeles
Frames totales	20 M
Frames por individuo	20,000
e	10

Tabla 3.3 Configuración PSO.

Y la configuración para ED.

Parámetro	Valor
Población inicial	20
Cruza	0.7
Mutación	0.8
Tamaño frame	64 x 64 píxeles
Frames totales	20 M
Frames por individuo	20,000

Tabla 3.4 Configuración ED

Ya que aquí estamos tratando con un ambiente muy dinámico, para obtener el mejor élite se procede a realizar lo siguiente; una vez que se evalúan independientemente los individuos:

1. Se ordenan de mayor a menor con respecto al puntaje.
2. Se obtiene un conjunto de tamaño e de los mejores individuos.
3. Se evalúan 10 veces cada uno de los individuos del conjunto.
4. El individuo con el mejor puntaje promedio de las 10 corridas es evaluado 100 veces, se obtiene el promedio de cada una de las 100 corridas y ese será el puntaje final.

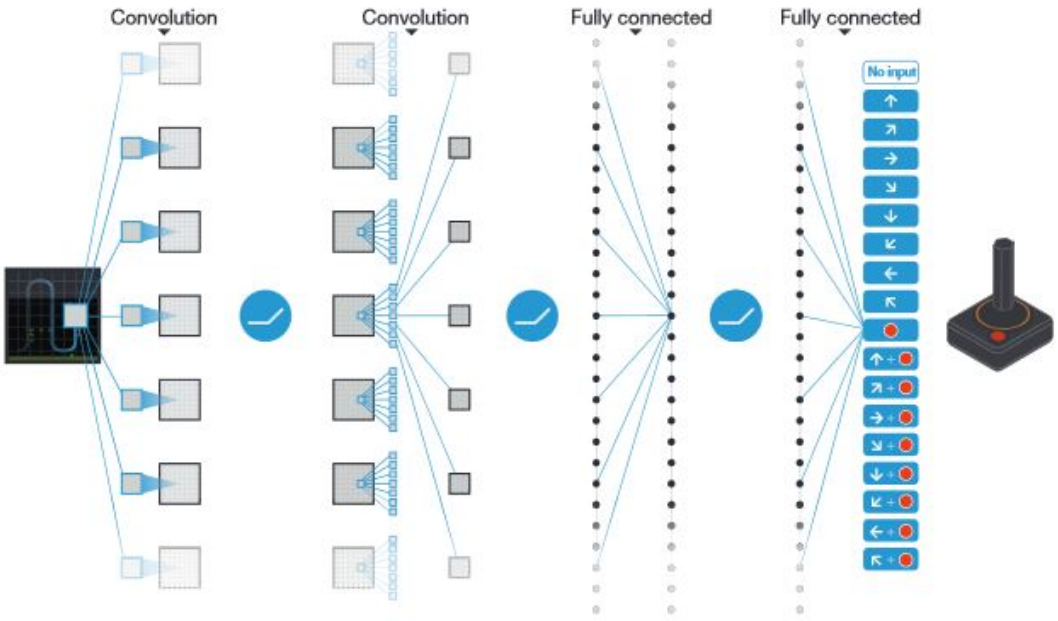


Figura 3.4 Arquitectura de red neuronal.

Red neuronal de (Mnih y col., 2015). La entrada de la red consiste de una imagen de $64 \times 64 \times 3$ (La imagen aquí se redujo y no se realizó ningún procesamiento de ella), seguida por tres capas convolucionales y dos capas completamente conectadas con una sola salida para cada acción válida. Cada capa oculta es seguida por una unidad lineal rectificada (esto es, $max(0, x)$).

Para PSO los individuos (partículas) están compuestos por dos vectores v y p, velocidad y posición respectivamente. El vector de velocidad es inicializado con 1's y el vector de posición es creado con números aleatorios entre 0 y 1. El vector de posición representa los pesos de la red neuronal (Figura 3.5). En el caso de ED se ignoran los puntos 2 a 4 y el mejor individuo es evaluado 100 veces y se promedia su puntaje para ser registrado. También se restringe cada peso de la red a estar entre -1 y 1. Se probará con los juegos: Asterix, Asteroids, Atlantis, Enduro, Frostbite, Kangaroo y Seaquest.

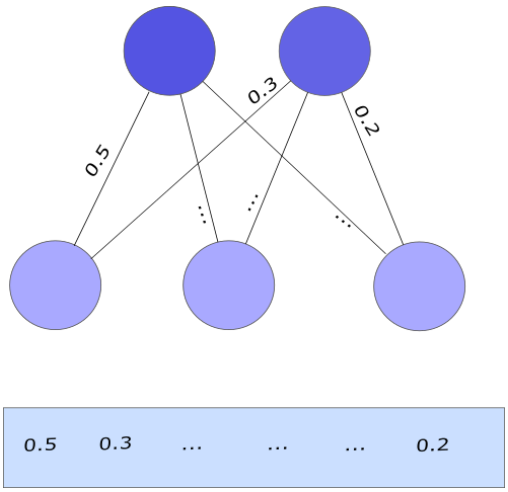


Figura 3.5 Representación de individuo en red.

Esta es la representación de una partícula o individuo en la red neuronal. Cada uno de los elementos del vector corresponde a un peso de la red neuronal, también incluye los sesgos.

4 Resultados

> Una computadora puede ser llamada "inteligente" si logra engañar a una persona haciéndole creer que es un humano.
>
> Alan Mathison Turing.
> 1912-1954

En esta sección se muestran los resultados obtenidos en los experimentos realizados con las configuraciones antes mencionadas. Primero se muestran los resultados de N-reinas, después el problema del vendedor viajero o TSP y finalmente se muestran los resultados obtenidos con la evolución de la red neuronal para jugar juegos de Atari®.

4.1 N-reinas

Las Tablas 4.1 y 4.2 muestran los resultados de cada una de las ejecuciones. La primera columna indica el número de reinas que se colocarán en el tablero, la columna **"Resueltos"** indica el número de veces (de las 10 o 50 ejecuciones) que el algoritmo alcanza una configuración de tablero válida. La tercera columna indica el número de instrucciones promedio de las ejecuciones[1]; solo es registrado para el cruce, que es la parte que estamos modificando. Finalmente, en la cuarta columna se indica el tiempo promedio en minutos.

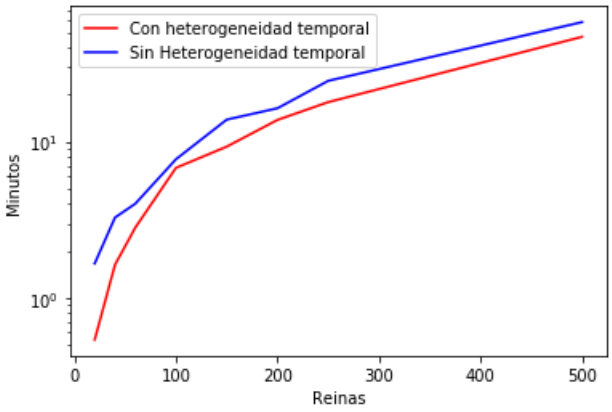
N	Resueltos	Instrucciones en Cruza	Tiempo
20	10	203,106.7	0.54 M
40	10	1,179,512.4	2.03 M
50	10	2,319,122.2	2.45 M
60	10	3,199,086.1	3.22 M
80	9	8,898,568.0	6.34 M
100	9	13,255,157.2	3.8 M
150	9	29,060,304.6	9.3 M
200	6	55,644,604.0	14.19 M
250	7	83,096,536.7	18.33 M
500	26	380,562,758.3	47.35 M

Tabla 4.1 Soluciones con heterogeneidad temporal en cruza

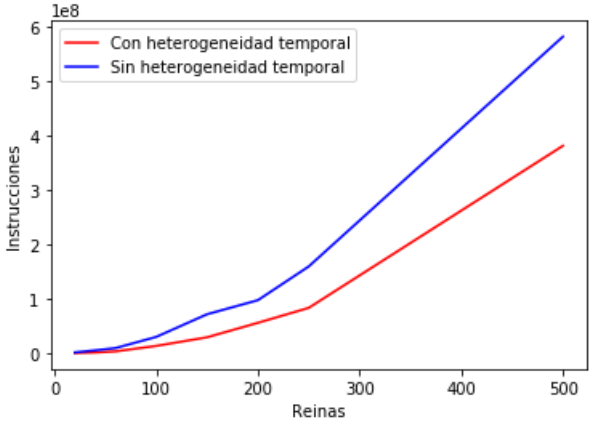
[1]Por cada combinación de las posiciones de los individuos un contador c es incrementado en 1.

N	Resueltos	Instrucciones en Cruza	Tiempo
20	10	1,359,720.7	2.06 M
40	10	5,133,783.2	3.27 M
50	9	11,408,601.2	6.25 M
60	10	9,276,517.8	4.01 M
80	10	19,565,327.2	6.36 M
100	8	29,796,699.0	8.12 M
150	6	71,354,126.8	14.25 M
200	9	97,183,364.6	16.38 M
250	6	159,084,196.8	24.54 M
500	21	580,915,501.2	58.31 M

Tabla 4.2 Soluciones sin heterogeneidad temporal en cruza



(a) Reinas y tiempo

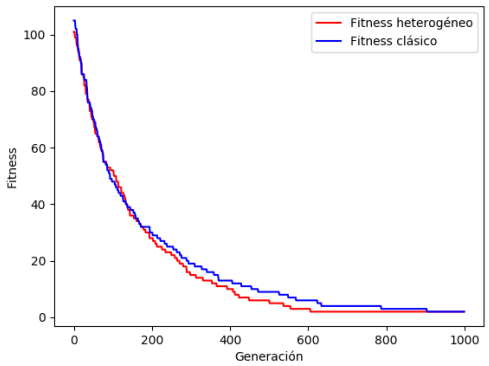


(b) Reinas e instrucciones

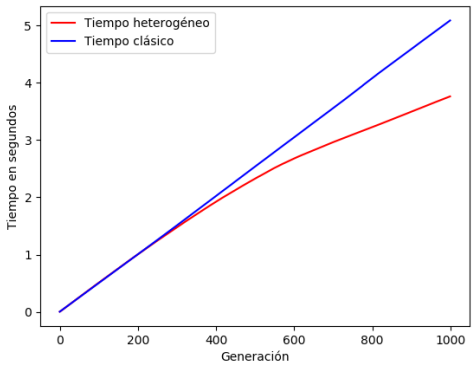
Figura 4.1 Relación entre reinas, tiempo e instrucciones

La Figura 4.1 permite visualizar el efecto de la heterogeneidad temporal con el aumento de reinas y tiempo (ver Figuras 4.1(a) y 4.1(b)). El gráfico parece indicar que cuanto mayor es la complejidad del problema, los tiempos de búsqueda están bien manejados. Del mismo modo, se observó que tener los individuos más aptos sin cruzar a través de las generaciones y permitir cruzar los no tan aptos permite un salto óptimo local, de hecho, no cruzar a toda la población permite disminuir el tiempo de ejecución del algoritmo y las instrucciones que se llevan a cabo. De igual

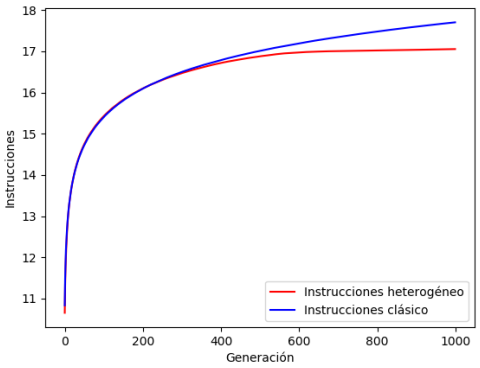
manera, la evaluación de la función objetivo se evalúa un número menor de veces[2].



(a) Generaciones y Fitness



(b) Generaciones y tiempo



(c) Generaciones e instrucciones

Figura 4.2 Relación entre generaciones, fitness, tiempo e instrucciones.
Resultados para el caso de n = 200 reinas.

[2]Esto es porque no hay necesidad de evaluar en F un individuo u_i que no fue cruzado o mutado en la generación anterior.

La Figura 4.2 muestra gráficas del progreso del fitness, tiempo e instrucciones a través de las generaciones para el caso de 200 reinas, en donde podemos observar el rendimiento del algoritmo genético en tiempo e instrucciones.
En la gráfica 4.2(a) podemos apreciar como el fitness de la población decrece a través de las generaciones para tanto el algoritmo genético clásico como el heterogéneo. En la gráfica 4.2(b) vemos que a medida que las generaciones avanzan, en el algoritmo clásico el tiempo incrementa linealmente, y en el heterogéneo, el tiempo no es lineal después de aproximadamente 300 generaciones. Esto cambio del tiempo puede ser entendido cuando se observa la gráfica 4.2(c), ya que en ese punto aproximado vemos que las instrucciones del algoritmo heterogéneo ya no me comienzan a subir como en el clásico.

Entonces, el algoritmo genético heterogéneo realiza un menor tiempo en la búsqueda ya que no realiza tantas instrucciones y por ende no destruye las soluciones actuales tan rápido como el clásico, lo que le permite encontrar más rápido la solución. Dado que se obtuvieron buenos resultados, entonces se probó con 1000 reinas. Esta vez se usan 6000 generaciones con 300 de población inicial y solo se ejecutará una vez, obteniendo los siguientes resultados:

Resueltos	Instrucciones en Cruza	Tiempo	Heterogeneidad
1	1,496,961,310	191.31 M	Si
0	2,170,172,477	256.34 M	No

Tabla 4.3 Resultados 1000-reinas

La Tabla 4.3 muestra los resultados de ejecutar el algoritmo solo una vez, en donde, podemos observar que el algoritmo heterogéneo resuelve el problema en poco más de 3 horas, mientras que, el algoritmo clásico no lo resuelve con las generaciones

establecidas[3], lo que indica que requiere un poco más de tiempo. En la Figura 4.3 se muestra un individuo de la población de 1000 reinas en la generación 0, donde obviamente no es una solución correcta, también se muestra un individuo solución (ver Figura 4.4) y la mejor individuo encontrado por el algoritmo clásico (ver Figura 4.5). Las reinas que están bien ubicadas se representan con un pequeño asterisco, por otro lado, las reinas que están bajo ataque se representan con un círculo azul y se dibujan líneas verdes en todas las direcciones, excepto la vertical.

[3]Esto no significa que el AG clásico no pueda resolverlo, ya que una inicialización aleatoriamente buena puede llegar a converger igual que el heterogéneo.

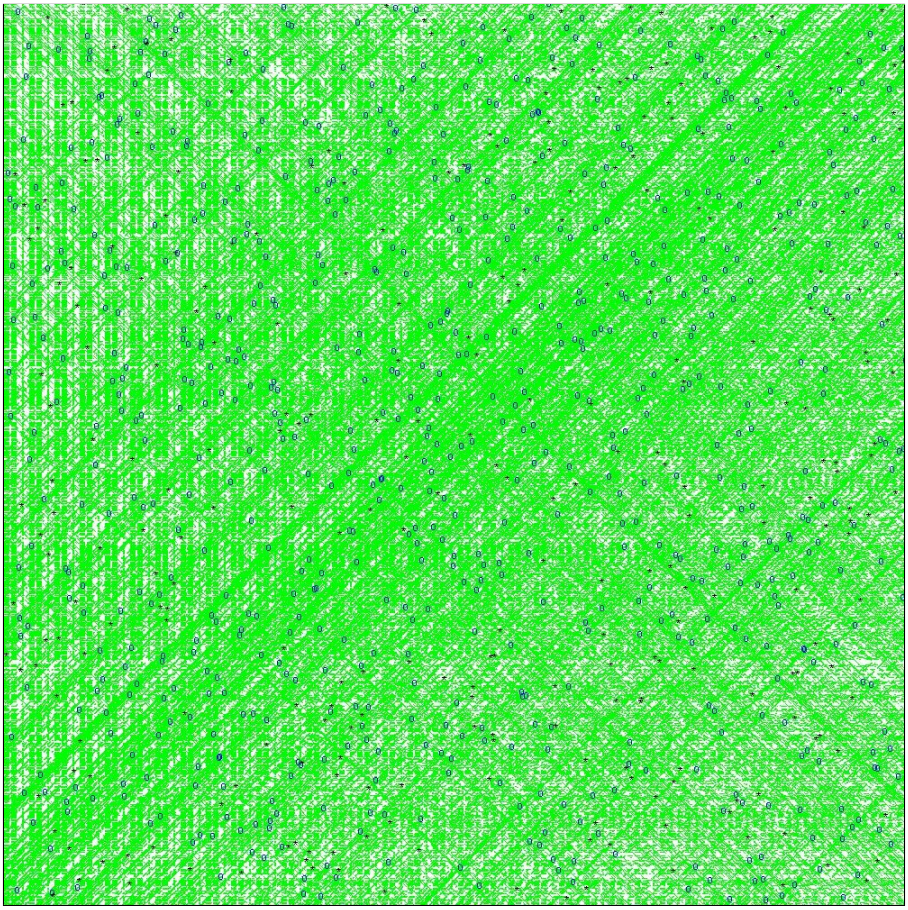


Figura 4.3 Individuo en generación 0, n=1000

Tablero con 1000 reinas en la generación 0. Líneas verdes indican intersección en reinas, los ***** indican una reina bien posicionada, **0** indica reina atacada.

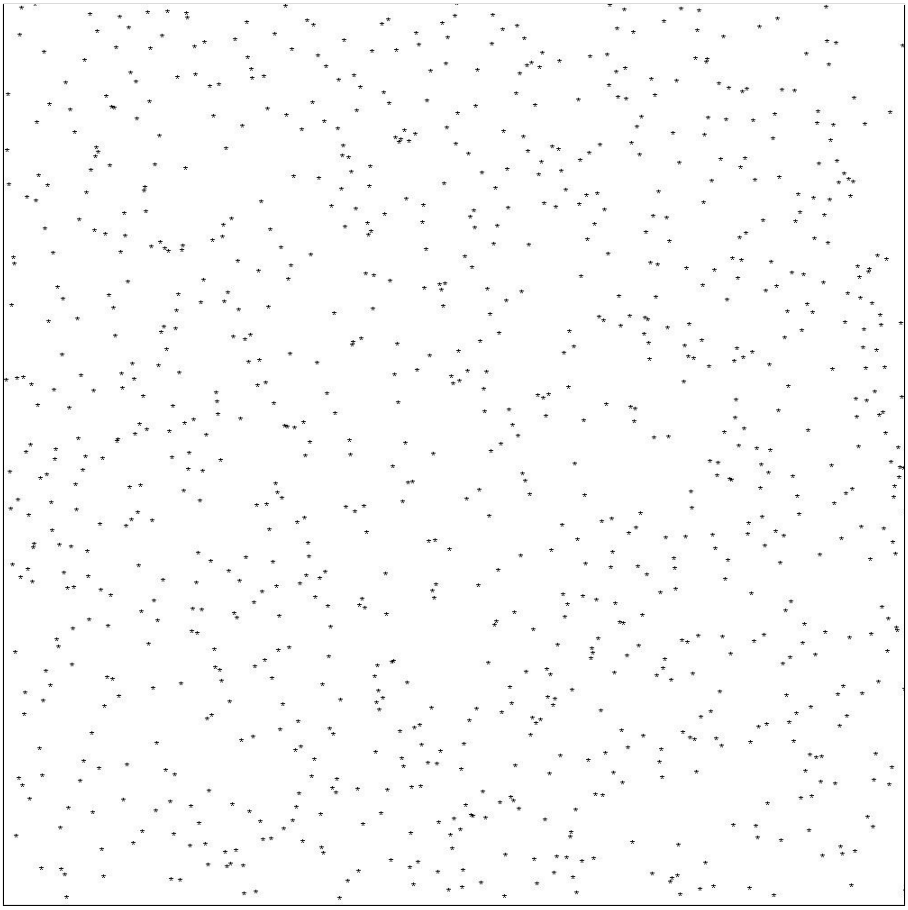


Figura 4.4 Una configuración óptima para n = 1000

Una configuración óptima para el caso de n = 1000 reinas usando el algoritmo con heterogeneidad. Los * indican una reina bien posicionada

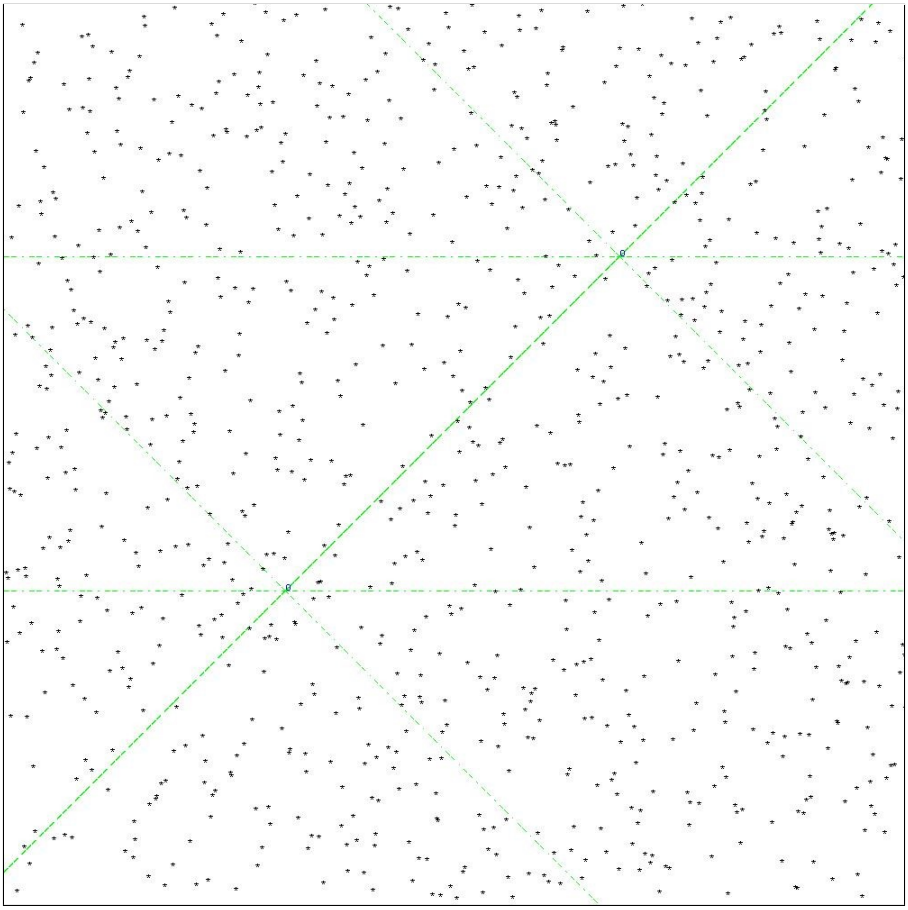


Figura 4.5 Configuración no válida para n = 1000

Una configuración no válida para el caso de n = 1000 reinas usando el algoritmo clásico. Líneas verdes indican intersección en reinas, los ***** indican una reina bien posicionada, **0** indica reina atacada.

4.2 Problema del vendedor viajero

A continuación se muestran los resultados obtenidos utilizando el algoritmo genético con y sin heterogeneidad aplicados al problema del agente viajero con 7,000 nodos.

Distancia	Instrucciones en cruza	Tiempo en horas	Heterogeneidad
24,032.29	211,337.7	87.24	Si
35,395.33	192,617.8	95.29	No

Tabla 4.4 Resultados TSP

En la Figura 4.6 podemos observar el resultado que obtuvimos al ejecutar el algoritmo genético usando heterogeneidad temporal durante 80,000 generaciones. En la Tabla 4.4 podemos observar que, aunque el algoritmo realizó más instrucciones en la cruza, se obtiene un mejor resultado en menos tiempo dando señales de que realmente podríamos acelerar el proceso de búsqueda con problemas más complejos que los que se muestran aquí. Sin embargo, por ahora esto será para un próximo trabajo.

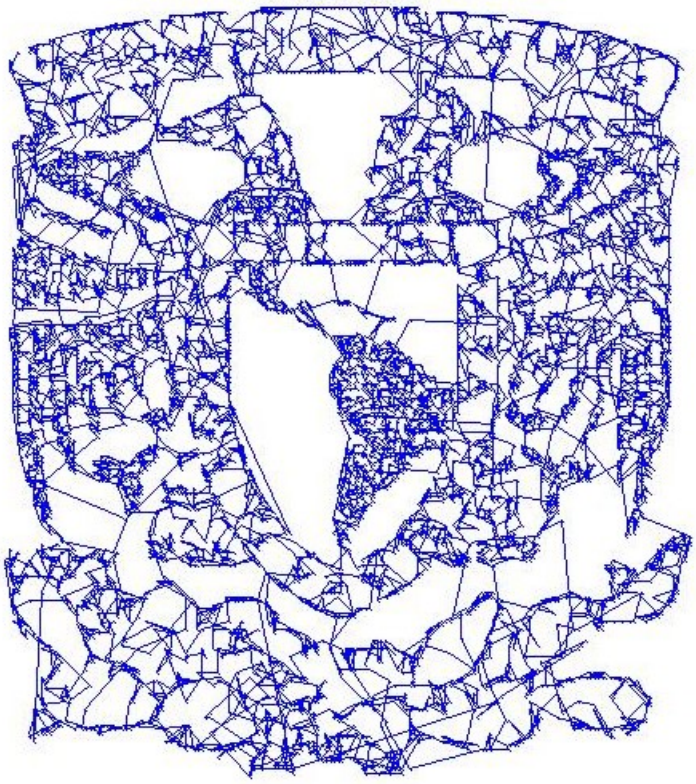


Figura 4.6 Solución TSP con heterogeneidad.

Camino encontrado por el algoritmo genético con heterogeneidad temporal en su población.

4.3 Juegos de Atari®

La Tabla 4.5 muestra los puntos que la red neuronal profunda obtuvo para cada uno de los juegos. Cada columna indica el algoritmo usado para el entrenamiento y las filas muestran el puntaje reportado para un juego en particular, entre ellos, podemos observar los resultados de las múltiples ejecuciones con PSO, ED y ED-H(con heterogeneidad). De igual manera que con los otros experimentos, vemos que a pesar de realizar una explotación menor en las soluciones actuales, se pueden llegar a obtener puntajes similares a los de ED sin modificaciones.

Además, vemos que los resultados en algunos juegos (Asteroids, Atlantis, Enduro, Frosbite y Kangaroo) llegan a superar a técnicas que siguen el gradiente y se requiere una cantidad menor de frames.[4]

Así como en algunos juegos se obtuvieron resultados que superan alguna técnica que sigue el gradiente, en otros se obtuvieron resultados malos. Por lo tanto, no es una técnica general para este tipo de dominios además que faltaría probar con otros juegos de Atari®.

Juego	DQN(200M)	ES(1B)	A3C(1B)	RS(1B)	GA(1B)	PSO	ED	ED-H
Asterix	4,349	1,440	22,140	1,197	1,850	617	1,065	937
Asteroids	1,365	1,562	4,475	1,307	1,661	1,072	1,102	1,423
Atlantis	279,987	1,267,410	911,091	26,371	76,273	26,265	21,994	29,146
Enduro	729	95	-82	36	60	30	25	25
Frostbite	797	370	191	1,164	4,536	550	759	699
Kangaroo	7,259	11,200	94	1,099	3,790	579	924	793
Seaquest	5,861	1,390	2,355	503	798	617	386	317

Tabla 4.5 Resultados Atari®

[4]Como se indicó en el capítulo 3, se ocuparon 20M para PSO, ED y ED-H.

5 Discusión, Conclusión y Trabajo a Futuro.

"El software es una gran combinación entre arte e ingeniería."

Bill Gates.

5.1 Discusión

Como acabamos de apreciar, en varias ejecuciones el efecto que tiene la heterogeneidad temporal da buenos resultados y permite encontrar las soluciones en un poco menos de tiempo debido a que realiza menos operaciones con los individuos, lo que permite acelerar el proceso de búsqueda. Sin embargo, esta modificación que se realizó en el algoritmo genético no significa que sea universal para todo tipo de problemas (Wolpert y Macready, 1997), por lo que es necesario probar esta modificación en otro conjunto de problemas para ver realmente su utilidad. Es decir, aquí ya hemos demostrado que esto ayuda con problemas con codificación entera, como N-reinas, TSP y en entornos estocásticos como aprendizaje por refuerzo, pero una buena idea es ver su comportamiento en problemas de codificación reales y otros entornos estocásticos.

De antemano ya se conocía que los algoritmos genéticos son suficientemente competentes para entrenar redes neuronales profundas para aprendizaje por refuerzo (Such y col., 2017). En este trabajo se utilizó la misma arquitectura de red neuronal del trabajo previo citado donde se realizaron las comparaciones con las demás técnicas que hacen uso del gradiente para optimizar los pesos de la red neuronal donde las técnicas de optimización propuestas (PSO, ED Y ED con heterogeneidad) lograron demostrar que, al igual que los algoritmos genéticos, estos algoritmos de búsqueda también pueden llegar a ser competentes para este complicado dominio.

5.2 Conclusión

Los resultados que se han obtenido con los algoritmos genéticos en problemas de recreación matemática (N-reinas), problemas combinatorios como TSP y en neuroevolución profunda, tanto en rendimiento, como en tiempos de búsqueda y escalabilidad del problema, sugieren que estos pequeños conjuntos de individuos "élite" permiten mejorar un sistema, que en el contexto de algoritmos de búsqueda podemos interpretarlo como una metodología que permite avanzar sobre el espacio de búsqueda, evitando los óptimos locales de una manera eficiente sin la necesidad de hacer una exhaustiva explotación de las soluciones.

5.3 Trabajo a futuro

Los resultados obtenidos en este trabajo de investigación nos llevan a creer que podemos explorar en problemas donde el espacio de búsqueda tiene más "picos locales", por ejemplo, otros dominios en aprendizaje por refuerzo profundo. El siguiente objetivo es responder a las preguntas:

1. **¿Qué heterogeneidad es la mejor para qué búsqueda?**
2. **¿Hay algunas búsquedas en las que la heterogeneidad no ofrezca ventajas o pueda ser desventajosa?**
3. **¿Por qué funciona la heterogeneidad en búsquedas?**

Para esto se propone utilizar los modelos NK de Stuart Kauffman (S. Kauffman y Levin, 1987; S. A. Kauffman y Weinberger, 1989). Estos modelos nos permiten generar superficies rugosas controladas por sus parámetros N y K, los cuales dan como resultado superficies con una cantidad arbitraria de mínimos y máximos locales en donde podríamos realizar varios experimentos intentando adaptar la población y responder las preguntas antes planteadas.

Referencias

Blum, Christian y Andrea Roli (sep. de 2003). "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". En: *ACM Comput. Surv.* 35.3, págs. 268-308. ISSN: 0360-0300. DOI: 10.1145/937503.937505. URL: https://doi.org/10.1145/937503.937505.

Bozikovic, M., M. Golub y Leo Budin (oct. de 2003). "Solving n-Queen problem using global parallel genetic algorithm". En: 104-107 vol.2. ISBN: 0-7803-7763-X. DOI: 10.1109/EURCON.2003.1248159.

Brockman, Greg y col. (2016a). *OpenAI Gym*. eprint: arXiv:1606.01540.

— (2016b). "OpenAI Gym". En: *CoRR* abs/1606.01540. arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540.

Cios, Krzysztof J. e Inho Shin (1995). "Image recognition neural network: IRNN". En: *Neurocomputing* 7.2, págs. 159-185. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(93)E0062-I. URL: http://www.sciencedirect.com/science/article/pii/0925231293E0062I.

Cocho, Germinal y col. (abr. de 2015). "Rank Diversity of Languages: Generic Behavior in Computational Linguistics". En: *PLOS ONE* 10.4, págs. 1-12. DOI: 10.1371/journal.pone.0121898. URL: https://doi.org/10.1371/journal.pone.0121898.

Cohen, Miri Weiss, Michael Aga y Tomer Weinberg (2015). "Genetic Algorithm Software System for Analog Circuit Design". En: *Procedia CIRP* 36. CIRP 25th Design Conference Innovative Product Creation, págs. 17-22. ISSN: 2212-8271.

DOI: https://doi.org/10.1016/j.procir.2015.01.033. URL: http://www.sciencedirect.com/science/article/pii/S2212827115000360.

Dörfler, Florian y Francesco Bullo (2014). "Synchronization in complex networks of phase oscillators: A survey". En: *Automatica* 50.6, págs. 1539-1564. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2014.04.012. URL: http://www.sciencedirect.com/science/article/pii/S0005109814001423.

Dörfler, Florian, Michael Chertkov y Francesco Bullo (2013). "Synchronization in complex oscillator networks and smart grids". En: *Proceedings of the National Academy of Sciences* 110.6, págs. 2005-2010. ISSN: 0027-8424. DOI: 10.1073/pnas.1212134110. eprint: https://www.pnas.org/content/110/6/2005.full.pdf. URL: https://www.pnas.org/content/110/6/2005.

Eshelman, Larry J. (1991). "The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination". En: ed. por GREGORY J.E. RAWLINS. Vol. 1. Foundations of Genetic Algorithms. Elsevier, págs. 265-283. DOI: https://doi.org/10.1016/B978-0-08-050684-5.50020-3. URL: http://www.sciencedirect.com/science/article/pii/B9780080506845500203.

Farhan, Ahmed S., Wadhah Z. Tareq y Fouad H. Awad (jul. de 2015). "Article: Solving N Queen Problem using Genetic Algorithm". En: *International Journal of Computer Applications* 122.12. Full text available, págs. 11-14.

Francisco C. Santos, Marta D. Santos y Jorge M. Pacheco (2008). "Social diversity promotes the emergence of cooperation in public goods games". En: *Nature* 454, págs. 213-216. ISSN: 1476-4687. DOI: 10.1038/nature06940. URL: https://doi.org/10.1038/nature06940.

Fu, Chunhua y col. (2018). "Solving TSP problem with improved genetic algorithm". En: *AIP Conference Proceedings* 1967.1, pág. 040057. DOI: 10.1063/1.5039131. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.5039131. URL: https://aip.scitation.org/doi/abs/10.1063/1.5039131.

Fukushima, K (1980). "Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". En: *Biological cybernetics* 36.4, págs. 193-202. ISSN: 0340-1200. DOI: 10.1007/bf00344251. URL: https://doi.org/10.1007/bf00344251.

Galeano, Haiver y P.-C Narváez (dic. de 2003). "Genetic algorithms for the optimization of pipeline systems for liquid transportation (1)". En: *CT y F - Ciencia, Tecnologia y Futuro* 2, págs. 2, 55-64.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201157675.

Hu, Xiaohui (jul. de 2003). "Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization". En:

Hu, Xiaohui y col. (jul. de 2003). "Swarm Intelligence for Permutation Optimization: Case Study of n-Queens Problem". En: *Proceedings of the IEEE Swarm Intelligence Symposium* 2003. DOI: 10.1109/SIS.2003.1202275.

Jebari, Khalid (dic. de 2013). "Selection Methods for Genetic Algorithms". En: *International Journal of Emerging Sciences* 3, págs. 333-344.

Kauffman, Stuart A. y Edward D. Weinberger (1989). "The NK model of rugged fitness landscapes and its application to maturation of the immune response". En: *Journal of Theoretical Biology* 141.2, págs. 211-245. ISSN: 0022-5193. DOI: https://doi.org/10.1016/S0022-5193(89)80019-0. URL: http://www.sciencedirect.com/science/article/pii/S0022519389800190.

Kauffman, Stuart y Simon Levin (1987). "Towards a general theory of adaptive walks on rugged landscapes". En: *Journal of Theoretical Biology* 128.1, págs. 11-45. ISSN: 0022-5193. DOI: https://doi.org/10.1016/S0022-5193(87)80029-2. URL: http://www.sciencedirect.com/science/article/pii/S0022519387800292.

Kennedy, J. y R. Eberhart (1995). "Particle swarm optimization". En: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4, 1942-1948 vol.4.

Koza, John R. (19 6 de 1990). *Non-Linear Genetic Algorithms for Solving Problems*. United States Patent 4935877. filed may 20, 1988, issued june 19, 1990, 4,935,877. Australian patent 611,350 issued september 21, 1991. Canadian patent 1,311,561 issued december 15, 1992.

Kuri, Angel y col. (oct. de 2003). "Classification of Sperm Cells According to their Chromosomic Content Using a Neural Network Trained with a Genetic Algorithm". En: vol. 3, 2253-2256 Vol.3. ISBN: 0-7803-7789-3. DOI: 10.1109/IEMBS.2003.1280246.

Lecun, Y. y col. (1998). "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE* 86.11, págs. 2278-2324.

Lin, C. (2009). "An Adaptive Genetic Algorithm Based on Population Diversity Strategy". En: *2009 Third International Conference on Genetic and Evolutionary Computing*, págs. 93-96.

López-López, Jaime, Diego Tejada-Arango y Jesús López-Lezama (jul. de 2016). "Planeamiento AC de la expansión de la red de transmisión considerando repotenciación de circuitos y ubicación de capacitores". En: *TecnoLógicas* 19, pág. 61. DOI: 10.22430/22565337.81.

Mahmoodabadi, M.J. y A.R. Nemati (2016). "A novel adaptive genetic algorithm for global optimization of mathematical test functions and real-world problems". En: *Engineering Science and Technology, an International Journal* 19.4, págs. 2002-2021. ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2016.10.012. URL: http://www.sciencedirect.com/science/article/pii/S2215098616304736.

Martın Abadi y col. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/.

Mcculloch, Warren y Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". En: *Bulletin of Mathematical Biophysics* 5, págs. 127-147.

Mnih, Volodymyr y col. (feb. de 2015). "Human-level control through deep reinforcement learning". En: *Nature* 518.7540, págs. 529-533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

Mohandes, Mohamed Ahmed (2012). "Modeling global solar radiation using Particle Swarm Optimization (PSO)". En: *Solar Energy* 86.11, págs. 3137-3145. ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2012.08.005. URL: http://www.sciencedirect.com/science/article/pii/S0038092X12002952.

Montesinos, P., Adela Guzmán y Jose Ayuso (mar. de 1997). "Optimización de redes de distribución de agua utilizando un algoritmo genético". En: *Ingeniería del agua* 4. DOI: 10.4995/ia.1997.2717.

Morales, José A. y col. (2016). "Generic temporal features of performance rankings in sports and games". En: *EPJ Data Science* 5, págs. 1-16.

Motter, Adilson, Changsong Zhou y Juergen Kurths (feb. de 2005). "Network Synchronization, Diffusion, and the Paradox of Heterogeneity". En: *Physical review. E, Statistical, nonlinear, and soft matter physics* 71, pág. 016116. DOI: 10.1103/PhysRevE.71.016116.

Nishikawa, Takashi y Adilson Motter (jun. de 2010). "Network synchronization landscape reveals compensatory structures, quantization, and the positive effect of negative interactions". En: *Proceedings of the National Academy of Sciences of the United States of America* 107, págs. 10342-7. DOI: 10.1073/pnas.0912444107.

Parsopoulos, K. E. y M. N. Vrahatis (2002). "Particle Swarm Optimization Method in Multiobjective Problems". En: *Proceedings of the 2002 ACM Symposium on Applied Computing*. SAC '02. Madrid, Spain: Association for Computing Machinery, págs. 603-607. ISBN: 1581134452. DOI: 10.1145/508791.508907. URL: https://doi.org/10.1145/508791.508907.

Perc, Matja ž y Attila Szolnoki (ene. de 2008). "Social diversity and promotion of cooperation in the spatial prisoner's dilemma game". En: *Phys. Rev. E* 77 (1), pág. 011904. DOI: 10.1103/PhysRevE.77.011904. URL: https://link.aps.org/doi/10.1103/PhysRevE.77.011904.

Storn, Rainer y Kenneth Price (ene. de 1995). "Differential Evolution: A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces". En: *Journal of Global Optimization* 23.

Such, Felipe Petroski y col. (2017). *Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning*. arXiv: 1712.06567 [cs.NE].

Sur Kolay, Susmita, Satyajit Banerjee y C. Murthy (ene. de 2003). "Flavours of Traveling Salesman Problem in VLSI Design." En: págs. 656-667.

Wolpert, D. H. y W. G. Macready (abr. de 1997). "No Free Lunch Theorems for Optimization". En: *Trans. Evol. Comp* 1.1, págs. 67-82. ISSN: 1089-778X. DOI: 10.1109/4235.585893. URL: https://doi.org/10.1109/4235.585893.

Yu, Jianbo, Shijin Wang y Lifeng Xi (2008). "Evolving artificial neural networks using an improved PSO and DPSO". En: *Neurocomputing* 71.4. Neural Networks: Algorithms and Applications 50 Years of Artificial Intelligence: a Neuronal Approach, págs. 1054-1060. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2007.10.013. URL: http://www.sciencedirect.com/science/article/pii/S0925231207003591.

Zhang, Yuanzhao, Takashi Nishikawa y Adilson E. Motter (jun. de 2017). "Asymmetry-induced synchronization in oscillator networks". En: *Phys. Rev. E* 95 (6), pág. 062215. DOI: 10.1103/PhysRevE.95.062215. URL: https://link.aps.org/doi/10.1103/PhysRevE.95.062215.