UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

Trabajo Fin de Grado

CURSO 2016/17

HERRAMIENTAS DE EVOLUCIÓN COLECTIVA PARA EL TRATAMIENTO DE PROBLEMAS DISTRIBUIDOS

Grado en Ingeniería en Tecnologías Industriales

ALUMNO

Daniel López Enseñat

TUTOR

Abraham Prieto García

FECHA

JULIO 2017

RESUMEN

Herramientas de evolución colectiva para el tratamiento de problemas distribuidos

En este trabajo se lleva a cabo el desarrollo y aplicación de un algoritmo distribuido Embodied Evolution (dEE) en un entorno real. Para ello, se hace un estudio de las técnicas de aprendizaje más utilizadas en el campo del *machine learning* y se aplican en diferentes problemas. En primer lugar, se utiliza un algoritmo genético simple sin sistema de decisión en un problema discreto. Posteriormente se resuelve un problema continuo mediante un algoritmo neuroevolutivo (NEAT). Por último, se desarrolla el algoritmo dEE para un problema continuo multiagente y se comprueba que funcione adecuadamente antes de que ser utilizado en el problema real

RESUMO

Ferramentas de evolución colectiva para o tratamento de problemas distribuidos

Neste traballo lévase a cabo o desenvolvemento e a aplicación dun algoritmo distribuído Embodied Evolution (dEE) nun entorno real. Para iso, faise un estudo das técnicas de aprendizaxe máis utilizadas no campo do *machine learning* e aplícanse a diferentes problemas. En primeiro lugar, utilízase un algoritmo xenético simple sen sistema de decisión nun problema discreto. De seguido, resólvese un problema continuo coa axuda dun algoritmo neuroevolutivo (NEAT). Por último, o algoritmo dEE é desenvolvido para un problema continuo multiaxente e compróbase que funcione de forma axeitada antes de ser usado no problema real.

ABSTRACT

Tools in collective evolution for distributed problems

Within the following Bachelor thesis, the development and implementation of a distributed Embodied Evolution (dEE) algorithm are carried out in a real environment. For this purpose, a study is conducted on the most used learning techniques in the field of machine learning and it is applied in different problems. First of all, it is adopted a simple genetic algorithm without a decision system in a discrete problem. Then, a continuous problem is solved by a neuroevolutive algorithm (NEAT). Finally, a dEE algorithm is developed for a multiagent continuous problem and it is checked that it works properly before its use in a real problem.

UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

TRABAJO FIN DE GRADO

CURSO 2016/17

HERRAMIENTAS DE EVOLUCIÓN COLECTIVA PARA EL TRATAMIENTO DE PROBLEMAS DISTRIBUIDOS

Grado en Ingeniería en Tecnologías Industriales

Documento 1

MEMORIA

Índice

Índice de Figuras

Índice de Tablas

1 INTRODUCCIÓN

El presente Trabajo Fin de Grado está centrado en el campo de aprendizaje máquina o *machine learning,* que consiste en un proceso de aprendizaje dentro de la Inteligencia Artificial (IA), dirigido a conseguir que una máquina sea completamente autónoma y puede tomar sus propias decisiones sin la intervención de una persona. En los últimos años este campo está adquiriendo una gran importancia debido al gran volumen y variabilidad de los datos que se maneja, haciendo imposible que una persona pueda sacar conclusiones válidas. Para ello, es necesario entrenar un aproximador o sistema de decisión, entre los que destacan las redes neuronales.

De manera incremental se han ido tratando problemas de mayor complejidad con diferentes técnicas del *machine learning* (se han utilizado para todos un aprendizaje no supervisado). En primer lugar se ha desarrollado un algoritmo genético simple para la solución de un problema discreto y estático. A continuación se resuelve uno continuo y dinámico usando redes neuronales de la librería NEAT y un algoritmo neuroevolutivo de la misma. NEAT consiste en un algoritmo capaz de ir aumentando la complejidad de la estructura de la red según las necesidades del problema a resolver. En tercer lugar, se trata un problema multiagente donde ya no evaluamos a un solo individuo, sino a un conjunto de ellos. Para resolver este problema (*shepherding)* se utiliza un algoritmo distribuido Embodied Evolution (dEE) desarrollado por el GII y por Pedro Trueba en su tesis [1].

En último lugar, se implantará en algoritmo dEE utilizado anteriormente para un problema multiagente en un entorno real (evacuación de una habitación con zonas de peligro potencial).

2 OBJETIVOS

El objetivo principal del trabajo es la aplicación de un algoritmo distribuido *Embodied Evolution* en un problema de ingeniería. No obstante, para alcanzar este objetivo se ha precisado de unas etapas intermedias para ir conociendo más en profundidad el campo de la inteligencia artificial. Dichas etapas o subjetivos son los siguientes:

- Familiarizarse con los algoritmos evolutivos resolviendo un problema discreto y estático mediante un algoritmo genético simple (problema de las 8 reinas)

- Aplicar redes neuronales junto con un algoritmo neuroevolutivo en un problema continuo de *shepherding*

- Implementación de un algotirmo distribuido Embodied Evolution en otro problema de pastoreo para corroborar su adecuado funcionamiento.

- Implementación del algoritmo EE para un problema de evacuaciones de emergencia

3 MACHINE LEARNING

Tal y como se comentó en la introducción, el trabajo que se presenta se enmarca dentro del campo de *machine learning*. Este proceso de aprendizaje es un tipo de Inteligencia Artificial (IA) dirigido al desarrollo de máquinas que puedan aprender y tomar decisiones por sí mismas, gracias a la detección de patrones dentro de un conjunto de datos, de manera que es el propio programa el que predice qué situaciones podrían darse o no.

Adquirió una gran importancia en los últimos años porque debido a la variabilidad de datos y sus crecientes volúmenes fue necesario la implementación del aprendizaje automático para el análisis de ésta magnitud de datos y su automatización para la creación de modelos. El estudio y el modelado del proceso de aprendizaje en sus múltiples manifestaciones constituye el *machine learning*.

En esta sección se describirá en que consiste el *machine learning* y diferentes aplicaciones que se han estudiado. En primer lugar no se describirá los diferentes tipos de aprendizaje que se disponen, ya que se irán explicando en los diferentes problemas donde se aplicarán cada uno de ellos.

3.1 Descripción

En este apartado se procede a describir a rasgos generales en que consiste el *machine learning*. Para ello comenzaremos con su clasificación

3.1.1 Clasificación

Dentro del *machine learning* se distinguen 3 tipos de aprendizaje según la cantidad de conocimiento aportado por el diseñador: Supervisado, no supervisado y de refuerzo.

3.1.1.1 Aprendizaje supervisado

Es el más utilizado y requiere de intervención humana, para la creación de datos que vienen etiquetados con el resultado correcto. Cuanto mayor sea la lista de datos mejor aprenderá la máquina sobre el problema a resolver Es decir, se utiliza cuándo el usuario conoce algunos resultados que la máquina ha de obtener para ciertos estímulos o entradas. Un ejemplo sería el aprendizaje de una máquina para que sea capaz de identificar los números o letras escritas por una persona.

3.1.1.2 Aprendizaje no supervisado

En los problemas de aprendizaje no supervisado el algoritmo es entrenado usando un conjunto de datos, con la diferencia de que este caso no van etiquetados con la respuesta correcta; nunca se le dice al algoritmo lo que representan los datos. La idea es que este pueda encontrar por sí solo patrones que ayuden a entender el conjunto de datos. Este método es similar al que utilizamos cuando somos pequeños para aprender a hablar. En un principio no somos capaces entendemos nada de lo que dice el resto pero a medida que vamos escuchando conversaciones, nuestro cerebro comienza a formar un modelo sobre cómo funciona el lenguaje y empezará a reconocer patrones y a esperar ciertos sonidos.

3.1.1.3 Aprendizaje por refuerzo

En estos problemas, el algoritmo aprende observando el mundo que le rodea. Su información de entrada es el *feedback* que obtiene debido a sus actos, es decir, aprende a base de ensayo-error. Un claro ejemplo se puede ver en los juegos, donde aplicas diferentes estrategias y perfeccionando aquellas que te hayan sido útiles para ganar el juego. A medida que se adquiere práctica, la estrategia va siendo cada vez mejor.

3.1.2 Estructura de decisión

En todo algoritmo de *machine learning* se dispone de una herramienta que recibe los *input* o estímulos y te devuelve una respuesta (*output*) y que es la que interesa optimizar para que devuelva la respuesta adecuada en cada caso. Estas pueden ser, entre otras, árboles de decisión y redes neuronales. Daremos más importancia a estas últimas porque son las más utilizadas en la actualidad y será necesario para entender siguientes secciones del trabajo.

3.1.2.1 Árboles de Decisión

Un árbol de decisión está formado por un conjunto de nodos de decisión (interiores) y nodos-respuesta:

- Un nodo de decisión está asociado a uno de los atributos y tiene 2 o más ramas que salen de él, cada una de ellas representando los posibles valores que puede tomar el atributo asociado. Es decir, un nodo de decisión es quién hace una "pregunta" al ejemplo analizado y según la respuesta el flujo toma una dirección u otra.

- Un nodo-respuesta está asociado a la clasificación que se quiere proporcionar, y nos devuelve la respuesta después de haber recorrido todo el árbol,

Para alcanzar un buen comportamiento del árbol de decisión se necesita un amplio abanico de ejemplos (datos).

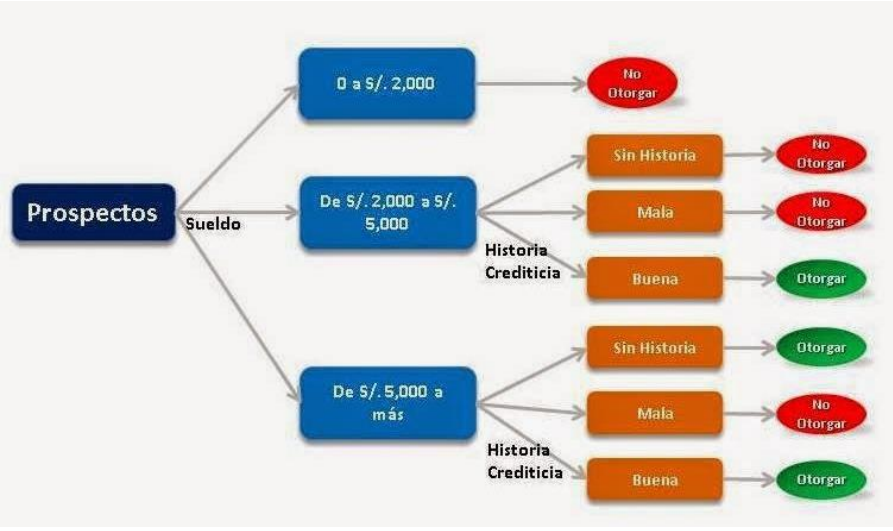


Figura 1: Ejemplo de Árbol de Decisiones. Fuente: contagrupoauthh.blogspot.com

3.1.2.2 Redes Neuronales

Una Red Neuronal Artificial (RNA) es un modelo matemático que intenta reproducir sistema nervioso, es decir, el cerebro humano. Está compuesta por un gran número de elementos interconectados (neuronas) trabajando al unísono para solucionar un problema específico. Las redes neuronales aprenden mediante ejemplos. Están configuradas para una aplicación específica, tales como reconocimientos faciales o clasificación de datos, entre otras muchas cosas, a través de un proceso de aprendizaje (*machine learning*).

Las redes neuronales, con su capacidad de sacar conclusiones válidas de complicados e imprecisos datos, pueden extraer patrones o detectar tendencias que para un humano o para otras técnicas sería imposible de detectar. Además, las redes tienen estas ventajas:

- Puede aproximar cualquier función, independientemente de su linealidad.
- Gran funcionamiento para complejos / abstractos problemas como el reconocimiento de imágenes.
- Una gran polivalencia a la hora de resolver problemas.
- Es la base de la IA.

No obstante, las redes neuronales también tienen desventajas:

- En algunas ocasiones, cuando se está ante una solución simple, funcionan mejor otros métodos como puede ser la regresión linear.
- Incrementar un poco la precisión (%) puede ser muy costoso computacionalmente hablando.

Nos vamos a centrar el tipo de red Perceptrón Multicapa porque es el tipo que se utilizará en los problemas posteriores.

3.1.2.2.1 Perceptrón Multicapa

El perceptrón multicapa (MLP sus silgas en inglés) es una de los tipos de redes más utilizados a día de hoy. Pertenecen a la clase general de estructuras llamada *feedforward neural networks*, un tipo básico de redes capaces de aproximar clases genéricas de funciones, incluyendo continuas e integrables. Las MLP son usadas en una gran variedad de modelos y problemas de optimización.

En la estructura de la ML, las neuronas están agrupadas en capas (*layer*). La primera y la última son llamadas *input* y *output* respectivamente, porque representan las entradas y las salidas de la red. El resto de capas, son llamadas las capas *hidden* u ocultas.

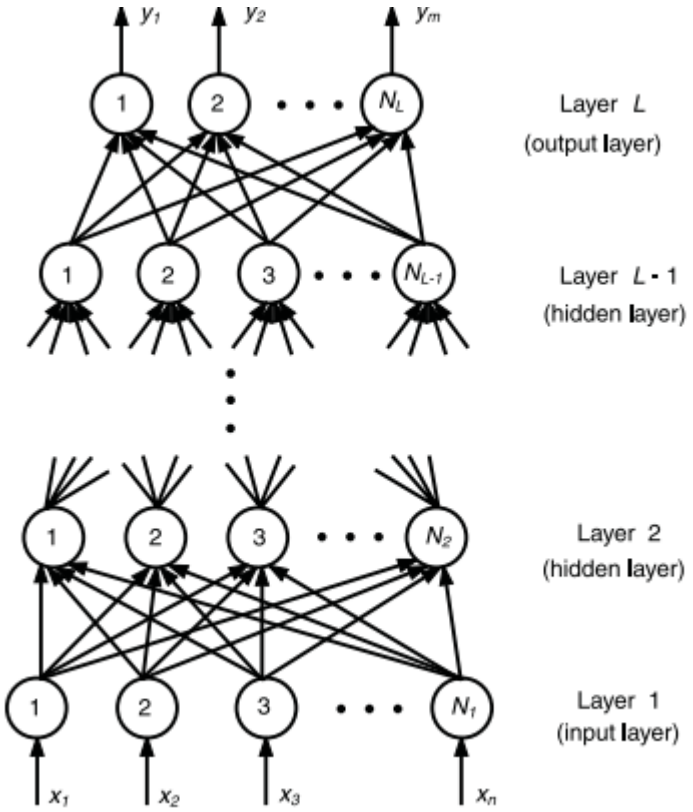


Figura 2. Estructura de una red MLP (Fuente: [2]*)*

Suponiendo que el número total de capas es n: La primera capa correspondería a la capa *input*, la enésima a la capa *output*; y desde la segunda hasta la n – 1 son capas ocultas [Figura 2].

X_i representa las entradas o estímulos de la red, las cuales serán procesadas dentro de la red. Posteriormente, representado con flechas, vemos las conexiones entre neuronas. Cada conexión, tiene asignado un peso (w), que multiplicará a la salida de la neurona anterior antes de entrar en la siguiente. Por último, y_i corresponde a la salida de la red.

En una red, cada neurona– exceptuando la de la capa de entrada – recibe y procesa estímulos procedentes de otras neuronas. La Figura X ilustra como procesa la información cada una de las neuronas en una MLP. Como se dijo anteriormente, cada entrada es multiplicada por su respectivo parámetro de peso y se hace un sumatorio con todos los productos resultantes, es decir, se realiza una "suma pesada" (γ). Esta suma pesada se le hace pasar por una función de activación (σ) produciendo el *output* de la neurona. Esta salida es a su vez la entrada de una o varias neuronas de la siguiente capa hasta llegar a la capa final.

Función de activación

La función utilizada debe estar acotada (en el eje y) y ser continua. La más utilizada para la función de activación es la función sigmoide:

$$\sigma(x) = \frac{1}{(1 + e^{-x})} \quad \textbf{(1)}$$

Como se puede observar en la figura X, la sigmoide tiene las propiedades de valer 1 para un valor de x infinito, y 0 para menos infinito.

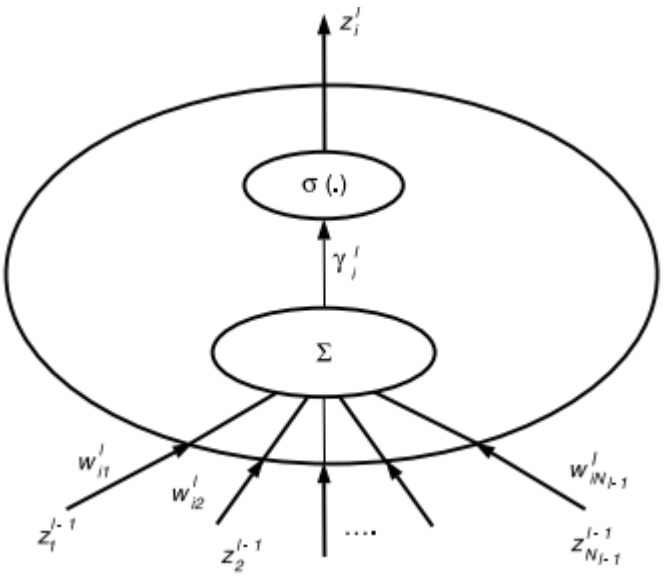


Figura 3: Esquema de una neurona de una red MLP (Fuente: [2]*)*

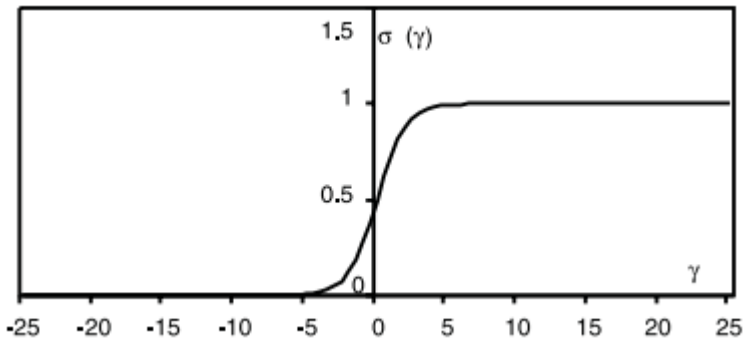


Figura 4: Esquema de una neurona de una red MLP (Fuente: [2]*)*

No obstante, la sigmoide no es la una función de activación que se utiliza, hay otras como puede ser la arcotangente:

$$\sigma(x) = \left(\frac{2}{\pi}\right) \arctan(x) \quad \textbf{(2)}$$

O también la tangente hiperbólica:

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (3)$$

Las neuronas de la capa de entrada simplemente retransmiten los estímulos del exterior a la primera capa oculta. Algunos autores no cuentan las neuronas de entrada como capa en una MLP, puesto que no se hace ninguna operación en ella. La función de activación para las neuronas de la capa de salida puede ser tanto una función logística (una sigmoide por ejemplo) como una simple función lineal ("suma pesada"). Elegir una u otra es en función del problema que se desee solucionar.

Bias

Si las todas las salidas de las anteriores neuronas (entradas de la actual) son 0, el resultado de la suma pesada es 0. Para crear las *bias* se asume una neurona anterior ficticia cuyo output es 1 ($z_0^{l-1} = 0$) y multiplicado por un parámetro de peso (w_0^{l-1}). Por tanto, la suma pesada será:

$$\gamma_i^{l-1} = \sum_{j=0}^{N_{l-1}} (w_j^{l-1}\, z_j^{l-1}) \quad (4)$$

El efecto de las que bias es que la suma pesada es igual a las bias cuando las salidas de todas las neuronas de la capa anterior sean 0.

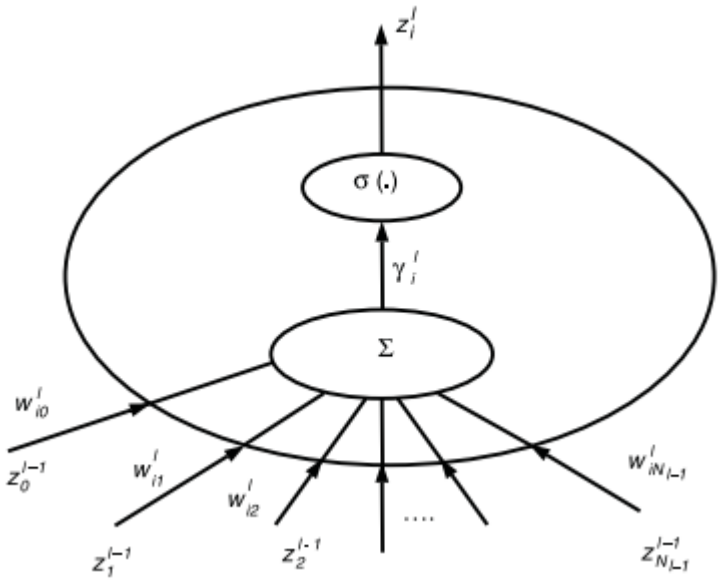


Figura 5. Esquema de una neurona de la capa L incluyendo bias (Fuente: [2]*)*

Procesado de la información

Dados los inputs $x = [x_1, x_2 \dots x_n]$ y los pesos w, las redes neuronales unidireccionales devuelven una salida $y = [y_1, y_2, \dots y_m]$ procedente de una MLP. Durante el proceso, los estímulos externos alimentas a las neuronas de la capa *input*, las las salidas de estas neuronas alimenta a las de la primera capa oculta, y así sucesivamente. Finalmente, las neuronas de la capa *L – 1* alimentan a las de las capa *output* (capa *L)*. El cálculo viene dado por:

$$z_i^1 = x_i;\ i = 1,2,3 \dots . N_1;\ n = N_1$$

Es decir, cómo es lógico el número de neuronas en la primera capa ha de ser igual al número de entradas externas que tengamos.

$$z_i^l = \sigma\left(\sum_{j=0}^{N_{l-1}} \left(wi_j^{l-1}\, z_{ij}^{l-1}\right)\right);\ i = 1,2, \dots N_1;\ l = 2,3, \dots . L \qquad (5)$$

Como ya se dijo, dentro de la neurona se realiza un sumatorio de cada entrada por su respectivo peso y se pasa por la función de activación (sigmoide generalmente).

La salida de las neuronas de la última capa será la salida de la red neuronal. Por tanto, ha de tener tantas neuronas en esta última capa como salidas se desee que tenga la red:

$$y_i = z_i^l, i = 1,2 \dots N_L, m = N_L$$

3.2 Aplicación en un problema discreto y estático

Después de haber descrito brevemente algunas características del aprendizaje automático, en este segundo apartado se intentará resolver el famoso problema de las 8 reinas, que se caracteriza por ser un problema discreto y estático. Se denomina problema discreto a aquellos que el número de soluciones son limitadas. Por lo general, estos problemas son más fáciles de resolver que uno continuo.

3.2.1 Descripción del problema

El problema de las 8 reinas consiste en colocar 8 reinas en un tablero de ajedrez (8x8) sin que ninguna sea capaz de eliminar a otra en un solo movimiento. Hay que tener en cuenta que una reina es capaz de trasladarse por el tablero el número de casillas que desee en las direcciones principales (horizontal o verticalmente) o en las direcciones diagonales.

Para la localización de las 8 reinas se precisarían 16 datos (2 para cada posición de cada reina en el tablero, es decir, una por coordenada), con lo que tendríamos completamente definida una configuración de tablero. Con el objetivo de disminuir las dimensiones de estos datos se lleva a cabo una simplificación del problema. Esta simplificación se basa en que sabemos de antemano que si 2 reinas están en la misma fila una elimina del tablero a la otra. Por tanto, podemos definir completamente la configuración del tablero con tan solo un vector de 8 números (genotipo), donde influye

el orden con el que están colocados. Estos números representarán en que columna (fenotipo) están colocadas las reinas teniendo en cuenta que la fila es el índice del número dentro del vector. Como es lógico, estos números son enteros y varían entre 1 y 8, provocando un espacio de búsqueda no continuo o discreto.

Como se mencionará más adelante, todo algoritmo genético precisa de una función de calidad. Concretamente, para este problema se requiere saber cómo es de bueno cada configuración del tablero. Para ello se tiene en cuenta el número de "avistamientos" de unas con otra:

$$Calidad = 1 - \frac{N_{diag} + N_{dir}}{N_{TOT}} \quad (6)$$

donde N_{diag} es el número de reinas que ve en las dos diagonales, N_{dir} es el número de avistamientos en las direcciones de las casiilas; y N_{TOT} es el número total de avistamientos que se podría dar en el caso de que todas estén alineadas (56 para un tablero 8x8)

3.2.2 Técnica de aprendizaje empleada

En este primer problema se va a utilizar uno de los algoritmos evolutivos más utilizado, llamado algoritmo genético que se describirá en este apartado.

Los Algoritmos Genéticos (AG) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización basado en el mecanismo de *selección natural* (supervivencia de los más fuertes), tomando una estrategia similar a la evolución vista en la naturaleza. Es decir, aquellas soluciones que tengan mayor capacidad de resolver el problema en cuestión tendrán más posibilidades de tener descendencia. Por tanto, el algoritmo genético actúa sobre un conjunto de soluciones potenciales llamada población. Cada una de estas posibles soluciones se les llama individuos, que representan cada una de las variables presentes en el proceso y que forman el cromosoma (individuo = cromosoma). Estos cromosomas se componen de una serie de valores que en gran parte de los casos está representado por número binarios.

Su gran utilidad proviene del hecho de que se trata de una técnica robusta que pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, No obstante, es cierto que para ciertos problemas muy específicos donde ya se ha desarrollado un algoritmo para solucionarlo, suele funcionar más rápido y eficientemente éste último, a pesar de que el AGS nos ofrece un buen funcionamiento. Por ello, los algoritmos genéticos suelen tener su campo de actuación en aquellos que no existe un algoritmo específico.

Como ya se dijo anteriormente, el algoritmo trabaja sobre la población, la cual evoluciona en cada paso o generación. Para saber cuán apto es cada uno de los individuos es necesaria una función de evaluación que esté acorde con el problema que se desea resolver. Para avanzar a la siguiente generación, se crean los nuevos individuos según dos estrategias básicas de evolución (cruce y mutación, cada una con una probabilidad de que suceda) y un posible elitismo. De esta manera se produce una nueva población, la cual reemplaza a la anterior y verifica que contiene una mayor

proporción de buenas características en comparación con la población anterior. Así, las buenas características se van propagando a través de la población. Favoreciendo el cruce a aquellos un mejor valor de aptitud, van siendo exploradas las zonas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético está bien diseñado, la población convergerá hacia una solución óptima en caso de que esta exista

.

Pseudocódigo de un Algoritmo Genético Simple:

```
begin /* Algoritmo Genetico Simple */
        Generar una poblacion inicial.
        Computar la funcion de evaluacion de cada individuo.
        while not Terminado do
        begin /* Producir nueva generacion */
                for Tamano˜poblacion/2 do
                        begin /*Ciclo Reproductivo */
                        Cruce ← Selección
                        If random() < P_rep then.
                                Mutación
                        end if
                        Evaluación
                        Nuvea generación ← Mutados
                end for
                if la poblacion ha convergido then
                        Terminado := TRUE
                end if
        end for
```

3.2.2.1 Operadores Evolutivos:

Como se ha visto en el pseudocódigo, los algoritmos genéticos disponen de 3 operadores evolutivos que llevan a cabo el reemplazo de unos individuos por los de la siguiente generación. Estos son: Selección, Cruce y Mutación.

Operadores de Selección

Encontramos varias formas de seleccionar los *padres* de la siguiente población:

- Elitista: Se escogen a los "n" mejores de la población y pasan intactos a la siguiente. Se suele combinar con otros métodos.
- Proporcional a la aptitud: La *selección de padres* se efectúa al azar pero favoreciendo a aquellos que tengan un valor de aptitud más alto, ya que a cada individuo se le asigna una probabilidad de ser *padre* que es proporcional a su valor de aptitud. Se dice que está basado en la ruleta sesgada, es decir, aquellos individuos que están bien adaptados al problema se escogerán varias veces en cada generación, mientras que aquellos están pobremente adaptados solo serán escogidos de vez en cuando.

- Por torneo: Se selecciona aleatoriamente “n” individuos de la población y entre ellos se escoge el que mayor aptitud posea. Si la población tiene “N” individuos, pues se repite este proceso “N” veces.

Cruce

Este operador utiliza los dos padres anteriormente seleccionados y devuelve uno o dos descendientes según el tipo de cruce que se utilice. Habitualmente el operador de cruce no se aplica a todos las parejas seleccionadas sino que tiene una probabilidad de que suceda comprendida entre 0.5 y 1.0. Los tipos que tiene son los siguientes:

- Cruce por 1 punto: Corta las ristras de los individuos por un punto al azar, para producir dos subristras iniciales (desde el principio del cromosoma hasta el punto de corte) y dos subristras finales (desde el punto de corte hasta el final). Se intercambian las subristras finales dando lugar los dos nuevos cromosomas.

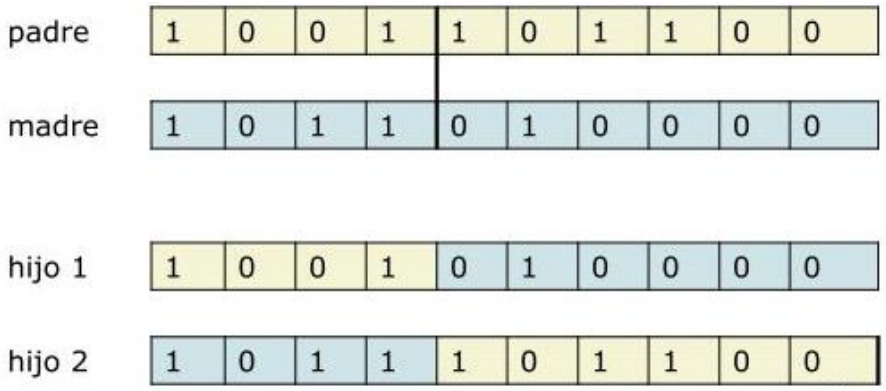


Figura 6. Cruce por un punto

- Cruce multipunto: El funcionamiento es similar al anterior solo que esta vez con más de un punto de corte.

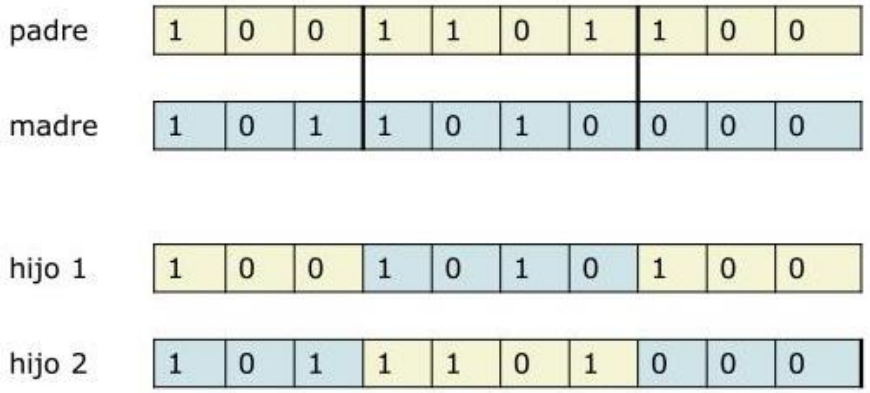


Figura 7. Cruce por 2 puntos

- Cruce Uniforme: En este tipo de cruce, cada gen de la descendencia se crea copiando el correspondiente gen de uno de los padres, escogido de acuerdo a una "máscara de cruce" generada aleatoriamente. Es decir, cada gen del descendiente tiene la misma probabilidad de proceder de cualquiera de los distintos progenitores.

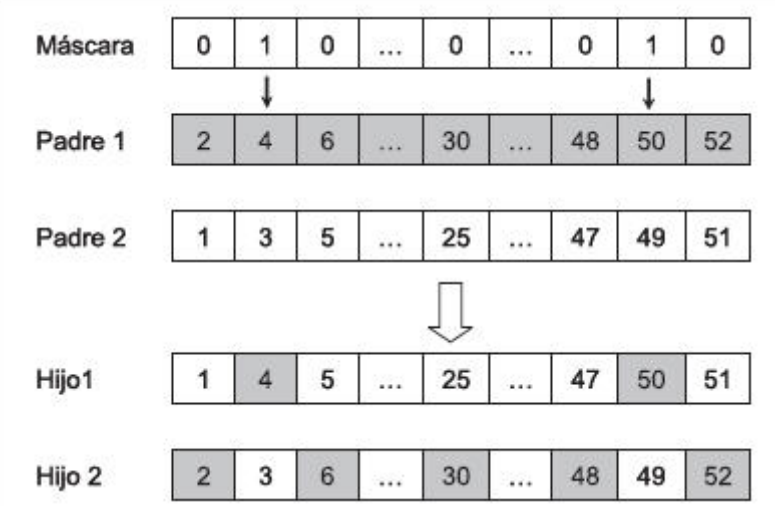


Figura 8. Cruce uniforme

Mutación

Proporciona un pequeño elemento de aleatoriedad en los individuos de la población. Si bien el operador de cruce es el responsable de buscar dentro de las posibles soluciones, el operador de mutación va ganando en importancia a medida que los individuos van convergiendo.

Consiste en introducir cambios aleatorios en los genes del cromosoma. Este cambio puede producirse en uno, varios o todos los genes. Cuando el objetivo es uno o varios, se escogen la posiciones a variar aleatoriamente y se varía por otro valor aleatorio o por el mismo con una pequeña variación.

A lo largo de las generaciones la probabilidad de mutación puede variar o no según se desee. Diferenciamos tres opciones posibles:

- Aumenta según va aumentando el número de generaciones. Tiene como objetivo mantener una cierta heterogeneidad cuando los individuos tienden a converger.

- Disminuye según va aumentando el número de generaciones, asegurando así una gran diversidad en la etapa inicial de la evolución.

- Constante durante todo el proceso

3.2.3 Algoritmo genético desarrollado

Centrándonos más en la resolución del problema, se desarrolla un código apoyado en un algoritmo genético con el fin de resolver el problema de las 8 reinas. Dicho algoritmo se basa la estructura general presentada en la sección anterior pero sin dejar de lado las necesidades concretas que requiere nuestro problema.

Figura 9. Diagrama del AGS.

Para describir el algoritmo de forma más eficaz iremos explicando punto por punto las diferentes fases del mismo presentes en el diagrama de la Figura 9.

En primer lugar, en lo que respecta a la inicialización, se ha decidido crear una población inicial completamente aleatoria, intentando así conseguir una gran diversificación inicial.

En segundo lugar, se define las características de los diferentes operadores evolutivos, lo que incluiría 'selección' y 'evaluación':

- *Elitismo*: En cada salto de generación se escoge a los dos con mejor *fitness* y pasan intactos a la siguiente generación. La cantidad de la élite no se ve variada en todo el proceso. Ser elegido aquí no les exime de poder ser elegidos para tener descendencia.

- Selección por *torneo*: Este proceso consiste en seleccionar al que mejor *fitness* tenga en un grupo de "n" individuos escogidos al azar. Se realiza dos veces para tener a los dos padres que serán recombinado. Este proceso (2 selecciones) se

repite las veces necesarias para llegar a los N individuos que tiene la población. Es decir, se repetirá (N-2)/2 veces.

- *Cruce:* Se ha elegido el cruce por un punto. En este caso, la probabilidad de que suceda el cruce es 1, es decir, todos aquellos que han sido elegidos como padres la van a sufrir la recombinación por cruce

- *Mutación:* Se cambia el valor de un solo gen. Se le suma a un gen aleatorio un valor aleatorio entre -4 y 4. La probabilidad puede variarse según se desee.

A continuación se evalúa a todos los individuos de la población y se comprueba si alguno de ellos es una solución lo suficientemente buena. En caso afirmativo termina el proceso, pero en caso contrario se vuelve al paso de selección. Se repite este proceso hasta llegar a la solución o hasta llegar al número máximo de soluciones.

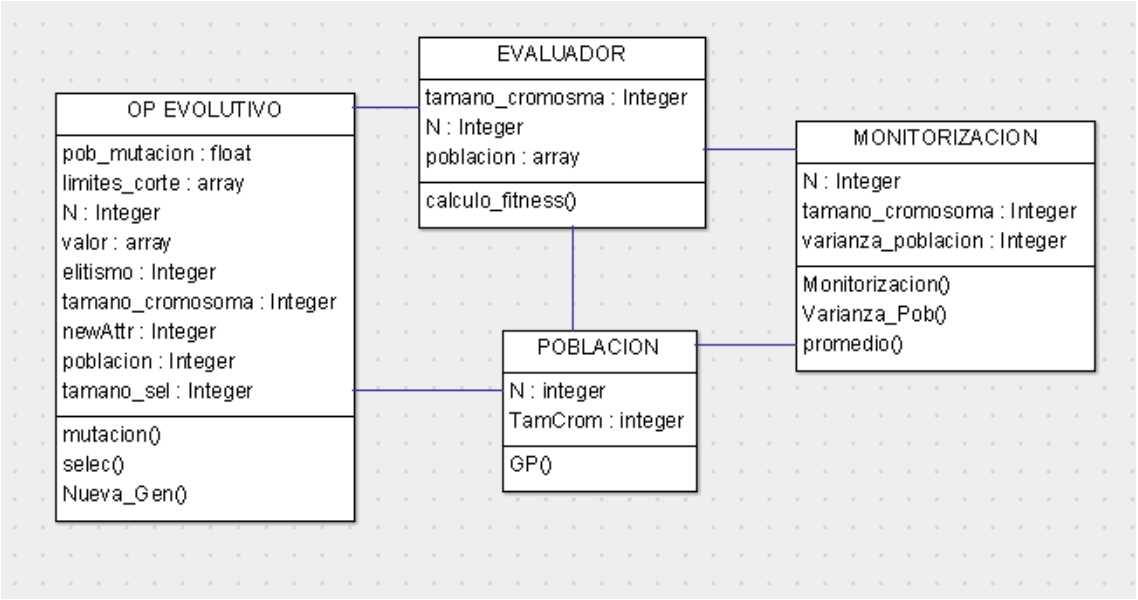


Figura 10. Diagrama de clases del algoritmo del problema de las 8 reinas

3.2.4 Resultados

Para la resolución del problema se ha optado por la configuración óptima obtenida en el análisis de sensibilidad que se realiza a posteriori para simplificar esta sección. Estos parámetros son los siguientes:

PARÁMETRO	VALOR
Probabilidad mutación	0.8
Tamaño población	110
Tamaño selección	3
Intensidad mutación	1
Elitismo	2

Tabla 1. Parámetros del AGS

Ahora se procede resolver el problema y obtener una solución del mismo:

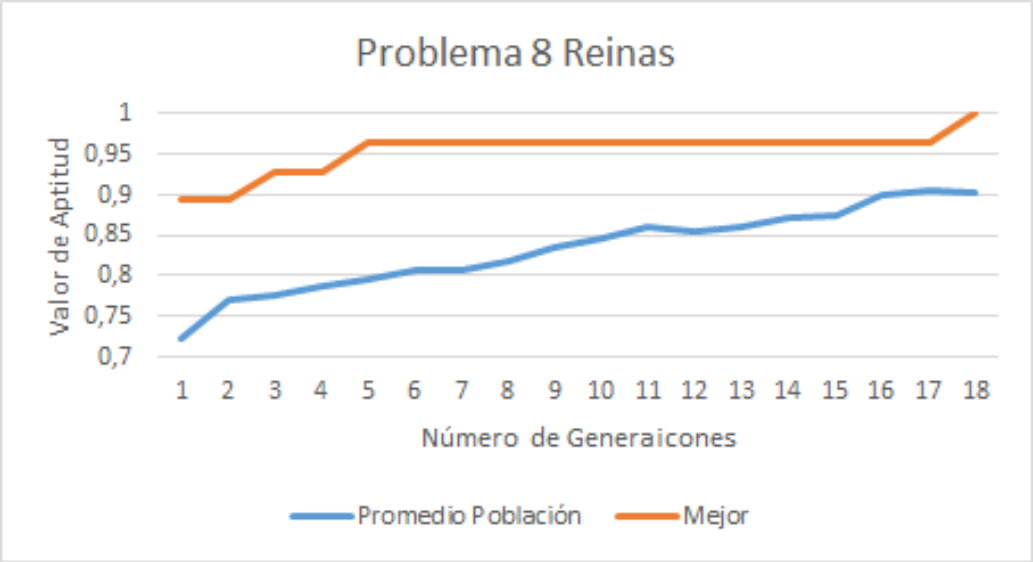


Figura 11. Calidad del mejor y la media durante el proceso

En la Figura 17 podemos observar cómo va mejorando la calidad de toda la población hasta encontrar la solución. La calidad del mejor nunca se ve disminuida en este problema puesto que con el elitismo siempre guardamos al mejor, y lógicamente no varía su calidad. El resultado de esta configuración ha sido bastante satisfactoria, puesto que ha encontrado una solución en tan sólo 18 generaciones.

A continuación se muestra la configuración tablero (fenotipo) que hemos obtenido tras la simulación, con un código genético (genotipo) que es [3, 6, 4, 1, 8, 5, 7, 2]:

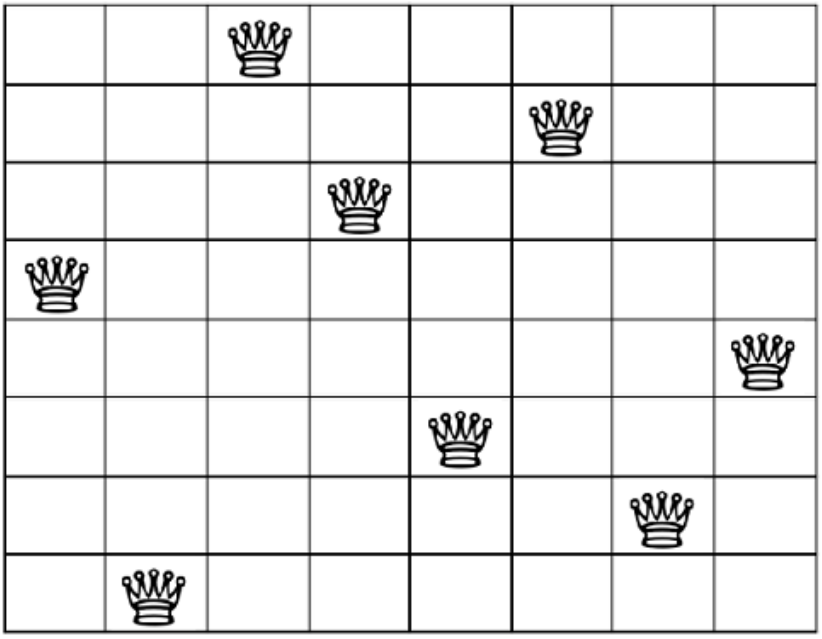


Figura 12. Configuración de tablero de una solución

Ninguna de las reinas vistas en la Figura 18 elimina del tablero a otra, por tanto podemos dar como válida la solución devuelta por el algoritmo genético desarrollado. Cabe destacar que estamos ante un problema discreto, donde hay un número limitado de soluciones válidas, al contrario que uno continuo.

3.2.5 Análisis de sensibilidad

Para conocer el problema con más profundidad y poder optimizar su eficacia se han realizado unas pruebas para ver cómo se comporta el AG variando los siguientes parámetros

- N (Tamaño de la población)
- Tamaño del torneo.
- Probabilidad de mutación

El objetivo de este análisis es ver cómo afectan estos tres parámetros tanto al número de iteraciones necesarias en caso de que encuentre solución como en el porcentaje de veces que encuentra una. Para ello, con cada configuración se harán 120 intentos, siendo un número significativo de repeticiones para sacar conclusiones válidas.

Por tanto, se medirán los siguientes indicadores de rendimiento:

- NIF: Promedio del Número de Iteraciones Finales, es decir, el número de generaciones que ha necesitado el algoritmo para encontrar la solución
- QF: Promedio de las calidades de todos los individuos en la última generación
- SP: Porcentaje del número de veces que se encuentra la solución en los 120 intentos que se realizan

La figuras 13 y 14 muestran los resultados de un conjunto de ejecuciones con la configuración que se muestra en la siguiente tabla:

PARÁMETRO	VALOR
TAMAÑO DE TORNEO	De 1 a 10 (paso: 1)
TAMAÑO POBLACIÓN	80
PROBABILIDAD MUTACIÓN	0.75

Tabla 2. Configuración del AGS en la primera prueba

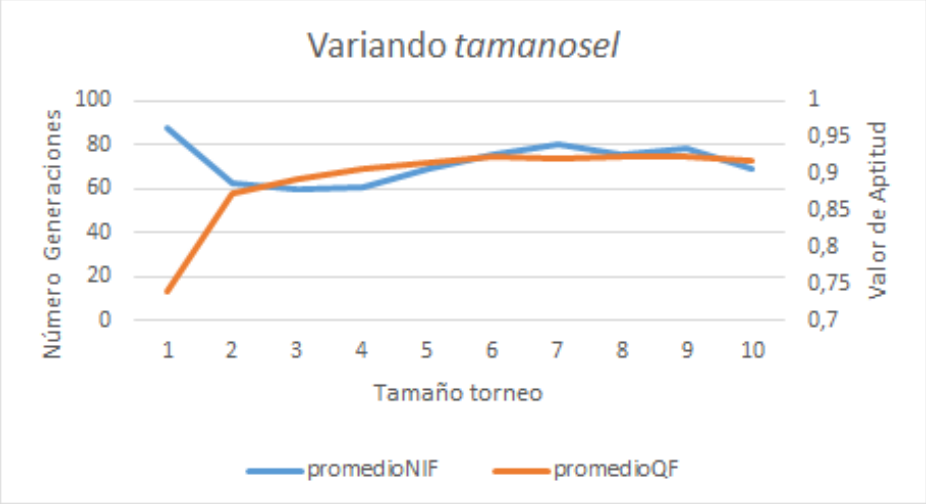


Figura 13. Número de generaciones y valor de aptitud frente al número de torneo

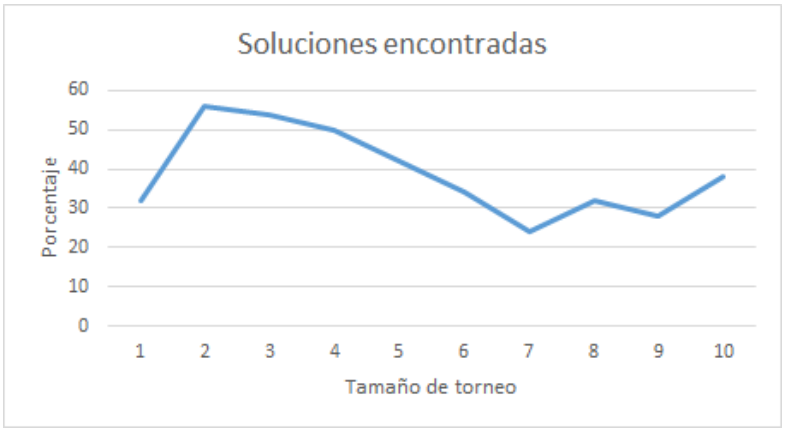


Figura 14. Soluciones encontradas frente al tamaño de torneo

En un comienzo, tanto el porcentaje de veces que encuentra la solución (figura 14) cómo el valor de aptitud medio de la población (figura 13) se ven aumentadas con el tamaño de torneo, hasta aproximadamente el valor de 3 para el caso del porcentaje de soluciones. No obstante, el número medio de generaciones que utiliza para encontrar la solución desciende hasta que llega a un torneo superior a 4, que vuelve a aumentar (figura 13).

Analizando ambas gráficas, escogemos un tamaño de torneo de 3. Es cierto que el porcentaje de soluciones desciende ligeramente en este valor, pero baja el número de generaciones necesarias y por tanto el coste y tiempo computacional.

Según se aumenta el tamaño de torneo vemos como desciende notablemente las veces que se encuentra solución (figura 14), a pesar de que aumenta la calidad media de la población en la última generación. Esto se debe a que la evolución se atasca en máximos locales, ya que minimizas las probabilidades de que se escoja uno con poca calidad, es decir, disminuyes la exploración.

Teniendo en cuenta estos resultados, la configuración de la segunda prueba es la siguiente:

PARÁMETRO	VALOR
TAMAÑO DE TORNEO	3
TAMAÑO POBLACIÓN	Desde 30 a 210 (paso: 20)
PROBABILIDAD MUTACIÓN	0.75

Tabla 3. Configuración del AGS en la segunda prueba

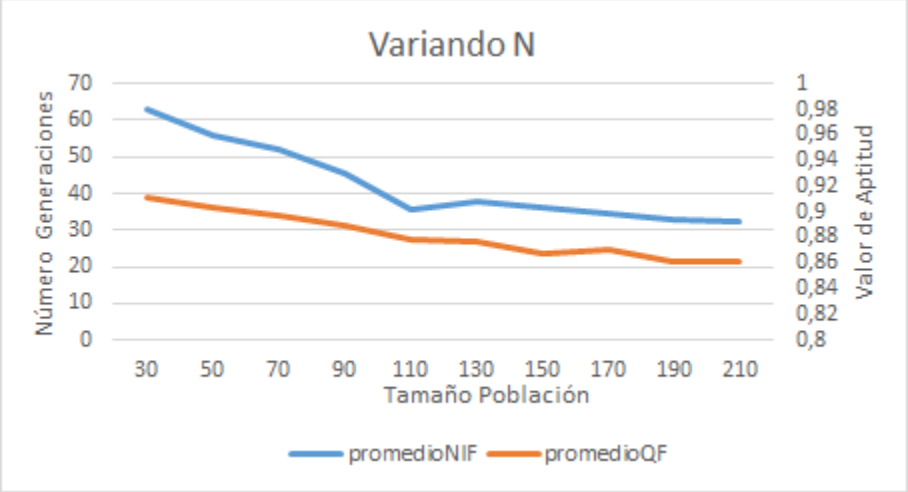


Figura 15. Valor de aptitud y número de generaciones frente al tamaño de la población

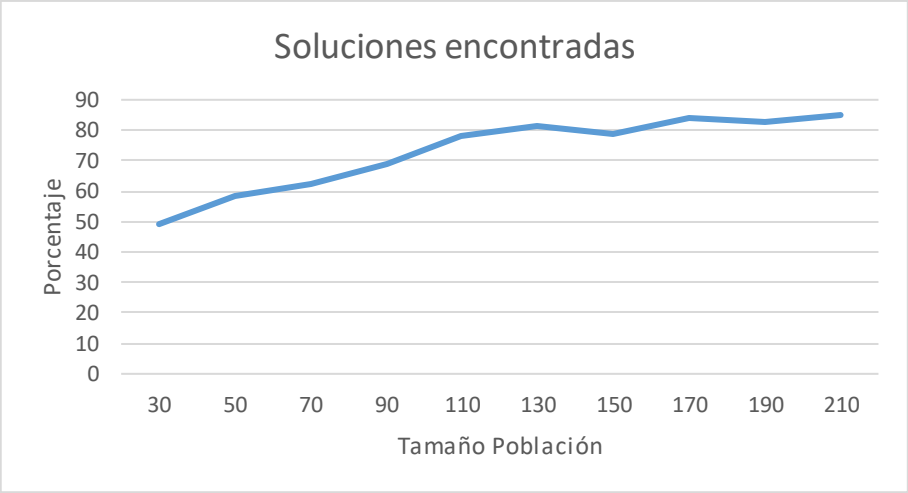


Figura 16. Soluciones encontradas frente a tamaño de la población

Este caso es bastante diferente al anterior. Por un lado de la balanza, se puede observar en las figuras 15 y 16 como nunca un aumento del tamaño de la población es perjudicial a la hora de resolver el problema, ya que conlleva una mayor exploración del espacio de búsqueda. La disminución de calidad promedio (figura 15) se debe a que el tamaño de torneo cada vez es menor en proporción al tamaño de la población. Pero por en el otro lado de la balanza nos encontramos que, como es lógico, cuanto mayor sea la población mayor será el número de operaciones a realizar, y por tanto mayor el tiempo. Intentando equilibrar ambos lados, se escoge un tamaño de 110, donde la subida del porcentaje de soluciones encontradas ya no es tan pronunciada, del mismo modo que sucede con la disminución del número de generaciones.

Para la última y tercera prueba, la configuración es:

PARÁMETRO	VALOR
TAMAÑO DE TORNEO	3
TAMAÑO POBLACIÓN	110
PROBABILIDAD MUTACIÓN	0 a 1 (paso: 0.1)

Tabla 4.Configuración del AGS en la tercera prueba

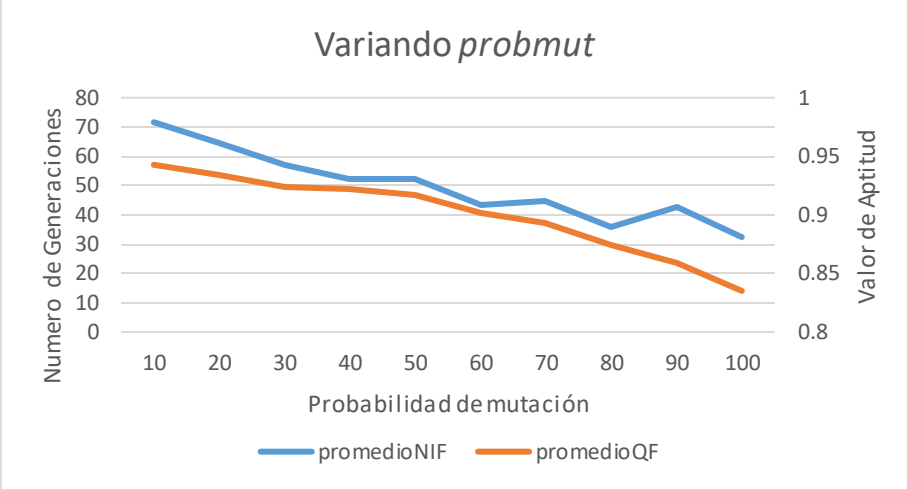


Figura 17. Número de generaciones y calidad frente a la probabilidad de mutación (%)

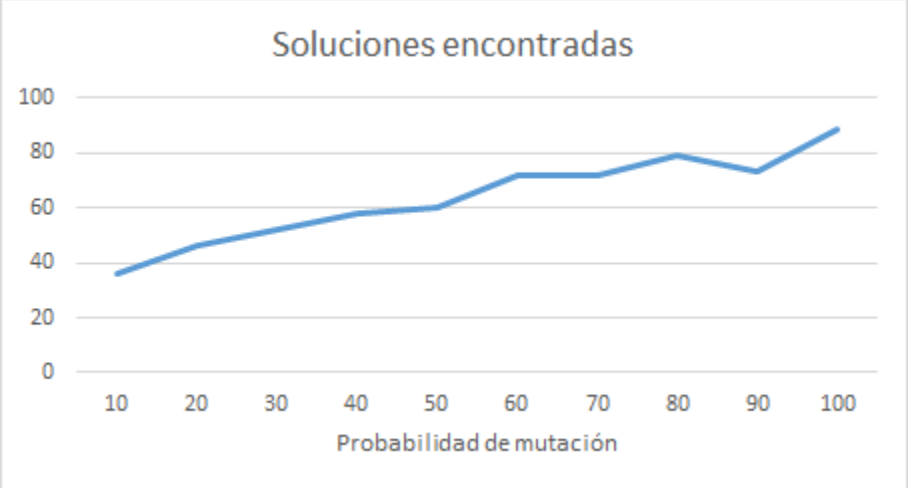


Figura 18. Soluciones encontradas frente a la probabilidad de mutación (%)

Esta vez, en un caso semejante al anterior. Un aumento de la probabilidad de mutación implica mayor porcentaje de soluciones encontradas (figura 18), menor número de generaciones y valor de aptitud (figura 17). Esto último se debe a que al variar tanto el

código genético de forma aleatoria no avanza hacia máximos de forma tan clara como cuando estaba baja, pero abarca más espacio de búsqueda. Elegimos una probabilidad de 0.8 para no excedernos demasiado ya que puede llegar a ser contraproducente.

Con todas estas pruebas podemos concluir que es un problema donde la mutación funciona muy bien, puesto que es un problema más bien de explorar el espacio de búsqueda que de explotar aquellos que tienen mucha calidad. No obstante, ha de encontrar el equilibrio entre exploración y explotación para alcanzar un comportamiento óptimo del algoritmo genético.

3.3 Aplicación en un problema continuo y dinámico

En este caso, se va a presentar un problema esencialmente diferente puesto que el espacio de búsqueda, al contrario que antes, es continuo y dispone de un número ilimitado de soluciones.

3.3.1 Descripción del problema

Se va a estudiar un escenario en el que se desea resolver un problema de *shepherding* [3] o pastoreo. Este tipo de problemas consisten en un agente externo que trata de guiar a uºn grupo de agentes a una posición. Para ello, tenemos un agente (que actúa de pastor o *shepherd*) que hará esta tarea gracias a una red neuronal artificial en su controlador que definirá su comportamiento y la cual irá mejorando su rendimiento según se avance en el proceso de evolución.

Como ya se explicó en el apartado de sistemas de decisión en *machine learning*, una red neuronal tiene unos estímulos de entrada. La sensorización que se proporciona al controlador de un agente tiene un gran impacto en el problema a resolver. Por un lado una sensorización con gran cantidad de datos mejora la complejidad del comportamiento que se puede generar, pero como contrapartida incrementa la dificultad del problema. Para este problema se ha considerado una sensorización total, es decir, aportamos al controlador del agente todas y cada una de las posiciones relativas a su velocidad y posición en forma de ángulo y distancia, además de la posición del punto objetivo. Por tanto, teniendo en cuenta que se simulan 15 ovejas tendremos una red

Respecto a la calidad, se ha desarrollado una función que nos devuelve un valor entre 0 y 1 según lo que ha avanzado el rebaño hacia la posición objetivo. Cuando el agente consigue llevar todas las ovejas a una distancia predefinida al objetivo se vuelve a inicializar tanto la posición del agente como la del rebaño para que tenga que volver a llevarlo. Entonces, para el cálculo de la calidad del individuo se utiliza la siguiente ecuación:

$$fitness = \left(\frac{\sum_{i=1}^{N_{obj}} \left\| P_{inicial} - P_{obj} \right\|}{N_{pasos}\ V_{max}}\right)_{OBJ} + \left(\frac{\left\| P_{inical} - P \right\|}{N_{pasos} V_{max}}\right)_{FUERA} \quad (7)$$

donde N_{obj} es el número de veces que consigue llevar el rebaño al objetivo, $P_{inicial}$ es la posición del centro de masas en la que se inicializa, P_{obj} e la posición del centro de masas cuando entra en el objetivo, N_{pasos} es el número de pasos que se realiza en la

simulación, V_{max} es la velocidad máxima de las ovejas, y P es la posición final del centro de masas cuando no se ha conseguido llevar el rebaño al objetivo y acaba la simulación.

Como se puede ver en la ecuación, se diferencia claramente dos sumandos. El primero se trata de la distancia normalizada que recorre el rebaño hacia el objetivo cuando alcanza el objetivo. En cambio, el segundo sumando es la distancia normalizada (pudiendo ser negativa) que recorre en dirección del objetivo cuando no se consigue introducir en el objetivo y finaliza la simulación, es decir, se llega al número de pasos que se establecieron.

3.3.2 Técnica de aprendizaje empleada

Una vez descrito el problema, se procede a la descripción de la técnica de aprendizaje que se va a utilizar para resolverlo. En este caso será un algoritmo neuroevolutivo de la librería NEAT, con un sistema de decisiones que será una red neuronal (como se dice en el apartado anterior) también de la misma librería.

3.3.3 Algoritmo neuroevolutivo de NEAT

Un algoritmo neuroevolutivo es aquel que está destinado a la optimización de una red neuronal artificial, tanto en lo que se refiere a los pesos y bias como a su topología. Para la descripción del algoritmo neuroevolutivo empezaremos por explicar la codificación genética que lleva asociada.

3.3.3.1 Codificación genética

Cada código genético incluye una lista de conexiones entre genes, indicando qué dos nudos están conectados, como podemos observar en la figura 20. Cada conexión específica el nudo del que nace la conexión, el nudo destino, el peso de la conexión, si está o no la conexión (un bit que nos lo indica) y un número innovación, que permite encontrar correspondencia en los genes durante el cruce, como se explicará más adelante. A pesar de que sólo se vaya a explicar el algoritmo para redes con una sola salida (cómo será nuestro caso), NEAT también puede evolucionar redes con más de una.

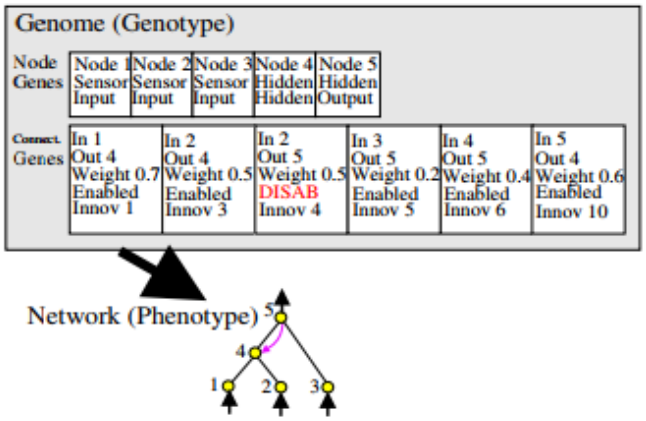


Figura 19. Genotipo y Fenotipo de una red (NEAT) (Fuente: [4]*)*

La mutación en NEAT puede cambiar los pesos de las conexiones y la estructura de la red. La mutación de los pesos es como en sistema neuroevolutivo común, cada gen puede ser variado o no según una función aleatoria. En cambio, la mutación a nivel estructural el cual expande el código genético ocurre de dos maneras (Figura 21):

- *Add conexion mutation*: una nueva y simple conexión es añadida entre dos nudos que previamente no estaban conectados

- *Add node mutation*: Una vieja conexión es eliminada y un nuevo nodo ocupa el lugar donde estaba esta. Dos nuevas conexiones son añadidas al código genético (una va desde el primer nudo de la vieja conexión hasta el nuevo nudo y la segundad desde el nuevo nudo hasta el nudo destino de la vieja conexión).

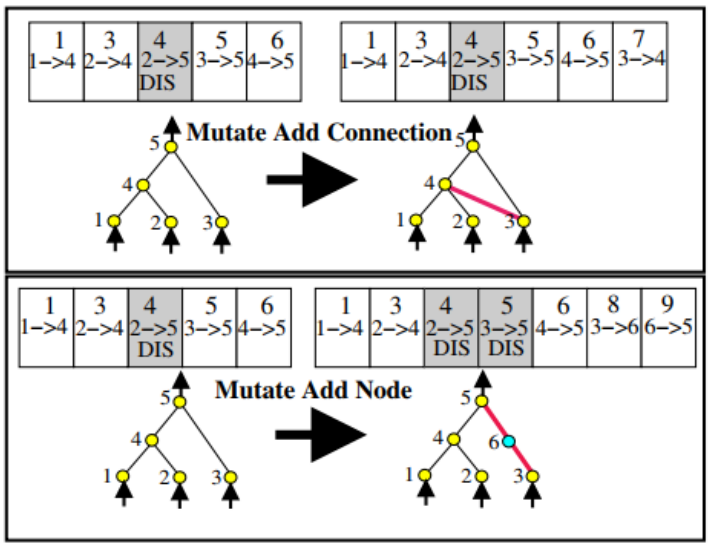


Figura 20. Las dos mutaciones estructurales que se realizan en NEAT. (Fuente: [4]*)*

Debido a la mutación, se crean códigos genéticos de diferentes tamaños, a veces con conexiones completamente diferentes aun estando en la misma posición del código. A continuación se explica cómo realiza NEAT el cruce.

3.3.3.2 Genes de seguimiento a través de notas históricas

Para mejorar el cruce, el sistema debe ser capaz de decir que genes ha de emparejar entre los individuos de la población. La observación clave es que dos genes que tienen el mismo origen histórico representan la misma estructura (con diferentes pesos), ya que nacen del mismo gen en el pasado.

El seguimiento del origen histórico de los genes requiere muy poco computacionalmente hablando. Cuando una nueva especie aparece (mediante una mutación estructural), un número innovación global es incrementado u asociado a este gen. Los números innovación representa la cronología de cada gen en el sistema. Como ejemplo, se pueden ver las dos mutaciones que ocurren en la Figura 21 una después de otra en el sistema. La nueva conexión creada en la primera mutación se le asigna el número 7, y las dos nuevas conexiones añadidas durante la segunda 8 y 9. Más adelante, durante el cruce, a la descendencia se le asignará el mismo número en cada gen; el número innovación no varía nunca. Así, el origen histórico de cada gen del sistema perdura durante la evolución. Ahora el sistema sabe que genes cruzar con cuales.

El método de cruce elaborado por NEAT es notable por su simplicidad. Dos estructuras cualquieras puede ser combinado sin necesidad de un análisis topológico, aunque aparentemente parezca que el problema pueda ser por sus topologías. Resolviendo el problema como uno de notas históricas, convirtiéndolo en un problema tratable y significativamente sencillo de solucionar.

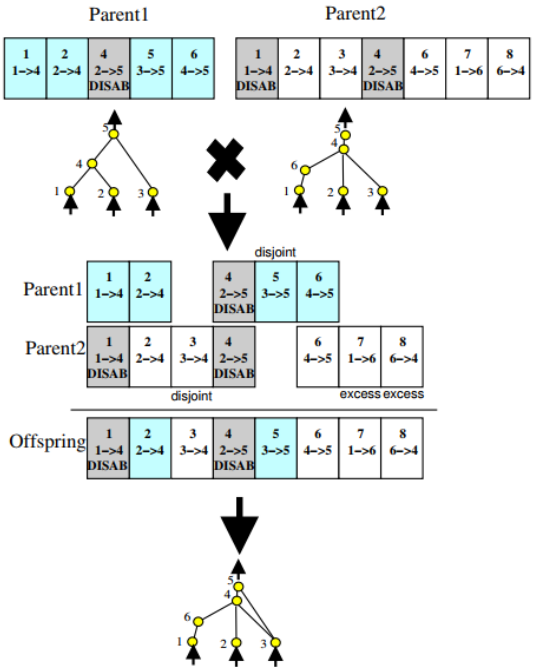


Figura 21 .Emparejamiento de dos redes con diferentes topologías gracias al número innovación (Fuente:[4]*)*

Como se puede observar en la Figura 22, a pesar de que el padre 1 y el padre 2 parecen diferentes, sus número innovación (se muestra en la parte de arriba cada gen) nos dice que genes se emparejan con cuales sin necesidad de realizar un análisis topológico de ambas redes.

3.3.3.3 Proteger innovación a través de la Especiación

Añadir una nueva estructura al sistema, como forma general, inicialmente reduce el valor de calidad. Sin embargo, NEAT especializa la población, para que los individuos compitan primero con los de su propio nicho en vez de que lo hagan con toda la población. De esta manera, la innovación topológica está protegida y tiene un tiempo para optimizar su estructura antes de competir con las demás especies de la población.

Las notas históricas mencionadas anteriormente hacen posible al sistema divide la población en especies basadas en un similitud topológicamente hablando. El número de genes que tiene uno y otro no (*excess* y *disjoints* en la Figura 22) nos indica la compatibilidad que hay entre estos. Cuanto menos compatibles sean dos genes, menos historia evolutiva comparten. Por lo tanto, se puede medir la compatibilidad (δ) de una pareja de códigos genéticos según el número de genes de exceso (E) y disjuntos (D), además de la diferencia media de los pesos de las conexiones ($\overline{W}$):

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \qquad (8)$$

Los coeficientes c_1, c_2 y c_3 se ajustan según queramos que unos factores u otros tengan más o menos importancia; y el factor N es el número de genes del código de mayor tamaño (normalizado).

La medida d distancia δ permite especializar usando un límite (δ_t). Los genomas son testeados una vez cada cierto tiempo: si la distancia entre los miembros elegidos aleatoriamente es menor que δ_t, se les coloca dentro de la misma especie. Cada genoma es colocado en la primera especia donde satisfaga las condiciones, para que no pueda estar en más de una al mismo tiempo.

En el proceso de reproducción de NEAT se usa *explicit fitness sharing* [2], donde los organismos de la misma especia comparten la calidad de su nicho, haciendo más difícil que una especia tome el control de la población. La calidad de una especie se calcula de la siguiente manera:

$$N_j' = \frac{\sum_{i=1}^{N_j} f_{ij}}{\bar{f}} \qquad (9)$$

donde N_j y N_j' son el viejo y el nuevo número de individuos en la especie j, f_{ij} es la calidad ajustada del individuo i en la población j, y $\bar{f}$ la media ajustada de toda la población.

3.3.3.4 Minimizar dimensiones

Típicamente, los algoritmos TWEANN (evolucionan las redes neuronales aumentando su estructura) empiezan con una población inicial aleatoria [3]. Tal diversidad topológica se introduce desde el principio porque nuevas estructuras no suelen sobrevivir en aquellos sistemas sin protección de innovación. Sin embargo, no se sabe a ciencia cierta que tal diversidad sea necesaria y útil. Además, para algunos problemas muchas de estas estructuras pueden ser demasiado complejas y estás perdiendo tiempo optimizándolas.

Por el contrario, NEAT comienza con una población uniforme de redes sin nudos ocultos. Porque NEAT protege la innovación con la especialización, lo que le permite empezar de esta manera (las estructura lo más simple posibles) y hacer crecer la estructura siempre y cuando sea necesario. Nuevas estructuras vas surgiendo según van sucediendo las mutaciones, y solo estas sobreviven cuando son útiles según la función de evaluación. De esta manera, NEAT busca una solución válida al problema con unas dimensiones de pesos mínimas, reduciendo significativamente el número de generaciones para encontrar una solución.

3.3.4 Código desarrollado

Tras la descripción del algoritmo neuroevolutivo de NEAT ahora se va a explicar el código desarrollado donde se implementa este algoritmo. A continuación se muestra el diagrama de clases de dicho código donde se ve que clases se relacionan con que clase y las diferentes variables y funciones que manejan:

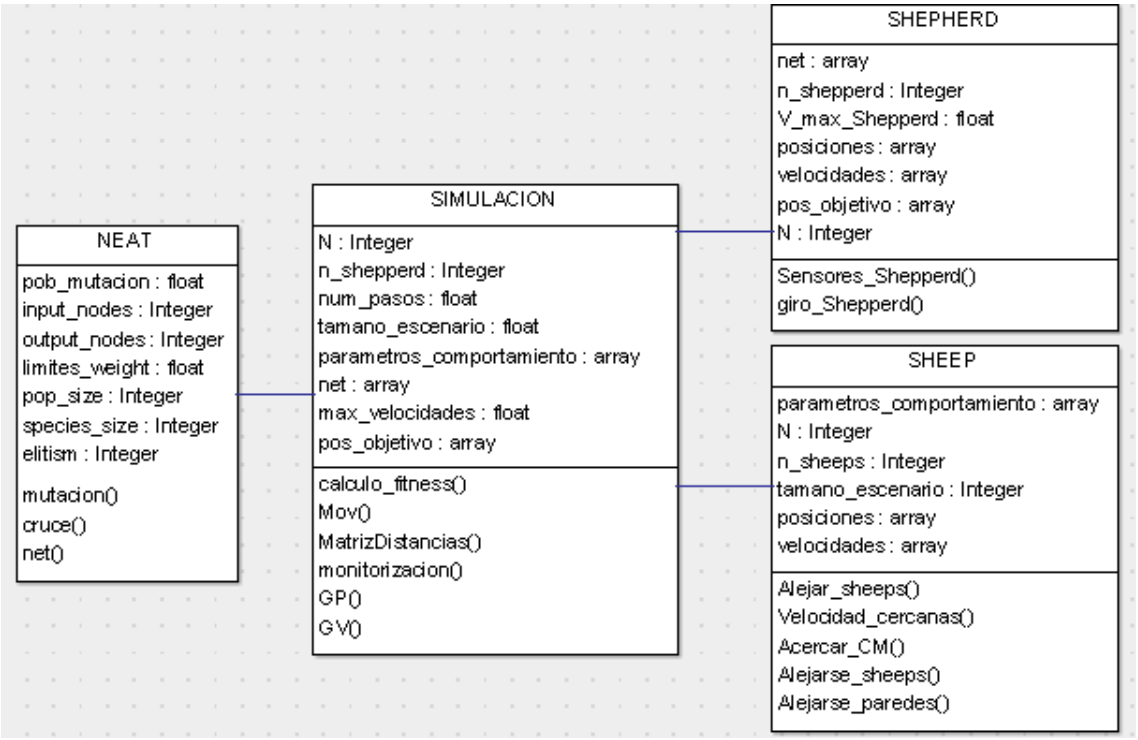


Figura 22. Diagrama de clases del problema continuo con 1 solo robot

Clases y relaciones entre ellas:

Simulación: Simula el movimiento del perro y del rebaño devolviendo a NEAT un *fitness* mediante la función anteriormente citada. Es la encargada de pedir a las clases oveja y perro la actualización de las velocidades y posiciones de estas y mostrarlo en pantalla en caso de que sea pedido por el diseñador.

- calculo_fitness(): Función encargada de ir evaluando los drones al mismo que tiempo que la simulación está en curso. No obstante, no se le devuelve a NEAT esta calidad hasta estar finalizada la simulación. Por ello estamos ante un algoritmo *off-line.*

- MatrizDistancias(): A partir de las posiciones de las ovejas y los perros calcula todas las distancias entre las diferentes personas para evitar futuros cálculos y que estos se repitan.

- monitorización(): Si el usuario lo desea mientras se evoluciona, o una vez finalizada la evolución, se puede ir mostrando en pantalla los movimientos de los diferentes individuos.

- GP(): Genera posiciones de la multitud y de los robots. Estas pueden ser aleatorias o no.

- GV(): Genera velocidades de las personas y de los robots. Estas pueden ser aleatorias o no.

- Mov(): Recibe la posición y la nueva velocidad de robots y personas y devuelve la nueva posición.

Shepjerd: Simulación le cede la red neuronal que define el comportamiento del perro. Esta tiene 32 *inputs* y un solo *output*. Con la ayuda de dos funciones, se calculará el nuevo vector velocidad del robot.

- Sensores_Robots(): Recibe la posición de los perros, de todas las ovejas y del objetivo y sus respectivas velocidades. Con ellos, realiza los cálculos necesarios para devolver a simulación los sensores que se hayan programado, como se ha explicado en el apartado de *Descripción del problema*.

- giro_Robots(): Convierte la salida de la red (un valor entre 0 y 1) en un giro de las velocidades de los perros entre ± el ángulo máximo que se ha programado.

Oveja: Esta clase es la encargada de definir el comportamiento de las ovejas. Ante exactamente los mismos estímulos externos todas actuarían de forma idéntica.

- Alejar_ovejas(): Entre ellas tienen una cierta repulsión para evitar choques o que ocupen la misma posición en el espacio. Cuanto más cerca estén entre ellas mayor será esta repulsión.

- Velocidad_cercanas(): Las ovejas tienden a la velocidad de las más cercanas para mantener la uniformidad del rebaño.

- Acercar_CM(): Se sienten atraídas por el centro de masas de las ovejas más cercanas, es decir, tienden a juntarse para formar el rebaño. Además, evita que

estas se separen una vez están juntas. A mayor distancia mayor será la atracción hasta un cierto límite, donde empieza a descender hasta hacerse 0.

- Alejarse_paredes(): Evita que se salgan de los límites de nuestro escenario.
- Alejarse_perros(): Simula el comportamiento de una persona ante un robot cercano (se aleja).

Todas y cada una de estas funciones se basan en el mismo tipo de ecuación, con la excepción de *Acercar_CM()*. Esta ecuación es la siguiente:

$$\overrightarrow{f_{ij}} = Max\left(1 - \frac{d_{ij}}{d_{umb}}\right)^{t} * \widehat{V_{ij}} \qquad (10)$$

donde $\overrightarrow{f_{ij}}$ es la fuerza debido a estar cerca de una pared, de un agente, etc…; Max es la norma máxima que podría alcanzar esta fuerza; d_{ij} es la distancia entre la oveja y el otro punto; d_{umb} es las distancia umbral a partir de la cual deja de tener efecto; t es un parámetro para variar la curvatura de la función; y $\widehat{V_{ij}}$ es el vector unitario que va desde la oveja al otro punto, o del otro punto a la oveja (según sean fuerzas de repulsión o de atracción).

Para el caso del centro de masas es similar, solo que la función está dividida en tres partes, como se muestra en la figura :

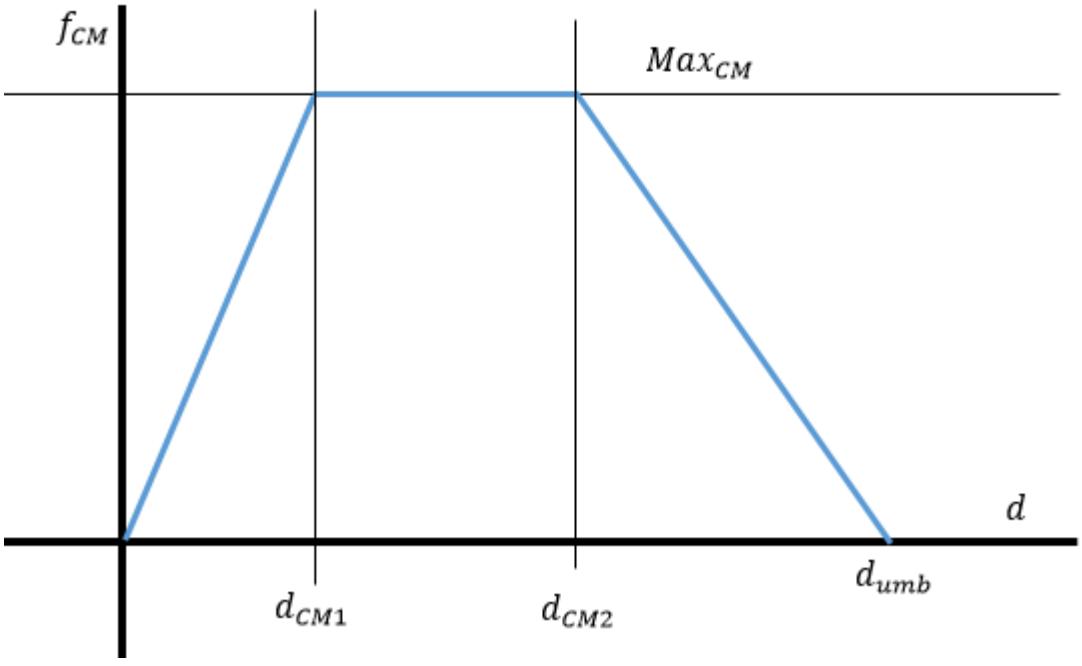


Figura 23: Representación gráfica de la influencia del centro de masas

3.3.5 Comparativa con un controlador manual

Tras haber definido todo el algoritmo utilizado, se va intentar resolver el problema de forma manual, es decir, realizar una función (que no se evoluciona) para intentar resolver el problema y ver que niveles de calidad somos capaces de alcanzar y compararlo con lo que se puede llegar con un algoritmo neuroevolutivo y una red neuronal artificial. Esta función recibe tres ángulos ($\widehat{(\overrightarrow{CM}, \overrightarrow{CR}}, \overrightarrow{V_p}$; $\widehat{\overrightarrow{V_r}, \overrightarrow{V_m}}$ y $\widehat{\overrightarrow{r_{obj}}, \overrightarrow{V_p}}$) y devuelve el giro entre -1 y 1 que debe hacer el perro, siendo -1 un giro máximo hacia la izquierda, 1 giro máximo hacia la derecha, y 0 seguir recto. Mediante unas operaciones trigonométricas y con vectores se calcula las coordenadas de la posición deseada por el robot (CD). Sabiendo esta posición y la velocidad que lleva el robot, miramos que ángulo ha de girar para ir en esa dirección. Si este es mayor que el máximo giro que puede realizar, el valor de la variable giro será 1 ó -1, según hacia qué lado esté. En la siguiente figura puede verse una representación simplificada del comportamiento manual desarrollado:

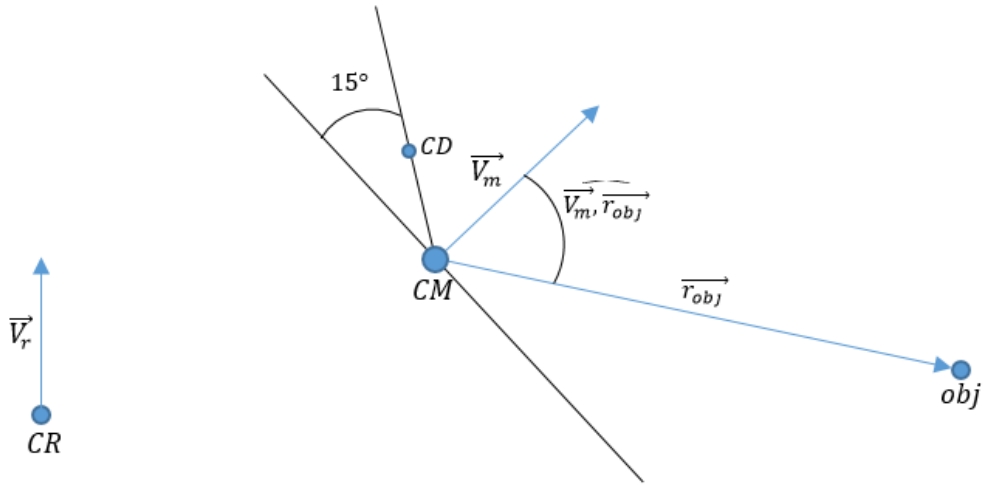


Figura 24. Representación de la configuración manual

Nomenclatura de la Figura 24:

- CR: Coordenada actual del robot
- $\overrightarrow{V_r}$: Vector velocidad del robot
- CD: Coordenadas deseadas por el robot
- CM: Coordenadas del centro de masas de la multitud
- $\overrightarrow{V_m}$: Velocidad media de las personas
- $\overrightarrow{r_{obj}}$: Vector que tiene origen en el centro de masas y destino en el punto objetivo
- obj: Coordenadas de la posición objetivo

En este caso, este comportamiento está definido según dos parámetros: El ángulo relativo entre la perpendicular a $\overrightarrow{V_m}$ y el vector que va desde CM hasta CD; y la distancia entre CM y CD.

Con esta configuración se ha alcanzado un valor de calidad de aproximadamente 0.3 en 500 pasos.

3.3.6 Resolución con NEAT

PARÁMETRO	VALOR
TAMAÑO POBLACIÓN	200
LÍMITE FITNESS	0.9
ELITISMO	3
PROBABILIDAD AÑADIR CONEXIÓN	0.3
PROBABILIDAD AÑADIR NODO	0.01
INTENSIDAD MUTACIÓN BIAS	1.0
PROBABILIDAD MUTAR PESOS	0.5
INTENSIDAD MUTACIÓN PESOS	1.8
LÍMITE DE COMPATIBILIDAD	6.0

Tabla 5. Parámetros del algoritmo de NEAT

A continuación se muestra cómo ha ido variando la calidad media de la población y la del mejor en cada una de las generaciones que se realizan. Para este problema, solo se hacen 40 generaciones debido al coste computacional que exige y por tanto el tiempo que se requiere (sobre 1 día) debido a la cantidad de cálculos que se realizan tanto para el movimiento de las personas como los cruces y mutaciones de las redes.

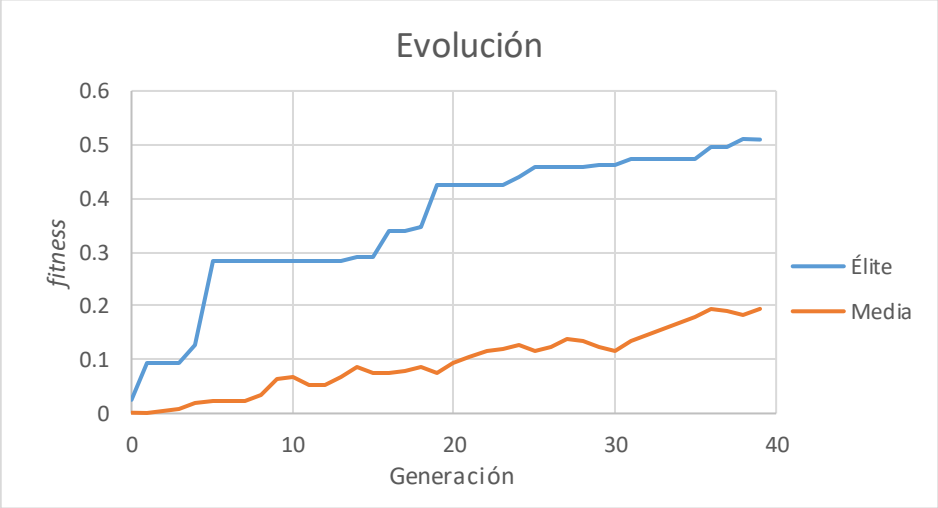


Figura 25. Representación de la calidad de la población durante la evolución

Se puede observar en la Figura 25 que la calidad del mejor de la población en ningún momento desciende. Se debe a que tenemos un elitismo de 3, y por tanto siempre

perduran los 3 mejores de cada generación intatctos para la siguiente. Como sucedía en el problema de las 8 reinas, la media de la población es muy baja, puesto que se inicia aleatoriamente, y siempre está muy por debajo del mejor.

En la siguiente figura, se aprecia que la desviación estandar en las primeras generaciones son exageradamente pequeñas a pesar de que se inicie en una población aleatoria. Esto se debe a que NEAT comienza con la estructura más simple posible y la va aumentado según lo vea necesario. Por tanto, a pesar de variar en los pesos, la desviación estándar en las primeras etapas de la evolución es muy próxima a 0. No obstante, según va avanzando la evolución y van apareciendo nuevas especies, y por tanto nuevas estructuras, aumenta hasta unos valores de aproximadamente 0.16.

Figura 26. Representación de la desviación estándar durante la evolución

Tras 40 generaciones se ha alcanzado un valor de calidad de 0.51. En las fotos de la figura 27 se puede observar el comportamiento del agente:

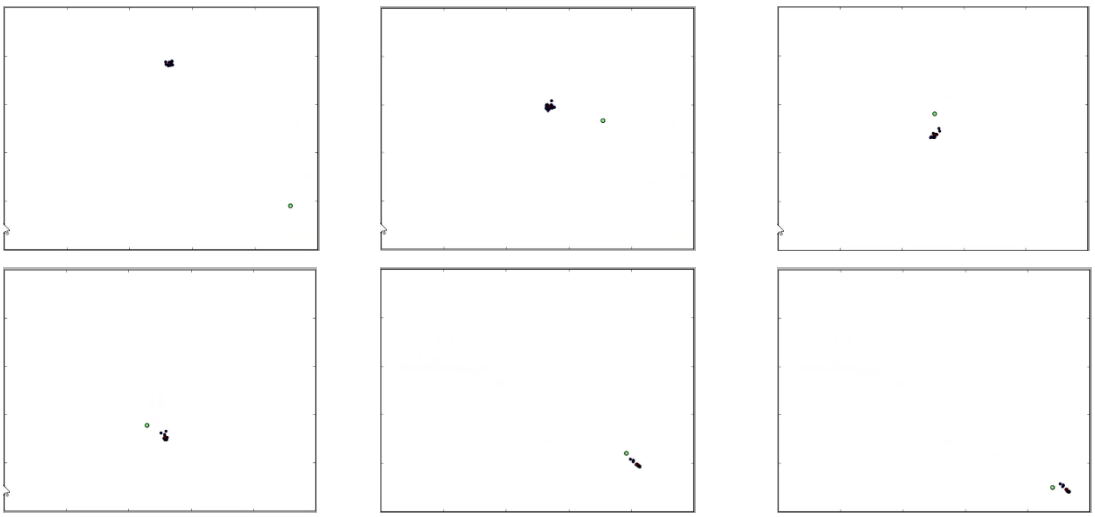


Figura 27: Un perro introduciendo las ovejas en el objetivo

3.3.7 Conclusiones

Una vez obtenidos los resultados tanto de la resolución manual como la resolución mediante NEAT ya se está en disposición de concluir si los resultados son satisfactorios o no. Para ello, se comparan las dos resoluciones.

En la gráfica de la Figura 28 se muestran 3 calidades: La calidad máxima teórica (rojo); la máxima calidad alcanzada mediante NEAT en 40 generaciones (azul); y la calidad obtenida con la configuración manual (verde).

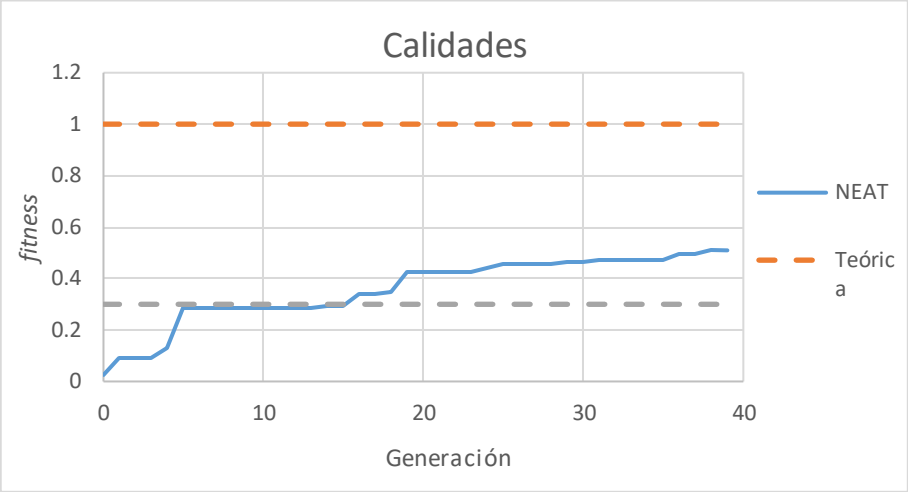


Figura 28. Gráfica comparativa entre el comportamiento manual y el obtenido con NEAT

En 40 generaciones, que corresponde a aproximadamente un día de cálculo en un procesador i5 de Intel, la calidad obtenida mediante evolución supera al mejor controlador que se ha conseguido programar de forma manual. Destacar que el valor de calidad máximo teórico, como ya se dijo en apartados anteriores, es imposible que se alcance.

3.3.8 Aplicaciones

Después de ver que los resultados obtenidos tras la evolución se han considerado satisfactorios, se ha hecho un estudio de las diferentes aplicaciones que tiene la técnica de pastoreo empleada en este problema. Entre otras se han destacado las siguientes:

- Robots que guían una manada
- Evacuación (guiando personas)
- Robots que mantienen a animales alejados de un lugar (por ejemplo un aeropuerto)
- Control de multitudes
- Limpiar el medioambiente (por ejemplo manchas de aceite en el océano)
- Mantener a gente alejada de zonas peligrosas como aguas no seguras, zonas de construcción, etc…

Más adelante, escogeremos la opción de la evacuación.

3.4 Aplicación en un problema continuo con varios agentes

Dentro de esta sección, a diferencia de las dos anteriores, se va a resolver dos problemas diferentes, aumentando un poco la dificultad en el segundo. Por ello, procedemos a explicar la técnica empleada antes de la descripción de ambos problemas.

3.4.1 Sistemas Multiagentes

Como ya se dijo, la gran diferencia de este problema es que ya no tenemos un solo individuo a evolucionar, por lo que tendremos que utilizar un algoritmo basado en un sistema multi-agente para poder llegar a una solución decente. En esta sección, se va a tratar las diferentes características que pueden tener estos algoritmos. Y a continuación se explicará en gran detalle en que consiste los algoritmos *Embodied Evolution* puesto que son los que se van a utilizar más adelante para la resolución de un problema de evacuación.

Un sistema multia-agente e puede definir como un conjunto de agentes que operan sobre la misma tarea en un entorno virtual o simulado. De forma general, suele ser aplicado para resolver problemas colectivos (aquellos que con un solo agente no es suficiente) y distribuidos. No obstante, también se puede utilizar en problemas que no requieren una solución colectiva, pero realizarla como tal conlleva a beneficios, es decir, mejorar con la calidad que podemos llegar a realizar el problema a resolver. Igual que en el caso de los algoritmos genéticos, en este trabajo también nos centraremos en los sistemas multiagentes, ya que se utilizará más adelante para la realización de un problema de evacuación. Un sistema multiagente tiene las siguientes ventajas:

- Aumentan la eficiencia en la resolución de la tarea si esta es paralelizable.
- Desglosa el problema global en unos más sencillos pudiendo alcanzar así trabajos mucho más complejos (de forma colectiva).
- Escalabilidad: Es sencillo añadir agentes al sistema.
- Su reutilización. Cambiando el simulador del entorno y la función que calcule tanto la calidad individual cómo la global
- Son robustos y tolerantes a fallos debido a su redundancia.
- Más rápido y eficiente debido a que la evolución se realiza a la vez que se simula en entorno, no sucede uno detrás de otro
- Son flexibles y escalables. Pueden estar formados con agentes de diferentes capacidades (sensores y actuadores) o con las mismas, y pueden ser tolerantes a cambios en el número de integrantes del equipo.
- Comunicación: Es imposible alcanzar una buena coordinación sin que exista una comunicación entre los agentes

3.4.2 Clasificación de un Sistema Multiagente

Existen varias características por las que se pueden clasificar los sistemas multiagentes, y ha habido en la literatura diferentes esfuerzos por hacerlo, aunque la gran mayoría de las clasificaciones coinciden en algunos criterios comunes. Por ejemplo, se puede clasificar según el tipo de coordinación (cooperar o competir), nivel de conocimiento de cada agente (individual o colectivo) y el tipo de organización que se lleve a cabo (centralizada o descentralizada).

En lo referente a la morfología se diferencian dos tipos: homogéneos y heterogéneos (como será nuestro caso). Los sistemas multiagentes homogéneos son aquellos donde los controladores de todos los agentes son idénticos, es decir, ante exactamente la misma situación todos actuarían de la misma forma. Por el contrario, los heterogéneos son aquellos que cada agente tiene su propio controlador y no tiene por qué tener comportamientos semejantes. Pueden tener controladores completamente diferentes o simplemente cambiar el código genético de la misma estructura de red neuronal.

Control

La arquitectura de control de un sistema multiagente es la encargad a de establecer cómo se gestiona el control del equipo en cuanto a la independencia de cada agente para tomar sus propias decisiones. El cómo se gestiona dicho control es crucial a la hora de que el sistema sea trasladable a la realidad. Distinguimos cuatro tipos de arquitecturas:

- Centralizadas: El grupo de agentes colabora de tal forma que ellos centralizan el control para asegurar los objetivos del grupo. Una de las inmediatas consecuencias que se enfrentas dentro de este esquema es que los agentes deben aceptar el control del agente central e, incondicionalmente, seguir sus indicaciones.

- Distribuidas o descentralizadas: los agentes realizan el control basándose en la información local, dejando de ser necesario un puesto de control central. Poseen mayor robustez pero su principal desventaja que presenta es que es mucho más complejo de diseñar el control coordinado, ya que como se dijo se basa en la información local de cada uno, y no de la información global.

- Híbridas: Combinan las dos arquitecturas anteriores. El control sigue estando basado en la información local de cada uno pero también utiliza información global

Comunicación

En el entorno, como ya se mencionó anteriormente los agentes mediante sus sensores sólo tienen acceso a la información local. Esta limitación se puede solucionar poniendo en contacto a los diferentes agentes y transmitir información unos a otros (información global relevante para la realización de la tarea global). En relación con las estructura de control, la comunicación de la información puede ser centralizada o distribuida, y como

es lógico, la complejidad de gestionar uno u otro es muy diferente. En el centralizado, el agente libre contiene toda la información global relevante y es el encargado de hacer llegar esta información al resto de agentes. En las estructuras descentralizadas, los agentes comparte la información entre ellos (todos con todos).

Coordinación

Para que un sistema multiagente realice un comportamiento colectivo, como se puede suponer, es necesario que exista una coordinación, provocando así un comportamiento más eficiente que si los comportamientos de ellos fuesen independientes. De forma aproximada, según la coordinación los sistemas se pueden desagregar en dos tipos: cooperativa y competitiva. En sistemas multiagente cooperativos los agentes persiguen el bien común (resolución de la tarea global). Por el contrario, en los sistemas competitivos los agentes persiguen objetivos en beneficio individual y en detrimento de otros robots, pudiendo dar lugar a una mejora en la elaboración del objetivo global.

3.4.3 Optimización de un Sistemas Multiagente

En este apartado de describe brevemente los tipos de algoritmos más importantes que existen para evolucionar al conjunto de robots teniendo en cuenta todas las características del anterior.

Estos algoritmos son semejantes a los de un solo agente, pero con claras diferencias que han de ser destacadas: La primera es en la codificación, ya que aparecen nuevas opciones. Se puede evolucionar una o varias poblaciones al mismo tiempo. Donde tenemos la segunda y la mayor diferencia respecto a los a los "mono-agentes" es en el estudio de la calidad. A pesar de ser una evolución conjunta (es la que realmente importa, puesto que esta indica como de bien o de mal está realizando la tarea global), los individuos han de tener una calidad individual o privada que nos indica cuanto está ayudando a la tarea global el agente evaluado. Un aumento de la calidad privada debe conllevar a un aumento de la calidad global, ya que si no jamás evolucionarían y sería imposible resolver el problema de manera eficiente. Es decir, cuando se maximiza la calidad global ha de maximizarse también la privada.

Tras el cálculo de estas calidades se procede a aplicar los diferentes operadores evolutivos (cruce y mutación como norma general), de manera semejante a los sistemas mono-agentes solo que puede aplicarse a uno o varios agentes. Por último, se creará la siguiente generación. Este bucle se repite hasta alcanzar una calidad o bien llegado a un límite de generaciones puesto por el usuario, como se puede observar en el siguiente pseudocódigo:

```
codificar en el genotipo parámetros de un agente o varios
inicializar población o varias poblaciones
loop
        for all genotipo en la población do
                formar equipo para evaluar
                evaluar equipo y asignar calidad al genotipo
        end for
        generar individuos nuevos
        reemplazar individuos de menor calidad
end loop
```

El esquema para evolucionar u optimizar un sistema utilizando un algoritmo evolutivo puede observar en el pseudocódigo anterior. La primera diferencia que hay sobre evolucionar un solo agente es en la codificación, ya que aparecen nuevas opciones. Se puede evolucionar una o varias poblaciones al mismo tiempo. Donde tenemos la segunda y la mayor diferente respecto a los a los "mono-agentes" es en el estudio de la calidad. A pesar de ser una evolución conjunta (es la que realmente importa, puesto que esta indica como de bien o de mal está realizando la tarea global), los individuos han de tener una calidad individual o privada que nos indica cuanto está ayudando a la tarea global el agente evaluado. Un aumento de la calidad privada debe conllevar a un aumento de la calidad global, ya que si no jamás evolucionarían y sería imposible resolver el problema de manera eficiente. Es decir, cuando se maximiza la calidad global ha de maximizarse también la privada. Tras el cálculo de estas calidades se procede a aplicar los diferentes operadores evolutivos (cruce y mutación como norma general), de manera semejante a los sistemas mono-agentes solo que puede aplicarse a uno o varios agentes. Po último, se creará la siguiente generación. Este bucle se repite hasta alcanzar una calidad o bien llegado a un límite de generaciones puesto por el usuario.

Clasificación de la optimización de estos sistemas según como y cuando se realice la evolución:

- Evolución *off-line*: La evolución y la simulación del entorno son procesos separados. Primero se simula el entorno durante un período de tiempo o de pasos y tras finalizar se devuelve la calidad.

- Evolución *on-line*: el proceso de evaluación ocurre de forma continua y al mismo tiempo que la evaluación. Es decir, cada cierto tiempo la evolución sucede durante la evolución.

- Evolución *on-board*: En este tipo de evoluciones del algoritmo se ejecuta de forma independiente en cada uno de los agentes. Otro termino más utilizado para llamar a este tipo de evoluciones es "Embodied Evolution" [5], ya que la evolución no depende de ningún elemento central.

- Evolución *off-board*: Es todo lo contrario al caso anterior. Los operadores del algoritmo se ejecutan de forma externa al agente.

3.4.4 Algoritmo distribuido Embodied Evolution

Como ya se dijo, en esta sección se van a resolver los problemas mediante un algoritmo distribuido Embodied Evolution. Dicho algoritmo canónico fue desarrollado por el Grupo Integrado de Ingeniería (GII) y por Pedro Jose Trueba en su tesis [1][1]. El procedimiento de EE es igual que al de un algoritmo genético corriente, y lo podemos dividir en tres procesos: Evaluación (continua), reproducción (selección y recombinación) y reemplazo.

Para entrar un poco en contexto, se explicará brevemente el trabajo desarrollado por Watson [6], donde presenta un el concepto global del paradigma y una implementación concreta, el algoritmo PGTA (*Probabilistic Gene Transfer Algorithm*). Este algoritmo está basado en el GA microbiano de Harvey, que se basa en la reescritura de porciones de individuos de baja calidad con pociones de otros com mayor calidad. En este caso, Watson lo adaptó para realizarlo de forma descentralizada y distribuida. La transferencia

se realiza de robot a robot, cuando ambos se encuentran en el escenario a una distancia (rango local). Cada uno de ellos transmite sus genes con mayor o menor frecuencia según su energía virtual, que va en relación con su rendimiento en la tarea. Cuando un robot está en posición de recibir estos genes, puede aceptarlos o no según una probabilidad inversamente proporcional a su nivel de energía. De esta manera, aquellos robots con alta calidad o energía tendrán una alta probabilidad de transmitir genes, pero una baja probabilidad de recibirlos. Entrando más en detalle, la tarea propuesta por Watson, consiste en ir hacia una luz situada en el centro del escenario. Cuando estos robots van hacia la luz su energía aumenta y cuando se alejan de ella esta disminuye. . Los controladores de los robots están reinaos por una red neuronal pero tiene ciertos comportamientos predefinidos, como por ejemplo volver a una posición aleatoria cuando han alcanzado el objetivo (luz).

A partir de este trabajo, han surgido numerosas variaciones dentro del paradigma de *Embodied Evolution*. Se distinguen 3 tipos: distribuida, encapsulada e híbrida, pero nos centraremos en el distribuido.

Para el caso de un algoritmo distribuido Embodied Evolution (en adelante dEE) los pasos a seguir son idénticos, por lo tanto, para la realización de un algoritmo canónico de dEE ese ha de seguir la misma estructura que en el caso de un algoritmo genético convencional, como se puede ver en el siguiente pseudocódigo:

for all *robot r en robots* ***do***

actuar en el entorno

calcular calidad privada

$P_{repro} \leftarrow \frac{S_{max}}{T_{max}}$ *{calcular probabilidad de reproducción}*

$P_{rep} \leftarrow \frac{1}{T_{esp}}$ *{calcular probabilidad de reemplazo}*

if *random* < P_{repro} ***then***

buscar individuos para reproducción

seleccionar individuo {Parámetro}

aplicar operadores genéticos de recombinación {Parámetro}

end if

if *tiempo de vida $* T_{mat} ***then***

if *random() < Prep or tiempo de vida $ Tmax* ***then***

reemplazar individuo por el nuevo

controlador[r] ← nuevoindividuo

end if

end if

end for

Dentro del bucle general (recorre todos los robots) se obtiene la calidad de cada uno de ellos y se comprueba si debe entrar en modo de reproducción. Si este modo se activa busca entre los individuos que cumplan los requisitos y elige al mejor de ellos (según los 3 requisitos o los que se hayan programado). Posteriormente, con el tiempo esperado de vida se calcula la probabilidad de que el robot r sea reemplazado, siempre y cuando lleve vivo un número de pasos mayor que el tiempo de madurez. Si no es

reemplazado, este sobrevivirá y permanecerá al menos un paso más en el entorno. En cambio, aquellos que si entren en el "if" el controlador pasará a estar ocupado por su descendiente.

3.4.4.1 Parámetros intrínsecos

Se han modelado los procesos mencionados anteriormente mediante probabilidades genéricas, tratando de independizar la tarea en el funcionamiento del algoritmo de la influencia del entorno. Estos modelos se ha parametrizado de una forma relativamente simple para tener una gran influencia en el comportamiento del algoritmo cuando estos parámetros sean variados. A continuación se explican todos los pasos del algoritmo canónico más en detalle.

3.4.4.2 Evaluación

Este proceso no tiene ninguna función genérica que lo modele ni tiene asociado ningún parámetro intrínseco, sino que es único para el problema que se va a resolver. Solo tiene el requerimiento de que para la evaluación en el algoritmo canónico se respete la naturaleza distribuida de este tipo de algoritmo.

3.4.4.3 Reproducción

Como se ha dicho con anterioridad, la reproducción en un EE es local y asíncrona, y surge de los encuentros de los agentes o robots en el escenario. En un algoritmo dEE existen tres parámetros que se generalizan independientemente del problema a resolver (siempre pueden ser modificados): el tamaño máximo de la venta de selección la política de selección y la probabilidad de utilizar unos operadores genéticos

.

Tamaño máximo de la venta de selección

La reproducción está regulada por una probabilidad uniforme que define la frecuencia con la que un agente entra en modo de reproducción en cada paso, es decir, si su código genético está disponible o para que se realice con él una operación de reproducción (ya sea un cruce de otro agente con él o de él con el otro, o una mutación). Esta probabilidad viene definida por dos parámetros propios de algoritmo: tiempo de vida máximo que puede alcanzar un robot en el entorno (T_{max}) y el que representa el tamaño máximo de la ventana de selección (S_{max}). Esta probabilidad se calcula mediante la siguiente fórmula:

$$P_{rep} = \frac{S_{max}}{T_{max}} \quad \textbf{(11)}$$

Cuando el robot se encuentra con el modo de reproducción activado, comprueba si entre todos los robots del alrededor hay alguno que esté también con el modo activo y además esté a su alcance. En caso de que no lo encuentre, permanecerá en modo activo hasta que lo encuentre y realizarán la operación de reproducción oportuna. El tamaño máximo de la venta de selección define el número máximo teórico de robots que puede encontrar con el modo de reproducción activado durante un tiempo de vida igual al tiempo máximo.

El tiempo de vida máximo es un valor fijo durante todo el proceso. Por lo tanto, el primer parámetro intrínseco que afecta a la reproducción.

Política de selección

En el algoritmo dEE se establecen tres criterios de aplicación que se utilizan en otros algoritmos EE: basado en genotipo, basado en un estado (por ejemplo basado en la posición espacial que ocupan) o basado en la calidad individual de los individuos:

- Criterio basado en genotipo: Se puede definir una medida de similitud de genotipo (por ejemplo la suma o norma de los códigos genéticos) y escoger individuos lo más semejantes posibles al del propio. Este criterio tiene utilidad desde el punto de vista de la especialización, ya que para mantener individuos especializados, se puede favorecer que un individuo siempre se reproduzca con otro individuo de su misma especie.

- Criterio basado en estado (posición espacial): Se escoge al robot más cercano al que está realizando la selección, que para problemas donde haya sub-tareas que estén divididas en diferentes partes del entorno puede ser útil para la especialización, puesto que los robots más cercanos estarán realizando la misma sub-tarea y serán de su misma especie.

- Criterio basado en calidad individual: Es el más común de los algoritmos genéticos convencionales, normalmente utilizado con un torneo en el que seleccionarán con mayor probabilidad los individuos de mayor calidad. Tiene gran interés para converger lo más rápido posible a soluciones de alta calidad. Cabe mencionar que un aumento de la calidad individual implica un aumento de la global, como ya se dijo.

Todos estos criterios definen el segundo parámetro intrínseco del algoritmo, que es la probabilidad que tiene un individuo de ser seleccionado. Ecuación con la que se calcula:

$$P_{eligibility} = \psi_e(\varphi_e, \gamma_e, [P_{cand}]) \quad (12)$$

donde la función ψ_e es la política de selección que depende de tres variables, que son los tres criterios explicados anteriormente: similitud de los genotipos (φ_e,), calidad del genotipo (γ_e) y su posición en el espacio (P_{cand}). Para el caso de que solo se utilice el criterio de la calidad (como será nuestro caso), la función pasará de depender de 3 variables a solo una. Devolverá un 1 para aquel individuo que tenga la mejor calidad y un 0 para los demás agentes.

Operadores Genéticos

Del mismo modo que en los algoritmos genéticos simples, el operador genético entra en acción cuando ya se ha seleccionado un individuo para su reproducción. Como ya se dijo en multitud de ocasiones, los operadores básicos de reproducción son la mutación y el cruce. El operador cruce quiere simular la reproducción biológica, puesto que necesita dos individuos (padres) para combinar sus genes. Sin embargo, para algunas codificaciones genotipo – fenotipo, el comportamiento de las descendencia está muy lejos del de los padres, provocando un abaja correlación entre la calidad de los padres y del hijo. En cambio, la mutación produce una pequeña variación en el código

genético del individuo implicando así una búsqueda local por las cercanías de la solución anterior.

Como tercer parámetro intrínseco, tenemos aquel que nos define la probabilidad de que el operador que se realice sea cruce o mutación, es decir, el balance entre ambos. Concretamente, este parámetro indica la probabilidad de que suceda la mutación (P_{ls}) y consecuentemente la de cruce, que sería la . Con este parámetro se puede variar lo que se quiera de explotación y exploración. A mayor valor de P_{ls} mayor será la explotación y menor la exploración.

En el pseudocódigo que se presenta a continuación ayuda a entender el proceso de recombinación y como y cuando se utilizan las probabilidades mencionadas anteriormente:

```
preferencia ← 0,5
porcionMutacion ← 0,15
porcionCruce  ← 0,85
if random < preferencia then
        α̃ ← genotipoEmisor
    else
        α̃ ← genotipoReceptor
    end if
    if random < P_ls then
        for i = 0 to length(α) do
            if random < porcionMutacion then
                genotipoMutado ←  α_i + random * pasoMutacion
            end if
        end for
        return genotipoMutado
    else
        for i = 0 to length(α) do
            if random < porcionCruce then
                genotipoCruzado ← genotipoEmisor
            else
                genotipoCruzado ← genotipoReceptor
            end if
        end for
        return genotipoCruzado
end if
```

3.4.4.3.1 Reemplazo

Cada robot tiene una probabilidad individual de ser reemplazado según el tiempo de vida esperado, que representa un valor medio del tiempo de vida para cada individuo según la calidad que tenga. Lógicamente, este tiempo ha de localizarse entre el tiempo de madure o mínimo (T_{mat}) y el tiempo máximo (T_{max}). El tiempo de madurez es aquel que se considera suficiente para que un individuo sea evaluado. El tiempo de vida máxima, en cambio, indica el mayor tiempo que se puede alcanzar en el proceso, para aquellos que tienen la calidad óptima. La resta entre estos dos valores se denomina L.

La ecuación con la que se calcula el tiempo de vida esperado depende de estos dos valores y de aquellos que se pueden ver en la siguiente figura:

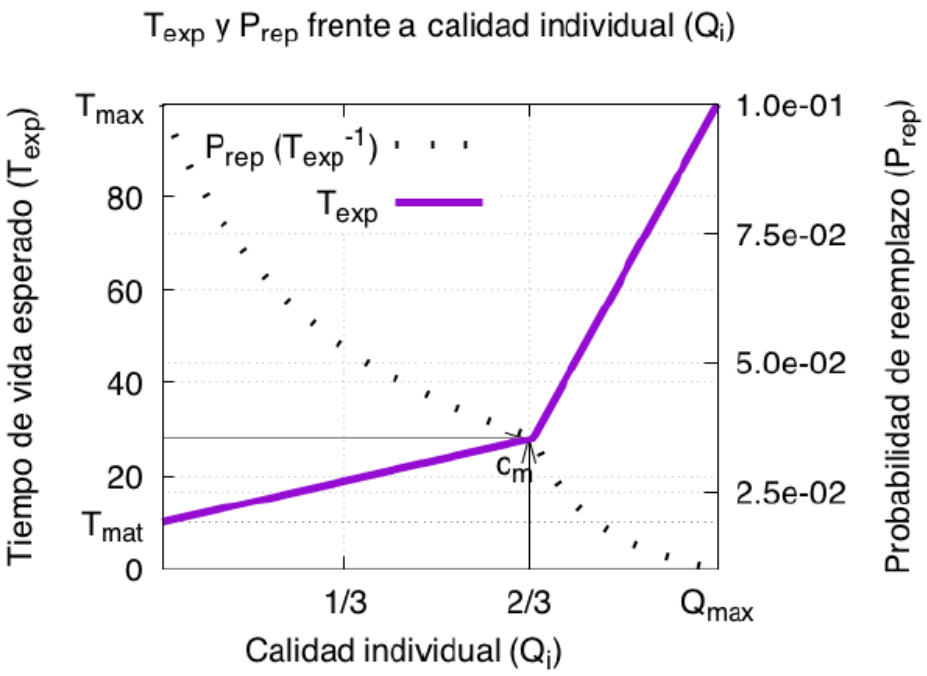


Figura 29. Representación de la función utilizada para el cálculo del tiempo esperado

Cómo se puede observar en la Figura 29, la función se ha diferenciado en dos partes lineales determinadas por el valor de Q_r. La primera está entre aquellos de calidad 0 y los de calidad $0.66Q_{max}$. A los individuos que están situados en esta parte de la gráfica se les considera mediocres y por ello el crecimiento del tiempo según aumenta Q_r es bastante pequeña, o dicho de otra forma, tiene un bajo valor de pendiente. Po otro lado, la segunda zona de dicha función (entre $0.66Q_{max}$ y Q_{max}) sí que tiene una gran pendiente, favoreciendo así que aquellos que de verdad tengan una buena calidad sean aquellos que sigan vivos.

De la Figura 29 podemos extraer dos parámetros intrínsecos. Q_i y Q_{max} son parámetros de evolución y T_{max} es un parámetro fijo; Como consecuencia obtenemos que estos parámetros intrínsecos son c_m y T_{mat}. La Figura 29 corresponde a una representación de esta función con los parámetros fijados en $T_{mat} = 10, c_m = 0.2$ y $T_{max} = 100$. El parámetro c_m es el coeficiente de mediocridad, con él podemos variar las pendientes de ambas zonas de la función. Un bajo valor de este implica que se penaliza aquellos individuos que tenga una baja calidad, y por tanto menor será su tiempo de vida esperado y mayor su probabilidad de ser reemplazado en cada paso de tiempo. Si se quiere penalizar menos a aquellos con menos calidad se debe poner un alto valor de c_m (próximo o igual a 1, puesto que como es lógico su valor está entre 0 y 1). No obstante,

a pesar de descender el número de reemplazos siempre se garantiza un número mínimo de ellos, aun cuando este vale uno. Por tanto, un bajo valor de c_m implica una mayor explotación y uno bajo una mayor exploración del espacio de búsqueda.

$$P_{rep} = \frac{1}{T_{esp}} \quad (13)$$

$$L = T_{max} - T_{mat} \quad (14)$$

$$Q_r = \frac{Q_i}{Q_{max}} \quad (15)$$

$$T_{esp} = c_m Q_r \frac{3Q_{max}}{2} L; \;\; if \; Q_r \leq 0.66 Q_{max} \quad (16)$$

$$T_{esp} = \left(c_m + \left(Q_r - \frac{2Q_{max}}{3} \right)(3 - 3c_m) \right) L; \;\; if \;\; Q_r > 0.66 Q_{max} \quad (17)$$

Como resumen, se muestra a continuación todos los parámetros intrínsecos que definen nuestro algoritmo canónico dEE:

1. Tiempo de vida máximo (T_{max})
2. Tamaño máximo de la ventana de selección (S_{max})
3. Probabilidad de la búsqueda local, es decir, de mutación (P_{ls})
4. Tiempo de madurez o mínimo de vida (T_{mat})
5. Coeficiente de mediocridad (c_m)
6. Política de selección (ψ_e)

3.4.5 Código desarrollado

Para la resolución de ambos problemas se va desarrollar un algoritmo basado en uno distribuido Embodied Evolution. Se empezará por decir que valores hemos establecido para los diferentes parámetros intrínsecos:

- T_{mat} → 40
- T_{max} → 10 x T_{mat}
- S_{max}→ 10
- P_{ls} → 1
- c_m → 0.3
- Política de selección → Calidad de los individuos.

Con estos parámetros el algoritmo queda prácticamente definido con todo lo explicado anteriormente. La única peculiaridad es, como ya se dijo el parámetro de la probabilidad de mutación, es que no dispone de cruce. En la mutación se cambian la cuarta parte de los genes sumándoles un número aleatorio entre -0.1 y 0.1.

3.4.6 Implementación del algoritmo

En esta sección se va a tratar de resolver dos problemas mediante técnicas de *shepperding* y optimizado con un algoritmo distribuido *Embodied Evolution*:

- Pastoreo sin obstáculos
- Evacuación de una habitación con zonas de peligro potencial

Los códigos desarrolladlo en ambos problemas son bastante similares. En la figura 30 se muestra el diagrama de clases para el primer problema.

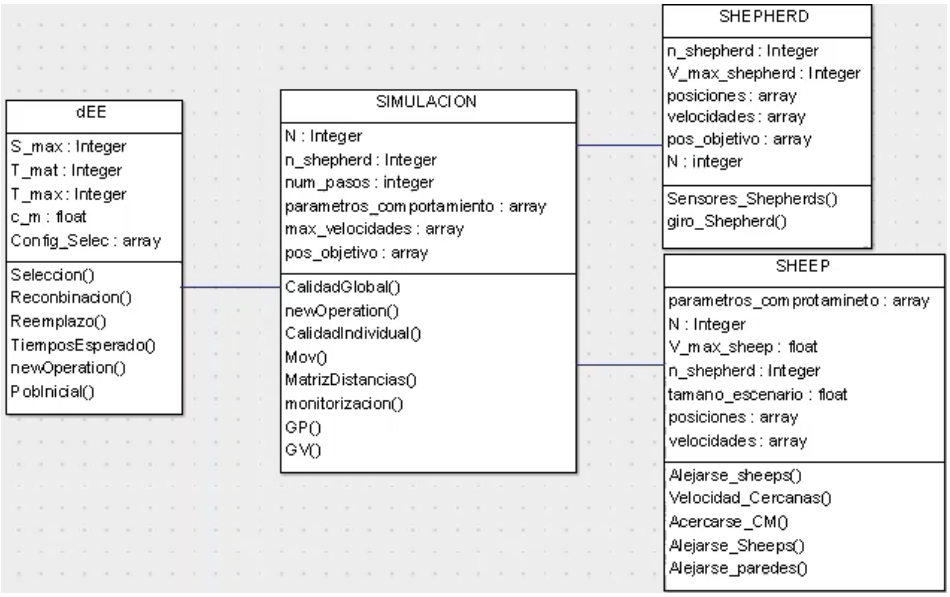


Figura 30. Diagrma de clases para el problema sin obstáculos

3.4.6.1 Problema continuo sin obstáculos

En el actual apartado se procede a resolver un problema bastante semejante al resuelto mediante NEAT. En este caso dispone de varios agentes, concretamente 10, que tratará de guiar a otros agentes no inteligentes a una posición objetivo. Para ello, utilizaremos el optimizador de sistemas multiagentes explicado en el apartado anterior (dEE).

Para la utilización de dicho algoritmo es necesario definir una calidad individual en cada instante de tiempo, calculada mediante la siguiente ecuación:

$$Q_i = \|P_1 - P_{obj}\| - \|P_2 - P_{obj}\| \quad \textbf{(18)}$$

donde Q_i es la calidad individual del perro i; P_1 es la posición que ocuparía la oveja en el siguiente instante de tiempo si solo fuese influenciado por el perro i; P_{obj} es la posición objetivo; y P_2 es la posición que ocuparía si se tiene en cuenta todas las fuerzas menos la producida por el perro i. Como podemos observar, se le puntuará positivamente a aquel que las esté acercando hacia el objetivo y negativamente a aquellos que las estén alejando. Como es lógico y aumento de esta calidad individual conlleva a un aumento en la global.

Resultados

Con todo el problema o el código desarrollado descrito procedemos a evaluar los resultados obtenidos tras una simulación de 15,000 pasos. En primer lugar hay que destacar la gran velocidad de estos algoritmos *on-line* frente a los *off-line.* Para que sirva de referencia, en este problema hemos necesitado, como ya dije 15,000 pasos, mientras que en el NEAT se necesitaban 10,000 para tan solo una generación, es decir, 400,000 para una evolución de 40 generaciones (casi 30 veces más en lo que a cálculos se refiere).

Centrándonos ya en los resultados obtenidos, vemos en la Figura 31 que en un principio no se una evolución clara pero como a partir de aproximadamente el paso 6000 comienza a ascender la calidad con una pendiente bastante pronunciada. Dicha ascensión termina aproximadamente sobre el paso 11000, donde incluso desciende ligeramente (algo habitual en este tipo de algoritmos).

Figura 31: Evolución de la calidad en el problema multiagente sin obstáculos

En la siguiente figura podemos observar la calidad global que tienen los 10 perros durante 5000 pasos sin variar su código genético, es decir, con su código resultante después de los 15000 pasos de evolución. La calidad varía entre los valores de 0.45 y 0.50. Es un valor que está muy cercano al alcanzado por un solo agente evolucionado con la librería NEAT. En un lado de la balanza nos encontramos que al ser 10 agentes en vez de 1 y pueden llegar a un comportamiento (global) mucho más complejo, pero en el otro lado tenemos que se ha utilizado un algoritmo propio y no uno de una librería y que a nivel individual se requiere un comportamiento mucho más completo y elaborado.

Figura 32. Calidad en 5000 pasos con agentes ya evolucionados

La pequeña variabilidad que se puede observar en la calidad de los agentes, se debe a la aleatoriedad a la hora de situar tanto a los perros como a las ovejas una vez llegan al objetivo. A continuación se muestra en la figura 32 la variación de los genomas a lo largo del tiempo (eje z). Como cada genoma viene definido por 6 genes, es imposible

representarlo, por lo que se representa en el eje x la suma de los 3 primeros genes, y en el eje y la de los 3 últimos. Se puede observar en el espacio de búsqueda explorado no es demasiado amplio debido a la inexistencia del cruce. No obstante, se ha alcanzado un comportamiento satisfactorio.

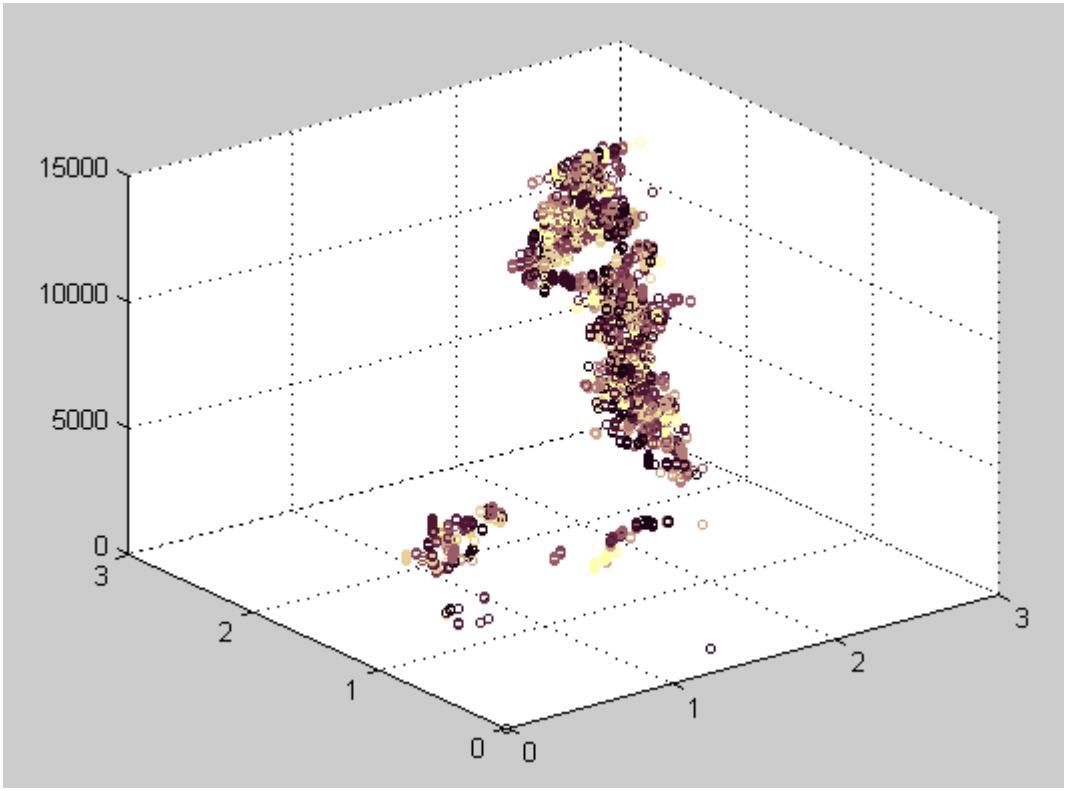
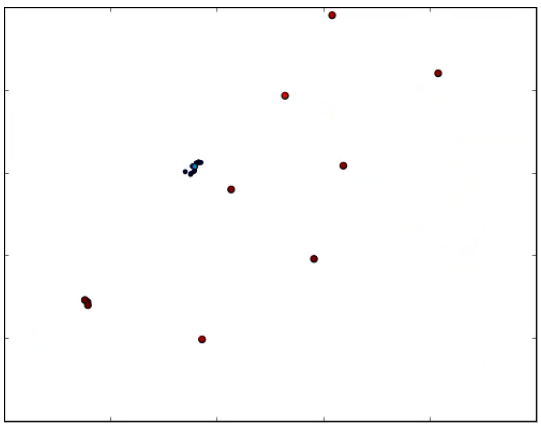
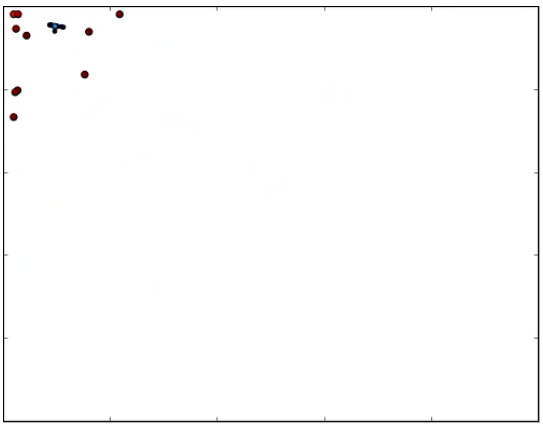


Figura 33: Variación de los genes con los número de pasos

Por último, se muestra una serie de fotogramas donde se puede apreciar el comportamiento que han desarrollado:





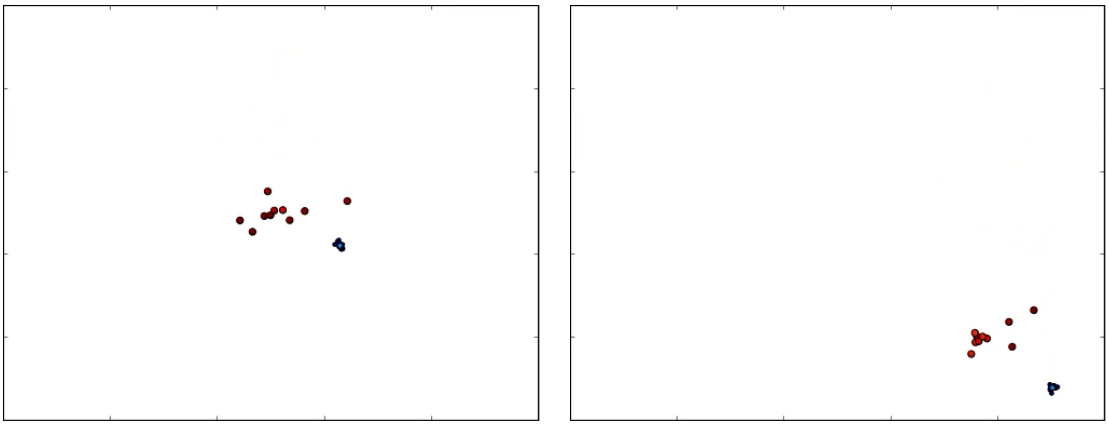


Figura 34: Cuatro fotogramas del comportamiento de los 10 agentes (sin obstáculos)

3.4.6.2 Problema continuo con obstáculos

En este problema, al contrario que todos los demás, ya no se tratará de perros y ovejas sino de drones que tratan de evacuar a un grupo de personas de una habitación con unos puntos de peligro potencial, donde las personas no podrán entrar. Estos puntos de peligro se considera que las personas no pueden verlos y por tanto no los esquivan por sí solos. Para este caso, el cálculo de la calidad global sí que se ve variado. Se parte de la misma ecuación, donde mirábamos cuanto avanzaba el centro de masas del grupo hacia el objetivo pero se le multiplica por un factor:

$$fitness = \left(1 - \frac{2\,N_Q}{N}\right) Q_{CM};\ if\ 2N_Q < N$$
$$fitness = 0; if\ 2N_Q \geq N \qquad \textbf{(19)}$$

Donde $fitness$ vuelve a ser la calidad global; N_Q la cantidad de gente que se ha quemado (entrado en la zona de peligro potencial) mientras se le lleva al objetivo; N es la cantidad de personas total a evacuar (15); y Q_{CM} es la calidad correspondiente al avance del centro de masas (ecuación 18).

En lo que respecta a la calidad individual esta vez está compuesta por dos sumandos: El primero es exactamente igual a la del anterior problema, es decir, con el que sabemos si están ayudando o no a que vayan hacia el objetivo. El segundo sumando es semejante al primero solo que respecto a los puntos de peligro potencial. Como es lógico, el segundo llevará un signo menos para penalizar cuando lo acerque y premiar cuando lo aleje. Para ser más justo, esta calidad se multiplica por un factor según a la distancia que estén las personas de estos puntos, premiando más aquellos que los alejan cuando de verdad hay peligro de entrar en dicha zona.

Resolución

En este último apartado de la sección vamos a analizar los resultados obtenidos en el problema de evacuación con sólo dos obstáculos no visibles para las personas. Como se puede observar en la Figura 34, estamos ante un problema donde la calidad global

es mucho más fluctuante que en los casos anteriores. Estos se debe a tres razones: La aleatoriedad a la hora de reinicializar las posiciones de drones y personas (igual que en los anteriores); la variabilidad al colocar las posiciones de los peligros potenciales; y a la dependencia de la calidad a las personas que entran en las zonas peligrosas (se pueden observar líneas verticales en el gráfico que llevan a 0 la calidad porque se han quemado más de la mitad de las personas durante el trayecto. De hecho, si se observan los gráficos de las Figuras 35 y 36 vemos como las dichas líneas verticales coinciden con las del otro.

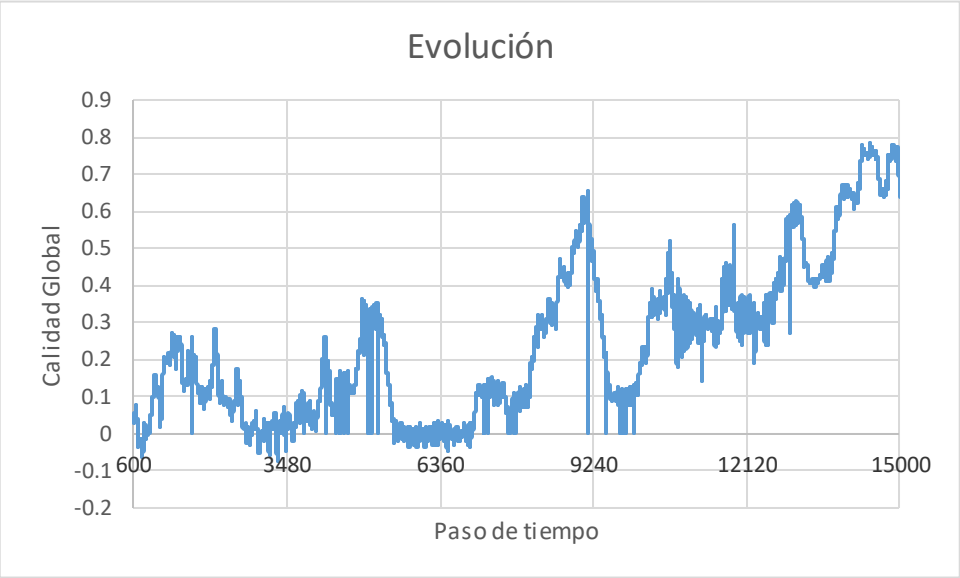


Figura 35: Evolución de la calidad global en el problema con obstáculos

A partir del paso número 10,000 se ve un aumento claro en la calidad global. Lógicamente esto implica un descenso en el número de personas que se adentran en las zonas peligrosas.

Por último, en lo que concierne a la resolución del problema, se puede apreciar un aumento claro de la pendiente de la gráfica N_{obj} frente al tiempo a partir del paso 10,000 y todavía más evidente a partir del 12,000.

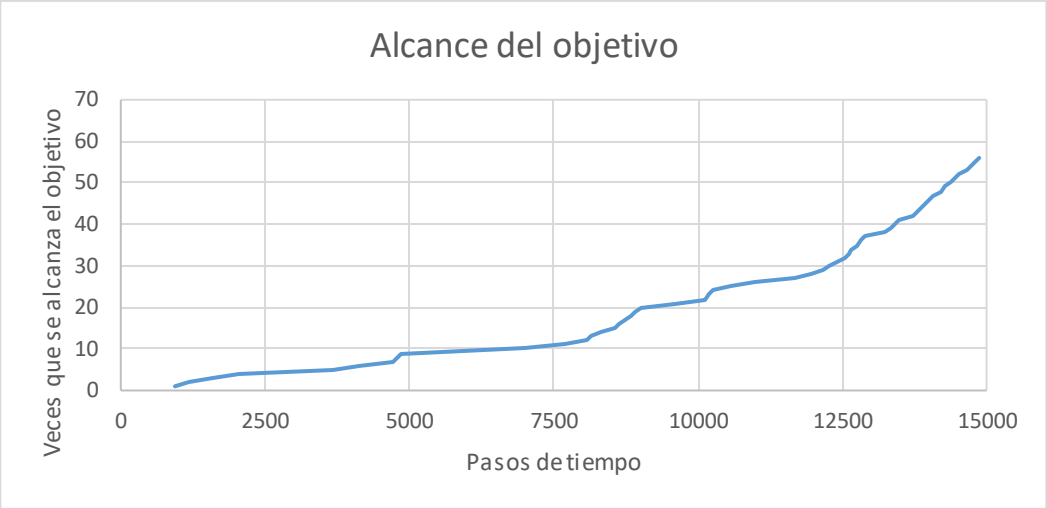


Figura 36: Número de veces que se alcanza el objetivo frente al tiempo (pasos)

Como se puede ver en la Figura 37, los controladores obtenidos al final de la evolución nos ofrecen un rendimiento más que satisfactorio. En 5,000 pasos tan sólo han entrado una vez en zonas de peligro y han mantenido una calidad bastante alta (0.7 aproximadamente) durante la mayor parte de la simulación

Figura 37: Representación de la calidad de los agentes ya evolucionados mediante dEE

Cabe destacar la velocidad de estos algoritmos para encontrar una solución satisfactoria, a pesar de ser un problema que comienza a tener cierta complejidad, como es en este caso. No obstante, esta velocidad se ve muy afectada por como sea la población inicial, puesto que en ciertas ocasiones no es capaz de encontrar una solución suficientemente válida debido a la gran explotación y a la tan poca exploración que posee.

En la figura 38 podemos observar como los diferentes controladores convergen de forma muy rápida, abarcando muy poco espacio de búsqueda.

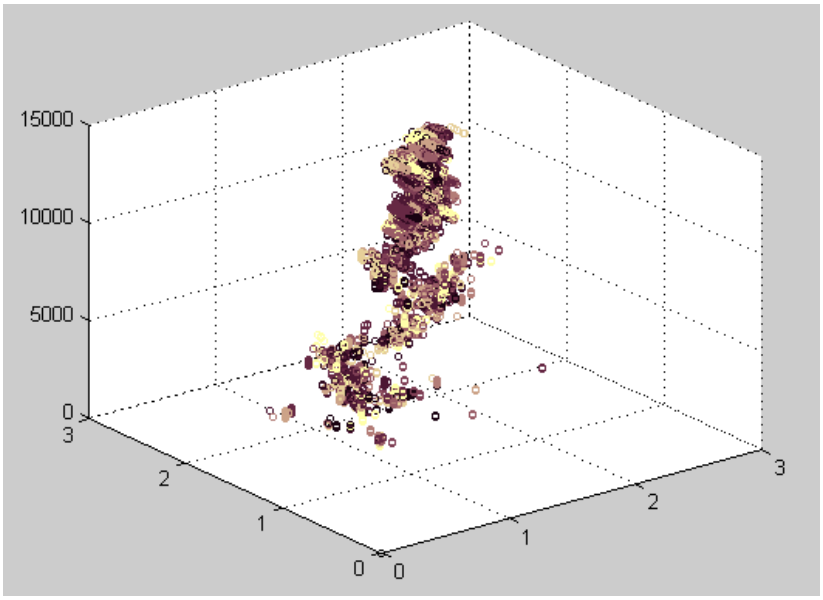


Figura 38: Evolución del genoma a lo largo del tiempo

Por último, en la figura 39 podemos apreciar como los agentes han desarropado un comportamiento que les permite evitar que las personas entren en las zonas de peligro (grises).

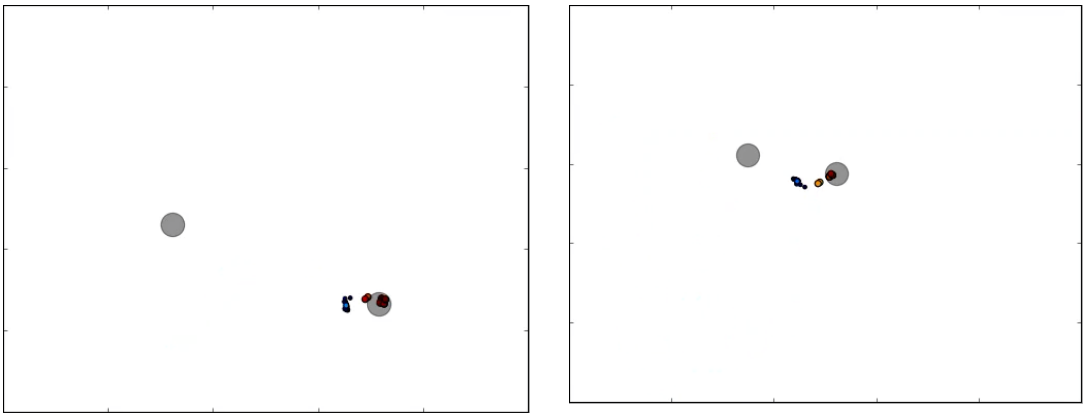


Figura 39: Dos fotogramas de los drones evitando que entren en las zonas grises (peligrosas)

4 APLICACIÓN EN UN ENTORNO REAL

En esta última sección se ha trasladado el algoritmo utilizado para optimización en problemas de shepperding colectivos a un entorno realista. En particular hemos decidido abordar un problema de evacuación en situaciones de emergencia. Para ello primero hemos estudiado trabajos que modelan el comportamiento en este tipo de situaciones y hemos generado un entorno de simulación realista a partir de los resultados del artículo de Tang, 2016 [4].

4.1 Problemas de evacuación

Cuando en situaciones de aglomeraciones ocurre una emergencia, tales como un incendio o una explosión, puede desembocar en un gran número de heridos y muertes. Estudios e investigaciones dicen han mostrado que la presencia de agentes bien entrenados presentes en el desastre, como es el caso de los bomberos, mejora significativamente la ratio de supervivencia en estas situaciones[7]. Sin embargo, la disponibilidad de información que posee estos agentes puede hacer que aún en este caso el comportamiento guiado no sea el óptimo y por ejemplo, determinados grupos de personas sean guiadas a las salidas más cercanas aun estando estas congestionadas, ralentizando así la evacuación. Para conseguir mejorar la efectividad de la evacuación se requiere de una cuidadosa organización de la evacuación y de aumentar la información disponible. Con el desarrollo reciente de los sensores, es posible instalar cámaras u otros sistemas de vigilancia en las salidas, si no existen ya. Con la información dada por estos sensores, la inteligencia artificial puede mejorar la eficiencia de los procesos de evacuación asistida que obtendríamos con la señalización o agentes tradicionales.

En este trabajo se ha estudiado la evacuación guiada por agentes artificiales (drones por ejemplo). Este ofrece varias ventajas, en primer lugar, los agentes pueden guiar hacer un guiado más rápido, puesto que llegan antes que los agentes tradicionales porque son desplegados dentro del propio edificio. En segundo lugar, con la información obtenida de los sensores del entorno a tiempo real, el robot es capaz de obtener la vía de escape más rápida, teniendo en cuenta la distancia que hay que recorrer y lo congestionada que esté cada una de las salidas y conocen en detalle el entorno. En tercer y último lugar, un robot no ve afectado su comportamiento ante una situación de pánico, algo que si podría suceder en el personal que se utiliza para esta tarea. Adicionalmente, los agentes de evacuación reales ponen en riesgo su vida en cada una de estas situaciones y por tanto el uso de agentes artificiales minimiza los daños personales.

Para obtener un comportamiento fiable lo primer que hemos hecho es estudiar y programar un modelo de comportamiento de multitudes en situaciones de emergencia.

4.1 Modelo del comportamiento de las personas

Una vez descritos los problemas de evacuación y las ventajas de usar un robot frente a otros métodos, se procede a definir el comportamiento de las personas.

A pesar de las variaciones, el comportamiento de las personas en este problema y en los anteriores comparte ciertos patrones, como por ejemplo alejarse de las paredes o de los drones. Para el cálculo de las diferentes fuerzas con las que posteriormente se obtiene la variación de la velocidad actual nos basaremos en las ecuaciones propuestas en [8].

4.1.1 Ecuaciones por las que se rige el movimiento

En esta subsección se aplica un modelo de fuerza colectiva o social para definir el comportamiento de la multitud al nivel individuo – individuo, porque la gente no siempre puede ir a la velocidad que desearía dentro de una multitud. Se modelan 3 fuerzas: fuerza propia (la que le lleva hacia la puerta que él desea ir), persona – persona fuerza (se ve influenciado por el resto de evacuados); persona – paredes interacción; y fuerza de repulsión persona - robot. Se considera una multitud de N personas y M robots, y la fuerza total f_i para el individuo i será:

$$f_i = f_i^S + \sum_{j=1;j\neq i}^{N} f_{ij}^I + \sum_{k=1}^{W} f_{ik}^W + \sum_{m=1}^{M} f_{im}^M \qquad (20)$$

donde f_i^S es la fuerza propia de evacuado "i", f_{ij}^I es la interacción de repulsión y la tendencia a la velocidad del evacuado "j"; f_{ik}^W la fuerza de repulsión del muro "k"; y f_{im}^M la fuerza de repulsión que ejerce el robot sobre la persona.

Se considera que las personas no ocupan ningún espacio pero sí se repelen unos a otros en caso de estar muy cerca para simularlo. Se modela una sencilla función para calcular el valor de esta fuerza

$$f_{ij}^{I_1} = F^I \left(1 - \frac{d_{ij}}{d_{umb}^I}\right)^{t^I} \widehat{V_{ij}} \qquad (21)$$

donde F^I es el valor máximo que puede alcanzar la fuera individuo – individuo; d_{ij} la distancia entre ambas personas; d_{umb}^I la distancia umbral a partir de la cual la persona "j" deja de tener efecto sobre la "i" (en esta ecuación); t^I es un parámetro para variar la forma de la recta / curva; y V_{ij} es el vector que va desde el individuo "j" hasta el "i" (normalizado). Para el caso de la fuerza de repulsión que ejerce el robot sobre la persona es exactamente igual. Lógicamente, cada uno tendrá diferentes parámetros.

Además de la fuerza de repulsión, para mantener la unidad del grupo se modela una fuerza con dirección y sentido a las n_{cer} personas más cercanas. Dicha fuerza, también está incluida dentro de las iteraciones individuo – individuo:

$$f_{ij}^{I_2} = K\ \widehat{V_m};\quad if\ d < d\_cer$$
$$f_{ij}^{I_2} = K - K\ (d_{umb} - d_{cer})\,(d_{ij} - d_{cer})\,\widehat{V_m};\quad if\ d_{cer} \le d_{ij} < d_{umb} \qquad \textbf{(22)}$$
$$K = \frac{F_{cer}^{I}}{N}$$

donde $f_{ij}^{I_2}$ es la fuerza que le obliga a llevar una velocidad parecida a la multitud; $\widehat{V_m}$ es una velocidad media ponderada de aquellas personas que estén a una distancia menor a d_{umb} (distancia umbral a partir de la cual deja de tener efecto el evacuado j sobre el i; d_{cer} es la distancia a partir de la cual la influencia deja de ser máxima y empieza a descender progresivamente; F_{cer}^{I} divido entre N (número total de personas) es la máxima magnitud que puede alcanzar.

Para el modelado de la fuerza de repulsión de las paredes se procede de una manera similar:

$$f_{ik}^{W} = F^{W}\left(1 - \frac{d_{ik}}{d_{umb}^{W}}\right)^{t^{W}}\ \widehat{V_{ik}} \qquad \textbf{(23)}$$

F^{W} es el máximo valor alcanzable de esta fuerza, es decir, lo que quieres que influya sobre el cambio de la velocidad de la persona; d_{ik} distancia entre la persona y la pared (en algunos casos sólo se considera la pared más cercana); d_{umb}^{W} como en el caso anterior, es la distancia en la que la pared deja de tener efecto, t^{W} parámetro que permite variar la forma de la curva; y $\widehat{V_{ik}}$ es el vector cuya dirección es la perpendicular al plano de la pared, con sentido que apunta hacia la persona y unitario

4.1.2 Ruta seleccionada por la persona

Cuando las personas se encuentran en una emergencia en un lugar, ellos suelen disponer de varias rutas de escape, de las cuales escogerá una u otra según los conocimientos que tengan sobre en el entorno y el pánico que tengan en ese momento. Se podría considerar dos clases de evacuados: aquellos que tienen conocimientos y aquellos que no. Lógicamente, tienen más probabilidades de escoger la ruta más corta aquellos que sí tienen conocimiento del entorno. No obstante, nuestro interés está en la actuación de los robots cuando sobre aquellos grupos que escogen las rutas incorrectas por lo tanto dentro de este trabajo se considerará que ningún grupo se dirige de manera autónoma a la ruta óptima de evacuación.

4.2 Optimización mediante algoritmos tipo shepherding

Un problema de evacuación y un problema del tipo *shepherding* son muy semejantes puesto que ambos consisten en el guiado de un grupo de agentes "no inteligentes" mediante unos agentes inteligentes. Por tanto, el problema puede ser resuelto de manera similar a los realizados hasta el momento.

En lo relacionado a la optimización, el algoritmo evolutivo que se va a utilizar es el algoritmo distribuido Embodied Evolution propio utilizado en apartados anteriores, ya que se trata de un problema multiagente.

4.3 Descripción del problema

Igual que en los demás problemas resueltos en este trabajo, a excepción del de las 8 reinas, estamos ante un problema continuo y dinámico. A rasgos generales es un problema bastante similar al anterior, pero con dos detalles que complican el problema notablemente. En primer lugar, vamos a tener dos tipos de zonas de peligro potencial: Aquellos que las personas pueden ver y escapan de ellos de forma autónoma, sin necesidad de ser guiados por los drones, y las zonas de peligros potenciales que las personas no son capaces de apreciar. El segundo cambio consiste en que el grupo de personas tiene una ruta que quiere llevar a cabo, diferente a la que el robot desea que realice. Es decir, en este caso tendremos dos puertas en dEE la habitación, una en la parte inferior derecha (objetivo de las personas) y otra en la esquina superior derecha (objetivo de los drones). Se considera que la puerta a la que desean ir las personas está colapsada o inaccesible, información que tienen los drones pero no las *personas.*

Por tanto, se simula una habitación de un edifico donde tenemos presentes dos zonas de peligro que son avistados por las personas y dos zonas de peligro potencial que las personas no se han dado cuenta de su existencia (como podría ser un suelo a punto de derrumbarse)

Para el cálculo tanto de la calidad individual como de la calidad global no hay ningún cambio respecto al último problema resuelto. No obstante, cabe destacar que a pesar de no penalizar concretamente cuando las ovejas se van por la puerta incorrecta sí que incurre en una disminución de la calidad global.

4.4 Resultados

Una vez definido todo el problema, ya estamos en disposición de poner a evolucionar los 10 agentes en el entorno y analizar los resultados obtenidos. Igual que en problemas anteriores la gráfica de la figura 40 oscila bastante debido a la gran variabilidad cada vez que lleva las personas a la puerta objetivo. A pesar de tener 4 zonas por las que no deben pasar las personas que ralentizan el avance se alcanza calidades del orden del 0,7

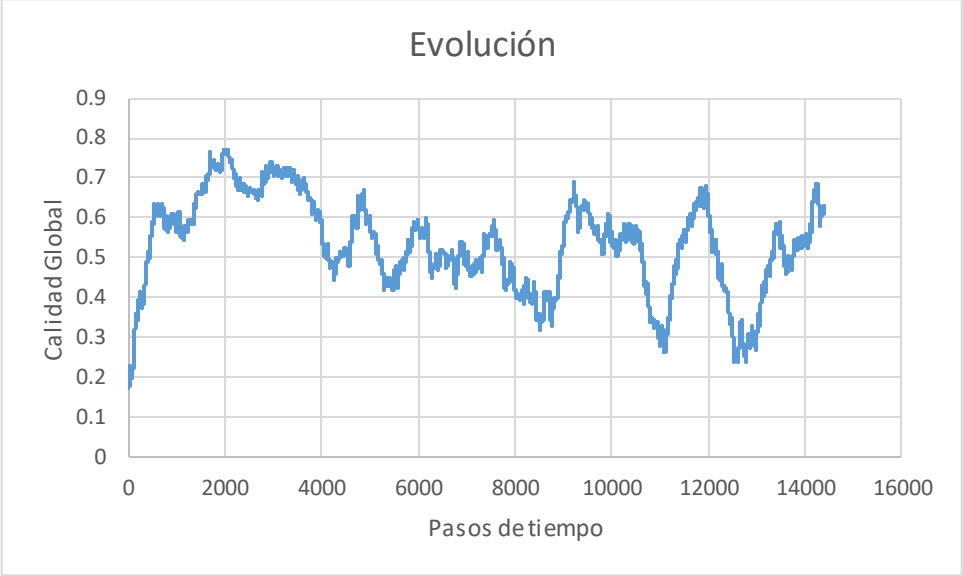


Figura 40. Evolución del problema de evacuación

En la figura 41 se muestra cómo se comportan los agentes una vez finalizado su evolución. Como se puede observar, el nivel de calidad global que alcanzan es bastante satisfactorio.

Figura 41. Calidad Global de la población evolucionada

Igual que en todos los casos anteriores, debido al algoritmo que se ha desarropado, las soluciones convergen muy rápidamente, esto nos permite encontrar rápidamente una solución satisfactoria pero por otro lado nos abre la puerta a modificar el algoritmo para realizar una exploración más exhaustiva y mejorar el resultado. No obstante, en los resultados particulares mostrados se ha encontrado una solución al problema que creemos que es difícilmente mejorable.

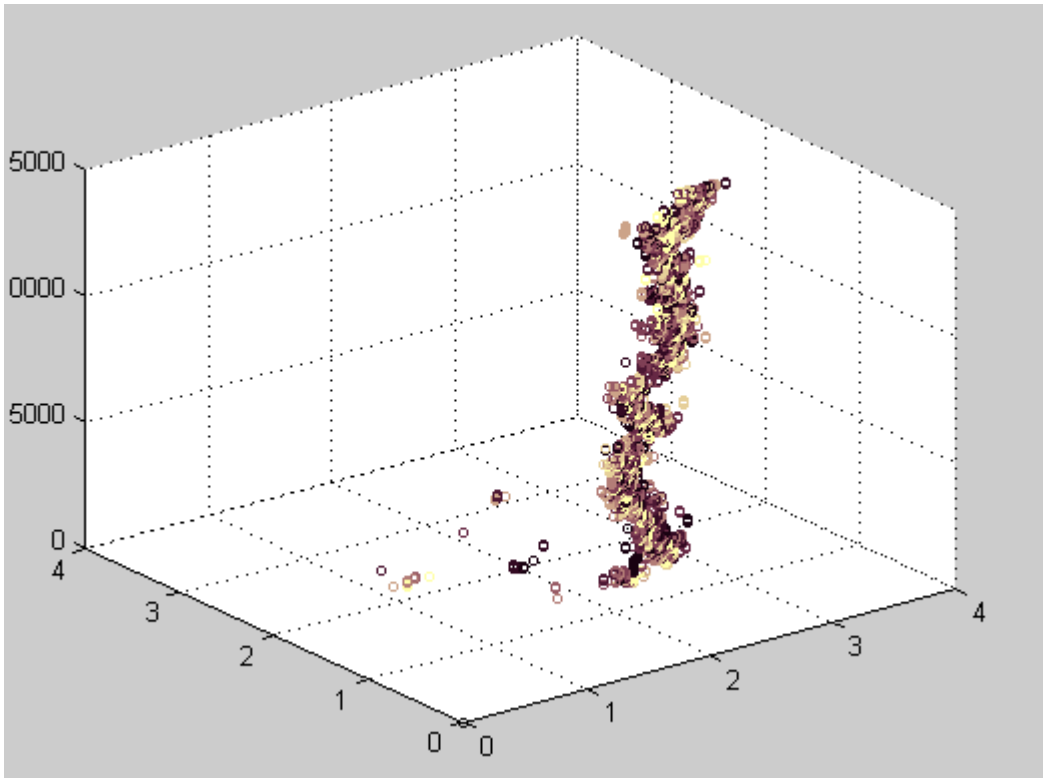


Figura 42. Variación de los genomas con el tiempo

En último lugar, se puede apreciar como los drones han aprendido a posicionarse formando un semicírculo detrás de ellos, llevándolos así hacia el objetivo al mismo que tiempo que se aseguran que las permanezcan todos juntos

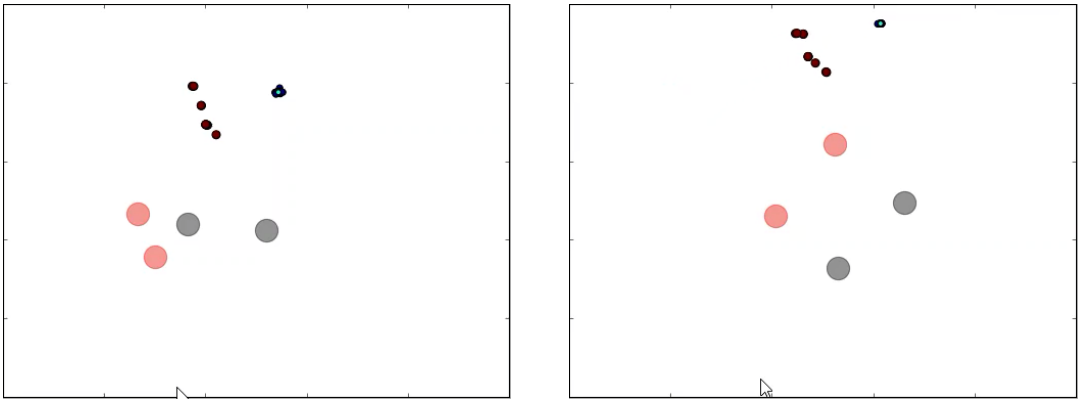


Figura 43. Comportamiento de los agentes

5 CONCLUSIONES

La primera conclusión a la que podemos llegar es que mirando los objetivos descritos en la segunda sección del trabajo hemos cumplido todos y cada uno de ellos. Se ha desarrollado por completo un algoritmo genético simple aplicado a un problema para familiarizarnos y adentrarnos en el mundo de la inteligencia artificial y los algoritmos genéticos. Además, gracias al análisis de sensibilidad hemos apreciado que un algoritmo genético, a pesar de gran polivalencia, cada problema neecesita su propia configuración de parámetros para obtener un buen resultado

Por otro lado, se ha aprendido a utilizar una de las mayores herramientas de este campo, como son las redes neuronales artificiales para el problema tipo *shepherding.* Además, fue necesario el desarrollo de un simulador en dos dimensiones en Python para poder observar y analizar las soluciones obtenidas tras la evolución.

A continuación, se programa un algoritmo distribuido Embodied Evolution basado en los realizado por el Grupo Integrado de Ingeniería de la UDC y el descrito por Pedro Trueba en su tesis [1] y se comprueba que funcione adecuadamente mediante un problema semejante al anterior. Una vez validado se procede a su aplicación en un problema real sobre la evacuación de una habitación en caso de una emergencia y comprobación de que tenga potencia suficiente para resolver problemas tan complejos como este. Además, cabe destacar que nos permite resolver un problema sin tener conocimientos previos de como se ha de solucionar.

6 BIBLIOGRAFÍA

[1] P. J. T. Martínez, "Optimización en sistemas multi-robot mediante Embodied Evolution," 2017.

[2] K. C. Gupta, "Neural Network Structures," *Neural Networks RF Microw. Des.*, pp. 61–103, 2000.

[3] J. M. Lien, S. Rodríguez, J. P. Malric, y N. M. Amato, "Shepherding behaviors with multiple shepherds," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2005, pp. 3402–3407, 2005.

[4] K. O. Stanley and R. Miikkulainen, "Efficient Evolution of Neural Network Topologies," *Evol. Comput. 2002. CEC'02. Proc. 2002 Congr. Evol. Comput.*, no. figure 1, pp. 1757–1762, 2002.

[5] N. Bredeche, E. Haasdijk, y A. Prieto, "Embodied Evolution for Collective Robotics : A Review," no. x, pp. 1–21.

[6] R. A. Watson, S. G. Ficiei, y J. B. Pollack, "Embodied evolution: Embodying an evolutionary algorithm in a population of robots," *Proc. 1999 Congr. Evol. Comput. CEC 1999*, vol. 1, pp. 335–342, 1999.

[7] A. Chaturvedi, A. Mellema, S. Filatyev, y J. Gore, "DDDAS for fire and agent evacuation modeling of the Rhode Island nightclub fire," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3993 LNCS-III, pp. 433–439, 2006.

[8] B. Tang, C. Jiang, H. He, y Y. Guo, "Human Mobility Modeling for Robot-Assisted Evacuation in Complex Indoor Environments," *IEEE Trans. Human-Machine Syst.*, vol. 46, no. 5, pp. 694–707, 2016.