# Un Entorno de Aprendizaje Neuroevolutivo: Screaming Racers

M. Sempere, F. Gallego, F. Llorens, M. Pujol, and R. Rizo

Department of Computer Science and Artificial Intelligence
University of Alicante
{mireia,fgallego,faraon,mar,rizo}@dccia.ua.es

**Key words:** Neuroevolución, Videojuegos, Inteligencia Articial de Nivel Humano.

**Abstract.**

En los últimos años se ha producido una gran evolución de los videojuegos. Esta evolución se une al hecho de que el mercado de los videojuegos ha experimentado un gran crecimiento. Por estos motivos, la inteligencia artificial (IA) es tenida en cuenta tanto por los propios desarrolladores de juegos como por los investigadores en el campo de la inteligencia artificial. Estos últimos tienen que tener en seria consideración los juegos de ordenador como un entorno interesante para desarrollar soluciones de IA. En este artículo se presenta un juego de simulación de carreras de coches como un entorno de experimentación donde desarrollar algoritmos de inteligencia computacional de nivel humano.

## 1 Introducción

En los últimos años se ha producido una gran evolución de los videojuegos. Los juegos actuales incluyen mejores representaciones gráficas, mundos virtuales cada vez más realistas, mayor interactividad, oponentes más inteligentes, etc.

Esta evolución se une al hecho de que el mercado de los videojuegos ha experimentado un gran crecimiento, superando en algunos casos incluso, los ingresos de la industria cinematográfica [1]. Al ser este mercado cada vez más competitivo, juega un papel muy importante el uso de la inteligencia artificial (IA) en el éxito o fracaso de un videojuego, convirtiéndose en una necesidad el uso de técnicas de IA para poder sobrevivir [2].

Son los propios desarrolladores los que reconocen el gran impacto que tiene la IA en el desarrollo de muchos juegos, debido a que los juegos actuales incluyen enemigos tácticos, compañeros, comentaristas o simplemente personajes de soporte que deben interactuar con el jugador y comportarse como humanos [3]. Por lo que hay que tener en cuenta que uno de los objetivos de la IA es estudiar y desarrollar sistemas inteligentes que emulen el comportamiento y las capacidades de las personas [4] (respuesta en tiempo real, razonamiento, creatividad, aprendizaje, comunicación, etc.)..

Por otro lado, los investigadores en el campo de la inteligencia artificial tienen que tener en seria consideración los juegos de ordenador como un entorno interesante para desarrollar soluciones de IA, ya que son muchas áreas de éste campo las que se pueden aplicar a los videojuegos: arquitecturas de agentes inteligentes, representación del conocimiento, navegación espacial, aprendizaje, planificación, colaboración,... [5] [4] [6].

En este artículo se presenta un juego de simulación de carreras de coches como un entorno de experimentación donde desarrollar algoritmos de inteligencia computacional de nivel humano. Este entorno permite realizar experimentos de aprendizaje no supervisado con pilotos de carreras virtuales. En el artículo se introducen algunas técnicas de IA utilizadas actualmente en el desarrollo de videojuegos y el algoritmo de computación neuroevolutiva NEAT utilizado en nuestro entorno. Finalmente se muestran resultados del trabajo realizado.

## 2 Técnicas de IA en juegos

En la literatura existen varias técnicas que tratan el uso de la IA en el desarrollo de videojuegos. Estas técnicas se pueden dividir en dos grandes grupos, las técnicas basadas en reglas y las que tienen en cuenta el aprendizaje y la adaptabilidad al comportamiento del jugador [2]. Las del primer grupo incluyen máquinas de estados finitos (FSM) y lógica difusa, mientras que en el segundo grupo se encontrarían redes neuronales y algoritmos evolutivos entre otras.

Centrándonos en el uso de técnicas de IA para el desarrollo de pilotos virtuales, es usual encontrar diferentes aproximaciones con el primer tipo de técnicas. En este caso, frecuentemente, se dispone de un circuito dividido en diferentes sectores (como las piezas de un Scalextric), que se estructura en memoria como una doble lista [7]. De este modo, una manera sencilla de construir un vehículo controlado mediante IA es definir una serie de líneas que atraviesan cada sector y que se utilizarán para guiar el coche. A partir de ellas, es posible señalar el camino óptimo de cada circuito, de manera que la técnica de IA desarrollada únicamente debe seguir esta línea.

A partir de esta estructura, los vehículos guiados por IA mantienen su localización en el circuito y utilizan una máquina de estados finitos para tomar la mejor decisión [8].

Un vehículo podría encontrarse en el estado STATE_OFF_TRACK. Este estado estimulará al piloto a volver a la trayectoria marcada, generando una línea desde la posición actual a uno de los puntos de la línea de conducción óptima del sector donde se encuentre el vehículo.

Estas técnicas se utilizan junto con otras que tienen en cuenta distintos parámetros, como la velocidad óptima, la dirección o el frenado de cada sector para cada tipo de vehículo de manera que no es necesaria una regulación manual [9].

Todas estas técnicas se basan en diferentes premisas, como información previa sobre líneas de conducción y otros trucos que producen la ilusión de inteligencia, pero que quedan muy lejos de la inteligencia real humana.

De modo opuesto a estas técnicas, existen aquellas que permiten el aprendizaje y evolución de los individuos. Este tipo de técnicas conllevan una serie de problemas. Por un lado, la depuración del código se convierte en una tarea ardua y difícil para el programador cuando se utilizan técnicas de redes neuronales o algoritmos genéticos. Por otro lado, la evolución de los personajes en tiempo real cuando se está jugando no está directamente controlada por los desarrolladores, y los personajes podrían aprender, evolucionar y por tanto comportarse de una manera que no es conveniente para el avance del juego. Por estas razones estas técnicas han sido utilizadas de manera offline, es decir, se utilizan durante la implementación del juego para el aprendizaje de los distintos personajes, pero éstos se incluyen en el juego ya evolucionados.

En nuestro caso adaptamos una perspectiva diferente con la aplicación de redes neuronales, algoritmos genéticos y computación neuroevolutiva. Estas ideas, aplicadas a nuestro sistema de inteligencia artificial constituyen un buen punto de partida sobre el que poder realizar experimentos con distintos algoritmos de inteligencia computacional de nivel humano. El sistema de inteligencia artificial utilizado combina sistemas multiagentes con algoritmos de neuroevolución, concretamente el algoritmo NEAT (NeuroEvolution of Augmenting Topologies).

### 2.1 Neuroevolución

La neuroevolución consiste en utilizar algoritmos genéticos para hacer evolucionar una población de redes neuronales [10]. El algoritmo genético tomará una población de redes neuronales y generará una nueva mejor o igual que la anterior, a partir de la recombinación, mediante el uso de operadores genéticos de cruce y mutación, de las redes neuronales de la población anterior.

Al utilizar un algoritmo genético para evolucionar una población, se está realizando una búsqueda en el espacio de los comportamientos, con la intención de encontrar un comportamiento óptimo.

Visto de forma práctica, el proceso de evolución considera cada red neuronal como el cerebro de un agente, en nuestro caso, un piloto virtual, dando lugar a una población de agentes que progresivamente van adaptando sus comportamientos al entorno. De esta forma, la neuroevolución representa un proceso de aprendizaje evolutivo.

En nuestro caso, existen dos aspectos clave en la creación de un algoritmo de neuroevolución [11]. El primero es el esquema de codificación utilizado para describir una red neuronal en términos genéticos. El segundo es el uso y la definición de operadores de selección, cruce y mutación adecuados.

### 2.2 Codificación de redes neuronales en genomas

Como se ha comentado anteriormente, uno de los principales problemas de la neuroevolución consiste en encontrar una representación adecuada de una red neuronal para que pueda ser utilizada por un algoritmo genético.

En el campo de la neuroevolución se han realizado multitud de aproximaciones para obtener la forma óptima de codificar y evolucionar genomas que representan redes neuronales.

De esta manera, es posible codificar una red neuronal como una secuencia de bits (GENITOR [12]), como una matriz binaria (Binary matrix encoding [13]) o generando una gramática (Grammar Based Encoding [14]), por ejemplo. Sin embargo, estas representaciones tienen un gran problema cuando dos representaciones genéticas distintas tienen exactamente el mismo comportamiento, de modo que su combinación mediante un operador de cruce habitual puede llevar a un empeoramiento de la especie. Este problema se conoce como el problema de las representaciones enfrentadas (Competing Conventions Problem, CCP) [15].

Para solventar este problema se realizaron diferentes intentos, pero ninguno de ellos dio buenos resultados hasta la aparición de Neuroevolution of Augmenting Topologies (NEAT).

### 2.3 NEAT

NEAT es un algoritmo capaz de evolucionar los pesos y las topologías de una población de redes neuronales, utilizando mecanismos de refuerzo para realizar un aprendizaje no supervisado [16] [17].

Utiliza dos tipos de genes: genes de enlace y genes neurona. Un gen enlace contiene información sobre las dos neuronas que conecta, el peso del enlace, si se encuentra o no habilitado y un *identificador de innovación*. El número de innovación es la principal característica del método NEAT para evitar el problema de las representaciones enfrentadas (CCP). Este identificador relaciona cada elemento estructural de un individuo con una base de datos de innovaciones estructurales global a toda la población de genomas. De esta manera, al utilizar el *identificador de innovación* cuando dos genomas tratan de combinar sus elementos estructurales, es posible identificar cuales son los genes comunes (mismo identificador de innovación) y no comunes.

Por otra parte, un gen neurona contiene la información relevante de una neurona, un identificador, una función asignada (entrada, salida, oculta o sesgo) y un BIT para indicar si es recurrente. De esta forma, un genoma en NEAT estará compuesto por dos vectores que contendrán la información de todos los genes enlace y neurona respectivamente (ver figura 1).

NEAT empieza con la mínima topología posible y añade neuronas y enlaces, por medio de dos de sus cuatro operadores de mutación. Cada vez que se añade una nueva estructura al genoma, se conoce como *innovación*. Estas innovaciones se almacenan en una base de datos global. Cuando una nueva neurona o enlace se añade al genoma, se referencia la base de datos y se le asigna un identificador de innovación, que nos dice si ha sido previamente descubierto por otro genoma o si es completamente nuevo. De esta manera, todos los genes de la población están identificados por este identificador de innovación.

Para realizar el cruce (ver figura 2), el operador recorre los genes enlace de los dos genomas progenitores, comparando sus números de innovación para determinar sobre cada gen si éste debe ser escogido para formar parte del hijo.

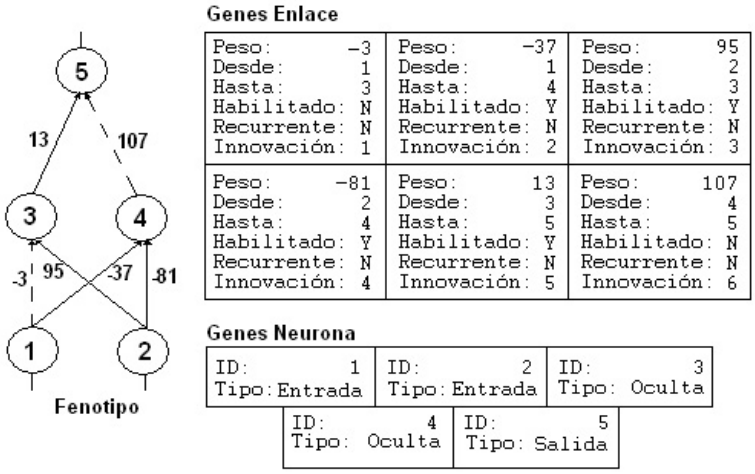


**Fig. 1.** Ejemplo de genoma codificado con NEAT.

Los genes son escogidos según dos premisas. Por una parte, cuando un gen con un determinado identificador de innovación se encuentra en ambos padres, el hijo hereda de forma aleatoria uno de ellos. Por otra parte, cuando un gen se encuentra solo en uno de los progenitores, si se trata del que tiene un mayor valor de idoneidad, el hijo lo hereda, sino, es descartado.

Por otra parte, NEAT proporciona una especificación para proteger las innovaciones con un comportamiento peor adaptado. Esta especificación trata de evitar que estos individuos desaparezcan prematuramente de la población.

## 3 Screaming racers

Screaming Racers es un videojuego de simulación de carreras donde los coches están controlados por agentes inteligentes que comienzan sin tener ningún conocimiento del entorno ni del vehículo. Estos agentes tratan de aprender y mejorar sus habilidades como pilotos, teniendo en cuenta que el único conocimiento que van a tener es el que ellos mismos sean capaces de aprender. El objetivo del juego es crear el mejor grupo artificial de pilotos para poder competir en cualquier circuito y contra cualquier otro equipo.

Este videojuego nos proporciona un entorno flexible donde experimentar con distintos algoritmos de inteligencia artificial de nivel humano.

### 3.1 Arquitectura interna

Este videojuego se ha diseñado como un sistema multiagente para beneficiarse de las ventajas de un modelo distribuido. La aplicación está dividida en dos partes: servidor y clientes. El servidor cuenta con un simulador que da vida

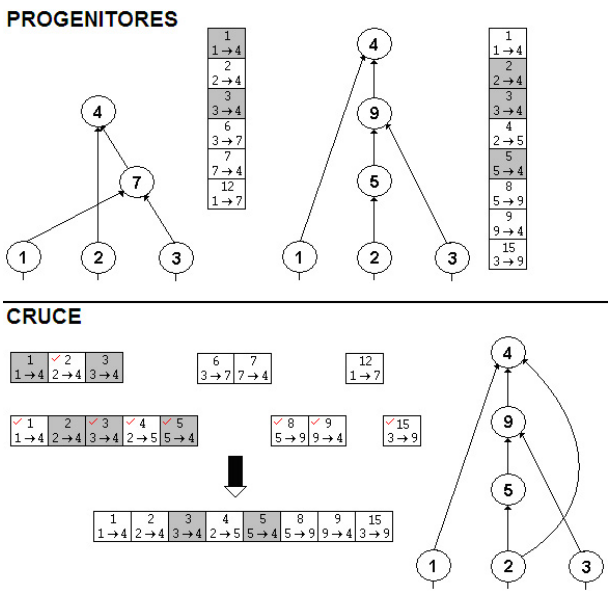


**Fig. 2.** Operación de cruce de NEAT. Los genes deshabilitados aparecen en gris

al entorno donde los agentes piloto se desarrollan, interactúan y aprenden. Los clientes serán utilizados por los jugadores para conectarse al servidor.

El diseño multiagente proporciona una serie de ventajas en lo que se refiere a los agentes piloto. Éstos pueden ser movidos desde los clientes al servidor y viceversa cuando sea necesario. Por este motivo se minimizan los problemas de la latencia en las comunicaciones. Por otro lado, los agentes piloto pueden sentir el entorno mediante una serie de sensores predefinidos y además se pueden comunicar entre ellos, permitiendo a los equipos compartir el conocimiento para beneficiarse de un comportamiento colaborativo.

En esta arquitectura cliente/servidor (ver figura 3), los principales componentes se encuentran dentro del servidor. Son los responsables de mantener el entorno virtual y el sistema multiagente, además de las comunicaciones entre los clientes. A continuación presentamos una breve descripción de cada uno de estos componentes:

- *Motor Físico*: su objetivo es el de aplicar las distintas leyes que rigen la física a todos los objetos que se encuentran en el entorno.
- *Agente Simulador*: es el responsable de gestionar y desarrollar cada sesión de entrenamiento o competición. Debe comunicarse con todos los agentes piloto para que estos puedan coordinar su funcionamiento. Funciona también como agente pizarra para comunicaciones entre equipos.
- *Motor Gráfico*: muestra en pantalla la apariencia de entorno y la interfaz de usuario.
- *Agente Controlador*: es el encargado de coordinar al resto de componentes.

– *Agente Comunicador*: implementa las tareas de comunicación entre servidor y clientes y viceversa.

– *Agentes Piloto*: cada uno de estos agentes implementa el cerebro de uno de los pilotos del simulador. Reciben información de sus sensores por medio del agente simulador y toman decisiones para actuar en consecuencia.

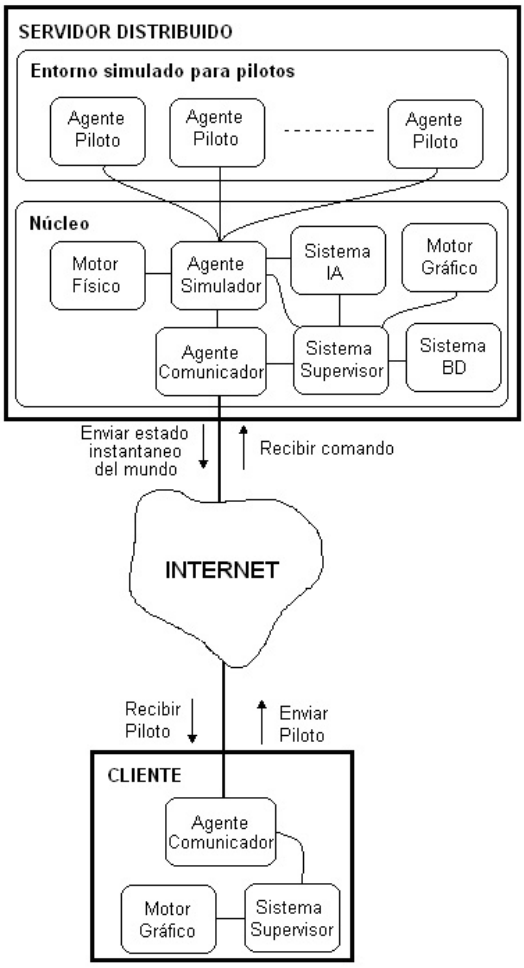


**Fig. 3.** Arquitectura interna de Screaming Racers.

## 4 Validación experimental

El proceso de experimentación se ha dividido en dos fases. Por un lado, realizar una comparación de diferentes algoritmos de neuroevolución para observar cual de ellos ofrecía mejores resultados. Por otro lado, estudiar el algoritmo seleccionado para comprobar su comportamiento.

Para la comparación de los diferentes algoritmos de neuroevolución se han tenido en cuenta los algoritmos: GENITOR, Binary Matrix Encoding (BME), Schiffman y dos variantes del algoritmo NEAT (con y sin especies).

La prueba llevada a cabo ha consistido en estudiar la evolución durante 500 generaciones utilizando una población de 100 individuos y una configuración estándar de parámetros. Los resultados mostrados en la figura 4 corresponden a la media obtenida por cada algoritmo tras realizar 2 pruebas en 5 circuitos diferentes. El gráfico muestra la evolución del mejor individuo de la población en cada algoritmo.

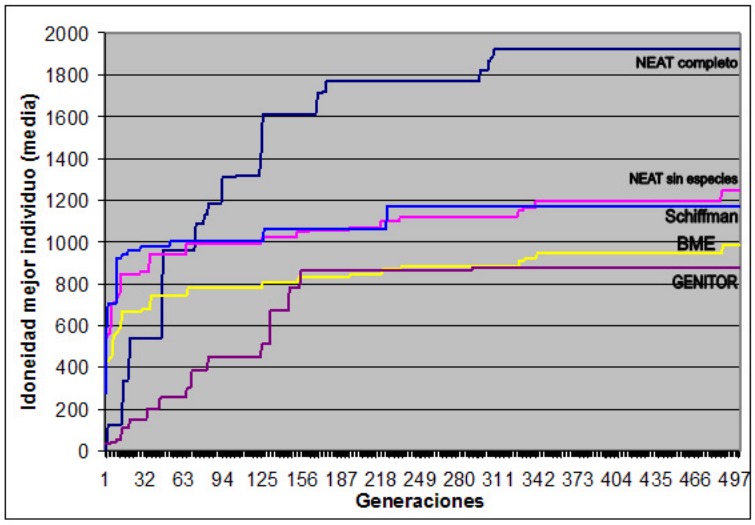


**Fig. 4.** Comparativa de aprendizaje de diferentes algoritmos de neuroevolución durante 500 generaciones.

Al observar estos resultados se deduce que el algoritmo más prometedor de los considerados para la experimentación es NEAT en su versión con especies. Por lo tanto, las siguientes pruebas han sido realizadas con este algoritmo.

Los siguientes experimentos exponen algunas habilidades adquiridas por los pilotos. En la figura 5 se muestran dos trazadas de un piloto. La imagen 5a muestra un piloto en edad temprana de evolución donde aún no ha desarrollado sus habilidades. En la imagen 5b se muestra un piloto entrenado tomando una curva de un circuito. Esta imagen muestra la validez de los resultados obtenidos.

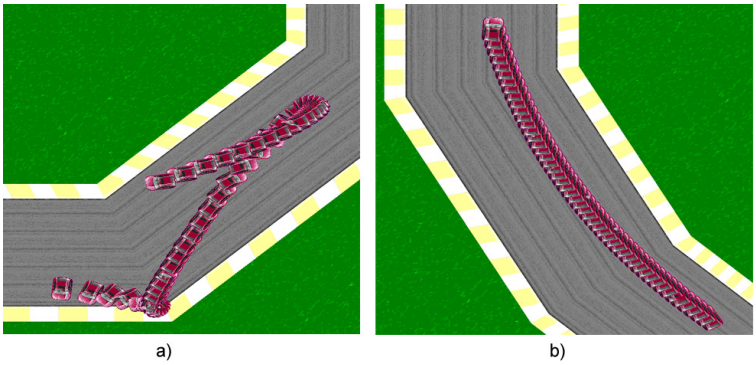


**Fig. 5.** Trazadas de un vehículo. a) Piloto sin evolucionar. b) Piloto entrenado.

## 5 Conclusiones

En este artículo se ha presentado un videojuego de simulación de carreras de coches que permite desarrollar algoritmos de inteligencia computacional de nivel humano y experimentar el aprendizaje no supervisado con pilotos de carreras virtuales. Este videojuego es una forma de combinar el algoritmo evolutivo NEAT con un sistema Multiagente.

Se han introducido algunas técnicas de inteligencia artificial utilizadas actualmente en el desarrollo de videojuegos, centrándose en la computación neuroevolutiva y concretamente, en el algoritmo neuroevolutivo NEAT.

Se ha realizado una comparación de distintos algoritmos de neuroevolución, mostrándose que NEAT supone una buena primera aproximación al problema.

Los futuros trabajos irán encaminados a combinar resultados de aprendizaje neuroevolutivo no supervisado con esquemas de aprendizaje supervisado y fases de desarrollo de los pilotos colaborativas y competitivas.

## References

1. Laird, J.E.: Using a computer game to develop advanced AI. IEEE Computer **34** (2001)
2. Johnson, D., Wiles, J.: Computer games with intelligence. Australian Journal of Intelligent Information Processing Systems **7** (2001) 61–68
3. Le Hy, R., Arrigoni, A., Bessière, P., Lebeltel, O.: Teaching bayesian behaviours to video game characters. Robotics and Autonomous Systems **47** (2004) 177–185
4. Laird, J.E., van Lent, M.: Human-level ai's killer application: Interactive computer games. Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (2000) 1171–1178
5. van Lent, M., Laird, J.E., Buckman, J., Hartford, J., Houchard, S., Steinkraus, K., Tedrake, R.: Intelligent agents in computer games. Proceedings of the National Conference on Artificial Intelligence (1999) 929–930

6. Kaminka, G.A., Veloso, M.M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S.: Gamebots: a flexible test bed for multiagent team research. Communications of the ACM **45** (2002) 43–45
7. Biasillo, G.: Racing and Racetrack for the AI. In: AI Game Programming Wisdom. (2002) 439–443
8. Biasillo, G.: Racing AI Logic. In: AI Game Programming Wisdom. (2002) 444–454
9. Biasillo, G.: Training an AI to Race. In: AI Game Programming Wisdom. (2002) 455–459
10. Yao, X.: Evolving artificial neural networks. Proceeedings if the IEEE **87** (1999) 1423–1447
11. Curran, D., O'Riordan, C.: Applying evolutionary computation to designing neural networks: A study of the state of the art. Technical report, National University of Ireland, Galway (2002)
12. Whitley, D.: The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. Proceedings of the Third International Conference on Genetic Algorithms and their Applications (1989) 116–121
13. Miller, G.F., Todd, P.M., Hedge, S.U.: Designing neural networks using genetic algorithms. Proceedings of the Third International Conference on Genetic Algorithms and their Applications (1989) 379–384
14. Hussain, T.S., Browse, R.A.: Network generating attribute grammar encoding. IEEE International Joint Conference on Neural Networks **1** (1998) 431–436
15. Hancock, P.J.B.: Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92) (1992) 108–122
16. Stanley, K., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10(2)** (2002) 99–127
17. Stanley, K., Miikkulainen, R.: Efficient evolution of neural network topologies. Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02) (2002)