UNIVERSIDAD DEL VALLE DE GUATEMALA

Facultad de Ingeniería

Ramificación neuro-evolutiva aplicada en videojuegos

Trabajo de graduación en modalidad de megaproyecto presentado por

Jorge Estuardo García Salazar

para optar por el grado académico de Licenciado en Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala

2018

Trabajo de Graduación

Ramificación neuro-evolutiva aplicada en videojuegos

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Ramificación neuro-evolutiva aplicada en videojuegos

Trabajo de graduación en modalidad de megaproyecto presentado por

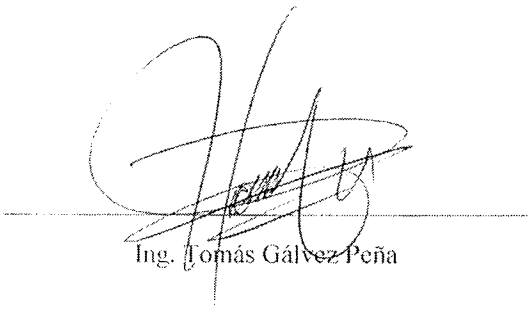
Jorge Estuardo García Salazar

para optar por el grado académico de Licenciado en Ingeniería en Ciencia de la Computación y Tecnologías de la Información

Guatemala

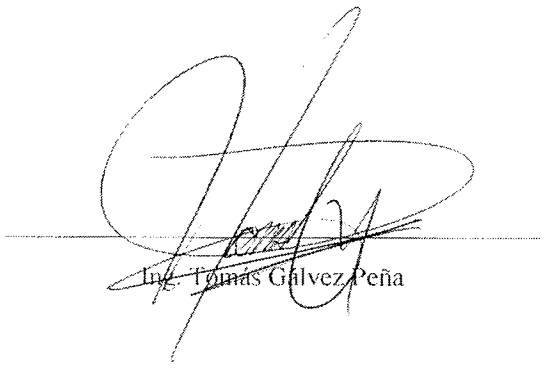
201

Vo.Bo.:

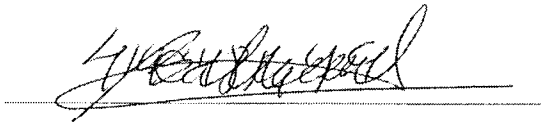


Ing. Tomás Gálvez Peña

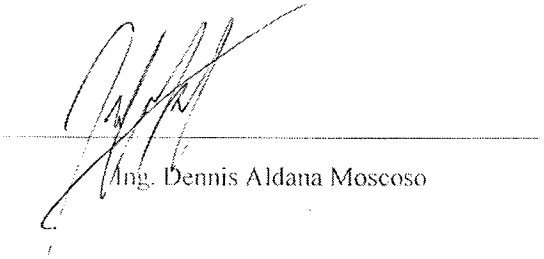
Tribunal Examinador:



Ing. Tomás Gálvez Peña



Ing. Alhvi Balcarcel Rodas



Ing. Dennis Aldana Moscoso

Fecha de aprobación: Guatemala, 5 de diciembre 2018

Índice

Resumen

Los algoritmos neuro-evolutivos son una categoría de algoritmo genético, la cual consiste en generar redes neurales mediante algoritmos evolutivos. Esta forma de inteligencia artificial se ha aplicado en muchas áreas, incluyendo el campo de los videojuegos. En este proyecto se experimenta con la implementación del algoritmo neuro-evolutivo NEAT, para la generación de agentes capaces de resolver juegos de la consola Atari. Se implemento el algoritmo de forma tal, que los agentes que se evolucionan para resolver los juegos, son agentes que fueron anteriormente creados y evolucionados para resolver otro juego de Atari similar. Esto con el fin de descubrir, si el reciclar agentes puede, o no presentar algún beneficio al momento de utilizar algoritmos neuro-evolutivos en el campo de los videojuegos. Los resultados indican, que este proceso puede ayudar a disminuir el número de iteraciones en el algoritmo para poder crear agente que genero los resultados deseados, aunque más experimentación es necesaria para concluir sobre la aplicabilidad de esta metodología.

I. Introducción

La inspiración para este proyecto nació al descubrir la existencia de los algoritmos neuro-evolutivos, y como los mismo pueden ser utilizados para crear bots o agentes que resuelven una amplia gama de problemas, entre ellos la creación de bots que juegan videojuegos. El concepto de una red neural artificial está inspirado en las redes neurales del cerebro animal y los algoritmos genéticos están inspirados en el proceso evolutivo de las especies, es entonces natural pensar que se pueden traer más conceptos del mundo biológico y traerlos al mundo de la computación.

Al entender el concepto básico de cómo se forma una red neuronal mediante un algoritmo neuro-evolutivo, se pensó nuevamente en el mundo biológico, y como las especies de un determinado entorno, evolucionan para adaptarse mejor a los cambios que tengan el mismo. Por ejemplo, si un nuevo depredador es introducido al ecosistema, u ocurre un cambio abrupto en la temperatura. En el caso de este proyecto, la idea es simular este proceso mediante uso de videojuegos como los entornos.

El proyecto involucró el uso de cuatro juegos como ambientes: dos juegos base, sobre los cuales se aplicó el algoritmo genético para obtener especies de agentes, que tratan de resolver el juego; y dos juegos derivados de los juegos base. Las especies resultantes del algoritmo sobre los juegos base, se usaron como población inicial para una nueva aplicación del algoritmo sobre los juegos derivados. Los dos videojuegos derivados se escogieron, de manera que fueran similares a los juegos base.

En este contexto resolver o sobrevivir a un videojuego, es lograr que el bot termine un nivel, logre llegar a cierto puntaje, o que llegue a cierto lugar dentro del juego. En este proyecto se utilizaron juegos de Atari, los cuales consisten en alcanzar la mayor cantidad de puntos posibles, antes de ser eliminado por una serie de enemigos, por lo cual si el algente lograba conseguir un puntaje similar o mejor al del experimentador el juego se daba por resuelto.

El proyecto debía de hacerse de tal manera que se resolvieran al menos tres juegos: uno base y dos derivados de este. Esto con el fin de saber si los agentes resultantes del juego base, se comportarían de manera distinta en los diferentes juegos derivados. Para este proyecto se decido utilizar cuatro videojuegos: dos como juegos base, y dos como derivados. De esta forma se pudo analizar si los agentes de determinado juego base, se comportaban mejor que los del otro.

Las pruebas realizadas demostraron que, el utilizar las especies generadas por el algoritmo genético sobre juegos base, como población inicial en los juegos derivados, conlleva a puntajes similares, que los que obtenidos al utilizar el algoritmo desde cero en juegos derivados. Inclusive en algunos casos, la metodología de reciclar los bots, con lleva en alcanzar los mismos puntajes en un menor tiempo.

A. Objetivos

1. Objetivo general

Brindar más información sobre los algoritmos neuro-evolutivos en la generación de agentes automatizados que resuelven videojuegos, mediante el experimento de utilizar los agentes generados al resolver un videojuego como la población inicial para resolver un segundo videojuego.

2. Objetivos específicos

Verificar la viabilidad de la implementación, analizando si este método es capaz de generar redes neurales artificiales que puedan alcanzar la aptitud deseada en cada juego.

Determinar si bajo este método se logra o no, mejorar el tiempo de computación o número de iteraciones, que le tomaría al algoritmo con una heurística inicial aleatoria, alcanzar un determinado puntaje en el juego.

B. Necesidades del proyecto

Para llevar este experimento a cabo fue necesario escoger videojuegos similares. Estos juegos tenían que permitir que el experimentador recolectará la información de salida, para así poder obtener las variables de entrada a la red neuronal. Estos datos de salida deben poder obtenerse de manera tal, que con cada acción que realice la red neural (por ejemplo, presionar un botón del control), haya nueva información de parte del juego, la cual se convertirá en la nueva información de entrada. Podemos ver las acciones como una variable a, y al videojuego como una función $vj(a)=r$ donde r es el premio otorgado por el juego respecto a la acción. La aptitud de cada agente se calculará en base a la suma de los premios r que se obtienen a partir de cada acción.

$$vj(a) = r$$

La información de salida de un juego puede ser recibida de varias maneras, como por ejemplo una matriz que represente los colores de la pantalla del juego en cada momento. Otra opción pueden ser los datos en la memoria RAM en cada momento del videojuego (opción utilizada para este experimento). En el mejor de los casos la información de salida del juego sería únicamente la significativa para el jugador, por ejemplo, tener únicamente la información de donde está el enemigo, movimientos realizados por el mismo etc. Este último ejemplo a pesar de ser lo más optimo (ya que la red neural tendría como entrada únicamente la información significativa), es difícil conseguir videojuegos que provean dicha información, y depurar manualmente la información de salida es muy laborioso. Además, para el propósito de este experimento es necesario tener tres juegos con el mismo formato de salida, por lo cual no solo habría que filtrar la información de cada juego, sino que también habría que hacerlo de tal manera, que la información de los juegos tenga el mismo formato.

En un mundo ideal todos los juegos no solo serían similares para un humano, sino que también estarían programados, de tal forma que el intercambio de datos estaría en el mismo formato y lenguaje. Los juegos existirían para la misma plataforma (por ejemplo, Windows o Atari), y además serian de código abierto. Este escenario es lógicamente improbable y si existe un caso de esto, no serviría como representación del mundo real.

La idea de experimentar con al menos tres videojuegos era para tener un juego base, y tener dos juegos similares al base con algún cambio o complejidad adicional a las mecánicas del juego. Esto es con fin de ver si los bots que resuelven el primer juego, son capaces de evolucionar de forma tal que ahora resuelvan los otros dos juegos. El cambio en complejidad puede ser muy variado, desde un cambio en el número de posibles acciones (dando al bot más opciones dentro del juego), o un cambio en las mecánicas del videojuego (por ejemplo, en un juego se dispara hacia el frente y en otro hacia los lados).

Debido a esto todos los juegos escogidos tienen la misma mecánica básica, la cual consiste de moverse a los lados y disparar al frente, con el fin de eliminar la mayor cantidad de enemigos posibles antes de ser eliminado por los mismo. Por lo tanto, se buscaron juegos base sin ninguna mecánica adicional a esta, para los juegos derivados se escogieron juegos que agregaran algún cambio menor a esta mecánica, o añadieran una (Como por ejemplo tener la opción de disparar un proyectil más grande). Sobre estos juegos se habla más a detalle en la sección *"Elección de videojuegos"* capítulo VI.

II. Neuro-evolución

La neuro-evolución es una forma de inteligencia artificial que utiliza algoritmos genéticos para generar redes neurales (Stanley 2017). Este tipo de algoritmo pertenece a la rama de "Deep Learning", que es una subrama del Aprendizaje de Máquina (Bengio *et al.*, 2013). La Neuro-evolución está inspirada en el proceso de evolución biológica del sistema nervioso y aplica abstracciones de la evolución natural, para generar redes neuronales artificiales (ANNs por sus siglas en inglés). Entrenar una ANN para resolver un problema, involucra encontrar los mejores parámetros posibles para los pesos de sus conexiones, así como la mejor topología. La idea básica de la Neuro-evolución es entrenar la ANN con un algoritmo genético, estos son una heurística de búsqueda inspirada en la teoría evolutiva de Charles Darwin (Risi y Togelius, 2017).

Una importante decisión de diseño, al momento de utilizar algoritmos neuro-evolutivos es la representación de la red neural en el código, ya que según la misma se determina como funcionaran los métodos de mutación y recombinación ("crossover" en inglés) (Risi y Togelius, 2017). Para los propósitos de este proyecto se escogió NEAT como algoritmo el cual maneja de representación de los cromosomas de una manera específica (de esto se hablará más a detalle en la sección "*Como funciona NEAT*" capítulo III)

Ya que un algoritmo neuro-evolutivo está basado en los algoritmos genéticos y en las redes neuronales artificiales, es importante tener un entendimiento básico de este tipo de algoritmos, es por ello por lo que a continuación se explica, de manera que se entiendan los conceptos base para entender el proyecto.

A. Algoritmos genéticos

Los algoritmos genéticos están inspirados en los procesos de la selección natural y la supervivencia del más fuerte. Estos son utilizados generalmente para generar soluciones de alta calidad a problemas de optimización o problemas de búsqueda. Esto utilizando operaciones basadas en procesos biológicos como: la mutación, recombinación y selección (Mitchell, 1996).

Al utilizar algoritmos genéticos, se empieza con una población de soluciones candidatas, llamados individuos, cromosomas, o fenotipos. Estos individuos luego son expuestos a un proceso evolutivo, para poder encontrar la mejor solución. Cada solución candidata tiene un conjunto de propiedades, llamadas "cromosoma", o alternativamente "genotipo", estas propiedades pueden ser mutadas y alteradas (Whitley, 1994). Si bien el cromosoma es el agente o individuo, el "genoma" es el conjunto de las características de este.

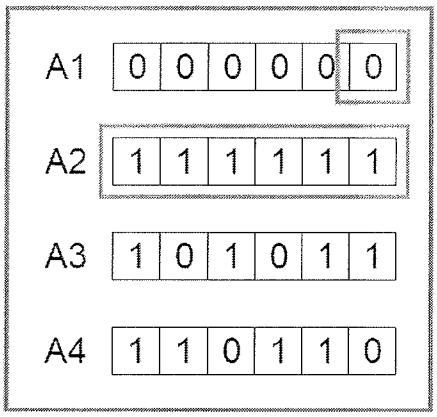


Figura 1. Representación gráfica de una población de 4 individuos, donde cada uno está formado por un arreglo de bits de tamaño 6. El cuadro púrpura señala la población, el cuadro verde señala un cromosoma y el cuadro rojo señala un único gen o alelo.

Para resolver un problema de optimización, usualmente se comienza el proceso con una población generada de manera aleatoria (como es el caso en este proyecto), luego se comienza el proceso iterativo de evolución, donde en cada iteración se genera un nuevo grupo de individuos. Cada iteración es llamada una generación, en cada generación, todos los individuos son puestos a prueba para calcular su aptitud (*fitness* en inglés). Los individuos más aptos dentro de la población son seleccionados de forma estocástica y el genoma de cada individuo es modificado mediante recombinación y/o mutación, estos nuevos individuos conforman la siguiente generación, la cual es utilizada en la siguiente iteración. Normalmente un algoritmo genético termina al alcanzar un máximo número de generaciones, o cuando se consiga la aptitud deseada.

Por lo general un algoritmo genético requiere de: una representación de los individuos en las generaciones, y de una función de aptitud. Una representación genética estándar es un arreglo de bits (Whitley, 1994), pero arreglos de otros tipos de dato o estructura también pueden ser utilizados (como es el caso en NEAT). Esta representación genética es útil debido a que dos o más arreglos pueden ser alineados uno con el otro, facilitando el proceso de recombinación (esto no significa que es obligatorio usar arreglos del mismo tamaño). Luego de tener tanto la función de aptitud, como la representación genética, se empieza el proceso con una población de soluciones las cuales se irán mejorando de manera iterativa mediante operaciones de recombinación y mutación, para luego seleccionar los individuos más aptos de cada generación, repitiendo el proceso.

1. Operaciones

Selección. El proceso de selección consiste en seleccionar los individuos más aptos de cada generación. Normalmente se escoge a la porción más apta de la población y únicamente a la misma se le permite reproducirse para que sus genes pasen a la siguiente generación. Cabe mencionar que no es inusual implementar el algoritmo de forma tal, que se conserve cierto número individuos (por lo general los más aptos) para pasar a la siguiente generación sin mutación alguna (en este proyecto se aplica esta metodología).

Mutación. Esta operación consiste en alterar las características de cierta de cantidad de especímenes de forma aleatoria. La mutación tiene una probabilidad de ocurrir en cualquier gen y en cualquier cantidad de estos.

Recombinación, para cada par de individuos que se reproduzcan (cromosomas padres), se escoge de manera aleatoria un punto de recombinación dentro de los genes. Luego se intercambian entre cromosomas las subsecuencias de genes, de antes y después del punto de recombinación, de esta manera se generan dos combinaciones distintas, estas son la descendencia de los dos cromosomas originales.

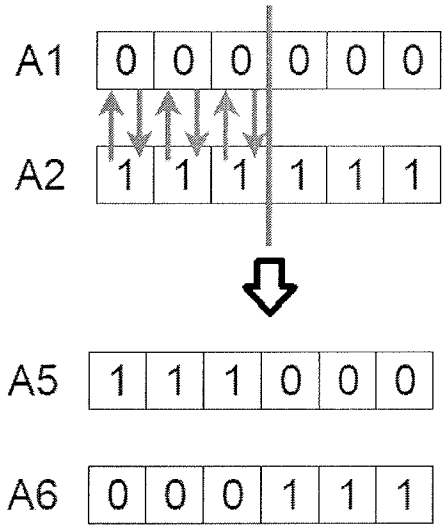


Figura 2. Proceso de Recombinación entre A1 y A2. En esta representación gráfica de la línea roja representa el punto de recombinación (en este caso a la mitad), a partir de esta línea intercambian las subsecuencias de genes, para así crear la descendencia: A5 y A6.

B. Redes neurales

Las redes neurales artificiales son sistemas computacionales inspirados por las redes neuronales biológicas, que constituyen parte del cerebro de los animales. (Frontiers, 2018). Las ANN (por sus siglas en inglés *"artificial neural network"*) están basadas en una colección de unidades conectadas, llamadas neuronas o neuronas artificiales. Cada neurona en la red es capaz de recibir una señal de entrada, para procesarla y mandar una señal de salida. Cada neurona está conectada con al menos otra neurona, y a cada conexión se le asigna un número real llamado peso sináptico, este es un número real que refleja el grado de importancia en la red para cada conexión.

Las neuronas se clasifican entre los tipos de: entrada, escondidas o salida. Usualmente hay tantas neuronas de entrada, como características en el set de datos, de la misma forma suele haber

tantas neuronas de salida como las posibles opciones en un conjunto de valores discretos, que conformarán una predicción a partir de los valores de entrada. El número de neuronas en las capas escondidas varía mucho con cada tarea. La arquitectura de una ANN es llamada topología, entiéndase la estructura de nodos y conexiones que forman una red neuronal. Tradicionalmente la topología es determinada por el programador antes de entrenar el algoritmo (Kearney, 2016). En el caso del algoritmo que se usara para este proyecto NEAT este paso no es necesario.

La gran ventaja con las redes neuronales es el hecho de que son capaces de utilizar información que es desconocida previamente y esta oculta en la data. El proceso de captura de esta información se llama: entrenamiento de la red neuronal. Existen tres tipos principales tipos de entrenamiento supervisado, no supervisado y por refuerzo. El entrenamiento supervisado significa que la red neural conoce los valores de salida deseados, y respecto a esto ajusta los valores de sus conexiones. El entrenamiento no supervisado significa que el valor deseado de los nodos de salida es desconocido, en este caso al sistema se le provee con un conjunto de patrones o datos y se deja que busque un grade de convergencia entre los valores de salida, en un determinado número de iteraciones. En el aprendizaje por refuerzo, el agente se entrena en base a recompensas o castigos los cuales son otorgados por el ambiente, en el cual acciona el agente, buscando maximizar las recompensas (Russell, 2010), este es el tipo de aprendizaje que se utiliza para este proyecto.

1. Red neuronal prealimentada

MLF por sus siglas en inglés (*"Multi-layer feed-forward"*), estas redes neuronales son las más populares y son aplicadas a una gran cantidad de problemas. Una red neural prealimentada consiste en neuronas que son ordenadas en capas (figura 3) El valor de una neurona i es igual a la sumatoria de todas las neuronas de entrada xj multiplicadas por el peso en la conexión wij, donde j representa una neurona de entrada. A esta sumatoria se le puede sumar un valor entero llamado "bias" este sirve para darle un mayor o menor peso a los datos de salida de cada neurona (esto es totalmente opcional).

El valor de la neurona i luego de procesar sus entradas es γi donde.

$$\gamma i = bias + \sum_{j\ entradas} wij \cdot xj$$

Este proceso se repite para cada neurona hasta llegar a las neuronas de salida, el valor de salida de cada neurona también puede ser pasada por una función de activación. La función de activación de un nodo define la salida de ese nodo dada una entrada o un conjunto de entradas, la función de activación más común es la función de Sigmoid. Esta función que tiene una curva en forma de "S", y tiene un dominio de todos los números reales, con un valor de retorno monótonamente incrementado devolviendo valores dentro del rango de 0 y 1 (*Auer 2008*).

$$S(x) = \frac{e^x}{e^x + 1}$$

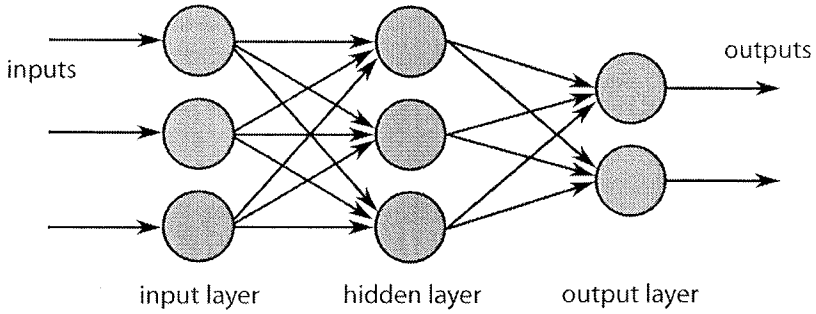


Figura 3. Representación gráfica de una red neuronal prealimentada. La red tiene 3 neuronas de entrada tres neuronas en la capa oculta y 2 neuronas de salida.

III. *"Neuroevolution of Augmented Topologies"* (NEAT)

"Neuroevolution of Augmented Topologies" (NEAT), Neuro-evolución de topologías aumentadas, es un algoritmo genético (GA) desarrollado por Ken Stanley y Risto Miikkulainen en 2002. La representación de los cromosomas que se utiliza en este algoritmo evita los problemas que tenían las representaciones utilizadas anteriormente (Como se menciona en la sección B *Antecedentes del algoritmo*). La técnica clave de NEAT yace en sus marcadores históricos, los cuales permiten :(1) hacer recombinación entre diferentes topologías, (2) proteger la innovación estructural mediante especiación y (3) generar una red neuronal de manera incremental, partiendo de una estructura mínima/ Esto resulta en un aprendizaje más rápido. El entendimiento de este algoritmo es parte esencial del proyecto por lo cual, en esta sección se explica más a detalle cómo funciona el mismo.

A. ¿Porqué NEAT?

La idea para este proyecto surgió al ver una implementación de este algoritmo, resolviendo el juego de Super Mario Bros. Además, para los fines de este proyecto el algoritmo se adecua bastante bien ya que el mecanismo de especiación que posee este mismo, permite una comparación más directa con el mundo real bilógico, esto es importante ya que la idea para este experimento es tomar especies y llevarlas a otros ambientes, para ver cómo evolucionan en los mimos. La analogía biológica sería sacar de su habitad a una población de una especie animal, para llevarla a un habitad diferente esperando cambios en su evolución.

En el mundo de los algoritmos neuro-evolutivos NEAT marca un antes y un después, ya que resolvió muchos de los problemas que existían (como se menciona en el apartado B *Antecedentes del algoritmo*), y hasta la fecha es el algoritmo más popular y usado en el campo. (Stanley, 2017). NEAT ademas fue el método de optimización utilizado en el acelerador de partículas Tevatron para encontrar la estimación de masa más precisa del quark. (Whiteson, 2006)

Stanley (2002:2) hablando sobre la importancia de NEAT:

> « NEAT is also an important contribution to GAs because it shows how it is possible for evolution to both optimize and complexify solutions simultaneously, offering the possibility of evolving increasingly complex solutions over generations, and strengthening the analogy with biological evolution. »

B. Antecedentes del algoritmo

Antes de la creación de NEAT se habían creado varios sistemas que también evolucionaban la topología de una red neuronal, y sus pesos. Estos métodos consisten de varias ideas de cómo se

debería implementar un TWEANN "*Topology and Weight Evolving Artificial Neural Networks*", esto es evolucionar los pesos y además la topología de una red neuronal.

Uno de estos algoritmos es *Structured Genetic Algorithm* (sGA) donde una cadena de bits representa la matriz de conexiones de la red, este método es notable por su simplicidad, ya que opera casi igual que un algoritmo genético normal. A pesar de esto tiene muchas limitaciones, primero que el tamaño de la matriz es igual al cuadrado del numero de nodos en la red, segundo que el tamaño de la cadena debe ser el mismo para todos los organismos (por lo tanto, el mismo número de nodos y conexiones) y tercero, el hacer recombinación mediante una cadena de bits no garantiza que la misma va a resultar en una combinación útil.

Debido a que una cadena de bits no es la manera más natural de representa una red, en la mayoría de los casos de TWEANNs se utiliza una codificación que represente estructuras de grafos de forma más explícita, como *Parallel Distributed Genetic Programming* (PDGP) la cual usa una representación dual, la primera representación es un grafo y la segunda es un genoma con la definición de los nodos, especificado las conexiones de entrada y salida. La idea es que se utiliza una representación o la otra dependiendo de qué operación se estuviera aplicando, para mutar la topología o recombinar la misma, se utiliza el grafo, y para alterar los pesos se utilizaba la representación lineal. Este método comparte una de las limitaciones de sGA, al no poder entrecruzar organismos de diferentes tamaños (número de nodos) por lo cual para entrecruzar se intercambian subgrafos del mismo tamaño, pero no podemos asegurar que los subgrafos intercambiados mediante PDGP son los adecuados para generar una descendencia funcional. (Stanley y Miikkulainen, 2002)

1. Convenciones que compiten

Uno de los principales problemas al momento de usar algoritmos Neuro-evolutivos es las convenciones que compiten o "*Competing Conventions Problem*" (Montana and Davis, 1989; Schaffer *et al.*, 1992). El problema de las convenciones que compiten consiste en tener múltiples soluciones para resolver un problema de optimización de pesos, en una red neuronal. Al aplicar recombinación sobre una pareja de genomas que representan la misma solución, lo más probable es que la descendencia generada por la recombinación de ambos genomas, no logre resolver el problema en cuestión. En la figura 4 se muestra este problema con dos simples redes que resuelven una suma de dos números, obteniendo la misma solución. En este con las tres neuronas A, B y C, se puede representar la misma solución de 6 formas distintas (3!). ¡En general una red con n neuronas ocultas puede tener n! soluciones con funcionalidad equivalente (Stanley y Miikkulainen, 2002). En la figura 4 podemos ver como se perdió un gen de información en la descendencia, que en este caso representa un tercio de la información.

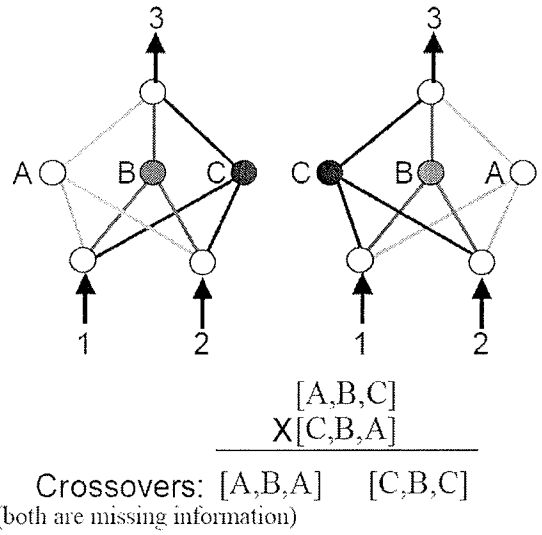


Figura 4: Problema de convenciones que compiten. Ambas redes resuelven le mismo problema (una suma de las entadas), aunque sus ocultos estén en diferente orden. La figura muestra que, en los dos genomas obtenidos al aplicar recombinación, se perdió información C en primer genoma y A en el segundo.

Este problema se complica más en el contexto de TWEANN, la complicación adicional, reside en que la red puede representar soluciones similares, teniendo topologías totalmente distintas, y genomas de tamaños diferentes. Las soluciones que existían anteriormente a NEAT no lograban combinar topologías, de tal manera que la descendencia fuera representativa de sus ancestros.

La intuición principal detrás de NEAT, se origina del problema de lograr representar diferentes estructuras de la misma manera, como genomas de diferentes tamaños, cromosomas diferentes con los mismos genes en posiciones diferentes etc. Resulta que la naturaleza resuelve un problema bastante similar en la reproducción sexual, los genomas en naturaleza tampoco tienen un tamaño fijo, estos se pueden cruzar mediante un proceso llamado *amplificación de genes*.

La naturaleza también tiene una solución para las convenciones que compiten, en este caso la naturaleza utiliza homología: dos genes son homólogos si son alelos si comparten el mismo gen ancestral (Radding, 1982; Sigal and Alberts, 1972). La idea principal de NEAT es que el origen histórico de dos genes es evidencia directa del a homología, entiéndase que si dos genes son homólogos si comparten el mismo origen. NEAT implementa una *sinapsis artificial* basada en marcadores históricos (como se explica en la sección "*Rastrear genes mediante marcadores históricos*" capítulo IV), permitiendo añadir nuevos nodos o nuevas conexiones, sin perder el historial cada gen en el transcurso de la ejecución. (Stanley y Miikkulainen, 2002)

2. Innovación mediante especiación

Al utilizar TWEANNs la innovación de un cromosoma, es darle una nueva estructura a la topología a través de mutación. El problema es que frecuentemente al añadir una nueva estructura a la topología, la aptitud decae. Es muy poco probable que, al introducir un nuevo nodo a la red, este genere una función útil, ya que, aunque el nuevo nodo tuviera el potencial de generar un beneficio, para ello primero los pesos de las conexiones de este deben ser calibrados. Para una estructura nueva logre tener alguna utilidad primero necesita optimizarse. Pero una red con una estructura nueva no tiende a sobrevivir, en un algoritmo genético convencional solo los individuos con la mejor aptitud se reproducen. Por lo tanto, es necesario implementar un método que proteja estas redes con estructuras nuevas, ya que en ellas podría existir una posible mejor solución.

En la naturaleza, diferentes estructuras tienden a estar en diferentes especies que compiten en diferentes nichos, de esta manera la innovación está- protegida de manera implícita por nicho. En las redes neurales se puede hacer lo mismo si agrupamos a las estructuras innovadoras en su propio nicho, de esta forma podrían optimizarse dentro de su propio nicho antes de competir contra toda la población. La especiación ya era motivo de estudio dentro del mundo de los algoritmos genéticos, pero no eran casi utilizados en la neuro-evolución. Para poder utilizar especiación se requiere una función que nos diga si dos genomas pueden estar dentro de la misma especie o no, es difícil formular una función que logre este objetivo, cuando se habla de redes con topologías potencialmente distintas. (Stanley y Miikkulainen, 2002)

NEAT propone una solución para saber si dos genomas pertenecen a la misma especie. Utilizando información histórica sobre los genes, la población puede ser dividida fácilmente en especies. NEAT utiliza un método ya existente "*explicit fitness sharing*", esto es forzar a organismos de la misma especie a compartir la recompensa por su aptitud (como se explica en la sección "*Protección la innovación mediante especiación*" del capítulo IV). La versión original de *"explicit fitness sharing"* agrupa a los individuos conforme a su rendimiento en vez de su similitud genética (Stanley y Miikkulainen, 2002) En el caso de NEAT es más conveniente la idea de agrupar los genes por su similitud genética, debido a que esto puede ser medido mediante la información histórica de los genes, y se ajusta más a lo que sucede en el mundo biológico.

IV. Cómo funciona NEAT

A. Genetic Encoding

NEAT fue diseñado para atender los tres retos antes mencionados. Cada genoma incluye una lista, indicando que nodo apunta que nodo. Cada conexión especifica el peso entre los nodos, la conexión puede estar activada (bit activado), o no (bit desactivado), el peso se indica en ambos casos.

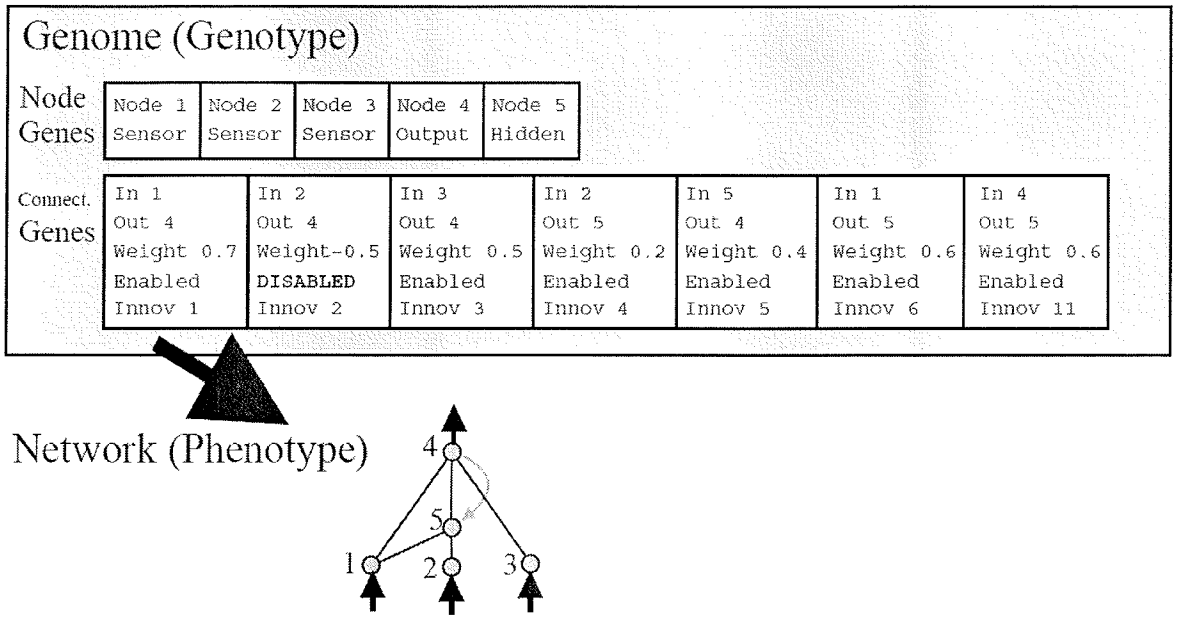


Figura 5: Una representación gráfica del genotipo y fenotipo de un genoma. Se especifican cuales son los nodos de entrada, salida y los ocultos. Podemos observar que el gen numero dos esta deshabilitado, por lo cual no se muestra en el fenotipo.

Además, se indica el *número de innovación* en cada gen, lo cual permite el proceso de recombinación y tener un registro de las innovaciones (como se indica en el apartado B *Rastrear genes mediante marcadores históricos*).

Las mutaciones en NEAT pueden cambiar tanto los pesos como la topología de la red. Los pesos se mutan como en cualquier algoritmo evolutivo, modificando sumando o restando valores aleatorios cuyo rango y desviación son escogidos por el experimentador. Las mutaciones a la estructura suceden de dos maneras (Figura 6), en ambos casos la mutación expande el tamaño del genoma agregando nuevos genes. La primera manera es agregar una nueva conexión entre dos nodos, que no estaban previamente conectados asignando un peso aleatorio a la conexión. La otra forma (agregar nodo), se agrega un nodo en medio de una conexión ya existente, esta conexión se deshabilita y se agregan las dos nuevas conexiones. La conexión entre el nodo de partida y el nuevo

nodo recibe un peso de 1, y la nueva conexión entre el nuevo nodo y el nodo destino original, recibe el valor de la conexión anterior. Esta forma de agregar un nodo se creó de manera de que la mutación causara el efecto mínimo posible.

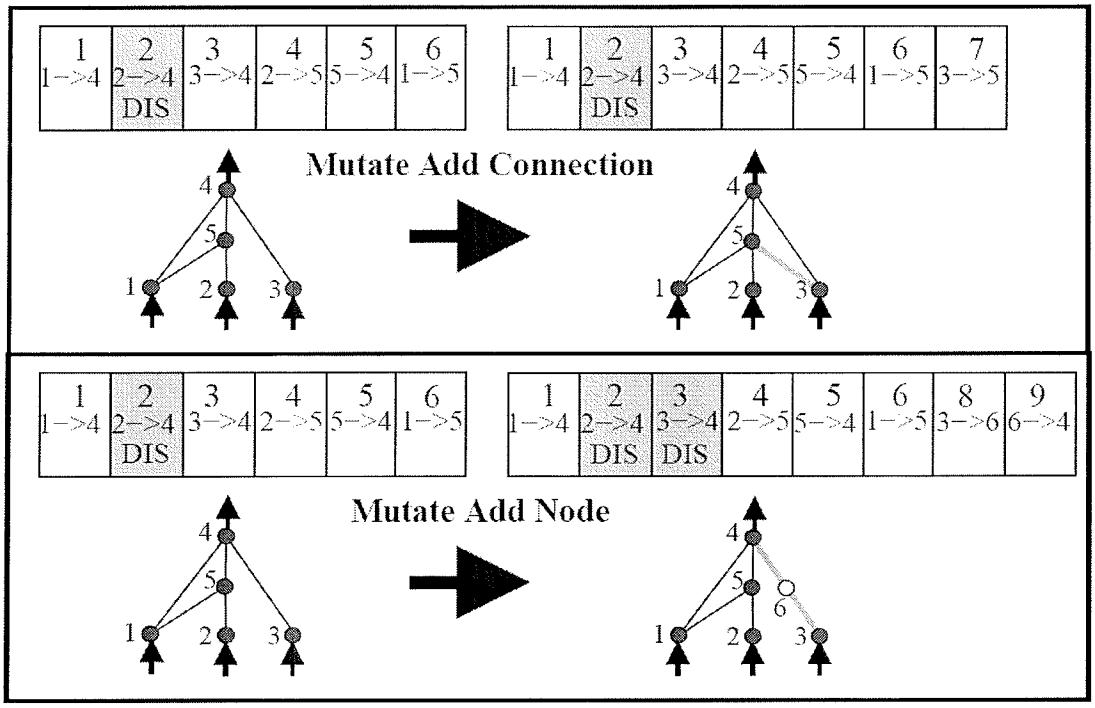


Figura 6: Las dos formas de mutación estructural en NEAT. Podemos ver como en la primera mutación se agrega un gen indicando la nueva conexión entre 3 y 5, además se le asigna un nuevo valor de innovación 7. En el caso de la segunda mutación se deshabilita la conexión entre 3 y 4, y se agregan dos nuevos genes. Una conexión entre 3 y 6(el nuevo nodo), y otra conexión entre 6 y 9, a cada una se le asigno un numero de innovación (8 y 9). Nótese que no se utiliza el numero 7 ya que ya fue utilizado en otra mutación.

Mediante mutación los genomas se hacen cada vez más grandes. Podemos tener genomas de diferentes tamaños, que probablemente tengan conexiones diferentes en la misma posición, es decir en el mismo número de gen. Esto causa una versión más compleja de convenciones que compiten, ya que se debe encontrar una manera de hacer recombinación de genomas variados de una manera que represente la unión entre los mismos sin perder datos importantes. En la siguiente sección se explica cómo se logra resolver este problema (Stanley y Miikkulainen, 2002).

B. Rastrear genes mediante marcadores históricos

Para aplicar la recombinación de los genomas, se debe hallar una manera de saber que genes concuerdan con otros genes, de otros individuos en una población con topologías diversas. La observación clave es que dos genes con el mismo origen histórico representan la misma conexión (aunque posiblemente con pesos diferentes, o uno podrá estar deshabilitado y el otro no), ya que ambos derivan del mismo gen ancestro de algún punto en el pasado.

Rastrear el origen histórico de un gen toma muy poco poder de cómputo (Stanley y Miikkulainen, 2002). Cada vez que un nuevo gen aparece se incrementa el *número global de innovación* y es asignado a dicho gen. De esta forma, el número de innovación nos da un registro cronológico del orden en el que fueron apareciendo los genes. Como ejemplo podemos ver la figura 6, donde podemos suponer que una mutación sucede después la otra. A la conexión creada por la primera mutación se le asigna un el número de innovación 7, en el caso de la segunda mutación el número global de innovación se incrementa, por lo cual se les asignan los números 8 y 9 a sus conexiones. En el futuro, si se aplica recombinación sobre estos estos genomas, la dependencia va a heredar los mismos números de innovación. Los números de innovación nunca son cambiados, de esta forma el origen histórico de cada gen es conocido durante la evolución.

Un posible problema es que en la misma generación mutaciones diferentes contengan la misma estructura, y reciban números de innovación diferentes. Sin embargo, al guardar una lista de las innovaciones que se han creado en la generación actual, podemos identificar los genes con la misma estructura y por lo tanto asignarles el mismo número de innovación.

El manejo de los marcadores históricos le da a NEAT una amplia capacidad, ya que ahora este sistema puede distinguir que genes coinciden. Al momento de aplicar recombinación entre genomas, los genes con el número de innovación se alinean (Figura 7). Estos genes son llamados *matching genes*, al español, *genes que coinciden*. Los genes que no coinciden son considerados genes de *exceso* y *disjuntos* (*excess y disjoint* en la Figura 7), dependiendo si están dentro o fuera del rango de los números de innovación del otro genoma. Estos genes representan una estructura que no existe en el otro genoma. Para crear la descendencia, en el caso de los genes que coinciden, los genes son escogidos de manera aleatoria de entre los dos padres. En el caso de los genes de exceso o disjuntos son siempre escogidos del padre con mejor aptitud, si ambos tuvieran la misma aptitud, se puede escoger entre tomar los genes de cualquiera de los dos padres o tomar los de ambos, pero no una combinación de estos. Los genes que antes fueron deshabilitados ahora tienen un 25% de probabilidad de ser reactivados durante la recombinación, permitiendo el uso de genes antiguos de nuevo

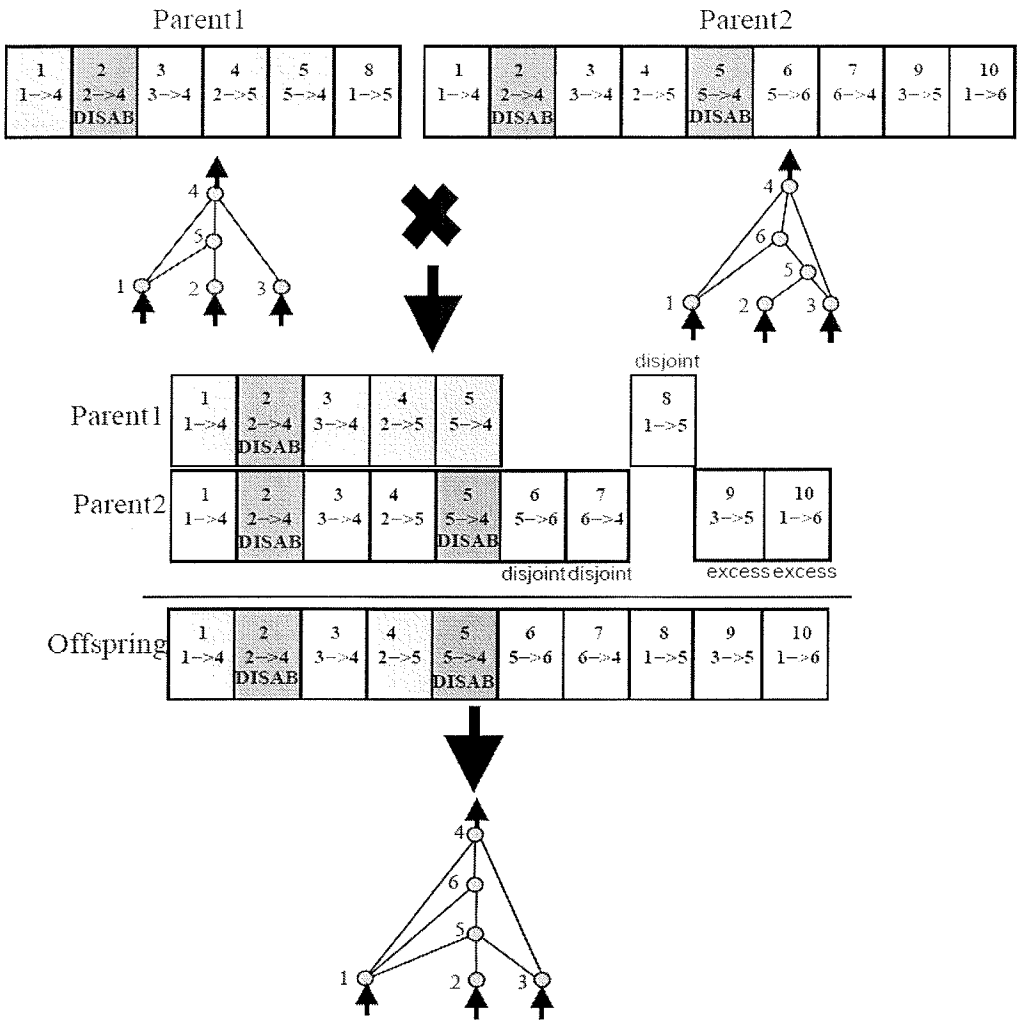


Figura 7: Pareo de genomas con diferentes topologías, mediante el número de innovación. El número de innovación presente en cada gen nos permite saber qué genes de ambos padres coinciden entre sí. De esta manera, sin necesidad de ningún análisis topológico, se puede crear una nueva estructura que combina las partes en las que los padres coindicen, como las que no. La descendencia escoge de manera aleatoria los genes que coinciden, de entre los dos padres. En el caso de los genes de exceso (los que no coinciden al final) y disjuntos (los que no coinciden al medio) se toman los del padre más apto.

Agregando nuevos genes a la población, y logrando recombinar los genomas con diferentes estructuras, este sistema logra crear diversas topologías, evitando así el problema de las convenciones que compiten. Pero resulta que como se explica en la siguiente sección, no es posible

lograr mantener innovación dentro de las estructuras sin aplicar un método que proteja a las mismas. En la siguiente sección se explica cómo se logra esto mediante especificación (Stanley y Miikkulainen, 2002).

C. Protección la innovación mediante especiación

Como fue antes mencionado, separar la población total en especies permite, que los organismos compitan dentro de sus propios nichos, en vez de con la población entera. De esta forma la innovación de las topologías es protegida donde tienen tiempo de optimizarse. En NEAT es posible gracias al tener la data de los marcadores históricos.

El número de genes que no concuerdan ("*excess and disjoint*"), entre dos genomas es la manera natural de medir la compatibilidad entre los dos genomas. Mientras más genes de disjoint/excess existan entre dos genomas, menor es la cantidad de historial genético que comparten. Sabiendo esto podemos calcular la distancia de compatibilidad entre dos genomas δ. En NEAT esto se calcula mediante una combinación lineal del número de genes: exceso E y disjuntos D, y el promedio de las diferencias entre los pesos de los genes de conexión que coinciden $\overline{W}$ (incluyendo los genes deshabilitados)

$$\delta = \frac{c1E}{N} + \frac{c2D}{N} + c3 \cdot \overline{W}$$

Los coeficientes c1, c2 y c3 nos permiten ajustar la importancia que se les quiera dar a cada uno de los factores. El factor N es el número de genes en el genoma más largo, el normalizador N puede ser configurado a uno si los genomas con los que se está trabajando no son muy grandes, por ejemplo, genomas de menos de 20 genes (Stanley y Miikkulainen, 2002). Se puede intuir que elegir los valores correctos de los coeficientes es un factor importante de la experimentación, este aspecto será explorado más adelante en la implementación del algoritmo.

Mediante el uso de la distancia δ entre dos genomas, podemos especificar el umbral de compatibilidad δ_t. Básicamente sería el valor máximo de distancia, dentro del cual dos genomas se consideran de la misma especie. En cada generación los genomas se van situando dentro de especies. Cada especie existente es representada por un genoma aleatorio dentro de la especie, siendo este de la generación anterior. Un genoma g de la generación actual, es situado dentro de la primera especie en la que g sea compatible. Si g no es compatible con ninguna especie existente, entonces se crea una nueva con g como representante. También se mantiene una lista ordenada de todas las especies.

NEAT utiliza "*explicit fitness sharing*" como mecanismo de reproducción, donde especímenes de la misma especie comparten su aptitud. Es importante notar que una especie por muy buen desempeño que tenga no debe volverse demasiado grande, para que ninguna especie se apodere de la población, lo cual esencial para que la evolución de las especies funcione. Por esto calculamos la aptitud ajustada f'_i para un organismo *i*, esta se calcula mediante la distancia δ entre cada otro organismo j en la población donde f_i es la aptitud obtenida por el individuo.

$$f'_i = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))}$$

La función sh() "*sharing function*" devuelve un valor 0 cuando la distancia $\delta(i,j)$ es mayor al umbral δ_t de lo contrario de vuelve un valor de 1(Spears, 1995). Por lo tanto $\sum_{j=1}^{n} sh(\delta(i,j))$ básicamente genera un conteo de los genomas que pertenecen a la misma especie que i según lo que definimos con nuestro umbral δ_t .

A cada especie se le da un numero de descendencia, en proporción de en la suma de las f'_i de los organismos que conforman la especie. De esta manera la suma de f'_i se hace menor al momento de existir muchos organismos de la misma especie ya que $\sum_{j=1}^{n} sh(\delta(i,j))$ aumenta volviendo cada f' un valor más pequeño. Las especies luego proceden a eliminar los miembros de la especie más débiles (menor aptitud) para luego reproducirse.

D. Minimizando la dimensionalidad a través de un crecimiento incremental de una estructura mínima

Los algoritmos TWEANN usualmente empiezan con una población de topologías aleatorias. Esta diversidad debía ser introducida desde el comienzo, ya que nuevas estructuras rara vez sobrevivían con los métodos que se usaban anteriormente. Con NEAT no tenemos ese problema ya que la innovación está protegida por la especiación. Pero ¿Empezar con una población variada trae algún beneficio? Al iniciar con una población con topologías creadas al azar, no hay forma saber qué estructuras son necesarias. Lo que sabemos es qué, esto es costoso a nivel de computo, debido a que cada conexion deriva en un mayor número de dimensiones que deben ser calculadas, para poder optimizar la red. Por lo cual, al empezar con una población con topologías aleatorias, el algoritmo puede estar perdiendo mucho esfuerzo optimizando estructuras que al final resulten innecesarias.

En contraste NEAT comienza con una población uniforme sin nodos ocultos, únicamente los nodos de entrada y salida. Ya que NEAT protege su innovación con especiación, puede empezar de manera mínima y hacer crecer únicamente las estructuras que vayan prevaleciendo, mediante mutaciones. De esta forma NEAT busca un menor número de dimensiones, reduciendo el número de generaciones necesarias para llegar a una solución.

E. Configuración de parámetros

La configuración de parámetros en NEAT son todas las variables y coeficientes que deben ser seleccionadas por el experimentador, como por el umbral δ_t , los coeficientes c1, c2, c3, o la

probabilidad de agregar una nueva conexión o nodo. En el artículo original de "*Neuroevolution of Augmented Topologies*" se menciona una configuración de parámetros base que se utilizada para resolver una serie de problemas. Se hace mención que estos parámetros fueron descubiertos mediante experimentación, y que cada problema puede ameritar una configuración distinta.

En el artículo se hace mencionan dos configuraciones, la primera fue utilizada en todos los problemas resultas (*"Verification: Evolving XORs"*, *"Pole Balancing as a Benchmark Task"* y *"Pole Balancing Comparisons"*) menos uno, y la otra únicamente fue utilizada en el problema de *"Double pole, no velocities"* o DPNV. La configuración cambia principalmente ya que en el problema de DPVN se utiliza una población de 1,000 con el fin de tener más especies para un problema más complejo, para el resto de los experimentos se utilizó una población de 150. Los coeficientes para medir la compatibilidad fueron c1 = 1 c2 = 2 y c3 = 0.4 en el caso de DPNV c3 = 3, de esta forma que la diferencia entre pesos es más significativa al diferenciar especies (debido a que la población de 1,000 tiene espacio para más especies). Si una especie X no mejora luego de 15 generaciones se le quita el permiso a reproducirse (estancamiento= 15). La probabilidad de que el peso de una conexión mute, es de 80% con una probabilidad de cambiar de manera uniforma del 90% y una probabilidad del 10% de cambiar a un valor a aleatorio. La probabilidad de que un gen sea deshabilitado es del 75% en caso de que ya estuviera deshabilitado en alguno de los dos padres. En cada generación 25% de la dependencia viene de mutaciones sin recombinación. En las poblaciones pequeñas la probabilidad de agregar un nuevo nodo es del 3% y la probabilidad de agregar una nueva conexión del 5%, en el caso de la población grande la probabilidad de agregar una nueva conexión es de 30%, porque con una población más grande se puede tener más diversidad de topologías generadas con las conexiones. La función de activación utilizada es la función de sigmoide, esto permite una normalización en los nodos ya que la función resulta en valores entre 0 y 1. También se menciona que por lo general se debe agregar conexiones más seguido de lo que se agregan nodos.

Los datos de esta sección son importantes, ya que estas configuraciones mencionadas, fueron utilizadas como línea de partida para este experimento de Ramificación Neuro-evolutiva. En el capítulo VI se hablará más a detalle de cómo se configuro el algoritmo para optimizar el resultado y el tiempo de ejecución del mismo.

V. Metodología

A. Herramientas

Parte importante de este proyecto fue seleccionar las herramientas a utilizar, ya que como se mencionó anterior mente es importante tener un marco de trabajo adecuado para poder manejar las entradas y salidas de los videojuegos. Pero también es importante saber que herramientas se utilizarían, para las demás partes del proyecto, como puede ser la ejecución del algoritmo NEAT, o guardar los resultados obtenidos en los múltiples experimentos.

La primera herramienta que se busco fue una que nos permitiera interactuar directamente con los videojuegos y tener un control sobre el intercambio de datos, entre el jugador y el videojuego. La primera idea fue buscar juegos de código abierto, el problema con esto, es que es difícil encontrar videojuegos que sean de código abierto aún más encontrar tres similares. Además, si se encontraran tres juegos distintos de código abierto habría que entender el código y lenguaje de cada uno de ellos para luego convertir las entradas y salidas al mismo formato, por todo lo mencionado esta opción de descarto. Otra opción que se consideró fue utilizar un emulador de una consola de videojuegos que nos permitiera guardar el color de los pixeles de la pantalla, para así tomar estos como valores de entrada para la red neural.

1. Open AI Gym

La herramienta que mejor satisfacía las necesidades de este proyecto para la interacción de los juegos es Open AI Gym, es un set de herramientas hecho para desarrollar y comparar, algoritmo de aprendizaje por refuerzo o "*reinforcement learning*". Está diseñado para que se le pueda enseñar a los agentes múltiples trabajos, desde cómo caminar, hasta jugar videojuegos de Atari.AI Gym no asume nada sobre la estructura del agente desarrollado y es compatible con librerías de computación numérica como TensorFlow.

AI Gym fue desarrollado por OpenAI, una compañía sin fines de lucro, dedicada a la investigación de la inteligencia artificial. La misión de Open AI es crear inteligencias artificiales de propósito general seguras, y asegurarse que los beneficios de estas inteligencias artificiales sean distribuidos de manera amplia y uniforme.

La librería de Open AI Gym es una colección de pruebas de problemas (ambientes), que se pueden utilizar para trabajar con algoritmos de aprendizaje por refuerzo, permitiendo al usuario utilizar cualquier algoritmo. Gym implementa un ciclo de agente-ambiente, donde el agente realiza una acción y el ambiente de la devuelve una observación y un premio. La razón por lo cual se decidió utilizar esta herramienta es que posee una amplia gama de ambientes para resolver, entre ellos hay varios videojuegos, además de la facilidad que provee el ciclo agente-ambiente para el propósito de este proyecto, ya que podremos tener acceso a las observaciones y premios para todos los juegos.

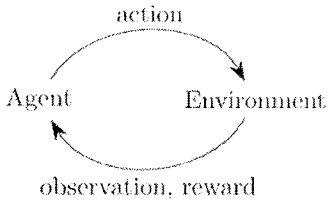


Figura 8. Ciclo agente-ambiente, se muestra como el ambiente envía una observación y un premio luego de recibir una acción del agente.

AI Gym posee cuatro tipos diferentes tipos de ambientes:

- Control clásico y juguete de texto: estos son tareas pequeñas, como balancear un poste mediante un carro, donde el marco de acción del agente es moverse a la izquierda o a la derecha. Estos problemas están diseñados para empezar a utilizar la herramienta.
- Algorítmico: crea computaciones como sumatorias, o volver una secuencia de datos. A pesar de que este tipo de tareas son fáciles para una computadora moderna, el reto es aprender únicamente con ejemplos.
- Atari: Nos permite jugar los clásicos juegos de Atari 2600. Gym integro ALE "*Arcade Learning environment*" el cual es un ambiente que ha tenido un gran impacto en el mundo del a investigación del aprendizaje por refuerzo. Este es el único ambiente que usaremos para este proyecto ya que la idea es usar videojuegos comerciales reales.

Esta herramienta se adapta perfectamente al propósito de este proyecto ya que no solo nos brinda toda la información que necesitábamos, más la ventaja adicional de tenerlo en el ciclo agente-ambiente. También se decidió utilizar esta herramienta debido posee los ambientes de los juegos de Atari dentro de los cuales hay varios juegos que tiene similitud incluso podría decirse que algunos juegos son copias de otros.

2. NEAT-Python

Para facilitar el uso del algoritmo NEAT se utilizó NEAT-Python, la cual es una implementación del algoritmo sin ninguna otra dependencia además de Python. NEAT-Python permite configurar cada parámetro del algoritmo en un archivo de configuración, esto es esencial para poder experimentar y ajustar el algoritmo a cada problema. NEAT-Python también permite utilizar otras configuraciones adicionales que se usan en neuro-evolución, que no necesariamente forman parte del algoritmo original de NEAT. Posteriormente se explicará que parámetros se utilizaron para ejecución del algoritmo, y la razón de porque se escogió cada valor.

NEAT-Python también tiene muchas herramientas que fueron de mucha utilidad para la realización de este proyecto, como la facilidad de guardar reservas de las generaciones, y los datos de las aptitudes durante las generaciones. Es gracias a estas herramientas que se pudo hacer un mejor análisis sobre la utilidad del experimento, ya que se pudieron compararon diferentes historiales de aptitud y graficarlos.

B. Elección de videojuegos

Luego de escoger OpenAI Gym como herramienta, se filtraron las opciones a posibles juegos a utilizar, siendo estos los juegos de Atari que proporciona la herramienta. Por suerte dentro de los juegos de Atari existen juegos muy similares unos de los otros, lo cual es perfecto para poner en prueba este experimento.

En una observación inicial dentro de los juegos de Atari, se notaron dos grupos de juegos similares, los juegos de naves donde las naves se mueven a los lados y los juegos que son similares a Pacman.

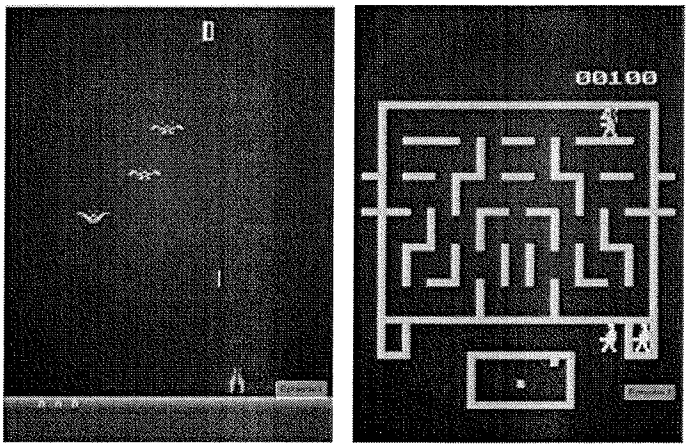


Figura 9. Juegos de Atari, a la izquierda está el juego Demon Atack y a la derecha Wizard of Wor.

Al observar más detenidamente ambos grupos de juegos, se notó que muchos juegos se parecían a Pacman únicamente de forma estética, pero no se parecían en las mecánicas del juego, por lo cual fueron descartados como sujetos de prueba para este proyecto. Por ende, se decidió usar los juegos de naves que fueran similares como sujetos de prueba para el proyecto.

Se escogieron cuatro videojuegos, los cuales son: Space Invaders, Demon Atack, Beam Rider y Assault. Los cuatro son videojuegos que consisten en alcanzar el mayor puntaje posible a medida que se eliminan a los enemigos. Se decidido utilizar dos videojuegos como posible juego base y dos videojuegos como los juegos derivados.

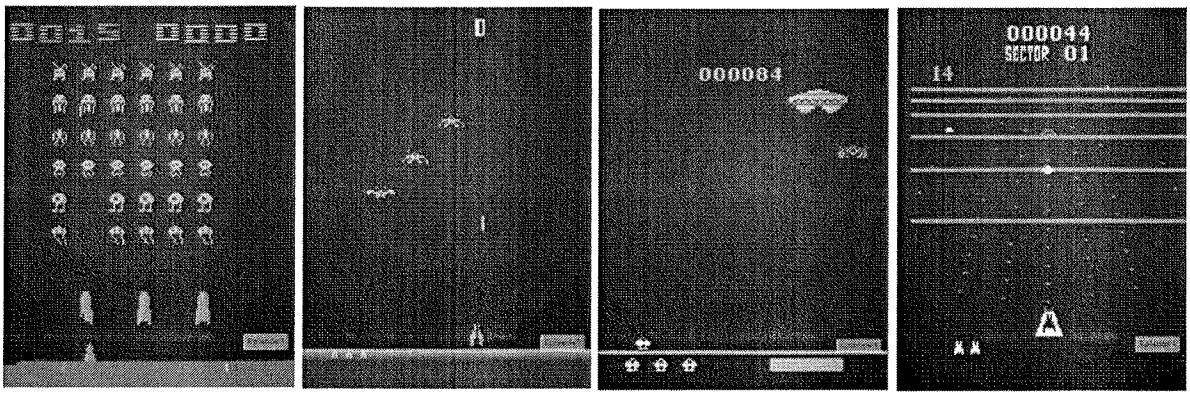


Figura 10. Se muestran los cuatro videojuegos de Atari utilizados en el siguiente orden: Space Invaders, Demon Atack, Assault y Beam Rider.

Los videojuegos que se usaron como juego inicial o juego base fueron Space Invadors y Demon Atack, debido a que estos juegos eran más simples y no incorporan ninguna otra mecánica, además de la de moverse a los lados y disparar, cabe mencionar que aunque las mecánicas de ambos son similares Space Invadors es un juego más lento y además permite al jugador cubrirse detrás de tres estructuras hasta que estas sean destruidas, a simple vista Demon Atack parece un poco más similar a los juegos hijo. Los juegos que se utilizaron como juegos hijo fueron Assault y Beam Rider, cada uno posea cambios a las mecánicas de juego. A continuación, se da una breve descripción de cada uno de los juegos.

- Space Invaders: El fin de este juego es eliminar a la mayor cantidad de enemigos que se mueven en la parte superior de la pantalla, disparándoles proyectiles, estos también son capaces de dispararnos. Los enemigos se mueven lentamente hacia abajo y si logran llegar a la parte baja de la pantalla el juego termina. También hay tres estructuras que permiten cubrirse de los proyectiles enemigos, pero también bloquean los proyectiles propios, estas estructuras se van gastando a medida que se les disparan proyectiles hasta el punto de desaparecer. El juego se empieza con tres vidas las cuales se pierden al recibir un proyectil enemigo, pero si los enemigos llegan a la parte baje de la pantalla se pierde sin importar la cantidad de vidas.

- Demon Atack: En este juego también se acumulan puntos a medida que eliminamos enemigos. Existen varios tipos de enemigo con diferentes patrones de movimiento, los cuales también nos pueden disparar proyectiles. El juego termina cuando luego de ser eliminado tres veces por los proyectiles enemigos.

- Assault: Al igual que los otros juegos se tienen tres vidas las cuales se pierden al recibir un misil enemigo, y se acumulan puntos al eliminar enemigos. La mayor diferencia en este juego es que ahora podemos disparar proyectiles a nuestros lados, ya que los enemigos pueden bajar hasta estar a la altura de la nave, para luego moverse hacia la misma.

- Beam Rider: En este juego tenemos más opciones, ya que tenemos tres proyectiles grandes que podemos disparar. Otra diferencia que tiene con los otros tres juegos es que a pesar de que el moviente también es de forma lateral, solo nos podemos mover dentro de las cinco posiciones fijas. Además, este es el juego que es más diferente gráficamente de los otros tres.

C. Acciones

OpenAI Gym nos da rango de acciones que podemos tomar en cada ambiente, por ejemplo, si en Space Invaders hay 6 posibles movimientos el ambiente acepta únicamente 6 acciones diferentes, las cuales son representados con números entero (en estate caso números en el 0 y 5). A pesar de que los cuatro juegos utilizados pertenecen a la misma videoconsola, las posibles acciones que nos da Gym para cada uno de los juegos son distintas.

Al generar nuestras redes neuronales mediante NEAT debemos especificar el número el número de nodos de entrada y de salida que tendrá cada red. Se decidido utilizar cada nodo de salida de la red neural como las posibles acciones que hay en cada ambiente. De forma que al final del proceso de prealimentarían se escoja el nodo de salida con el valor más alto y se tome dicha acción.

El hecho de que los ambientes tengan diferentes números de posibles acciones es un problema, ya que planeamos reciclar las redes neuronales de un ambiente al otro, y estas redes tienen que tener la misma cantidad de nodos de salida. Por ello se normalizaron las acciones posibles, lo cual hace sentido ya que en el mundo real los cuatro videojuegos funcionan en la misma consola con el mismo control. La diferencia de posibles acciones existe debido a que algunos videojuegos no hacen nada con ciertas entradas del control, mientras en otros juegos si utilizan estas entradas. Algunos números representaban acciones bastante diferentes (por ejemplo, el valor 1 es la acción de no hacer nada en Assault, pero en el resto de los juegos es la acción de disparar)

Tabla 1

Acciones que se toman originalmente en cada ambiente respecto al número entero de entrada.

Acción	Space Invadors	Demon Atack	Assault	Beam Rider
0	No hacer nada	No hacer nada	No hacer nada	No hacer nada
1	Disparar proyectil	Disparar proyectil	No hacer nada	Disparar proyectil
2	Moverse a la derecha	Moverse a la derecha	Disparar proyectil	Disparar proyectil grande
3	Moverse a la izquierda	Moverse a la izquierda	Moverse a la derecha	Moverse a la derecha
4	Moverse a la derecha y disparar	Moverse a la derecha y disparar	Moverse a la izquierda	Moverse a la izquierda
5	Moverse a la izquierda y disparar	Moverse a la izquierda y disparar	Moverse a la derecha y disparar	Moverse a la derecha y disparar
6	No existe	No existe	Moverse a la izquierda y disparar	Moverse a la izquierda y disparar

Tabla 2

Acciones que se toman en cada ambiente respecto al número entero de entrada, luego de normalizar las acciones de forma congruente.

Acción	Space Invadors	Demon Atack	Assault	Beam Rider
0	No hacer nada	No hacer nada	No hacer nada	No hacer nada
1	Disparar proyectil	Disparar proyectil	Disparar proyectil	Disparar proyectil
2	Moverse a la derecha	Moverse a la derecha	Moverse a la derecha	Moverse a la derecha
3	Moverse a la izquierda	Moverse a la izquierda	Disparar a la izquierda	Moverse a la izquierda
4	Moverse a la derecha y disparar	Moverse a la derecha y disparar	Disparar a la derecha	Moverse a la derecha y disparar
5	Moverse a la izquierda y disparar	Moverse a la izquierda y disparar	Moverse a la izquierda y disparar	Moverse a la izquierda y disparar
6	Se redirige a 0	Se redirige a 0	No hacer nada	Disparar proyectil grande

D. Configuración del experimento

Para el desarrollo de este proyecto se utilizó el lenguaje Python, principalmente porque es el lenguaje que soporta OpenAI Gym, además Python posee librerías que pueden ayudar a realizar redes neuronales como Numpy y Tensorlow. Gym soporta oficialmente Linux y OS X, Python 2.7 y 3.5. Por comodidad del experimentador se utilizó inicialmente una máquina virtual de Linux dentro de Windows, debido a que el algoritmo necesita bastante poder de procesamiento se decidió cambiar de mitología para no perder procesamiento en la máquina virtual o el huésped de la misma. Se instalo la herramienta de Ubuntu 18.04 LTS para Windows 10 la cual nos permite utilizar el bash de Linux dentro de Windows y se instalaron todas las dependencias necesarias para trabajar con Gym, además se instaló y configuró VcXsrv para poder visualizar las partidas al momento de renderizarlas. Para la configuración del algoritmo de NEAT, inicialmente se intentó apegarse lo más posible a la configuración recomendada en el documento académico de "*Evolving Neural Networks through Augmenting Topologies*".

1. Observación del ambiente

OpenAI Gym provee dos posibles métodos de observación de los ambientes, al momento de usarlos los juegos de Atari. Una opción nos provee con los valores RGB de cada pixel de la imagen en cada cuadro. La otra opción nos permite observar la memoria RAM de la consola. Se escogió usar la opción, que observa la memoria RAM ya que este consiste únicamente de 128 bytes, a diferencia de la matriz de pixeles en la que se tienen tres valores por cada píxel y el tamaño de la matriz de pixeles es de 160 x192 (la resolución de la consola Atari). Debido a esto todas las ANNs creadas tiene 128 nodos de lectura y 7 nodos de salida uno para cada posible acción.

2. Función de aptitud

La función de aptitud toma una lista de genomas (población) y le asigna a cada uno de ellos su valor de aptitud. Para cada uno de los genomas:

1. Generamos la ANN que corresponde al genoma
2. Empezamos con valor de aptitud igual a cero
3. Inicializamos el ciclo agente-ambiente de Gym con el respectivo ambiente (videojuego)
4. Se inicializa la propagación de la ANN con la observación del ambiente como entrada
5. Se toma la acción respectiva a la salida de la ANN y se la damos al ambiente
6. Se toma el premio y la nueva observación del ambiente
7. Se suma el premio a la aptitud.
8. Repetir el ciclo con la nueva observación hasta que el videojuego se termine

(código en la sección de anexos)

Esta función vuelve al algoritmo muy lento, debido a que calcula la aptitud de cada uno de los individuos en cada generación una por una, por ello se implementó multiprocesamiento para poder calcular las aptitudes de varios genomas de forma concurrente.

Para esto se utilizó el paquete de multiprocesamiento de Python y la función *pool*, esta nos permite controlar un grupo de procesos de trabajo a los que se les puede asignar tareas, también se puede definir cuantos núcleos del procesador puede usar el grupo de procesos.

Otro problema encontrado al momento de calcular la aptitud fue que muchas veces la función tomaba mucho más tiempo, debido a que el agente se quedaba en un punto donde no estaba generando puntos por eliminar enemigos, pero tampoco era alcanzado por ningún proyectil enemigo. Por ello se implementó un contador para asegurarse de que no se quede en un estado donde han transcurrido varios pasos, pero no ha generado ningún incremento en la aptitud, si ha llegado a esta cantidad de pasos el ciclo termina y se devuelve el puntaje obtenido hasta ese momento como aptitud del genoma.

3. Almacenamiento de información

Para poder analizar los resultados de ejecución era importante tener alguna manera de almacenar los datos, por suerte NEAT-Python provee la opción de usar reportes, para este programa se utilizaron dos tipos de reportes, uno de control y otro de estadísticas. Con los reportes de control podemos indicar un intervalo de generación para guardar a la población entera, esto lo logra utilizando Pickle es cual es un modulo de Python que permite serializar objetos y guardarlos en archivos. Los reportes estadísticos en cambio nos pueden brindar datos sobre la aptitud en las generaciones o la cantidad de especies en cada generación, estos datos son guardados en un archivo formato csv. Además de esto, también se guardaron los genomas ganadores de cada ejecución, esto para poder mostrar de manera gráfica y así poder tener una noción de lo que están haciendo los agentes, además de esta forma se ahorra la tarea de volver a correr el algoritmo desde el último punto guardado, cada vez que quisiéramos ver al agente ganador jugar, esto se implementó mediante Pickle.

E. Configuración del algoritmo

En el documento original de *"Neuroevolution of Augmented Topologies"* se mencionan dos configuraciones recomendada, se decidido iniciar las pruebas con la confirmación para problemas complicados, adaptándolo de forma tal que tuviera sentido en el contexto de los ambientes utilizados. Estos valores fueron modificados a medida que se realizaron más pruebas, buscando masificar la aptitud de las especies y minimizar el tiempo ejecución.

Tabla 3

Configuración inicial del algoritmo NEAT

Población	1,000
Función de activación	Sigmoid
Coeficientes de compatibilidad(conexiones) c1 y c2	1.0
Coeficiente de compatibilidad (pesos) c3	3.0
Probabilidad de añadir una conexión	0.15
Probabilidad de eliminar una conexión	0.1
Probabilidad de añadir un nodo	0.3
Probabilidad de eliminar un nodo	0.1
Umbral de compatibilidad δ_t	4.0
Porcentaje de reproducción por especie	0.2
Estancamiento	15

Con esta configuración se iniciaron las primeras pruebas, se inició una ejecución durante 100 generaciones, utilizando como ambiente el juego de Space Invaders. Esta ejecución brindo los siguientes resultados.

- Tiempo de ejecución: 8 horas con 54 minutos
- Aptitud del mejor genoma: 625

Debido al largo tiempo de espera se hicieron más pruebas reduciendo la población a un total de 150, este número también se tomó de las recomendaciones de NEAT. Con este cambio a la población también se ajustaron los valores el resto de las variables, ya que al tener una población más pequeña no hay tanto espacio para una gran cantidad de especies, por lo cual se realizaron cambios en los coeficientes que corresponden a la compatibilidad entre especies y evitar el problema de que cada individuo sea una especie distinta. También se hizo una prueba cambiando las conexiones iniciales de forma tal que todos los nodos de entrada estuvieran conectados a los nodos de salido, este cambio se realizó con la idea de que al tener las conexiones que probablemente el algoritmo terminaría generando de igual forma.

Sin conexiones iniciales:

- Tiempo de ejecución: 1 horas con 9 minutos
- Aptitud del mejor genoma: 625

Con los nodos de entrada y salida conectados

- Tiempo de ejecución: 2 horas con 22 minutos
- Aptitud del mejor genoma: 880

Se puede notar que la ejecución con la población más grande no represento ninguna mejora respecto a las ejecuciones con una población de 150 individuos, a pesar de haber tardado mucho más que estas. Esta diferencia entre las configuraciones se vuelve mucho más cerrada si tomamos

en cuenta que la aleatoriedad del ambiente, ya que los videojuegos manejan muchas variables aleatorias. Esto se puedo comprar al observar a los agentes ganadores jugar, ya su puntuación fue muy variada en los tres casos dando resultados entre 500 y 880.

La similitud tan grande entre las tres pruebas se le atribuye a la probabilidad asignada de agregar y eliminar nodos y conexiones, ya que al porcentaje ser tan bajo no logra generar muchas variaciones dentro de las 100 generaciones. Respecto si es mejor empezar con o sin conexiones, pareciera que la configuración con conexiones resulta en una mejor aptitud. Pero la diferencia es mínima, aún más considerando la aleatoriedad de los resultados, y que ambos agentes ganadores parecieran comportarse igual. Debido a que empezar la ejecución sin conexiones es un proceso mucho más rápido, parece que es la mejor opción, además que conserva el principio de NEAT sobre empezar con una topología mínima. La decisión de empezar con redes sin conexiones se vuelve más obvia al observar la graficas de cómo va mejorando la aptitud en cada generación.

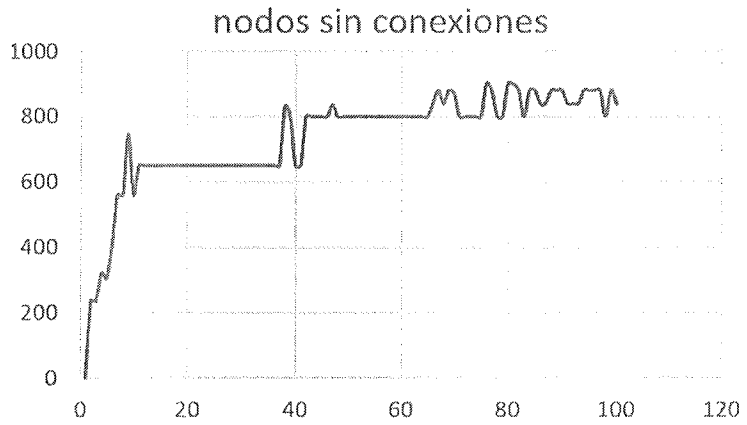


Figura 11. Gráfico de dispersión, de aptitud máxima por generación. Este es el gráfico para la configuración sin conexiones iniciales, podemos apreciar como mejora la aptitud respecto al número de generación.

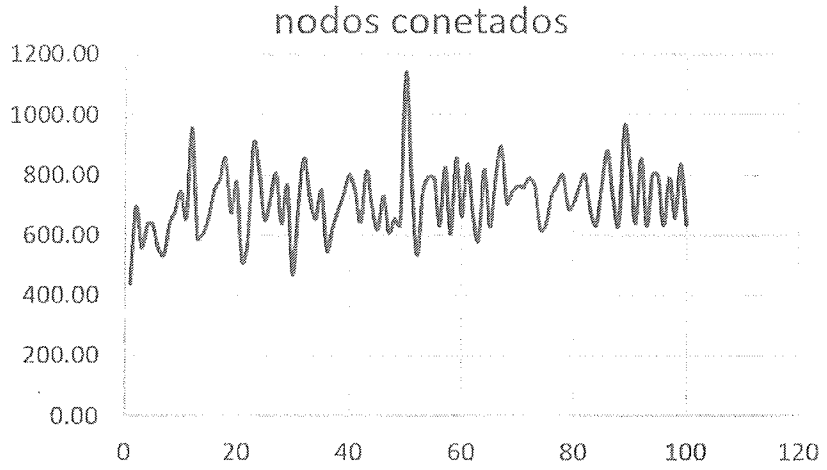


Figura 12. Gráfico de dispersión, de aptitud máxima por generación. Este es el grafico para la configuración con conexiones iniciales, a pesar de que se empieza con una mejor aptitud, esta no mejorar mucho respecto al número de generación, además varía mucho de valor dando la impresión de que la aptitud depende mucho más de la aleatoriedad.

Otra observación importante que se realizo es que en ambos casos no se estaban generando nuevas especies, es decir que los 150 individuos perecen siempre a la única especie, esto demuestra que el umbral de compatibilidad δt es muy alto, por lo cual se disminuyó. Al disminuir el umbral y permitir la existencia de más especies, se mejoraron los resultados en aptitud.

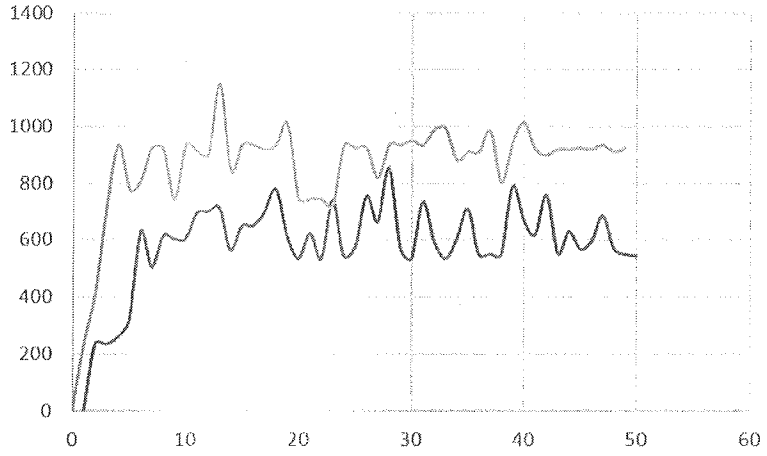


Figura 13. Gráfico de dispersión, de aptitud máxima por generación, de dos diferentes ejecuciones. la línea azul representa la ejecución con un valor $\delta t=3$, y la línea naranja la ejecución con $\delta t=2$.

Podemos comprobar que el tener un umbral de compatibilidad más bajo, brinda un mayor crecimiento de la aptitud, esto probablemente se deba a que con la configuración de δt=2 se tiene más de una especie, lo cual abre paso a la innovación. (en los archivos de reporte, podemos ver como la primera configuración mantuvo una sola especie mientras al cambiar δt=2 se obtuvo una variad de especies a lo largo de las generaciones)

Al final de varias pruebas se pulieron más los parámetros, pensando en las necesidades del problema que se quiere resolver sin dejar de tomar en cuenta los valores recomendados y la lógica detrás de los mismos. Esta es la configuración que se utilizó para el resto de las ejecuciones.

Tabla 4

Configuración del algoritmo NEAT

Población	200
Función de activación	Sigmoid
Coeficientes de compatibilidad(conexiones) c1 y c2	1.0
Coeficiente de compatibilidad (pesos) c3	0.4
Probabilidad de añadir una conexión	0.3
Probabilidad de eliminar una conexión	0.15
Probabilidad de añadir un nodo	0.2
Probabilidad de eliminar un nodo	0.1
Umbral de compatibilidad δ_t	2.0
Porcentaje de reproducción por especie	0.2
Estancamiento	15

VI. Resultados

Se resolvieron los cuatro juegos de manera individual simulando cada ambiente durante 100 generaciones. Luego se procedió a resolver Beam Rider y Assault, partiendo de las especies generadas en Demon Atack y Space Invaders, estas ejecuciones también fueron de 100 generaciones para compararlas con las originales. Nótese que el puntaje para cada juego se calcula de forma diferente. (el código y las tablas de datos se encuentran en los anexos)

Resultados finales de las tres ejecuciones:

Beam Rider:

- Tiempo de ejecución: 8145 segundos
- Aptitud del genoma ganador: 996
- Aptitud promedio de la población: 799
- Especies :14

Beam Rider partiendo de Space Invaders

- Tiempo de ejecución: 8735 segundos
- Aptitud del genoma ganador: 996
- Aptitud promedio de la población: 897
- Especies :4

Beam Rider partiendo de Demon Atack

- Tiempo de ejecución: 8735 segundos
- Aptitud del genoma ganador: 1332
- Aptitud promedio de la población: 860
- Especies :4

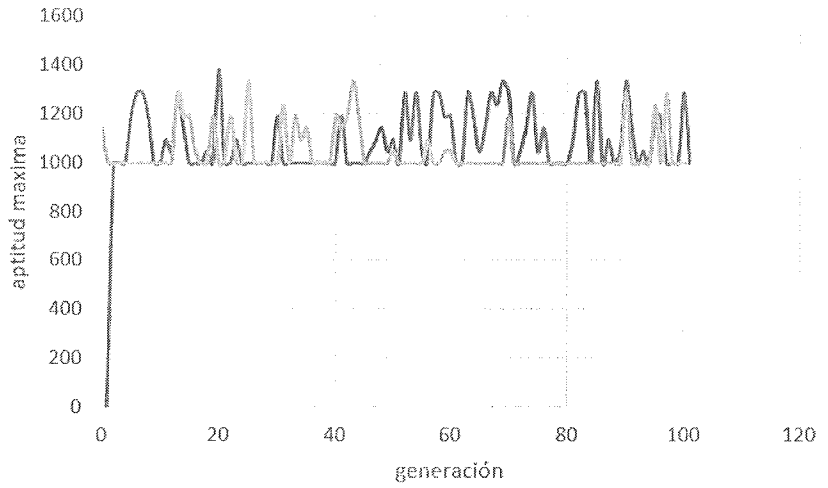


Figura 14. Gráfico de dispersión, de aptitud máxima por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Beam Rider empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Space Invaders. Podemos observar que no existe una diferencia significativa entre las dos líneas, además del origen.

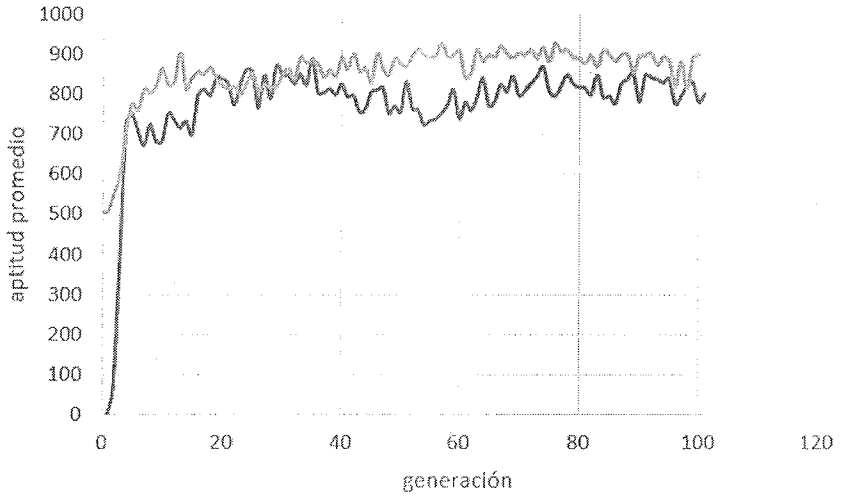


Figura 15. Gráfico de dispersión, de aptitud promedio por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Beam Rider empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Space Invaders. Se puede apreciar que la ejecución basada en pase invadir se mantiene por arriba de la otra ejecución en casi todas las generaciones, esto tiene concordancia con los resultados finales de las ejecuciones.

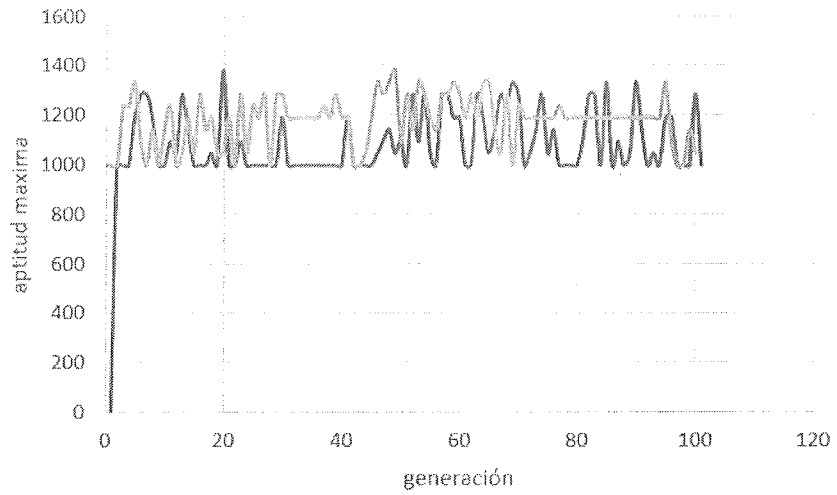


Figura 16. Gráfico de dispersión, de aptitud máxima por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Beam Rider empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack. Se puede apreciar que después de la generación numero 40 la línea amarilla obtuvo valores más altos que la azul, lo cual parecería indicar que la utilización de las especies de otro ambiente pudo haber brindado una pequeña ventaja.

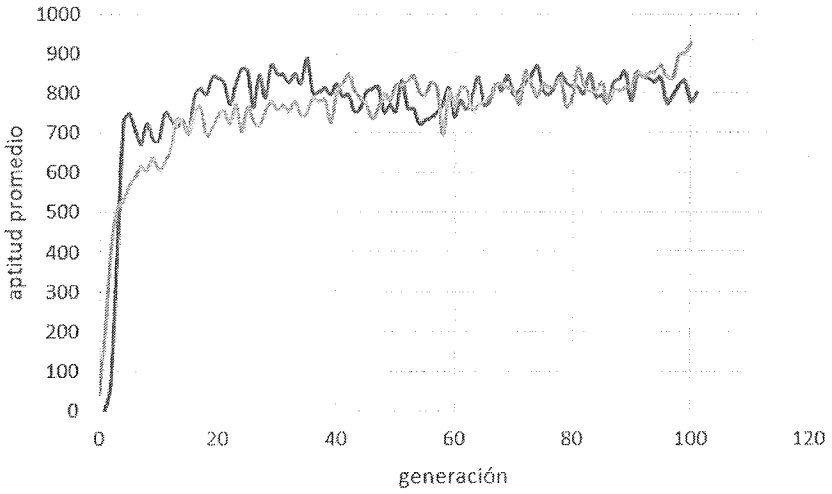


Figura 17. Gráfico de dispersión, de aptitud promedio por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Beam Rider empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack. Se puede observar que la ejecución que empieza con los genomas aleatorios tiene ventaja durante las primeras 40 generaciones, pero luego la otra ejecución es capaz de alcanzarla e incluso superarla al final.

Resultados finales de las tres ejecuciones:

Assault:

- Tiempo de ejecución: 3341 segundos
- Aptitud del genoma ganador: 819
- Aptitud promedio de la población: 438
- Especies :5

Assault partiendo de Space Invaders

- Tiempo de ejecución: 2714 segundos
- Aptitud del genoma ganador: 714
- Aptitud promedio de la población: 428
- Especies :8

Assault Rider partiendo de Demon Atack

- Tiempo de ejecución: 2844 segundos
- Aptitud del genoma ganador: 1802
- Aptitud promedio de la población: 488
- Especies :6

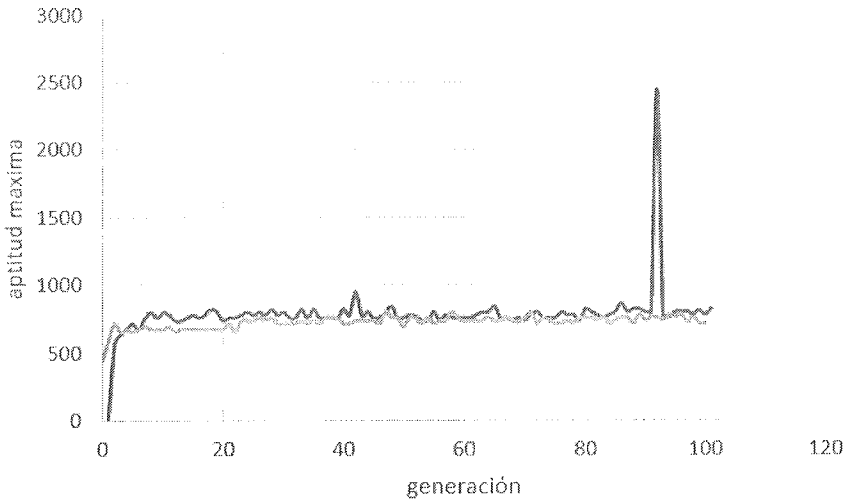


Figura 18. Gráfico de dispersión, de aptitud máxima por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Space Invaders. Podemos observar que no existe una diferencia significativa además de un punto atípico entre la generación 80 y 100.

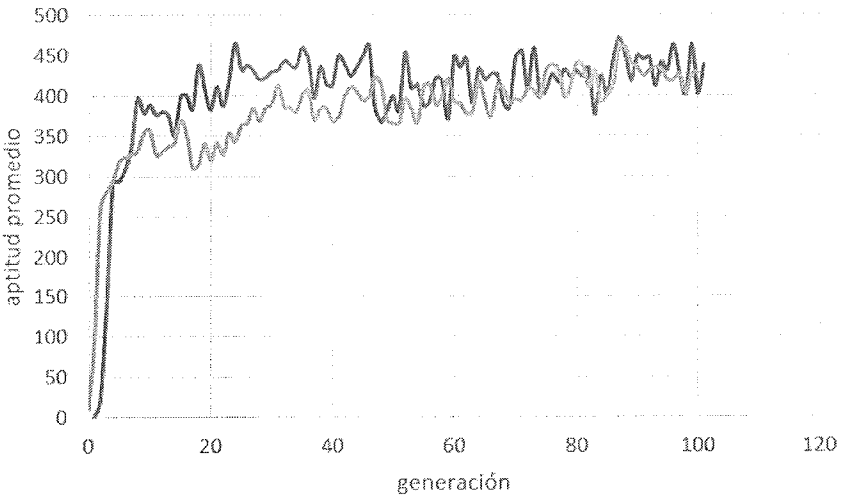


Figura 19. Gráfico de dispersión, de aptitud promedio por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Space Invaders. Se puede observar que la ejecución que empieza con los genomas aleatorios tiene ventaja durante las primeras 40 generaciones, pero luego la otra ejecución es capaz de alcanzarla e incluso superarla en ciertos puntos.

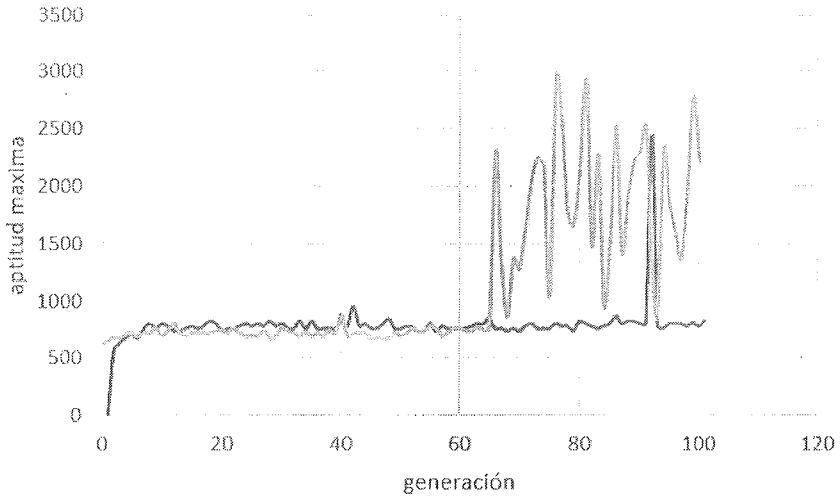


Figura 20. Gráfico de dispersión, de aptitud máxima por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack. Se puede apreciar que después de la generación numero 60 la línea ejecución con las especies recicladas toma una ventaja bastante significativa, pero con valor muy variados.

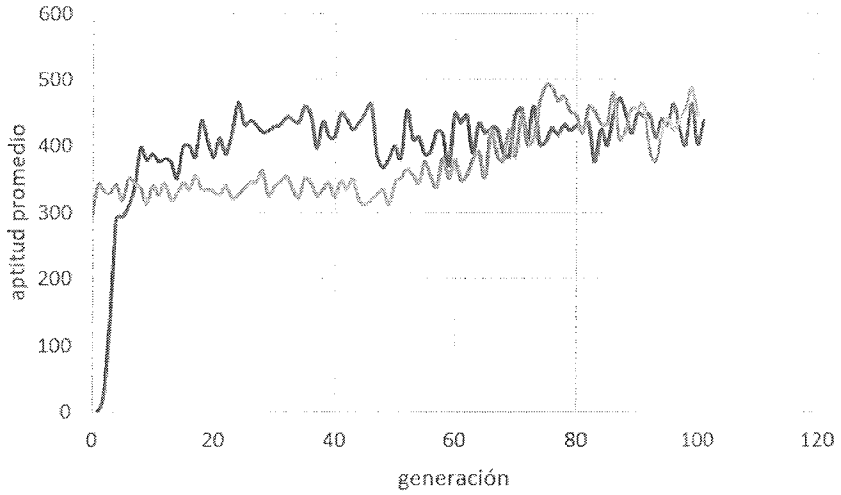


Figura 21. Gráfico de dispersión, de aptitud promedio por generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack. Se puede observar que la ejecución que empieza con los genomas aleatorios mantiene un promedio mayor durante casi todas las generaciones, pero aproximadamente luego de la generación 70, la ejecución basada en las especies de generadas en Demon Atack logra Alcanar el valor de la otra ejecución he incluso pasarla.

VII. Discusión

Podemos ver la ventaja que parecen tener las ejecuciones las ejecuciones que parten de un juego base, es alcanzar los máximos (o máximos locales) en una menor cantidad de iteraciones, y además que inician con una mejor aptitud, en contraste con sus contrapartes que inician todas con una aptitud de 0. Era de esperarse que estas ejecuciones iniciarán con ventaja, ya que los agentes de estas, inician con redes neuronales más evolucionadas, sin embargo no era seguro que estas redes neurales fueran a producir una aptitud positiva, pero este fue el caso para los cuatro experimentos.

La ejecución de Assault que parte de Demon Atack parece ser un caso diferente al resto, ya que por la línea que traza las aptitudes de los juegos que parten de un juego base, no suelen separarse mucho de sus contrapartes de las ejecuciones que parten desde cero. En el caso de esta ejecución las aptitudes máximas después de la generación 60 aumentan de gran manera, estas aptitudes se mantenían alrededor de 700, pero después de este punto la aptitud salta a 2,305 y se mantiene oscilante entre 3000 y 1,000, mientras las aptitudes de la ejecución que parte desde cero se mantienen cerca de 700. En este caso específico parece que iniciar con una población de cromosomas que parten de otro ambiente, nos da una ventaja bastante importante, ya que se obtienen aptitudes máximas hasta cuatro veces mayores a las de la otra ejecución. También es importante notar que a partir del punto en el que la ejecución en cuestión incrementó su aptitud, esta empezó a variar mucho más.

Para saber por qué se creó un incremento tan repentino en la aptitud máxima de las especies que parten de Demon Atack, se decidió observar el comportamiento de ambos agentes ganadores al final de la generación número 100. Esta observación reveló que el agente que inicia con los genomas nuevos no es capaz de disparar a los lados, mientras el agente que parte de los genomas de Demon Atack ya es capaz de disparar a la izquierda, esto explica porque los agentes de esta ejecución logran obtener puntajes mucho más altos, ya que, a partir de la segunda oleada de enemigos, es necesario disparar a los lados para poder sobrevivir. Parece que el haber empezado con los agentes de otro juego le permitió a esta ejecución llegar a este punto primero.

Debido a que este experimento resultó tan diferente a los otros tres, se retomaron ambas ejecuciones del mismo, por otras 100 generaciones más, revelando los siguientes datos.

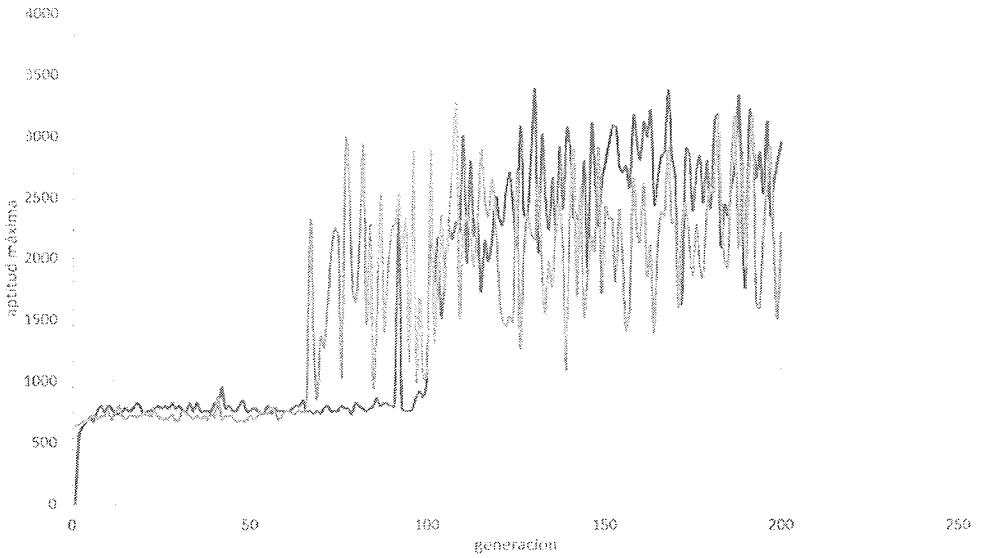


Figura 22. Gráfico de dispersión, de aptitud máxima por 200 generaciones, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack.

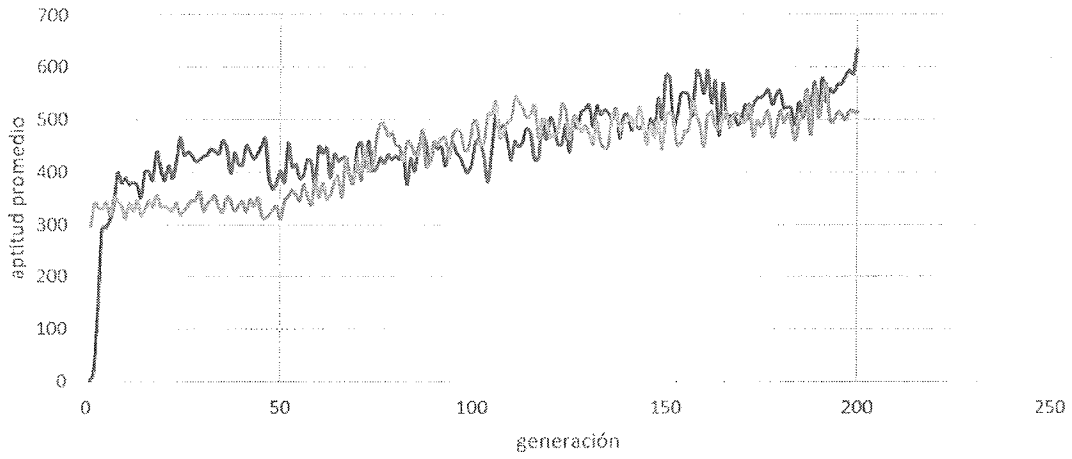


Figura 23. Gráfico de dispersión, de aptitud promedio por 200 generación, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack.

Los gráficos de dispersión nos muestran que, la ejecución de Assault que parte desde cero también alcanza aptitudes más altas, aproximadamente luego de la generación 100. Pareciera que la ejecución que parte desde cero alcanza el mismo punto en el que la aptitud máxima incremente de manera rápida rondando entre 3,000 y 2,000, estas aptitudes máximas también varían mucho, al igual que en la otra ejecución.

Ambas ejecuciones alcanzan un punto en el que la aptitud aumenta mucho, pero en ambos casos esta aptitud máxima se vuelve mucho más variada, alcanzando puntajes que son hasta tres veces más pequeños tres o más veces más grandes al puntaje anterior. Al ver este este comportamiento tan variado en los genomas ganadores de cada generación se decidió observar el comportamiento de estos agentes dentro del juego.

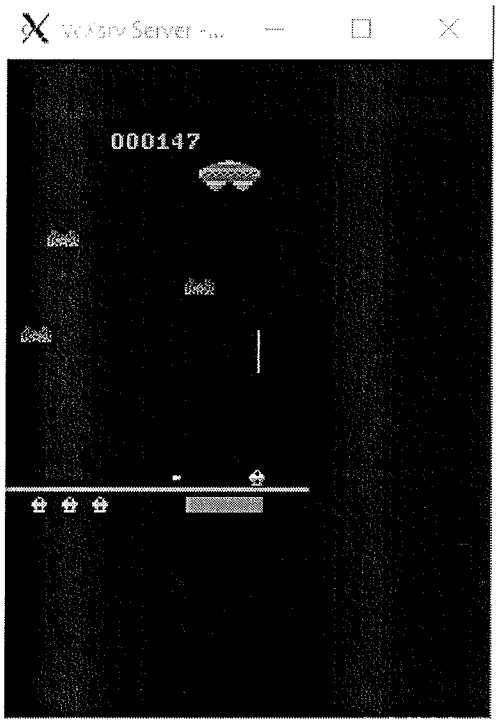


Figura 24. Captura de la partida del agente ganador de la ejecución de 200 generaciones de Assault, en la que se partido de los agentes que resultaron de resolver Demon Atack. El agente dispara hacia arriba y hacia la izquierda en todo momento.

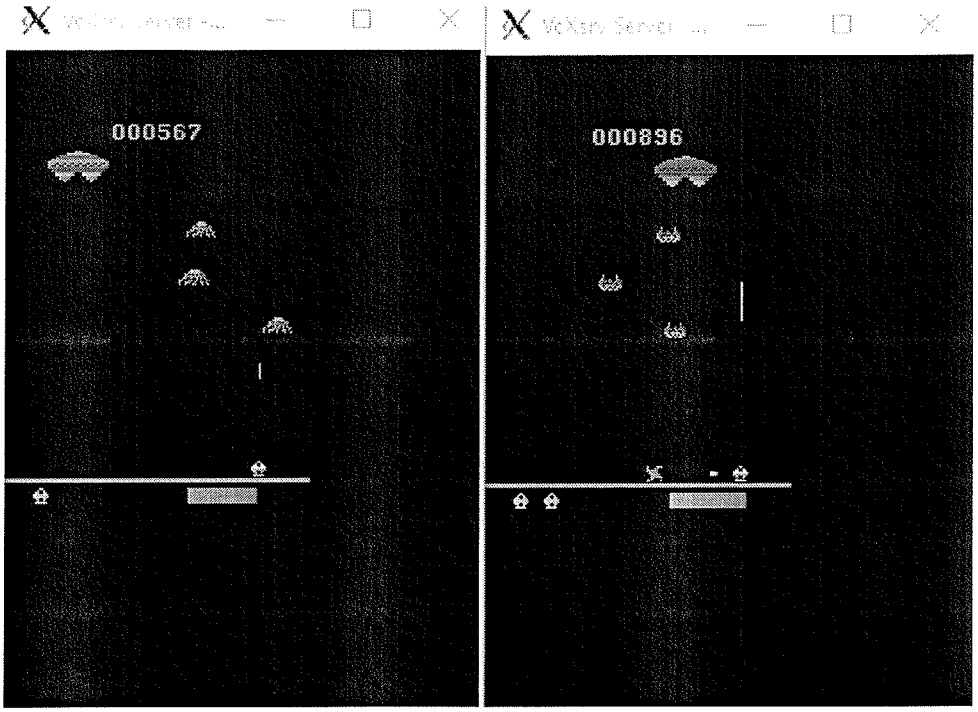


Figura 25. Capturas de la partida del agente ganador de la ejecución de 200 generaciones de Assault, en la que se parte con agentes generados desde cero. El agente dispara hacia arriba en todo momento y dispara a la izquierda únicamente al ser amenazado.

Ambos agentes ganadores mostraron un comportamiento similar, pero con diferencias interesantes. Ambos disparan en todo momento hacia arriba, mientras se mueven a la derecha, para luego quedarse en extremo derecho y nunca más moverse de ahí. El agente que inicia con genomas nuevos se mueve a la derecha de a poco, mientras el otro agente se mueve de golpe, pero en ambos casos parece que estos movimientos iniciales son algún tipo de reacción a los proyectiles enemigos. La otra diferencia radica en que el agente que parte de la ejecución de los genomas de Demon Atack, dispara a la izquierda en todo momento luego de llegar al extremo derecho de la pantalla, mientras el agente de la ejecución que parte con genomas creados desde cero, dispara únicamente al a izquierda cuando está en peligro de ser golpeado por este costado. En otras palabras, este agente dispara de manera reactiva, en contraste con el otro que dispara a la izquierda en todo momento. A pesar de esto ambos métodos parecen ser igualmente efectivos y en este punto alcanzar un mayor o menor puntaje podría depender más de la aleatoriedad del ambiente/juego.

El correr los agentes ganadores de forma individual nos revela más información, no solo para apreciar el comportamiento de estos, sino que también se pudo observar una variación en los puntajes, la cual es mayor a la que nos muestra la gráfica de aptitudes máximas, ya que en estas podemos ver la aptitud máxima o aptitud promedio de entre todas las especies, pero no como varía la aptitud de un mismo agente o cromosoma. En este caso ambos agentes mostraron una variación de puntajes muy grande entre partidas, los puntajes observados estuvieron entre 500 a 2000 puntos.

Debido a la aleatoriedad del juego, no sabemos si la diferencia entre los puntajes de los agentes ganadores de la generación 200, se deba a que uno de los agentes sea más eficiente, o no. Por ello se ejecutaron 1,000 veces ambos agentes, para poder saber con qué frecuencia estos obtienen mejores o peores puntajes, y así poder determinar cuál de estos agentes es superior. Por conveniencia llamaremos "*agente derivado*" al agente ganador, el cual fue generado por NEAT al usar la última generación de genomas creados al resolver el juego Demon Atack, como generación cero. Llamaremos "*agente bruto*" al agente ganador, el cual fue generado por la ejecución de NEAT partiendo de nuevos genomas generados de manera aleatoria.

A continuación, se muestran dos histogramas, los cuales muestran los puntajes obtenidos por ambos agentes durante las 1,000 partidas.

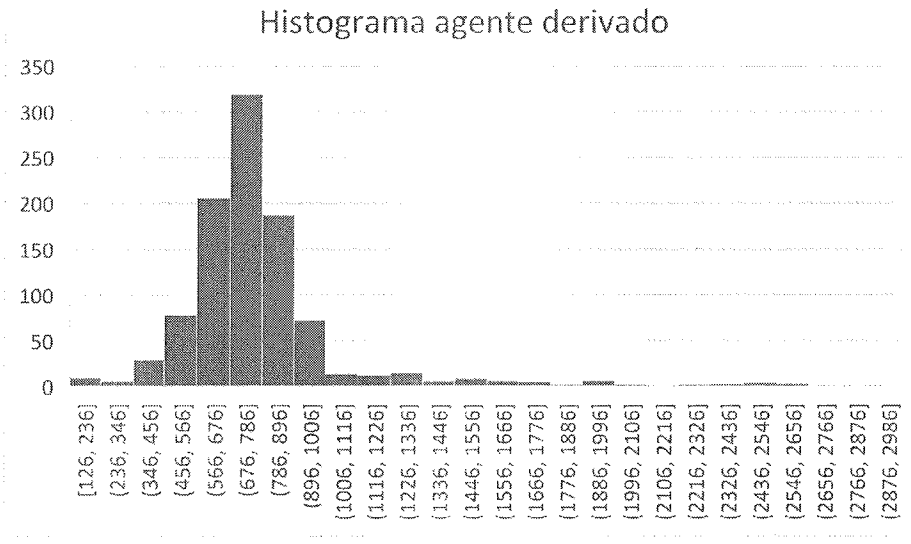


Figura 26. Histograma de los puntajes obtenidos durante 1,000 partidas por el agente derivado, con una media de 789.5 y una desviación estándar de 327.43

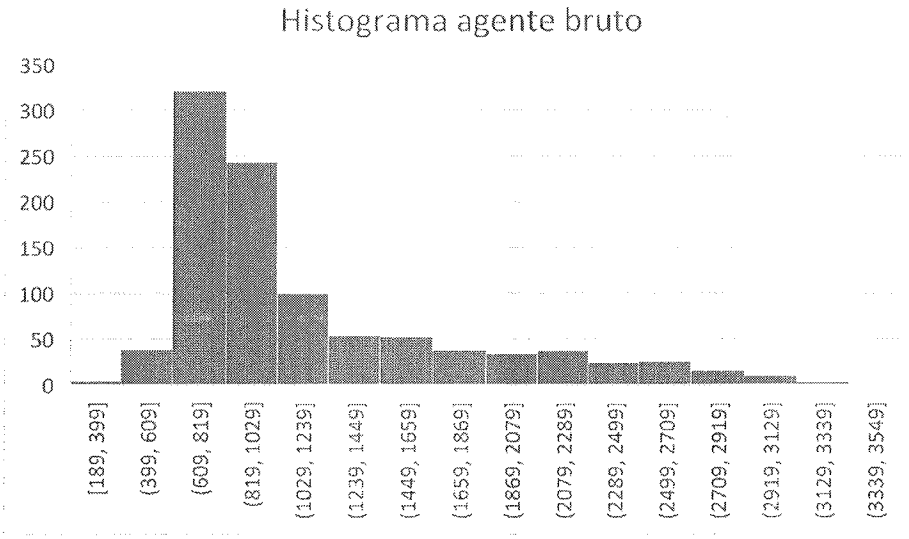


Figura 27. Histograma de los puntajes obtenidos durante 1,000 partidas por el agente bruto, con una media de 1167.27 y una desviación estándar de 599.57

Podemos observar que el agente derivado acumula la mayoría de sus puntajes entre 566 y 896 puntos, con una media de 789.5, además son pocas las veces que este agente obtuvo puntajes entre 2,000 y 3,000, lo cual se puede ver en parte reflejado en su desviación estándar de 327.43. Por otra el agente bruto acumulo la mayoría de sus puntos entre 609 y 1029, con una media de 1167.27, además este agente logró obtener más puntos en el rango de 2,000-3,0000 con una desviación estándar de 599.57.

A simple viste se puede notar que el agente bruto obtuvo un mejor desempeño a lo largo de los 1,000 juego ya que los puntajes con mayor frecuencia son mejores que los del agente derivado, esto se vuelve más evidente al comparar las medias, siendo la media del agente bruto 32% mayor. Otro punto a favor del agente bruto es que sus puntos están más distribuidos hacia la derecha (donde están los mayores puntajes), de forma que mientras más alto el puntaje menor es la frecuencia, formando una curva hacia abajo en la gráfica. En el caso de agente derivado casi todos los puntajes se acumulan en el mismo lugar, con muy pocas ocurrencias hacia la derecha del gráfico, se puede apreciar cómo se forma una campana de gauss en el gráfico.

Podemos entonces afirmar que el agente bruto es superior al agente derivado, lo cual es lo intuitivo, debido a que el agente bruto elimina los proyectiles que llegaban por los costados, mientras que el agente derivado disparaba al a izquierda sin patrón alguno. Aunque podría considerarse contra intuitivo, que la ejecución que parte desde cero haya generado un mejor agente en la generación número 200, ya que estaba una clara desventaja durante las primeras 100 generaciones. Parece ser que era únicamente cuestión de tiempo para que esta ejecución no solo alcanzará a la otra, sino que además la superara, claro que esto no puede ser confirmado únicamente con estos datos. Debido a esto se dejaron seguir ambas ejecuciones durante otras 100 generaciones para un total de 3000.

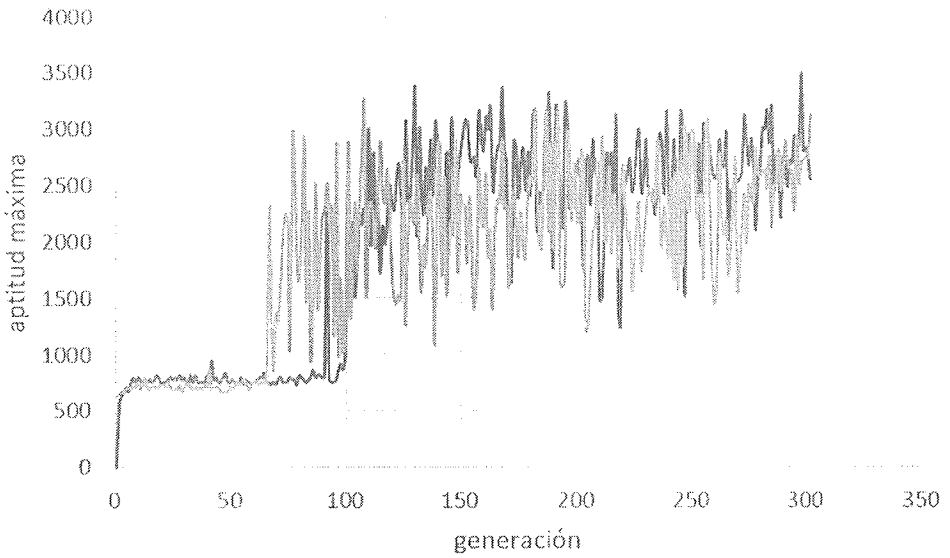


Figura 28. Gráfico de dispersión, de aptitud máxima por 300 generaciones, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack.

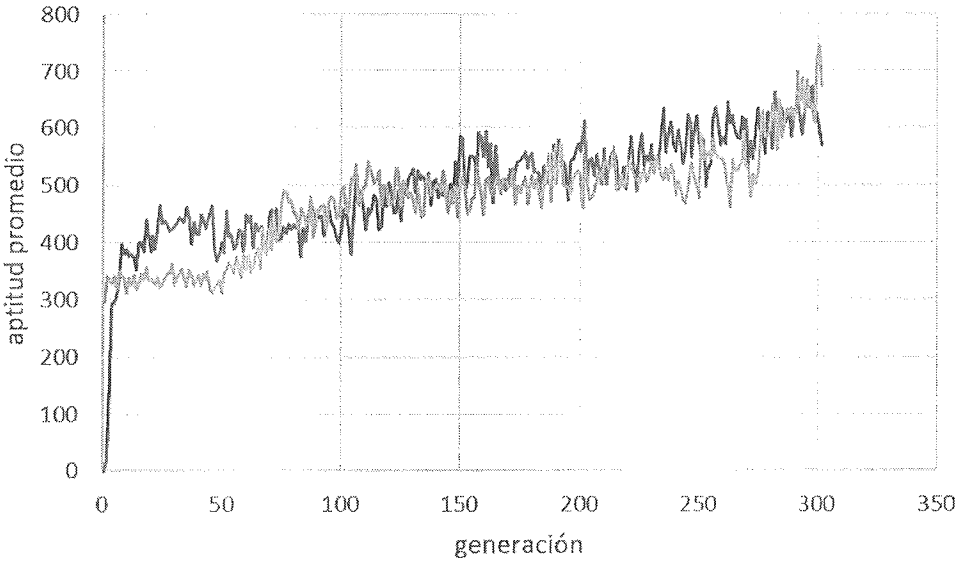


Figura 29. Gráfico de dispersión, de aptitud máxima por 200 generaciones, de dos ejecuciones distintas. La línea azul representa la ejecución de Assault empezando con genomas iniciales aleatorios, la línea naranja representa la ejecución empezando con las especies que resultaron del ambiente de Demon Atack.

Podemos ver que los puntajes promedio de ambas ejecuciones se asemejan mucho, y los puntajes máximos siguen siendo muy diversos, debido a que estos puntajes son tan variados se volvió a analizar a los agentes ganadores de la generación 300. El nuevo agente bruto parece tener el mismo comportamiento que el agente bruto que se creó con las primeras 200 generaciones, pero el nuevo agente derivado parece ser capaz de esquivar los proyectiles que vienen desde arriba, esto moviéndose un poco a la izquierda para luego regresar al extremo derecho. De nuevo para tener más información sobre el desempeño, se crearon histogramas de los puntajes obtenidos por los agentes durante 1,000 partidas.

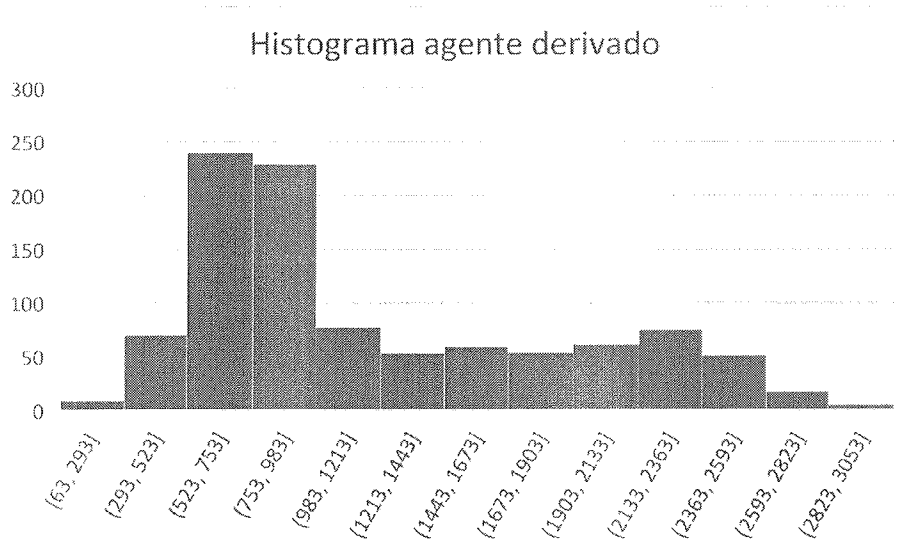


Figura 30. Histograma de los puntajes obtenidos durante 1,000 partidas por el agente derivado, con una media de 1203.25 y una desviación estándar de 661.97

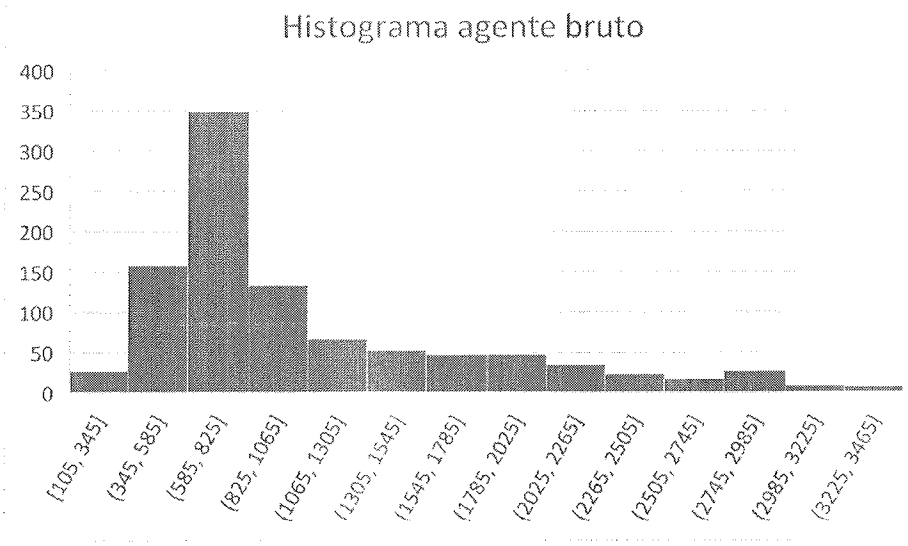


Figura 31. Histograma de los puntajes obtenidos durante 1,000 partidas por el agente bruto, con una media de 1073.81 y una desviación estándar de 692.35

Esta vez parece ser que el agente derivado es el que obtiene un mejor desempeño a lo largo de los 1,000 juegos ya que los puntajes con mayor frecuencia son mejores que los del agente bruto, el cual tiene una media de 1073.81 y el agente derivado una media de 1203.25 siendo 11% más grande. El comportamiento del agente bruto de esta generación parece ser muy similar al de la generación 200, ambos tienen menos ocurrencia a medida que los puntajes aumentan, y tanto la media como la desviación son similares, aunque ambos indicadores son mayores en el nuevo agente. En el caso del agente derivado hay un cambio bastante notorio, esta vez los puntos están bastante mejor distribuidos hacia la derecha, sin ningún patrón aparente, esto se puede ver reflejado en la desviación estándar de 661.35 la cual es el doble de la desviación de del agente derivado de la generación 200. Podemos entonces confirmar que el nuevo agente derivado tiene un mejor desempeño, esto se puede atribuir al nuevo comportamiento que le permite esquivar ciertos proyectiles.

Gracias a estas nuevas 100 generaciones podemos darnos cuenta de que no podemos saber cuál de las dos ejecuciones va a ser la mejor en un determinado punto, ya que ambas han mostrado ser superior en diferentes generaciones.

VIII. Conclusiones

Al analizar los resultados de los dos ambientes base, podemos darnos cuenta de que las ejecuciones que parten de Demon Atack tienen ventaja contra las que parte de Space Invaders, con la excepción de las aptitudes promedio en el juego de Beam Rider (Las diferencias de los promedios de las aptitudes máximas son 134 y 396 puntos en favor de Demon Atack). Esto podría deberse a que Demon Atack es un juego más similar a los dos juegos derivados, de inicio sabíamos que el juego de Space Invaders, no solo era más lento en sus movimientos, sino que además tiene una mecánica que ninguno de los otros juegos posee, que es la de cubrirse.

En términos generales parece ser que la ventaja que tienen las ejecuciones que parten de un juego base es el número de generaciones que les toma alcanzar cierto puntaje, y que en ningún caso estas ejecuciones iniciaron con un puntaje máximo de 0. El mejor ejemplo de que estas ejecuciones parten con ventaja fue el caso de Assaut en el que la ejecución que parte de Demon Atack, alcanza puntajes mucho más altos en las primeras 100 generaciones, pero luego de estas 100 generaciones alcanza puntajes iguales o mejores, por lo demostrado en las pruebas a partir de este punto cualquiera de los dos podría generar el mejor agente. Por ende, no se puede decir que un método es mejor al otro, únicamente se puede concluir que las ejecuciones que parten de un juego base pueden llegar a dar una ventaja en alcanzar un determinado puntaje en un menor número de generaciones.

Es difícil generar conclusiones, únicamente con los resultados obtenidos con estos experimentos, principalmente por la naturaleza aleatoria de los videojuegos. Esto es un factor importante ya los agentes que generaron los mejores puntajes en una generación pueden haberlo hecho únicamente por suerte, o puede darse el caso que un agente que pudiera haber generado un gran puntaje no se reproduzca por mala suerte. Esto también dificulta el análisis de los agentes ganadores ya que el agente ganador de una generación puede no ser realmente el mejor de la población. Esta aleatoriedad se puede apreciar principalmente en los gráficos de dispersión de aleatoriedad máxima, ya que a pesar de que se toma el mejor valor de entre los 200 cromosomas, se puede apreciar lo mucho que varían los resultados. Debido a esta aleatoriedad resultaron aún más útiles la especiación que permite que agentes que hayan podido tener mala suerte generen dependencia, y el estancamiento que permite que una especie que no esta generando buenos resultados sobreviva n generaciones antes de ser eliminada.

A pesar de todo podemos considerar el experimento como exitoso ya que mediante la metodología propuesta se lograron alcanzar resultados igual de buenos, que al empezar con genomas nuevos. Incluso teniendo ventaja en alcanzar en alcanzar determinados puntajes de forma más rápida, cumpliendo así todos los objetivos planteados para este proyecto. Esto abre la puerta a realizar más experimentos utilizando esta metodología en otro tipo de problemas ajenos a los videojuegos.

Pero, ¿es una mejor alternativa empezar NEAT con genomas que provengan de otro ambiente, si se desea resolver un videojuego? Los resultados parecen demostrar que la respuesta es: Sí. Siempre y cuando el objetivo sea reducir el número de generaciones y se cumplan las siguientes condiciones:

1) Los cromosomas o agentes que provengan de otro ambiente ya se deben de poseer, ya que sería mucho más eficiente resolver el juego desde cero, a tener que resolver otro juego previamente.
2) Ambos videojuegos deben ser lo suficientemente similares de acuerdo a: la cantidad de acciones posibles, el desenlace de las acciones, los valores de entrada a la red neuronal y la cantidad de los valores de entrada a la red deben ser los mismos.

IX. Recomendaciones

Hacer pruebas con al menos un juego base que sea lo más distintos posible, a los juegos base utilizados en el experimento, esto con el fin de saber que tan necesario que los juegos base sean similares a los juegos derivados, y si esto es o no un beneficio para conseguir una menor cantidad de generaciones, para resolver los juegos derivados. Pare el caso de un experimento como este, se podrían utilizar otros juegos de Atari que no fuesen de naves que disparan a enemigos (genero *"Shoot 'em up")* como PackMan Pong o Breakout.

Realizar pruebas con mayor cantidad de generaciones, de ser posible en mayores ordenes de magnitud, de manera que se obtenga la mayor aptitud posible en todas las pruebas. De esta forma también se podría verificar si existe alguna mejora o deterioro en las aptitudes de los agentes derivados, al estar las redes neuronales mayormente acopladas a los juegos base.

Repetir las pruebas con un diferente método de toma de datos a la que se utilizó en este experimento (memoria RAM del sistema), de forma que se pueda tener una mejor intuición de que es lo que están interpretando los agentes. Una posible opción es utilizar los colores por píxel, aunque esto tomaría más procesamiento, también podría brindar más información.

Como contramedida de la aleatoriedad de las aptitudes generadas por los agentes, una buena opción podría ser: calcular la aptitud segun al puntaje promedio obtenido por el agente en una serie de partidas. La desventaja de este método es que, hubiera aumentado mucho el tiempo de ejecución, multiplicando el tiempo que lleva evaluar la aptitud de cada agente, por el número de partidas.

X. Referencias bibliográficas

Auer, Peter; Harald Burgsteiner; Wolfgang Maass (2008). *A learning rule for very simple universal approximators consisting of a single layer of perceptrons*. Editorial Board.

Barricelli, Nils Aall. (1963). *Numerical testing of evolution theories. Part II. Preliminary tests of performance, symbiogenesis and terrestrial life*. Acta Biotheoretica.

Bengio, Y.; Courville, A.; Vincent, P. (2013). *Representation Learning: A Review and New Perspectives*. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Frontiers. (2018). *Artificial Neural Networks as Models of Neural Information Processing*. Frontiers Research Topic.

Kassahun, Yohannes; Sommer, Gerald. (2005). *Efficient Reinforcement Learning Through Evolutionary Acquisition of Neural Topologies*. Christian Albrechts University.

Kearney, William. (2016). *Using Genetic Algorithms to Evolve Artificial Neural Networks*; Colby College.

Mitchell, Melanie. (1996). *An Introduction to Genetic Algorithms. Cambridge*, MA: MIT Press.

Risi, Sebastian; Togelius, Julian. (2017). *Neuroevolution in Games: State of the Art and Open Challenges*. IEEE Transactions on Computational Intelligence and AI in Games.

Russell, Stuart J.; Norvig, Peter. (2010). *Artificial Intelligence A Modern Approach*. Prentice Hall.

Stanley, Kenneth O. (2017, 13 de julio*). Neuroevolution: A different kind of deep learning*. O'Reilly Media. Rcuperado de https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning .

Stanley, Kenneth; Miikkulainen, Risto. (2002) *Evolving Neural Networks through Augmenting Topologies*. The MIT Press Journals.

Stuart J. Russell; Peter Norvig. (2010) Artificial Intelligence: A Modern Approach, Third Edition, Prentice Hall.

Whiteson, Daniel (2006) Stochastic Optimization for Collision Selection in High Energy Physics. Unverisity of Texas.

Whitley, Darrell. (1994). *A genetic algorithm tutorial*. Statistics and Computing.

XI. Anexos

El código desarrollado para este proyecto de graduación, así como también los reportes generados se pueden encontrar el siguiente repositorio *https://github.com/Jorest/Ramificacion-Neuro-evolutiva*