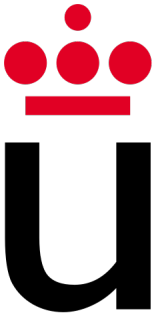

Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN MATEMÁTICAS

Curso Académico 2021/2022

Trabajo Fin de Grado

SIMULACIÓN DE ESTRATEGIAS DE CAZA DE ANIMALES GREGARIOS A PARTIR DE METAHEURÍSTICAS

Autor: Daniel Carmona Pedrajas

Directora: Clara Simón de Blas

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se explica cuál es la idea general del proyecto y su origen. Este proyecto detalla la implementación y análisis de metaheurísticas que se han empleado en un sistema de simulación, tema sobre el que versa el TFG de software con título *DISEÑO Y DESARROLLO DE UNA APLICACIÓN PARA LA SIMULACIÓN DE ECOSISTEMAS* por lo que se referenciará dicho documento a lo largo de esta memoria para facilitar la comprensión del problema propuesto.

1.1 Idea general

El principal objetivo de este TFG es la implementación de metaheurísticas para el modelado de caza de animales gregarios, de tal manera que cada animal simulado use esta herramienta para maximizar su utilidad dentro de la manada.

1.2 Motivación

Siempre he estado interesado en los animales, recuerdo ir de pequeño a casa de mi abuela y pedirla que me pusiera documentales en vez de dibujos animados. Con el paso del tiempo empecé a percibir una cierta belleza en lo cruda que es la naturaleza, no hay ni buenos ni malos, simplemente seres vivos luchando por su supervivencia.
Cuando cursé Fundamentos Biológicos en el primer año de carrera realizamos una práctica que simulaba la competición por los recursos entre especies usando una hoja de cálculo.

Fue en ese momento que reparé en la relación evidente que tienen las matemáticas con cualquier otro campo del conocimiento y en el potencial que hay en la fusión de las distintas disciplinas. Siempre se han hecho grandes avances cuando esto ha ocurrido, por ejemplo, *El origen de las especies* nació de la unión entre biología y geología y rompió con todas las teorías de evolución anteriores, lo que lo ha convertido en uno de los textos científicos más importantes de todos los tiempos.

Es por todo esto que he decidido usar todo lo aprendido en el Doble Grado en Ingeniería del Software y Matemáticas y lo poco que sé de biología para hacer este TFG. Tengo claro que no supondrá ninguna revolución aunque espero que salga algo interesante de aquí.

Capítulo 2

Objetivos

En este capítulo se expondrán los objetivos del proyecto para responder a la siguiente cuestión: qué metaheurística es más eficiente a la hora de decidir el posicionamiento de un depredador para maximizar su utilidad dentro de la manada.

2.1 Objetivo general 1

Implementar metaheurísticas para los algoritmos de optimización de estrategias de maximización de utilidad de un depredador dentro de su manada.

2.1.1 Objetivos específicos 1

1. Definir el problema.
2. Seleccionar las metaheurísticas a implementar.

2.2 Objetivo general 2

Comparar las distintas estrategias metaheurísticas, con el fin de determinar la mejor estrategia de resolución del problema de optimizar la caza de las especies depredadoras.

Capítulo 3

Optimización y metaheurísticas

En este capítulo se definirá el concepto de optimización matemática, se hará la distinción de problemas P y NP y finalmente se describirá la necesidad de búsqueda de estrategias de optimización heurísticas y metaheurísticas además de definir sus características.

3.1 Optimización

La optimización [1] es el proceso de optimizar o perfeccionar. La optimización matemática consiste en el uso de ecuaciones matemáticas y algoritmos para resolver problemas con el propósito de encontrar la mejor solución posible dentro de las soluciones factibles. En este caso, la mejor solución posible es la que maximiza la utilidad de un depredador sin perjudicar la de los demás.

3.1.1 Elementos de la optimización

En el proceso de optimización encontramos los siguientes elementos [2]:

- **Espacio de soluciones**: Es la codificación de las soluciones factibles, es decir, las soluciones que cumplen las restricciones del problema. Determina el tamaño del espacio de búsqueda de cada problema donde encontramos las restricciones y limitaciones de recursos del problema.
- **Objetivo**: Es el predicado matemático que expresa el propósito a alcanzar dentro del espacio de soluciones. Da lugar a la función objetivo del problema de decisión.

- **Función objetivo**: Permite asociar a cada solución factible un valor que determina su calidad. Es una correspondencia f entre un espacio de soluciones X y $\mathbb{R}$

- **Vecindad**: Dada una solución $x \in X$, la vecindad $N(x)$ de esa solución es un subconjunto del espacio de soluciones X que contiene soluciones "próximas" a la solución considerada.

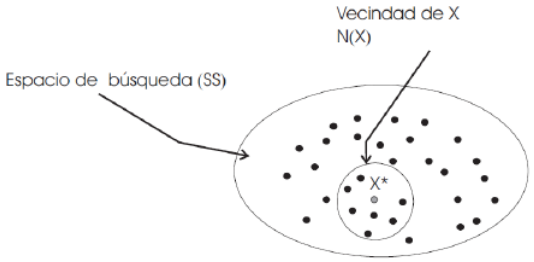

Figura 3.1: Vecindad de una solución. Fuente: [2]

- **Distancia**: Sea X el espacio de soluciones factibles para un problema concreto se puede definir una función distancia $dist(x, y)$ entre cualquier par de puntos $x, y \in X$:

$$dist : X \times X \longrightarrow \mathbb{R}$$

Se puede definir la vecindad $N(x) \subseteq X$ de la solución x como:

$$N(x) = \{y \in X : dist(x, y) \leq \varepsilon, \quad \varepsilon > 0 \in \mathbb{R}\}$$

- **Movimiento**: Cada solución x dentro del espacio de soluciones factibles X tiene asociado un vecindario $N(x) \subseteq X$ que consiste en aquellas soluciones que se pueden alcanzar desde x con un movimiento. Se define un **movimiento** m sobre X como una operación $\oplus$ que permite construir una solución vecina $S = X \oplus m, \quad S \in N(X)$. $M(S)$ es el conjunto de todos los posibles movimientos que se pueden realizar desde S.

3.1.2 Definición de un problema de optimización

Con todos los elementos anteriores definidos, podemos plantear un problema de optimización de esta manera:

Sea X un espacio de soluciones factibles Sea f una función objetivo

$$f(x) : x \in X \longrightarrow \mathbb{R}$$

Entonces el **problema de optimización** se define como el proceso de búsqueda de $\hat{x}$ que verifique

$$\hat{x} \in X : f(\hat{x}) \leq f(x), \forall x \in X$$

Para un problema donde el objetivo es minimizar la función f.

Estas soluciones $\hat{x}$ pueden ser óptimos locales o globales:

- **Óptimo global**: Dado un problema de optimización se dice que una solución factible $x \in S \subseteq X$ es un óptimo (mínimo) global si:

$$\forall y \in S \longrightarrow f(x) \leq f(y)$$

- **Óptimo local**: Dado un problema de optimización y una estructura de vecindad N, se dice que una solución factible $x \in S \subseteq X$ es un óptimo mínimo local con respecto a $N(x)$ si:

$$\forall y \in N(x) \longrightarrow f(x) \leq f(y)$$

3.1.3 Optimización multimodal

Dentro de la optimización, se puede encontrar la optimización multimodal que se ocupa de las tareas de encontrar todas o la mayoría de las soluciones múltiples (al menos localmente óptimas) de un problema, en lugar de una única y mejor solución.

3.2 Clasificación de problemas según resolución

Antes de proceder a la clasificación de los problemas es necesario aclarar el concepto de algoritmo, complejidad y Máquina de Turing

3.2.1 Algoritmos

Un **algoritmo** es un conjunto ordenado de pasos exentos de ambigüedad que resuelven un problema. Se puede medir la eficacia de un algoritmo según el tiempo y los recursos que utilicen [3].

- **Tiempo**: En cuanto al tiempo empleado en realizar todos los pasos del algoritmo. Si el tiempo es *polinomial* (p.ej: n^2 pasos para ordenar una lista de n elementos) decimos que es un algoritmo *eficiente*. Si el tiempo es exponencial (p.ej: 2^n pasos para ordenar una lista de n elementos) decimos que es un algoritmo *ineficiente*.

- **Espacio**: En cuanto al espacio en memoria utilizada por un ordenador al resolver el problema.

3.2.2 Máquinas de Turing

Una **Máquina de Turing** [4] es un dispositivo capaz de adoptar un estado determinado conectado a un cabezal de lectura/escritura con el que puede leer y escribir símbolos en una cinta infinita. Podemos representar matemáticamente una máquina de Turing MT de la siguiente manera:

$$MT = (\Gamma, \Sigma, \cdot, Q, q_0, F, f)$$

Donde:

- Γ es el alfabeto de símbolos de la cinta

- $\Sigma \subseteq \Gamma$es el alfabeto de símbolos de entrada

- $\cdot$ es el símbolo blanco que no pertenece a Σ

- Q es un conjunto finito de estados

- $q_0 \in Q$ es el estado inicial

- $F \in Q$ es el conjunto de estados finales

- f es una función de transición $f : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ [1]

Una Máquina de Turing es **determinista** si tiene una única configuración a la que transitar a partir de una configuración dada, en cambio, si posee varias configuraciones a las que transitar es **no determinista**.

[1] L para Left y R para Right

Dada cualquier Máquina de Turing no determinista, existe una Máquina de Turing determinista que acepta el mismo lenguaje, es decir, el no determinismo no aporta mayor poder de computación a las Máquinas de Turing.

3.2.3 Problemas decidibles vs indecidibles

Se pueden dividir los problemas en grandes grupos, los problemas **decidibles** y los problemas **indecidibles** [5]. Se dice que un problema es **indecidible** cuando no existe un algoritmo que permita resolverlo, ni siquiera con tiempo y recursos ilimitados. No se puede determinar si el algoritmo se dentendrá en algún momento ni cuando lo hará. Por contraposición un problema es **decidible** cuando existe o puede existir un algoritmo para su resolución.

Los problemas decidibles se dividen a su vez en dos tipos de problemas, los problemas **P** y los problemas **NP**.

3.2.4 Problemas P vs NP

- **Problemas P**: Problemas que pueden ser resueltos en con un algoritmo de complejidad polinomial por una máquina de Turing determinista.

- **Problemas NP**: Problemas cuyas soluciones pueden ser verificadas en tiempo polinómico por una máquina de Turing no determinista. Aunque hasta el momento muchos de estos problemas requieren de tiempo exponencial para ser resueltos, como por ejemplo encontrar los factores primos de un número muy grande.

- Todo problema P es NP ya que si P se puede resolver en tiempo polinomial, también se puede verificar en tiempo polinomial. Que todo problema NP sea P no se ha conseguido demostrar y se asume que P$\neq$ NP

- **Poblemas NP-Completos**: Los más complejos de resolver y se intuye que no son problemas P. Un problema C es NP-Completo si es un problema NP y si todo problema NP se puede reducir a C en tiempo polinomial.

- **Problemas NP-Hard** son problemas NP-Completos pero que no pertenecen a NP.

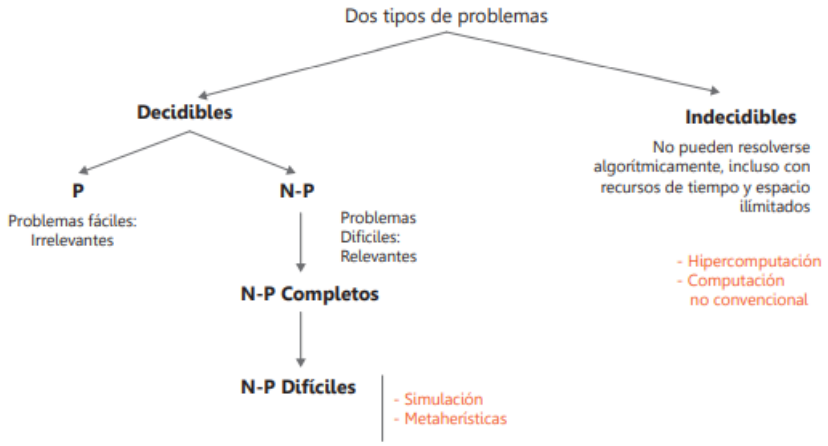

Figura 3.2: Clasificación de los problemas P y NP. Fuente: [5]

3.3 Heurísticas y metaheurísticas

Las heurísticas y metaheurísticas son herramientas utilizadas en la resolución de problemas NP-Completos y que dan soluciones relativamente buenas a los mismos.

3.3.1 Algoritmos heurísticos

Un algoritmo heurístico [2] es un procedimiento simple a menudo basados en el sentido común que priorizan la sencillez y la rapidez sacrificando optimalidad.

Existen dos tipos de algoritmos heurísticos:

- **Constructivos**: Procedimientos que son capaces de construir una solución a un problema dado. La forma de construir la solución depende de la estrategia seguida.

    1. **Estrategia voraz**: Partiendo de una semilla se va añadiendo un elemento de la solución paso a paso de manera que el elemento elegido en cada iteración produce una mejora en la solución parcial.
    2. **Estrategia de descomposición**: Se divide sistemáticamente el problema en subproblemas más pequeños de forma recursiva creando soluciones parciales que serán combinadas al final del algoritmo.
    3. **Estrategia de reducción**: Identifica las características que contienen las soluciones buenas conocidas y se asume que la solución óptima también las tendrá, reduciendo así el espacio de búsqueda.

4. **Estrategia de manipulación del modelo**: Simplifica el problema original para obtener una solución al problema simplificado y a partir de esta extrapolar una solución al problema original.

- **Búsqueda**: Parten de una solución factible dada y a partir de ella intentan mejorarla.

1. ***First improvement***: Mejoran la solución seleccionando el primer movimiento que la mejore dentro de su vecindad.
2. ***Best improvement***: Examina la vecindad de la solución original y todos los posibles movimientos eligiendo el que mayor incremento produzca en la calidad de la solución.
3. **Aleatorizada**: Para una solución factible dada y una vecindad asociada a esa solución, se seleccionan aleatoriamente soluciones que pertenecen a esa vecindad.

3.3.2 Algoritmos metaheurísticos

El término **metaheurística**[6] se obtiene anteponiendo el sufijo *meta* a heurística, significa superior a las heurísticas. Una metaheurística es una forma de mejorar una heurística en cuanto a rendimiento.

Al igual que ocurre con las heurísticas, existe una gran variedad de metaheurísticas como se ilustra en la figura 3.3. Las más importantes son:

- **De Relajación**: Utilizan modificaciones del modelo original haciendo el problema más fácil, cuya solución facilita la resolución del problema original.

- **Constructivas**: Establecen mejores estrategias de selección de elementos para las soluciones parciales que se crean en las heurísticas constructivas.

- **Búsqueda**: Recorren el espacio de soluciones de una manera más eficiente que las heurísticas, evitando así quedarse atrapadas en óptimos locales.

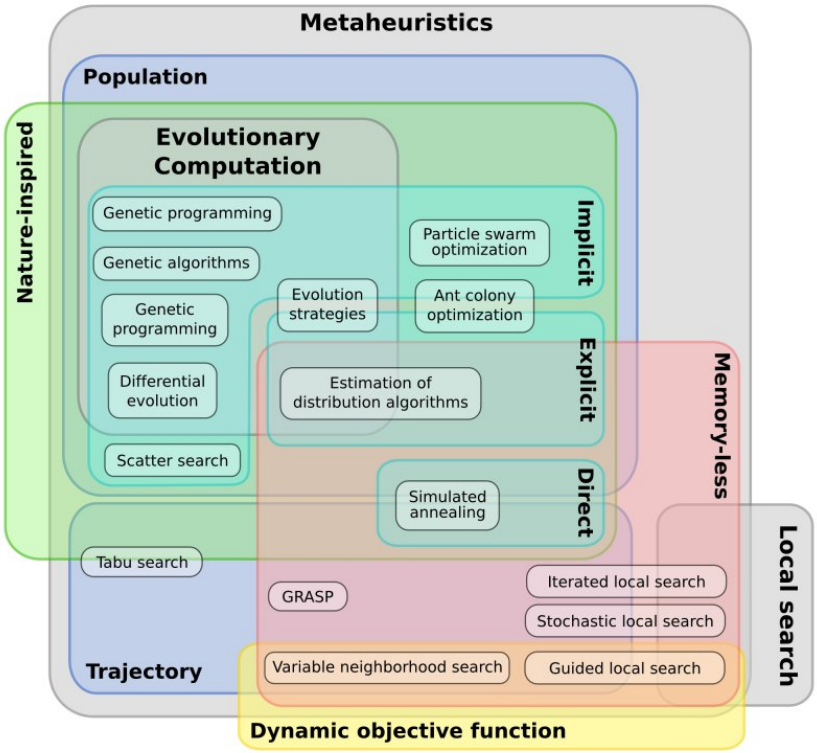

Figura 3.3: Clasificación de las metaheurísticas. Fuente: [6]

- **Metaheurísticas evolutivas**: Establecen estrategias para conducir la evolución en el espacio de búsqueda de conjuntos de soluciones (usualmente llamados poblaciones) con la intención de acercarse a la solución óptima con sus elementos.

3.3.3 Exploración y explotación

Más adelante en el texto encontraremos mencionados estos dos conceptos. La **exploración** es la fase de la metaheurística en la que se busca por el espacio de soluciones distintas vecindades donde es posible que se encuentre una solución óptima global. Una vez detectadas estas vecindades, decimos que la metaheurística entra en fase de **explotación** cuando busca las soluciones óptimas locales dentro de dichas vecindades.

Capítulo 4

Estado del arte

En este capítulo se presentarán dos formas en las que se han modelado las estrategias de caza de lobos.

4.1 Simulación de caza a partir de reglas simples

Para ofrecer una explicación alternativa a la aparente complejidad de las estrategias de caza de los lobos, un grupo de investigadores de la Asociación de Perros de Asistencia AEPA-Euskadi creó un modelo computacional basado en agentes para simular este comportamiento. Los agentes tienen las siguientes propiedades:

- Los agentes son homogéneos, es decir, no existe una jerarquía entre ellos.
- Los agentes no usan comunicación explícita; toda la información es obtenida a partir de estímulos externos, como la posición de los demás agentes que obtienen a partir del sentido de la vista.
- Los agentes no cooperan explícitamente aunque se afectan unos a otros en cuanto a su distribución y su fase dentro del proceso de caza.
- Se asume que los agentes tienen un conjunto similar de objetivos.
- Los agentes son capaces de percibir a sus iguales y distinguen objetos del entorno en dos categorías: los "iguales" y "todo lo demás"

Las reglas que siguen los agentes son muy simples:

1. Moverse hacia la presa hasta que se encuentren a una distancia prudencial. Cada lobo se mueve en línea recta, independientemente de la trayectoria de los demás, hacia la presa. Al llegar a una distancia d de la presa se detiene. Si el lobo se acercase más podría ser dañado.

2. Cuando se llega a la distancia segura d, el lobo se recoloca alejándose de los demás que también estén a una distancia segura.

.

Los resultados que se obtienen se muestran a continuación y recuerdan a las estrategias que utilizan los lobos.

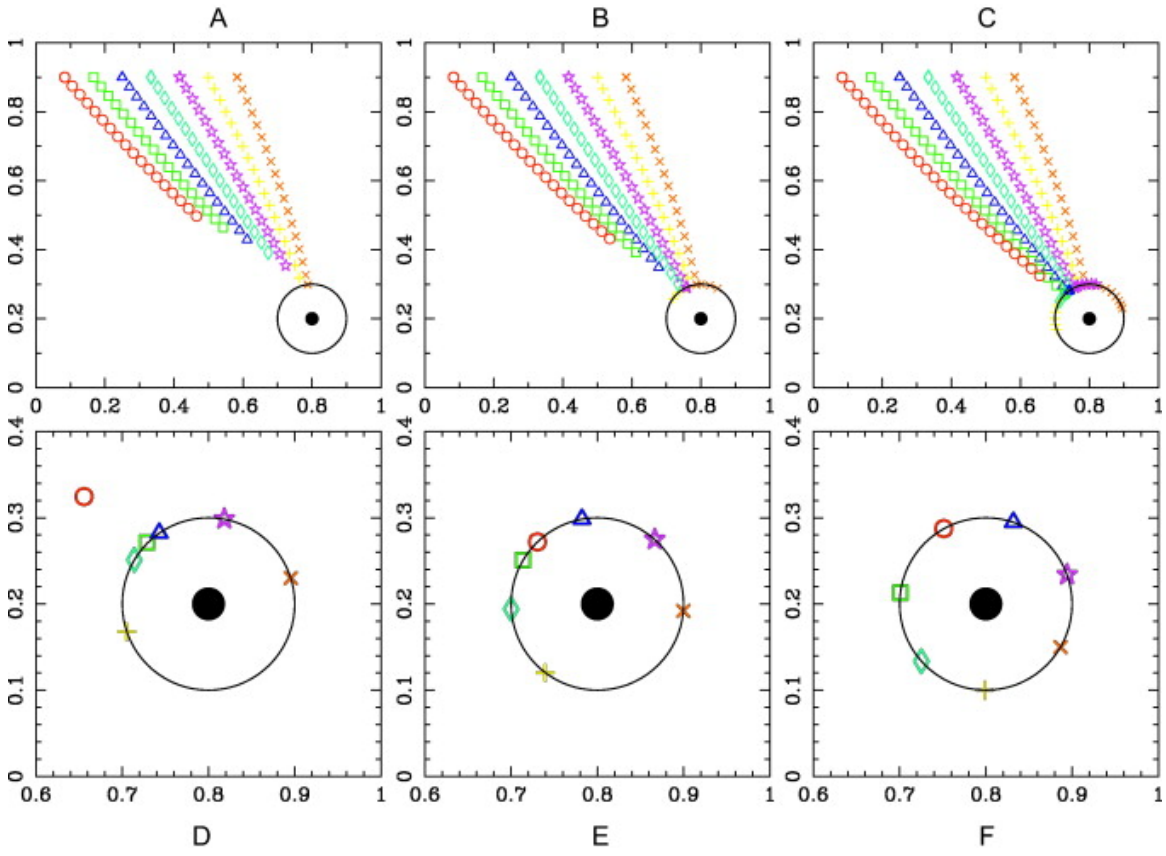

Figura 4.1: Lobos rodeando a presa estacionaria. Fuente: [7]

Los lobos parecen rodear a la presa (figura 4.1), e incluso llegan a ralentizarla (figura 4.2).

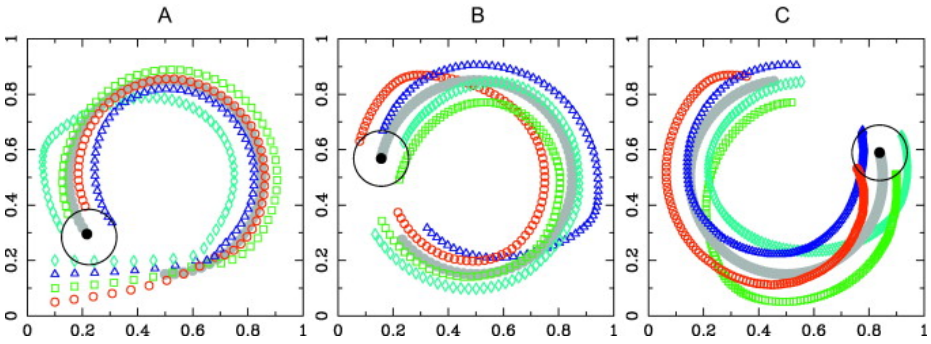

Figura 4.2: Lobos ralentizando a presa en movimiento. Fuente: [7]

También se pudieron observar aparentes comportamientos de emboscada (figura 4.3).

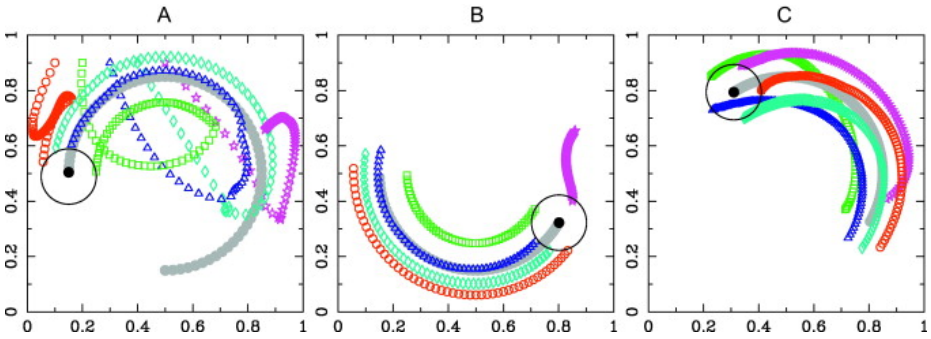

Figura 4.3: Lobos emboscando a presa en movimiento. Fuente: [7]

El experimento concluye que se pueden explicar las estrategias de caza de los lobos sin asumir un alto nivel de coordinación entre ellos.

4.2 Modelo bioinspirado de caza con robots

Otra manera de modelar la caza de animales es creando agentes que pasen por distintos estados. Alfredo Weitzenfeld llegó al siguiente diseño [8]:

- **Lobo Alpha**: Su comportamiento se define con tres estados como se presenta en la figura 4.4.

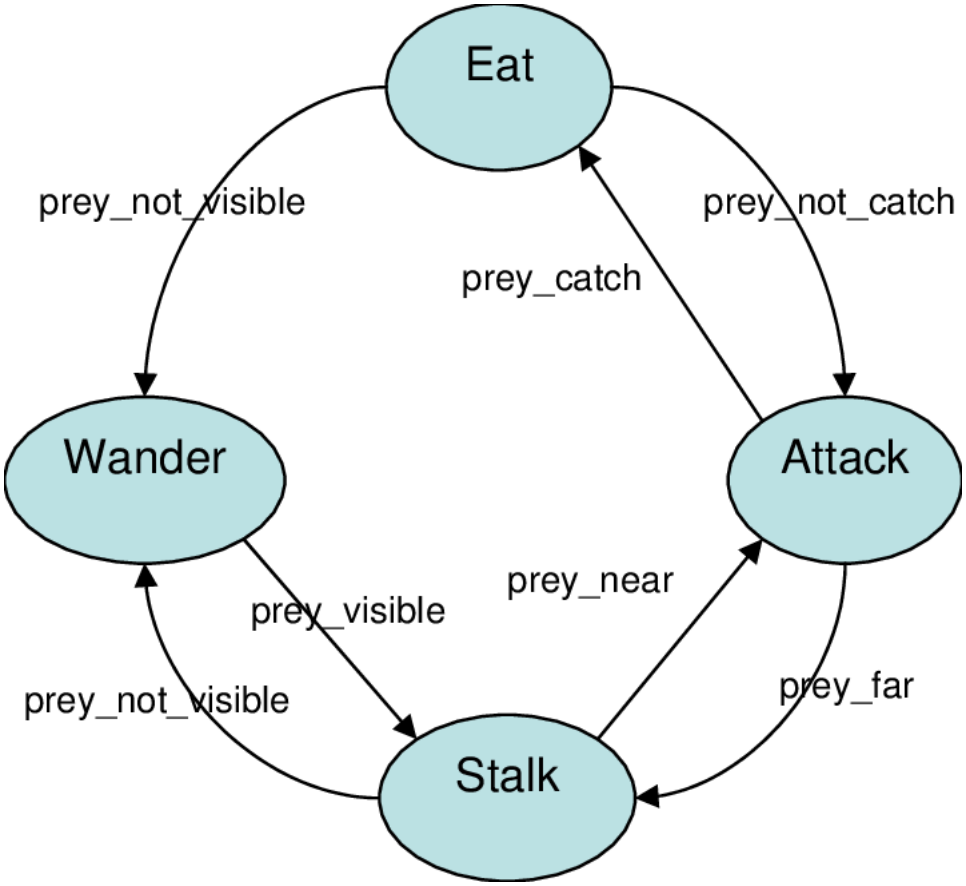

Figura 4.4: Diagrama de estados de un lobo Alpha. *Wander* (Explorar), *Stalk* (Acechar), *Attack* (Atacar), *Eat* (Comer). Fuente: [8]

  - **Explorar**: Se busca a una presa por el entorno, si la encuentra se transiciona al estado de **Acechar**.
  - **Acechar**: Desde este estado un lobo puede transicionar a **Explorar** si deja de ver a la presa o a **Atacar** si la presa está lo suficientemente cerca.
  - **Atacar**: En este estado el lobo se acerca a la presa hasta atraparla. Pasa al estado **Comer** si consigue inmovilizarla, en caso contrario vuelve al estado **Acechar** si se le escapa.
  - **Comer**: Estado en el que el lobo come. De este estado se pasa a **Explorar**.

- **Lobo Beta**: Su comportamiento se define con tres estados como se presenta en la figura 4.5.

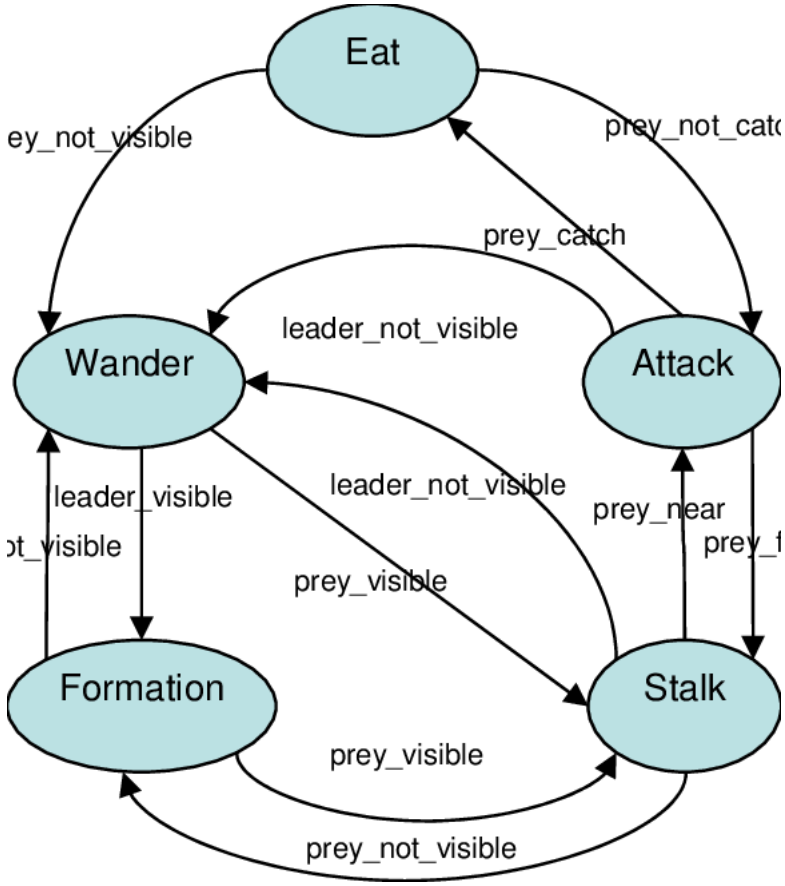

Figura 4.5: Diagrama de estados de un lobo Beta. *Wander* (Explorar), *Stalk* (Acechar), *Attack* (Atacar), *Eat* (Comer), *Formation* (Formación). Fuente: [8]

- **Explorar**: Se busca a la presa o al líder de la manada por el entorno. Si encuentra al líder pasa a **Formación**, si encuentra a la presa entonces transiciona al estado de **Acechar**.
- **Formación**: Mientras siga viendo al lobo Alpha, se mantiene cerca de él, si lo pierde de vista pasa al estado **Explorar**. Si encuentra a la presa pasa al estado **Acechar**.
- **Acechar**: Desde este estado un lobo puede transicionar a **Formación** si deja de ver a la presa y en caso de perder de vista al lobo Alpha pasa al estado **Explorar**. Cambia al estado **Atacar** si la presa está lo suficientemente cerca.
- **Atacar**: En este estado el lobo se acerca a la presa hasta atraparla. Pasa al estado **Comer** si consigue inmovilizarla o de nuevo al estado **Acechar** si se le escapa.

– **Comer**: Estado en el que el lobo come. De este estado se pasa a **Explorar**.

Además de esto también modela un comportamiento de repulsión entre los lobos para evitar choques entre ellos. En la figura 4.6 se muestra el estado **Cazar** que encapsula los estados previamente explicados y el estado **Evitar** al que se transiciona cuando un lobo ve a otro que no sea el Alpha para alejarse de él.

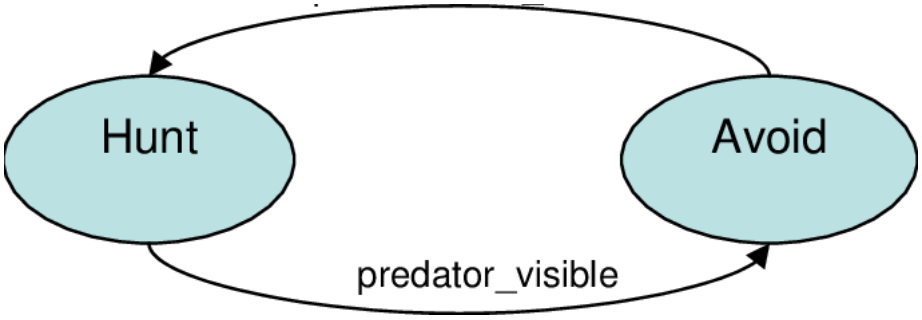

Figura 4.6: Diagrama de estados para el comportamiento de repulsión. *Hunt*(Cazar), *Avoid*(Evitar). Fuente: [8]

Capítulo 5

Definición del problema

En este capítulo se explicará cuál es el objetivo de usar las metaheurísticas. Posteriormente se definirá formalmente el problema.

5.1 Idea general

En este TFG se han implementado las metaheurísticas anteriormente explicadas para la simular la caza coordinada entre depredadores. El funcionamiento actual del sistema de simulación está detallado en la memoria del TFG de software con título *Diseño y desarrollo de una aplicación para la simulación de ecosistemas* en la sección 10.2. La manada de depredadores selecciona una presa del rebaño. Para cada uno de los depredadores se quiere obtener la posición óptima en la que se debería colocar para acercar a la presa al resto de la manada, sabiendo que la presa huirá de él como se ilustra en la figura 5.1. El depredador asume que la presa se moverá 15 unidades de distancia en dirección contraria a él.

La simulación es en tiempo real, es decir, los depredadores y las presas están en constante movimiento. Esto implica que la posición óptima de cada depredador cambia a lo largo del tiempo. Por esta razón, se actualizará la posición óptima o deseada de cada depredador cada medio segundo.

La hipótesis que se plantea es la siguiente: si todos los depredadores utilizan las metaheurísticas para calcular sus posiciones óptimas, acabarán acercándose o incluso rodearán a la presa para poder cazarla.

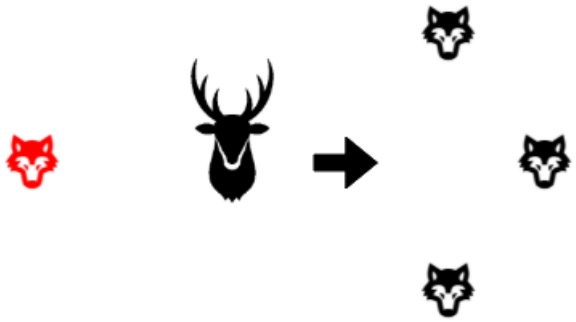

Figura 5.1: Diagrama de caza. Fuente: Elaboración propia

5.2 Formalización del problema

Se modela el problema de optimización a resolver de la siguiente manera.

- **Espacio de soluciones**: Es espacio de soluciones será $\mathbb{R}^2$. Todas las soluciones serán factibles aunque se inicializarán en un radio de 30 unidades de la posición inicial $\vec{x} = (x, y) \in \mathbb{R}^2$ del depredador.

- **Función objetivo**: Se minimizará la siguiente función objetivo f definida como:

$$f(\vec{p}) = max\{d(\vec{u}, \vec{p}), max\{d(\vec{u}, \vec{m}), \vec{m} \in M \backslash \{\vec{x}\}\}\} \tag{5.1}$$

  Siendo:

  1. $\vec{p} \in \mathbb{R}^2$ la posición a la que se moverá el depredador p.
  2. $\vec{m}$ la posición de un depredador perteneciente a la manada M, sin incluir $\vec{x}$.
  3. $\vec{u} \in \mathbb{R}^2$ la posición en la que quedará la presa si el depredador se mueve a la posición $\vec{p}$.
  4. $d : \mathbb{R}^2 \times \mathbb{R}^2 \longrightarrow \mathbb{R}$ la distancia euclídea entre dos puntos.

Las metaheurísticas recibirán:

1. La **posición del depredador** que se quiere mover. Se denota como $\vec{p}$
2. La **posición de la presa**. Se denota como $\vec{u}$
3. Las posiciones de los demás miembros de la **manada**. Se denotan como $M = \{m_1, m_2, \ldots, m_n\}$

Y devolverán la posición $\vec{s}$ que minimiza la función objetivo 5.1. En resumen, se quiere minimizar la máxima distancia de la presa a los depredadores.

5.2.1 Análisis del problema

Dada la incomputabilidad de $\mathbb{R}^2$, un ordenador no puede obtener la solución óptima al problema. Si se discretiza el espacio de soluciones entonces este problema sería un problema P ya que habría que recorrer un conjunto de puntos evaluando su optimalidad, tarea que se puede hacer en tiempo polinomial. ¿Por qué se usan metaheurísticas entonces? El espacio de soluciones puede llegar a ser demasiado grande, lo que daría un tiempo de computación elevadísimo aunque el algoritmo implementado fuera eficiente. Si hay que calcular cada medio segundo una posición óptima para todos los integrantes de una manada no existe eficiencia. Usar metaheurísticas que obtengan soluciones aceptables en poco tiempo soluciona ese problema. Además, la simulación pretende modelar estrategias de caza animal, que no siempre son exitosas, y con metaheurísticas se puede conseguir la sensación de movimientos erráticos que pueden producirse en la naturaleza.

Son por todas estas razones que se emplearán metaheurísticas de búsqueda multiagentes para la simulación del comportamiento de caza de los depredadores.

Capítulo 6

Metodología

En este capítulo se detallará en primer lugar el algoritmo que se ha utilizado para simular la huida de las presas. Posteriormente se explicará el funcionamiento de las distintas metaheurísticas implementadas para la simulación de caza, comenzando por el *Particle Swarm Optimization*, siguiendo con el *Grey Wolf Optimizer* y finalizando con el *Whale Optimization Algorithm*. Para cada algoritmo se hablará de su inspiración, modelado matemático, fases de exploración y explotación además de proporcionar el pseucódigo del mismo.

6.1 Algoritmo de huida

El algoritmo de huida principal que usarán las presas está inspirado en el algoritmo de Boids [9] que modela el comportamiento de un rebaño en el que cada individuo maniobra según las posiciones y velocidades de los agentes cercanos siguiendo unas simples reglas:

- **Separación**: Moverse para evitar chocar con agentes cercanos.
- **Alineación**: Girar hacia la orientación media de los agentes cercanos.
- **Cohesión**: Moverse para intentar quedar en el medio de los agentes cercanos.

Ya que en la simulación existen predadores y las presas necesitan llegar a un lugar seguro para evitar ser cazadas he añadido otras dos reglas:

- **Esquive**: Alejarse de predadores cercanos.

- **Huida**: Moverse hacia la zona segura.

Expresemos la fuerza que un agente i recibe de una agente j en cada caso dado un instante t:

Separación	$\vec{S}_{i,j}(t) = \begin{cases} U_j(t) - U_i(t) & \text{si } d(U_i, U_j) \leq R_s \\ \vec{0} & \text{si } d(\vec{u}_i, \vec{u}_j) > R_s \end{cases}$	(6.1)
Alineación	$\vec{A}_{i,j}(t) = \begin{cases} \vec{v}_j(t) & \text{si } d(U_i, U_j) \leq R_a \\ \vec{0} & \text{si } d(U_i, U_j) > R_a \end{cases}$	(6.2)
Cohesión	$\vec{C}_{i,j}(t) = \begin{cases} U_j(t) & \text{si } d(U_i, U_j) \leq R_c \\ \vec{0} & \text{si } d(U_i, U_j) > R_c \end{cases}$	(6.3)
Esquive	$\vec{E}_{i,j}(t) = \begin{cases} P_j(t) - \vec{u}_i(t) & \text{si } d(U_i, P_j) \leq R_e \\ \vec{0} & \text{si } d(U_i, P_j) > R_e \end{cases}$	(6.4)
Huida	$\vec{H}$	(6.5)

Tabla 6.1: Tabla de fuerzas ejercidas sobre un agente presa

Donde:

- $U(t) = \{x(t), y(t), z(t)\}$ es la posición de una presa en un instante t.
- $P(t) = \{x(t), y(t), z(t)\}$ es la posición de un predador en un instante t.
- $\vec{v}(t) = \{x(t), y(t), z(t)\}$ es la velocidad de cualquier agente en un instante t definida en la ecuación 6.7.

- $\vec{H}$ es un vector dirección que apunta hacia la zona segura.
- R_s es el radio [1] que indica a qué distancia afecta la fuerza de separación.
- R_a es el radio que indica a qué distancia afecta la fuerza de alineación.
- R_c es el radio que indica a qué distancia afecta la fuerza de cohesión.
- R_e es el radio que indica a qué distancia afecta la fuerza de esquive.

Una vez definidas todas las fuerzas, podemos expresar la fuerza $\vec{F}$ que actúa en un agente presa i en un instante t de la siguiente manera:

$$\vec{F_i}(t) = w_s * \frac{1}{M_s} \sum_{\substack{j=1 \\ j \neq i}}^{N} \vec{S}_{i,j}(t) + w_a * \frac{1}{M_a} \sum_{\substack{j=1 \\ j \neq i}}^{N} \vec{A}_{i,j}(t) + w_c * \frac{1}{M_c} \sum_{\substack{j=1 \\ j \neq i}}^{N} \vec{C}_{i,j}(t) + w_e * \frac{1}{M_e} \sum_{\substack{j=1 \\ j \neq i}}^{P} \vec{E}_{i,j}(t) + w_h * \vec{H} \tag{6.6}$$

Y obtenemos la velocidad de la siguiente manera:

$$\vec{v_i}(t) = \beta(t) * \vec{F_i}(t) \tag{6.7}$$

Donde:

- w_s es el peso que le damos a la regla de separación.
- w_a es el peso que le damos a la regla de alineación.
- w_c es el peso que le damos a la regla de cohesión.
- w_e es el peso que le damos a la regla de esquive.
- w_h es el peso que le damos a la regla de huida.
- M_s es el número de agentes presa tales que $d(U_i, U_j) \leq R_s$.
- M_a es el número de agentes presa tales que $d(U_i, U_j) \leq R_a$.
- M_c es el número de agentes presa tales que $d(U_i, U_j) \leq R_c$.

[1] El radio modela la distancia de visión del animal

- M_e es el número de predadores tales que $d(U_i, P_j) \leq R_e$.
- N es el número de agentes presa.
- P es el número de predadores.
- $\beta(t) = \begin{cases} \frac{v_{max}}{\|\vec{F}_i(t)\|} & \text{si } \|\vec{F}_i(t)\| > v_{max} \\ 1 & \text{en otro caso} \end{cases}$
- v_{max} es la velocidad máxima de un agente presa.

Por lo tanto la posición de los agentes se puede calcular mediante el siguiente sistema de ecuaciones diferenciales ordinarias con las correspondientes condiciones iniciales:

$$\frac{dU_i(t)}{dt} = \vec{v}_i(t) \qquad i = 1, 2, \ldots, N \tag{6.8}$$

Como la simulación funciona en pasos discretos dado que se consideran instantes de tiempo $t = \{1, \ldots, T\}$, la velocidad se calcula de una forma distinta para dar una sensación de movimiento fluido:

$$\vec{v}_{n,i} = \alpha_i(G * \vec{v}_{n-1} + (1 - G) * \beta_i(\vec{F}_{n,i})) \tag{6.9}$$

Siendo:

- G la fuerza de giro.
- $\alpha_i = \begin{cases} \frac{v_{max}}{\|(G*\vec{v}_{n-1}+(1-G)*\beta_i(\vec{F}_{n,i})\|} & \text{si } \|(G * \vec{v}_{n-1} + (1 - G) * \beta_i(\vec{F}_{n,i})\| > v_{max} \\ 1 & \text{en otro caso} \end{cases}$

6.2 Algoritmo Particle Swarm Optimization

El algoritmo ***Particle Swarm Optimization***[10] o ***PSO*** es una metaheurística de búsqueda multiagente en la que varios agentes pertenecientes a un enjambre recorren el espacio de soluciones intercambiando información entre ellos para localizar soluciones óptimas.

6.2.1 Idea general PSO

Este algoritmo está inspirado en el movimiento de bancos de peces. Cada agente de búsqueda de la metaheurística recuerda las siguientes piezas de información:

1. La mejor posición en la que ha estado.
2. La mejor solución de todo el enjambre.
3. La dirección en la que se está moviendo.

Con esta información, calcula un punto deseado al que ir, sumando los tres vectores que representan a cada elemento. A cada elemento de información le dará un peso aleatorio por lo que acabará en una posición aleatoria dentro del paralelogramo que se puede ver en la figura 6.1.

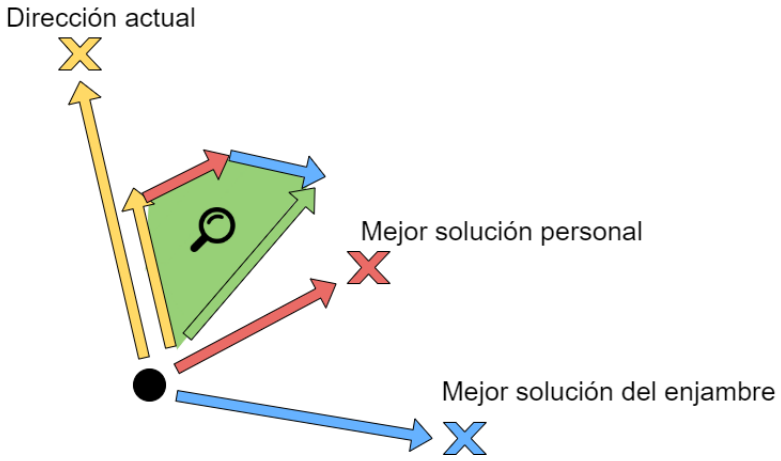

Figura 6.1: Diagrama PSO. Fuente: Elaboración propia

6.2.2 Modelado matemático del PSO

Definimos la **posición** de un agente i en una iteración t como $X_i^t = (x, y) \in \mathbb{R}$.

Esta posición se actualiza en cada iteración del algoritmo de la siguiente manera:

$$X_i^{t+1} = X_i^t + \vec{V}_i^{t+1} \tag{6.10}$$

Siendo

$$\vec{V}_i^{t+1} = w\vec{V}_i^t + c_1 r_1 (P_i - X_i^t) + c_2 r_2 (G - X_i^t) \tag{6.11}$$

Donde

- $w \in [0, 1]$ es la **inercia** del agente.
- $\vec{V}_i^t \in \mathbb{R}^2$ es la **velocidad** del agente i en la iteración t, el vector de color amarillo de la figura 6.1.
- $c1 \in [0, 2]$ una constante que le da un peso al **componente cognitivo** del algoritmo, es decir, cuánta importancia le da un agente a su mejor solución hasta el momento.
- $c2 \in [0, 2]$ una constante que le da un peso al **componente social** del algoritmo, es decir, cuánta importancia le da un agente a la mejor solución encontrada por el enjambre.
- $r1, r2 \in [0, 1]$ son valores escalares **aleatorios**.
- $P_i \in \mathbb{R}^2$ la **mejor solución** encontrada por el **agente** i.
- $G \in \mathbb{R}^2$ la **mejor solución** encontrada por el **enjambre**.

6.2.3 Exploración y explotación del PSO

A continuación se detalla el efecto de los distintos parámetros en la búsqueda de soluciones.

Los parámetros aleatorios $\boldsymbol{r_1}$ y $\boldsymbol{r_2}$ sirven para evitar que un agente se quede atrapado fácilmente en un óptimo local.

Cuando $\boldsymbol{w}$ es 1 la exploración del espacio de soluciones es el máximo, es decir, se buscan zonas donde encontrar un óptimo global, en cambio, cuando w está cercano a 0 la exploración es prácticamente nula y se pasa a una fase de explotación, donde se buscan óptimos locales. La mejor estrategia a seguir es decrementar w de 1 a 0 a lo largo de las iteraciones del algoritmo.

En cuanto a los parámetros $\boldsymbol{c_1}$ y $\boldsymbol{c_2}$ existen distintas posibilidades:

1. Si $c_1 \gg c_2$ la exploración está en un nivel alto dado que las soluciones no convergen al no haber intercambio de información entre los agentes.
2. Si $c_2 \gg c_1$ la explotación está en un nivel alto dado que las soluciones convergen rápidamente a la mejor solución encontrada por el enjambre. Esto puede llevar a encontrar soluciones óptimas locales ya que los agentes no dan importancia a sus propios descubrimientos que podrían estar cerca de la solución óptima global.

3. Si c_2 y c_1 tienen un valor parecido existe un equilibrio entre la exploración y la explotación.

6.2.4 Pseudocódigo del algoritmo PSO

El funcionamiento del algoritmo se detalla en la siguiente tabla:

Inicializar los parámetros $(N, c_1, c_2, w_{min}, w_{max}, Iteraciones)$
Inicializar las N soluciones iniciales de los agentes aleatoriamente
for 0:$iteraciones$ **do**
    $G \leftarrow$ Obtener la mejor solución
    Decrementar w
    for cada agente **do**
        Calcular la nueva velocidad $\vec{V}$ del agente a partir de la fórmula 6.10
        Calcular la nueva posición X del agente a partir de la fórmula 6.10
    end for
end for

Tabla 6.2: Pseudocódigo del algoritmo *PSO*

6.3 Algoritmo Grey Wolf Optimizer

El algoritmo ***Grey Wolf Optimizer***[11] o ***GWO*** es una metaheurística bioinspirada de búsqueda multiagente en la que varios agentes pertenecientes a un enjambre recorren el espacio de soluciones intercambiando información entre ellos para localizar soluciones óptimas.

6.3.1 Idea general GWO

Este algoritmo se inspira en la estrategia de caza del lobo gris. Esta especie está en lo alto de la cadena alimenticia y suele vivir manadas de entre 5 y 12 integrantes, cada uno con su rango dentro de la misma.

- **Alpha**: Un líder macho y una líder hembra. Toman las decisiones más importantes a la hora de cazar, seleccionar un lugar de descanso o migrar a otras zonas de su hábitat.
- **Beta**: Son lobos subordinados que ayudan a los alphas en la toma de decisión y otras actividades de la manada. Son los candidatos a sustituir a los alpha.

- **Delta**: Ayudan a las tareas de exploración, suelen ser ancianos, cazadores o vigilantes. Dominan a los omega.
- **Omega**: Son los lobos sumisos de la manada. Suelen alimentarse en último lugar y ayudan a cuidar a los cachorros de la manada.

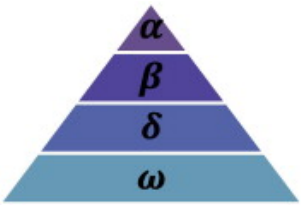

Figura 6.2: Jerarquía de una manada de lobos. Fuente: [11]

El algoritmo GWO utiliza esta jerarquía para clasificar las soluciones encontradas durante la búsqueda según su optimalidad. De esta manera, cada agente actualiza su posición dando mayor peso a las soluciones que representan un nivel más alto de la jerarquía. Esta estrategia se ilustra en la figura 6.3.

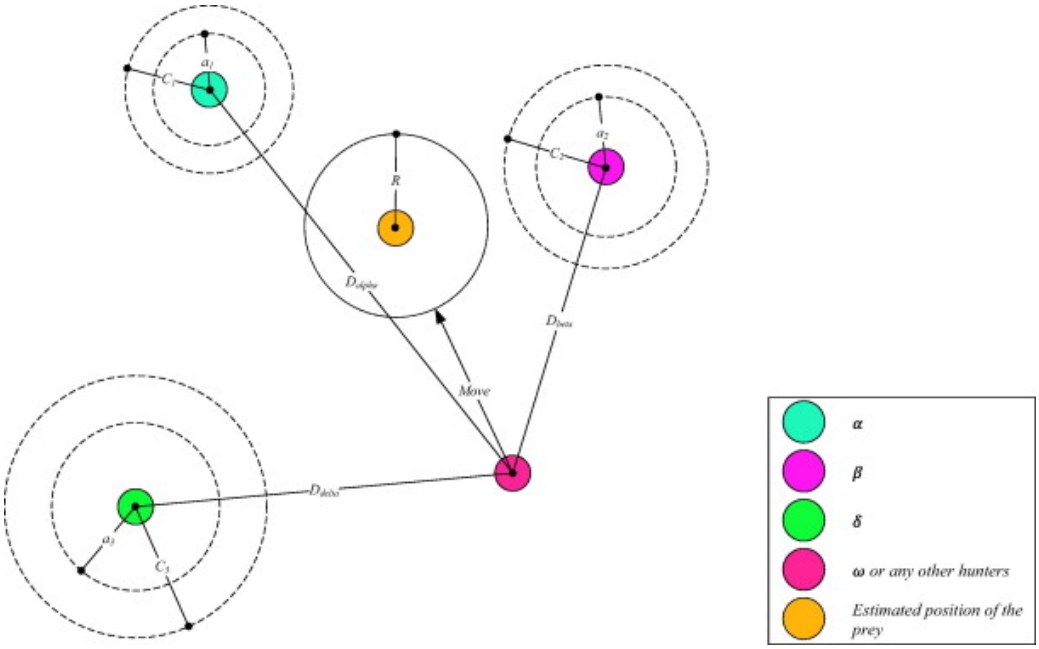

Figura 6.3: Actualización de la posición de un agente en GWO. Fuente: [11]

6.3.2 Modelado matemático del GWO

Definimos la **posición** de un agente i en una iteración t como $X_i^t = (x, y) \in \mathbb{R}$.

Cada posición representa una solución con una optimalidad:

- X_α^t es la mejor solución de la manada en la iteración t.
- X_β^t es la segunda mejor solución de la manada en la iteración t.
- X_δ^t es la tercera mejor solución de la manada en la iteración t.
- X_ω^t es cualquier otra solución en la iteración t.

En cada iteración sólo se actualizan las posiciones de los agentes ω de la siguiente manera:

$$X_\omega^t = \frac{X_1 + X_2 + X_3}{3} \tag{6.12}$$

Siendo:

$$X_1 = X_\alpha^t - D_\alpha \cdot \vec{A} \tag{6.13}$$

$$X_2 = X_\beta^t - D_\beta \cdot \vec{A} \tag{6.14}$$

$$X_3 = X_\delta^t - D_\delta \cdot \vec{A} \tag{6.15}$$

$$D_x = \mid C \cdot X_x^t - X_\omega^t \mid \qquad x \in \{\alpha, \beta, \delta\} \tag{6.16}$$

$$\tag{6.17}$$

Donde:

- $\vec{A} = 2\vec{a} \cdot r_1 - \vec{a}$
- $\vec{a}$ un vector cuyos componentes decrecen linealmente de 2 a 0 a lo largo de las iteraciones.
- r_1 un escalar aleatorio entre 0 y 1.
- $C = 2 \cdot \vec{r}_2$
- $\vec{r}_2$ un vector aleatorio con componentes entre 0 y 1.

Cabe recalcar que se necesitan más de 3 agentes para que el algoritmo funciones ya que las 3 mejores soluciones no se actualizan hasta ser superadas en optimalidad, por lo que no existiría exploración ni explotación durante la ejecución del algoritmo.

6.3.3 Exploración y explotación del GWO

Los parámetros que controlan la exploración y la explotación son $\vec{A}$ y C

- Las coordenadas de $\vec{\boldsymbol{A}}$ pertenecen al intervalo $[-2a, 2a]$. Si $|\vec{A}| \leq 1$ entonces el algoritmo entra en fase de explotación, ya que la nueva posición del agente estará entre su posición actual y la posición en la que estaría la presa (en este caso sería una de las mejores soluciones encontradas hasta el momento). Como se especifica en el apartado 6.3.2, las coordenadas de $\vec{A}$ decrecen con el tiempo, de esta manera se consigue una fase de exploración en las primeras iteraciones del algoritmo para ir transicionando a una fase de explotación. Se puede ver este comportamiento de manera gráfica en la figura 6.4.

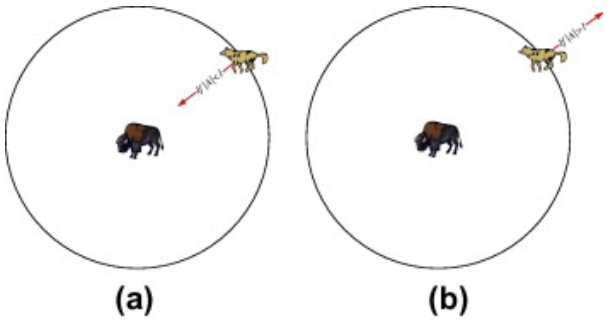

Figura 6.4: Explotación en (a). Exploración en (b). Fuente: [11]

- El parámetro $\boldsymbol{C}$ se utiliza para que los agentes diverjan de tal manera que se explore el espacio de soluciones de manera eficiente. Al tomar valores aleatorios entre 0 y 2, en cada iteración se da un peso distinto a las mejores soluciónes (fórmula 6.16), encontradas para sumar importancia a la decisión del agente de búsqueda y así evitar óptimos locales.

6.3.4 Pseudocódigo del algoritmo GWO

El funcionamiento del algoritmo se detalla en la siguiente tabla:

Inicializar las $N > 3$ soluciones $X_i \quad i = 1, 2, 3, \ldots, N$
Inicializar los parámetros a, $\vec{A}$ y C
for 0:*iteraciones* **do**
    $X_\alpha^t \leftarrow$ Mejor solución
    $X_\beta^t \leftarrow$ Segunda mejor solución
    $X_\delta^t \leftarrow$ Tercera mejor solución
    for cada solución X_ω^t **do**
        Actualizar la solución con la fórmula 6.12
    end for
    Actualizar a, $\vec{A}$ y C
    Actualizar X_α^t, X_β^t y X_δ^t
end for

Tabla 6.3: Pseudocódigo del algoritmo *GWO*

6.4 Algoritmo Whale Optimization

El algoritmo ***Whale Optimizacion***[12] o ***GWO*** es una metaheurística bioinspirada de búsqueda multiagente en la que varios agentes pertenecientes a un enjambre recorren el espacio de soluciones intercambiando información entre ellos para localizar soluciones óptimas.

6.4.1 Idea general WOA

Este algoritmo se inspira en la estrategia de red de burbujas que emplean las ballenas jorobadas para cazar bancos de peces.

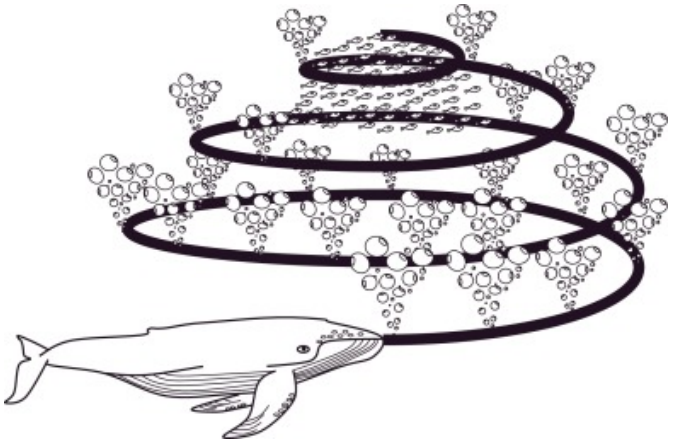

Figura 6.5: Estrategia de red de burbujas. Fuente: [12]

Esta técnica consiste en crear una espiral alrededor del banco de presas para ir reduciendo el espacio en el que pueden moverse como se ve en la figura 6.5

6.4.2 Modelado matemático del WOA

Definimos la **posición** de un agente i en una iteración t como $X_i^{t+1} = (x, y) \in \mathbb{R}$.

En cada iteración t se actualizan las posiciones de los agentes de la siguiente manera:

$$X_1^{t+1} = X_{best}^t - D_1 \cdot \vec{A} \tag{6.18}$$

Donde:

- $D_i = \mid C \cdot X_{best}^t - X_i^t \mid$
- $\vec{A} = 2\vec{a} \cdot r_1 - \vec{a}$
- $\vec{a}$ un vector cuyos componentes decrecen linealmente de 2 a 0 a lo largo de las iteraciones.
- r_1, r_2 un escalar aleatorio entre 0 y 1.
- $C = 2 \cdot r_2$

6.4.3 Exploración y explotación del WOA

La **fase de exploración** se consigue acercando a un agente a otro aleatorio con las siguientes fórmulas cuando $\mid \vec{A} \mid > 1$:

$$X_1^{t+1} = X_{rand}^t - D \cdot \vec{A} \tag{6.19}$$

$$D = \mid C \cdot X_{rand}^t - X_i^t \mid \tag{6.20}$$

Las posibles posiciones a las que viajará el agente se pueden ver en la figura 6.6.

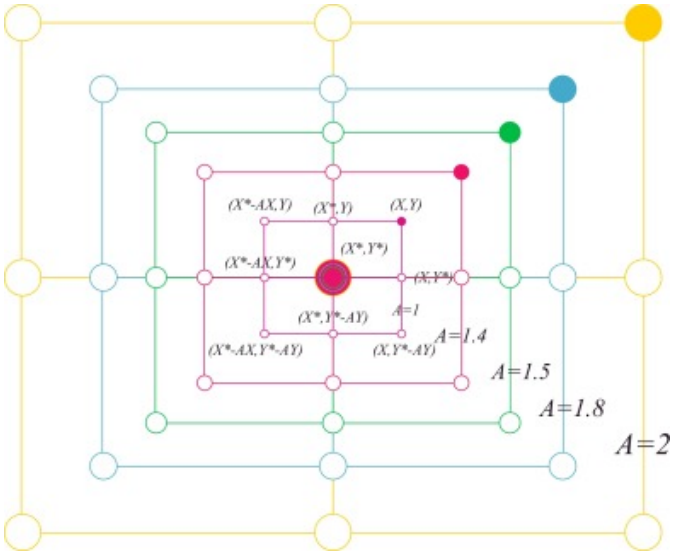

Figura 6.6: Posibles posiciones en fase de exploración según valores de A. $\vec{X}^*$ es el agente aleatorio. Fuente: [12]

Para la **fase de explotación** se modelan dos comportamientos de las ballenas.

1. **Constricción de la red**: Gracias al decrecimiento de $\vec{a}$ a lo largo de las iteraciones se transiciona a la fase de de explotación cuando el módulo de $\vec{A}$ es menor que 1.

2. **Movimiento en espiral**: Además de actualizar su posición mediante la fórmula 6.18, un agente también puede hacerlo a partir de la siguiente ecuación:

$$X_1^{t+1} = D' \cdot e^{bl} \cdot \cos(2\pi l) + X_{best}^t \tag{6.21}$$

Donde:

- $D' = \mid X_{best}^t - X_i^t \mid$
- b es un parámetro para definir la forma de la espiral logarítmica.
- l es un parámetro aleatorio entre -1 y 1.

El agente tiene un 50% de posibilidades de utilizar una de las dos fórmulas para actualizar su posición en cada iteración. Podemos ver en la figura 6.7 las dos formas de explotar el espacio de soluciones alrededor de las mejores soluciones encontradas en un determinado momento.

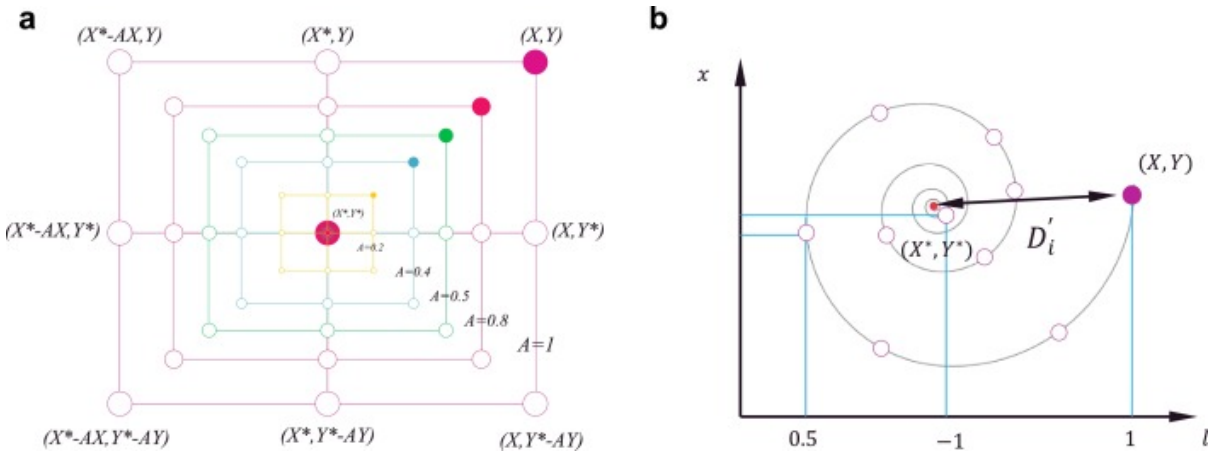

Figura 6.7: (a)Constricción de la red. (b)Movimiento en espiral. Fuente: [12]

6.4.4 Pseudocódigo del algoritmo WOA

Inicializar las N soluciones $X_i \quad i = 1, 2, 3, \ldots, N$
Inicializar los parámetros a, $\vec{A}$ y C
for 0:*iteraciones* **do**
    $X^t_{best} \leftarrow$ Mejor solución
    for las demás soluciones **do**
        Actualizar los parámetros a, $\vec{A}$, C, l, y p
        if $p > 0.5$ **then**
            Actualizar la posición del agente con la ecuación 6.21
        else
            if $\mid \vec{A} \mid < 1$ **then**
                Actualizar la posición del agente con la ecuación 6.18
            else
                Seleccionar un agente X_{rand} aleatorio y actualizar la posición del agente con la ecuación 6.19
            end if
        end if
    end for
    Actualizar la optimalidad de cada solución
end for

Tabla 6.4: Pseudocódigo del algoritmo *WOA*

Capítulo 7

Resolución del problema

En este capítulo veremos cómo resuelve cada algoritmo el problema definido en el capítulo 5. En primer lugar se plantearán dos problemas, uno con solución conocida y otro con solución desconocida, para ver cómo se comportan los distintos algoritmos.

Se realizarán varios experimentos variando número de iteraciones y número de agentes. Una vez analizado su rendimiento se realizará una comparación de los tres algoritmos para decidir cuál resuelve mejor el problema.

7.1 Problema específicos

Se plantean dos problemas específicos para testear las metaheurísticas.

7.1.1 Problema específico con solución conocida

Es difícil plantear un caso en el que sepamos la solución óptima a priori, por lo que se ha diseñado un problema específico en el que es sencillo calcularla. De esta manera se podrá comprobar que las metaheurísticas funcionan correctamente.

Las condiciones iniciales, representadas en la figura 7.1, del problema son:

- **Posición inicial del depredador**: $P = (0, 0)$
- **Posición inicial de la presa**: $U = (15, 15)$
- **Posiciones de la manada**: $M = \{(0, 30), (15, 45), (30, 30)\}$

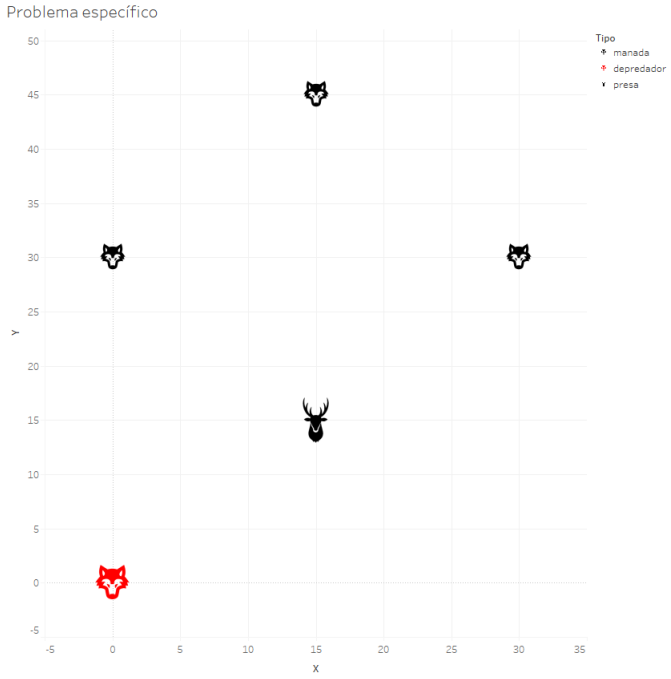

Figura 7.1: Condiciones iniciales del problema. Fuente: Elaboración propia

De esta manera sabemos que la posición óptima será $X = (15, 15)$ ya que la presa se moverá a la posición $U' = (15, 30)$ y quedará a la misma distancia de todos los depredadores por lo que aseguramos que la máxima distancia a ellos es la mínima.

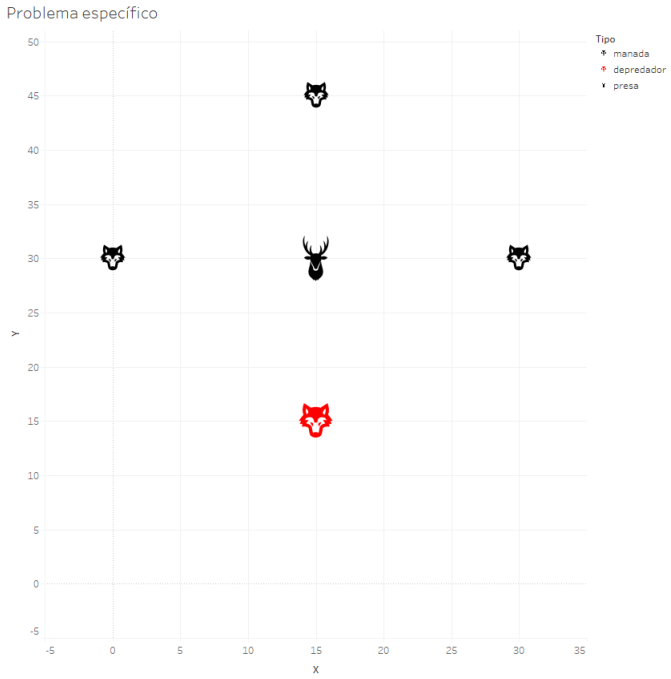

Figura 7.2: Solución del problema. Fuente: Elaboración propia

Se puede observar la solución en la figura 7.2 donde:

- El **lobo rojo** representa el depredador al que se le aplica la metaheurística.
- Un **lobo negro** representa a otro depredador de la manada.
- El **ciervo** representa a la presa que se quiere cazar.

7.1.2 Problema específico con solución desconocida

También se testearán los algoritmos con un problema con condiciones iniciales aleatorias.

Las condiciones iniciales, representadas en la figura 7.3 del problema son:

- **Posición inicial del depredador**: $P = (45, 90)$
- **Posición inicial de la presa**: $U = (60, 65)$
- **Posiciones de la manada**: $M = \{(10, 15), (80, 80), (70, 30), (20, 10), (60, 25)\}$

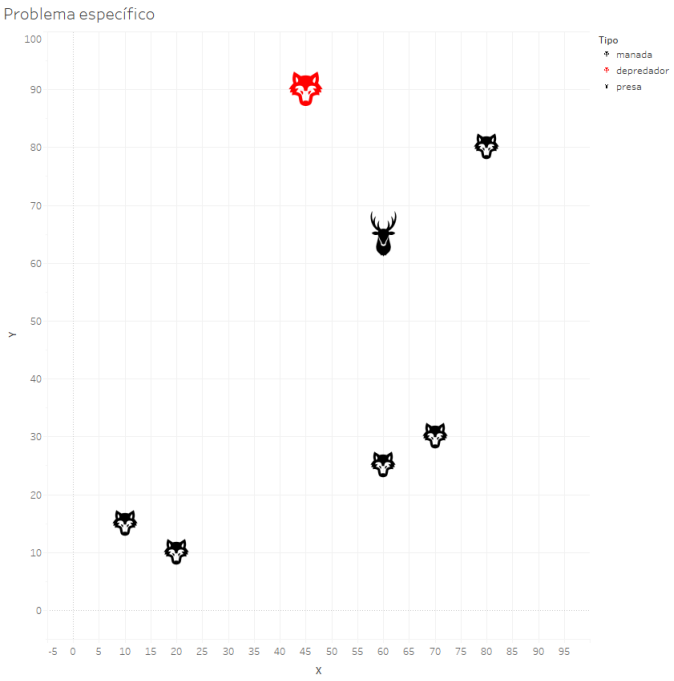

Figura 7.3: Condiciones iniciales del problema. Fuente: Elaboración propia

No se conoce la solución óptima a este problema a priori, tampoco si es única.

7.2 Algoritmo Particle Swarm Optimization

7.2.1 Resolución rápida del problema 7.1.1 con PSO

Se comienza ejecutando el algoritmo con 50 iteraciones para comprobar si se puede llegar a una solución óptima de forma rápida. Definimos el error de una solución como la distancia euclídea entre la solución encontrada y la solución óptima del problema 7.1.1 que es la posición (15,15).

- Con **3 agentes** obtenemos la solución $P = (14'648, 11'739)$ con un error de 3'27 unidades. No es la solución óptima pero se encuentra cerca de ella a pesar de haber hecho sólo 50 iteraciones.

- Con **5 agentes** obtenemos la solución $P = (14'729, 13'003)$ con un error de 2 unidades. Una solución más cercana a la óptima.

- Con **10 agentes** obtenemos la solución $P = (15'013, 14'813)$ con un error de 0'18 unidades. Prácticamente la solución que buscamos.

En la figura 7.4 se muestra un gráfico con mostrando el error que cada ejecución del algoritmo tiene. Se ve que el algoritmo con 5 agentes converge de una manera rápida hacia la solución, la explotación comienza muy pronto. Sin embargo, con 3 y 10 agentes las soluciones no comienzan a converger hasta las iteraciones finales.

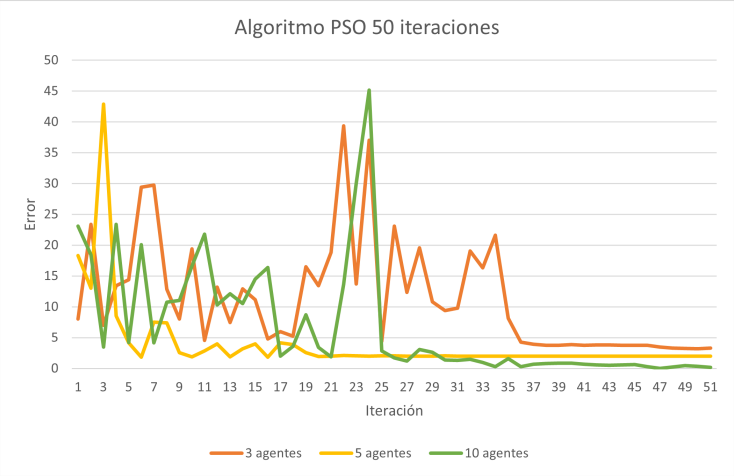

Figura 7.4: Gráfica de ejecución del algoritmo PSO con 50 iteraciones. Fuente: Elaboración propia

Podemos decir que éste algoritmo converge rápidamente hacia la solución óptima con un número reducido de iteraciones.

7.2.2 Resolución lenta para el problema 7.1.1 con PSO

Ahora se ejecuta el algoritmo con 500 iteraciones para comprobar si la convergencia hacia la solución óptima que se observa en la ejecución anterior se mantiene en el tiempo.

- Con **3 agentes** obtenemos la solución $P = (14'999, 14'938)$ con un error de 0'06 unidades.
- Con **5 agentes** obtenemos la solución $P = (14'999, 14'997)$ con un error de 0'002 unidades.
- Con **10 agentes** obtenemos la solución $P = (15, 14'999)$ con un error de 0 unidades.

Las tres ejecuciones han conseguido llegar a la solución óptima.

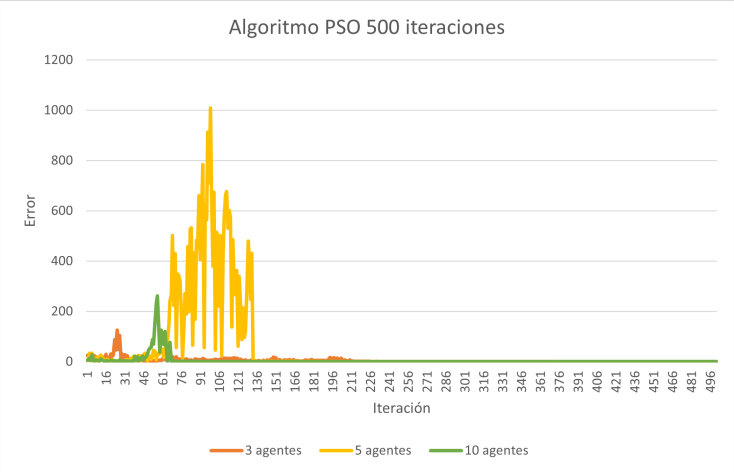

Figura 7.5: Gráfica de ejecución del algoritmo PSO con 500 iteraciones. Fuente: Elaboración propia

Ha ocurrido algo distinto cuando se ejecuta el algoritmo con más iteraciones: como se aprecia en la figura 7.5, con 5 agentes las soluciones tardan en converger, es decir, la fase de exploración es más duradera al contrario de lo que podría parecer en la ejecucion 7.2.1. La misma inversión ha ocurrido con 10 y 3 agentes, se converge mucho más rápido hacia la solución en ejecuciones con muchas iteraciones.

7.2.3 Resultados para el problema 7.1.2 con PSO

Dado que para este problema no conocemos la solución óptima, la calidad de las soluciones se medirá con su optimalidad, que es la máxima distancia que hay entre los depredadores de la manada y la presa. Se harán 500 iteraciones para resolver el problema:

- Con **3 agentes** obtenemos la solución $P = (78'58, 83'58)$.
- Con **5 agentes** obtenemos la solución $P = (63'774, 68'774)$.
- Con **10 agentes** obtenemos la solución $P = (84'507, 89'507)$.

Todas ellas tienen la misma optimalidad: 55'71 unidades es la máxima distancia a los demás miembros de la manada a la que se encontrará la presa cuando el depredador se mueva a esa posición.

Como se ve en la figura 7.6 tanto con 3 agentes como con 10, se converge rápidamente a una solución óptima. La fase de exploración vuelve a ser más duradera con 5 agentes.

Figura 7.6: Gráfica de ejecución del algoritmo PSO para el problema con solución desconocida. Fuente: Elaboración propia

Sólo se muestran 200 iteraciones en la gráfica ya que a partir de este punto las soluciones convergen.

Se puede comprobar el estado final del problema en la figura 7.7, con las tres soluciones obtenidas y la posición final de la presa, que se habrá movido aproximadamente a la posición (45,50).

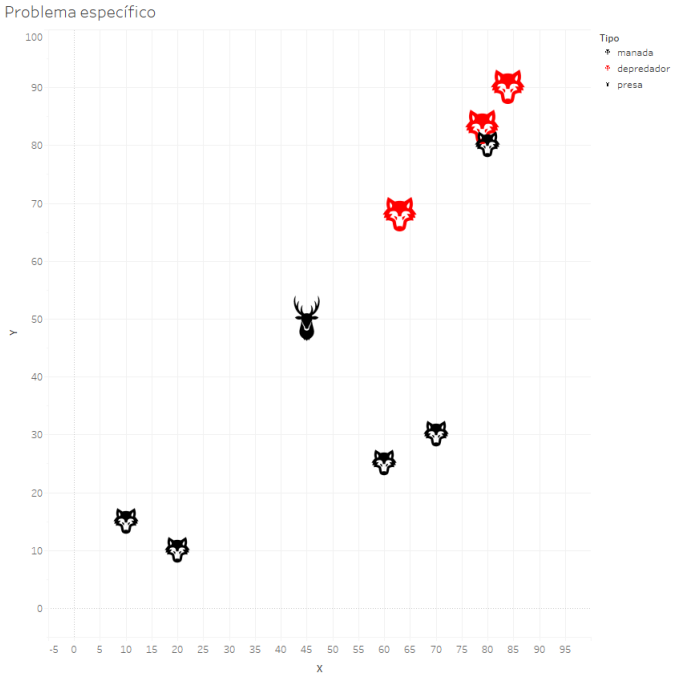

Figura 7.7: Soluciones óptimas obtenidas con PSO. Fuente: Elaboración propia

Es interesante que las tres soluciones sean distintas y tengan la misma optimalidad. Esto significa que existen varias soluciones óptimas. Puede deberse a que la presa siempre se moverá las mimas unidades de espacio independientemente de la distancia a la que se encuentre el depredador. Esta característica hace que las soluciones a este problema se puedan agrupar según lo siguiente:

Sea U la posición de la presa, U' la posición final de la presa, S la solución encontrada y x su optimalidad (que es la máxima distancia de U' a los depredadores), podemos decir que cualquier otra solución S' tal que el ángulo entre el vector $(1, 0)$ y el vector $\vec{US'}$ sea el mismo que con el vector $\vec{US}$ tendrán la misma optimalidad mientras que $d(U', S)$ y $d(U', S')$ sean menores que x.

7.2.4 Demostración del algoritmo PSO

En este vídeo `https://www.youtube.com/watch?v=yvNJa6JcPy8` se muestra el resultado de usar el algoritmo PSO para la simulación de caza. Cada depredador actualiza la posición a la que se quiere mover cada 15 frames de simulación. En el vídeo podemos ver puntos rojos, que representan a los depredadores, y puntos azules que representan a las

presas.

Los parámetros que se utilizan son:

- 50 iteraciones
- 3 agentes de búsqueda
- $c1 = 2.$
- $c2 = 2.$

7.3 Algoritmo Grey Wolf Optimizer

7.3.1 Resolución rápida del problema 7.1.1 con GWO

Se comienza ejecutando el algoritmo con 50 iteraciones para comprobar si se puede llegar a una solución óptima de forma rápida.

- Con **4 agentes** obtenemos la solución $P = (15'643, 7'772)$ con un error de 7'25 unidades. Está lejos de ser la solución óptima.
- Con **7 agentes** obtenemos la solución $P = (15'272, 11'217)$ con un error de 3'79 unidades. Algo mejor que la anterior, aun así sigue sin ser una buena solución.
- Con **10 agentes** obtenemos la solución $P = (14'082, 11'552)$ con un error de 3'56 unidades. Una solución un poco más cercana a la óptima.

En la figura 7.8, se ven zonas planas, esto es porque la mejor solución no se actualiza hasta encontrar una mejor. Cada llanura es una zona de exploración. Se puede asegurar que este algoritmo no obtiene soluciones óptimas en ejecuciones con pocas iteraciones, el menor error de los 3 experimentos en 3,56 el cual es bastante grande considerando que se comienza relativamente cerca de la solución óptima.

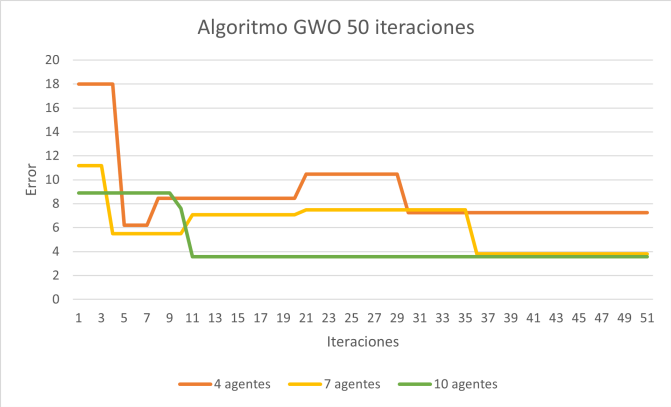

Figura 7.8: Gráfica de ejecución del algoritmo GWO con 50 iteraciones. Fuente: Elaboración propia

Una posible explicación a su bajo rendimiento es que si hay N agentes de búsqueda, sólo $N-3$ actualizan sus soluciones ya que los 3 mejores no realizan movimientos.

7.3.2 Resolución lenta para el problema 7.1.1 con GWO

Se realiza otro experimento con 500 iteraciones para comprobar si las soluciones obtenidas convergen hacia la solución óptima con un mayor tiempo de computación.

- Con **4 agentes** obtenemos la solución $P = (15'03, 12'607)$ con un error de 2'39 unidades. Sigue sin ser una buena solución.

- Con **7 agentes** obtenemos la solución $P = (15'035, 13'075)$ con un error de 1'92 unidades. Una pequeña mejora pero aún lejos de la solución óptima.

- Con **10 agentes** obtenemos la solución $P = (14'978, 14'293)$ con un error de 0'7 unidades. Una mejora significativa en cuanto a la anterior ejecución.

Ninguno de los tres experimentos ha dado un resultado bueno comparado con el algoritmo PSO.

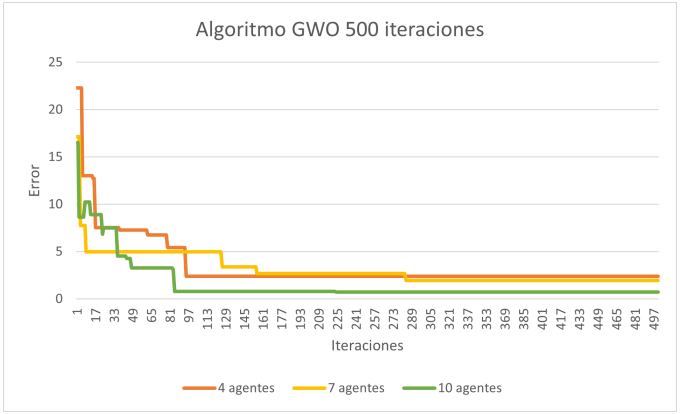

Figura 7.9: Gráfica de ejecución del algoritmo GWO con 500 iteraciones. Fuente: Elaboración propia

Podemos deducir de la figura 7.9, que el algoritmo GWO no es bueno en la fase de explotación.

7.3.3 Resultados para el problema 7.1.2 con GWO

Dado que para este problema no conocemos la solución óptima, la calidad de las soluciones se medirá con su optimalidad, que es la máxima distancia que hay entre los depredadores de la manada y la presa. Se harán 500 iteraciones para resolver el problema:

- Con **4 agentes** obtenemos la solución $P = (89'084, 87'848)$ con 55'84 unidades como optimalidad.

- Con **7 agentes** obtenemos la solución $P = (70'182, 75'212)$ con 55'71 unidades como optimalidad.

- Con **10 agentes** obtenemos la solución $P = (78'398, 82'756)$ con 55'71 unidades como optimalidad.

Aproximadamente 56 unidades es la máxima distancia a los demás miembros de la manada a la que se encontrará la presa cuando el depredador se mueva a la posición calculada. Se obtiene el mismo resultado que con el algoritmo PSO.

Como se ve en la figura 7.10 se converge rápidamente a una solución óptima.

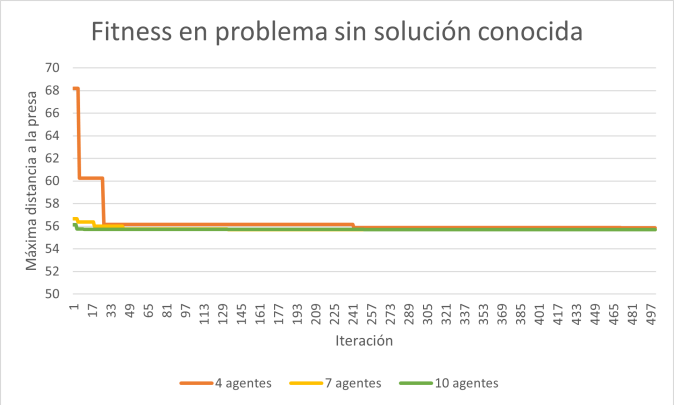

Figura 7.10: Gráfica de ejecución del algoritmo GWO para el problema con solución desconocida. Fuente: Elaboración propia

Se puede comprobar el estado final del problema en la figura 7.11, con las tres soluciones obtenidas y la posición final de la presa, que se habrá movido aproximadamente a la posición (45,50). Las soluciones obtenidas con en algoritmo GWO consiguen el mismo estado final de la presa.

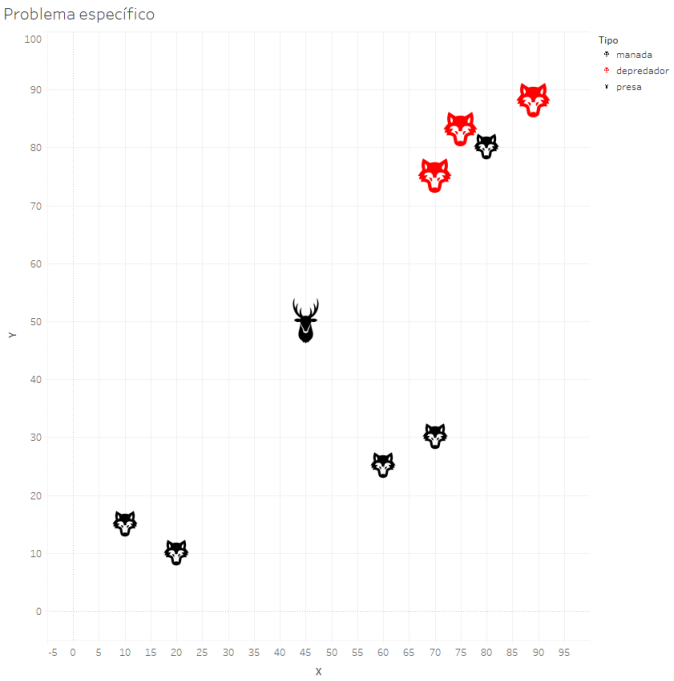

Figura 7.11: Soluciones óptimas obtenidas con GWO. Fuente: Elaboración propia

7.3.4 Demostración del algoritmo GWO

En este vídeo `https://www.youtube.com/watch?v=T01smXHUaCQ` se muestra el resultado de usar el algoritmo GWO para la simulación de caza. Cada depredador actualiza la posición a la que se quiere mover cada 15 frames de simulación. En el vídeo podemos ver puntos rojos, que representan a los depredadores, y puntos azules que representan a las presas.

Los parámetros que se utilizan son:

- 200 iteraciones
- 10 agentes de búsqueda

7.4 Algoritmo Whale Optimization

Es necesario recalcar que este algoritmo ha sido el único que ha necesitado una restricción en las soluciones factibles: deben estar a menos de 100 unidades de distancia de la posición del depredador. Si no se impone tal restricción, las soluciones nunca convergen.

7.4.1 Resolución rápida del problema 7.1.1 con WOA

Se comienza ejecutando el algoritmo con 50 iteraciones para comprobar si se puede llegar a una solución óptima de forma rápida.

- Con **3 agentes** obtenemos la solución $P = (15'072, 14'175)$ con 0'82 unidades de error. Cercana a la solución óptima para el número de agentes e iteraciones empleados.
- Con **5 agentes** obtenemos la solución $P = (15'037, 13'259)$ con 1'74 unidades de error. Peor resultado que con 3 agentes.
- Con **10 agentes** obtenemos la solución $P = (14'949, 12'451)$ con 2'54 unidades de error. El peor resultado de los tres experimentos.

En la figura 7.12, se observa una disminución del error mucho más gradual que con los otros algoritmos, aunque el algoritmo parece funcionar mejor con menos agentes y un

número reducido de ejecuciones. Esto puede deberse a que cuantos más agentes tenga la metaheurística, más exploración realiza antes de pasar a la fase de explotación. En el siguiente experimento se comprobará si es cierto.

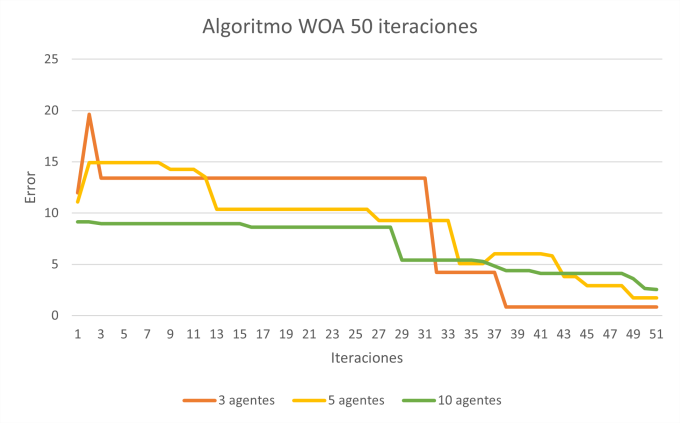

Figura 7.12: Gráfica de ejecución del algoritmo WOA con 50 iteraciones. Fuente: Elaboración propia

7.4.2 Resolución lenta para el problema 7.1.2 con WOA

Se realiza otro experimento con 500 iteraciones para comprobar si las soluciones mejoran con un número mayor de iteraciones.

- Con **3 agentes** obtenemos la solución $P = (15'021, 14'098)$ con un error de 0.91 unidades. Una buena solución con tan solo 3 agentes.
- Con **5 agentes** obtenemos la solución $P = (14'956, 14'061)$ con un error de 0.93 unidades. Una pequeña mejora pero aún lejos de la solución óptima.
- Con **10 agentes** obtenemos la solución $P = (14'999, 14'809)$ con un error de 0.19 unidades. Prácticamente la solución óptima.

En comparación con la ejecución anterior se ve que con las tres configuraciones hay un comportamiento similar en cuanto a la convergencia a las soluciones. Además ha ocurrido lo contrario que en la resolución rápida del problema: con 10 agentes se ha pasado mucho más rápido a la fase de explotación que con 3. Este algoritmo es capaz de obtener buenas soluciones, aunque no tan buenas como el algoritmo PSO. En la figura 7.13 se aprecia

una convergencia lenta y gradual hacia la solución. Esta metaheurística parece tener un buen equilibrio entre exploración y explotación.

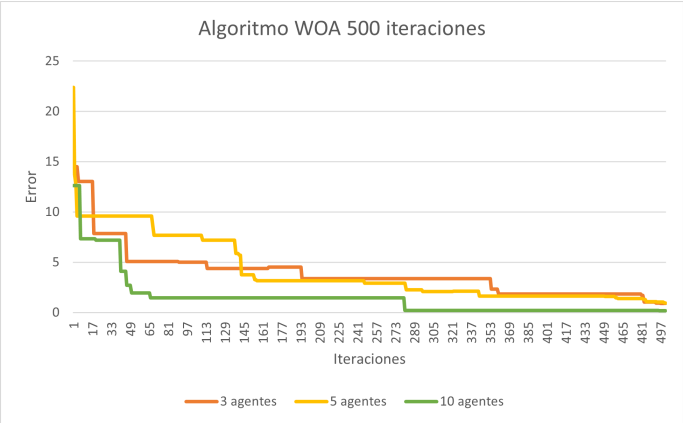

Figura 7.13: Gráfica de ejecución del algoritmo WOA con 500 iteraciones. Fuente: Elaboración propia

7.4.3 Resultados para el problema 7.1.2 con WOA

Dado que para este problema no conocemos la solución óptima, la calidad de las soluciones se medirá con su optimalidad, que es la máxima distancia que hay entre los depredadores de la manada y la presa. Se harán 500 iteraciones para resolver el problema:

- Con **3 agentes** obtenemos la solución $P = (88.702, 93.713)$ con 55'71 unidades como optimalidad.

- Con **5 agentes** obtenemos la solución $P = (88.611, 93.610)$ con 55'71 unidades como optimalidad.

- Con **10 agentes** obtenemos la solución $P = (88.746, 93.752)$ con 55'71 unidades como optimalidad.

De nuevo llegamos a soluciones con la misma optimalidad que con algoritmos anteriores. En este caso las soluciones son prácticamente las mismas

Como se ve en la figura 7.14 se converge rápidamente a una solución óptima.

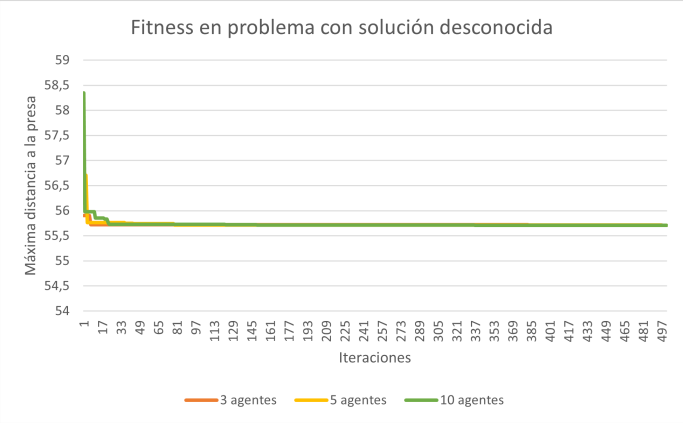

Figura 7.14: Gráfica de ejecución del algoritmo WOA para el problema con solución desconocida. Fuente: Elaboración propia

Podemos ver las soluciones en la figura 7.15, con las tres soluciones obtenidas prácticamente superpuestas y la posición final de la presa, de nuevo, en el punto (45,50).

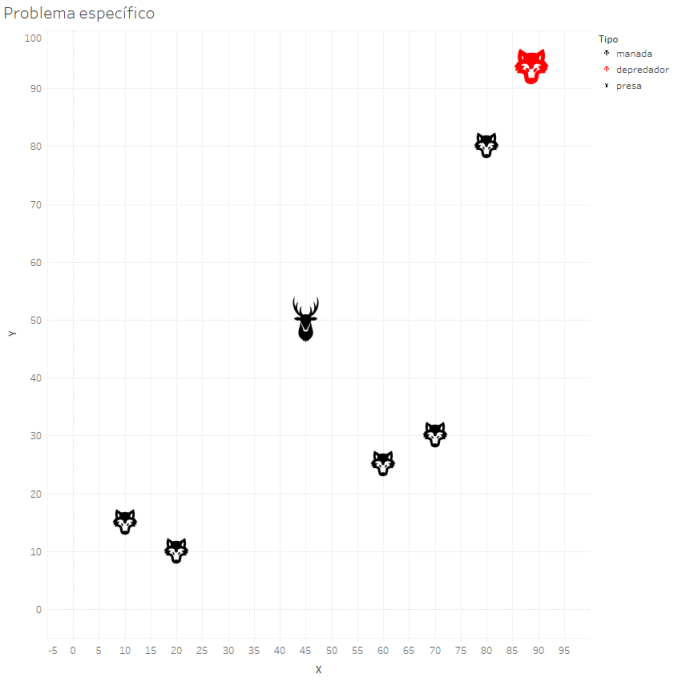

Figura 7.15: Soluciones óptimas obtenidas con WOA. Fuente: Elaboración propia

7.4.4 Demostración del algoritmo WOA

En este vídeo `https://www.youtube.com/watch?v=8X-GDPYd9OI&t=1s` se muestra el resultado de usar el algoritmo WOA para la simulación de caza. Cada depredador actualiza

la posición a la que se quiere mover cada 15 frames de simulación. En el vídeo podemos ver puntos rojos, que representan a los depredadores, y puntos azules que representan a las presas. Los parámetros que se utilizan son:

- 100 iteraciones
- 5 agentes de búsqueda

7.5 Comparativa de algoritmos

Dado que todas la metaheurísticas han tenido prácticamente el mismo rendimiento y soluciones en el problema 7.1.2 realizaremos la comparación tomando como referencia los resultados del problema 7.1.1 con 500 iteraciones y 10 agentes de búsqueda.

Se ilustran los errores de las soluciones óptimas en la figura 7.16.

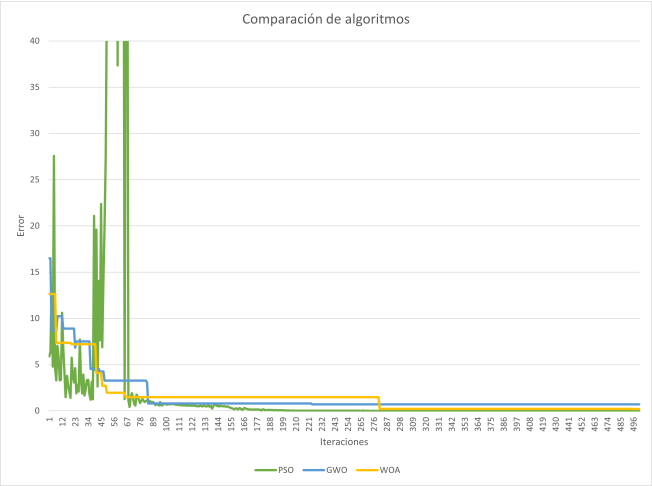

Figura 7.16: Comparación del error en las soluciones para PSO, GWO Y WOA. Fuente: Elaboración propia

En la siguiente tabla se hace la comparación numérica de los resultados.

ALGORITMO	SOLUCIÓN	ERROR
PSO	(15, 14'999)	0
GWO	(14'978, 14'293)	0'7
WOA	(14'999, 14'809)	0'19

Tabla 7.1: Comparación de soluciones y errores de las metaheurísticas.

En cuanto a soluciones la mejor se alcanza usando el algoritmo PSO. Aun así, la gráfica 7.16 nos da más información de cómo se comporta cada metaheurística en una misma situación. Mientras que el PSO comienza con una fase larga de exploración, los algoritmos GWO y WOA combinan la exploración con la explotación desde el inicio de una manera equilibrada, es por eso que el error baja gradualmente a lo largo de toda la ejecución. Tras la fase de exploración, el algoritmo PSO converge rápidamente hacia la solución óptima gracias a haber recorrido una gran parte del espacio de soluciones.

Si tenemos poco poder computacional es mejor usar los algoritmos GWO y WOA ya que proporcionan soluciones aceptables en poco tiempo. Si por el contrario podemos permitirnos una gran capacidad de computación, el algoritmo PSO da mejores resultados a largo plazo gracias a su exhaustiva búsqueda en el espacio de soluciones.

Capítulo 8

Conclusiones y trabajos futuros

En este capítulo se expondrán las conclusiones y se pincelará la posible implementación de una metaheurística genética para simular la evolución de las especies.

8.1 Conclusiones

Tras haber comprendido los conceptos básicos de optimización y metaheurísticas y haberme documentado sobre métodos de modelado de la caza me sentía preparado para afrontar el problema propuesto pero a la hora de usar las metaheurísticas encontré dificultades. Las metaheurísticas son una herramienta muy potente para la optimización como se ha podido comprobar a lo largo de este documento, aun así, comprender al completo cualquiera de los algoritmos para explotarlos al máximo lleva su tiempo. Es necesaria una gran capacidad de análisis para interpretar si una metaheurística está funcionando correctamente. Distinguir entre un error a la hora de escribir el código y una mala configuración de parámetros ha sido una de las partes más difíciles del proyecto. Realizar la comparación de los distintos algoritmos y analizar sus resultados me ayudó a comprender mejor cómo cada uno de ellos se comporta en las fases de exploración y explotación. Los resultados obtenidos han sido satisfactorios, como se puede ver en los vídeos de las demostraciones, emplear metaheurísticas en varios animales al mismo tiempo hace que emerjan interacciones aparentemente complejas que permiten simular la caza en grupo, por lo que podemos confirmar la hipótesis planteada en la sección 1.1.

8.2 Trabajos futuros

Se propone la siguiente regla para la simulación:

Las presas tiene un pseudo-ADN en forma de una cadena de 10 caracteres con distintas letras que definirán las características del animal. Por ejemplo: VTVTTTTTVV (V para velocidad, T para tamaño) significará que el individuo tiene velocidad 4 y tamaño 6.

El orden de los caracteres importan ya que una cría solo tendrá activos los 5 primeros y un adulto los tendrá activos todos.

Cuando dos adultos se reproduzcan y tengan una cría, el ADN de esta se creará mediante un algoritmo genético que combinará los ADNs de sus progenitores.

De esta manera se pretende encontrar la configuración de ADN óptima con la que las presas sean capaces de escapar de los depredadores o resistir su ataque.

Apéndice A

Código fuente

Código del algoritmo *PSO*: https://github.com/Dacarpe03/Ecosystem-Simulator/blob/main/EcoSim/Assets/Model/Hunting/PSOStrategy.cs

Código del algoritmo *GWO*: https://github.com/Dacarpe03/Ecosystem-Simulator/blob/main/EcoSim/Assets/Model/Hunting/GWOStrategy.cs

Código del algoritmo *WOA*: https://github.com/Dacarpe03/Ecosystem-Simulator/blob/main/EcoSim/Assets/Model/Hunting/WOAStrategy.cs

Apéndice B

Manual de usuario

Guía rápida para usar la aplicación.

B.1 Descarga e instalación

Si no dispone del archivo .zip con nombre "*EcoSimPhase2DanielCarmonaPedrajas.zip*" acceda al siguiente enlace y descarge el archivo pinchando sobre él como se muestra en la figura B.1.

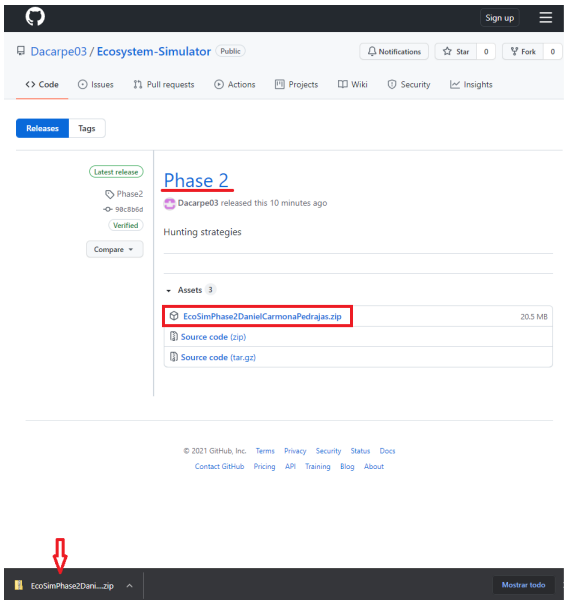

Figura B.1: Guía de descarga

Una vez descargado el archivo es necesario descomprimirlo. A continuación abra la

carpeta y haga doble click en el archivo *EcoSim* (ver figura B.2) para ejecutarlo. Antes de abrirlo, aclarar que la aplicación sólo puede cerrarse utilizando la combinación de teclas *Alt+F4*.

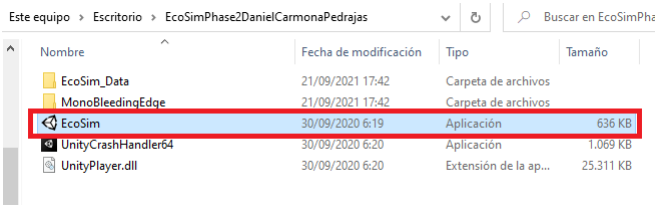

Figura B.2: Abrir el archivo

B.2 Menú de configuración

Al abrir el archivo se abrirá un menú como el que se presenta en la figura B.3.

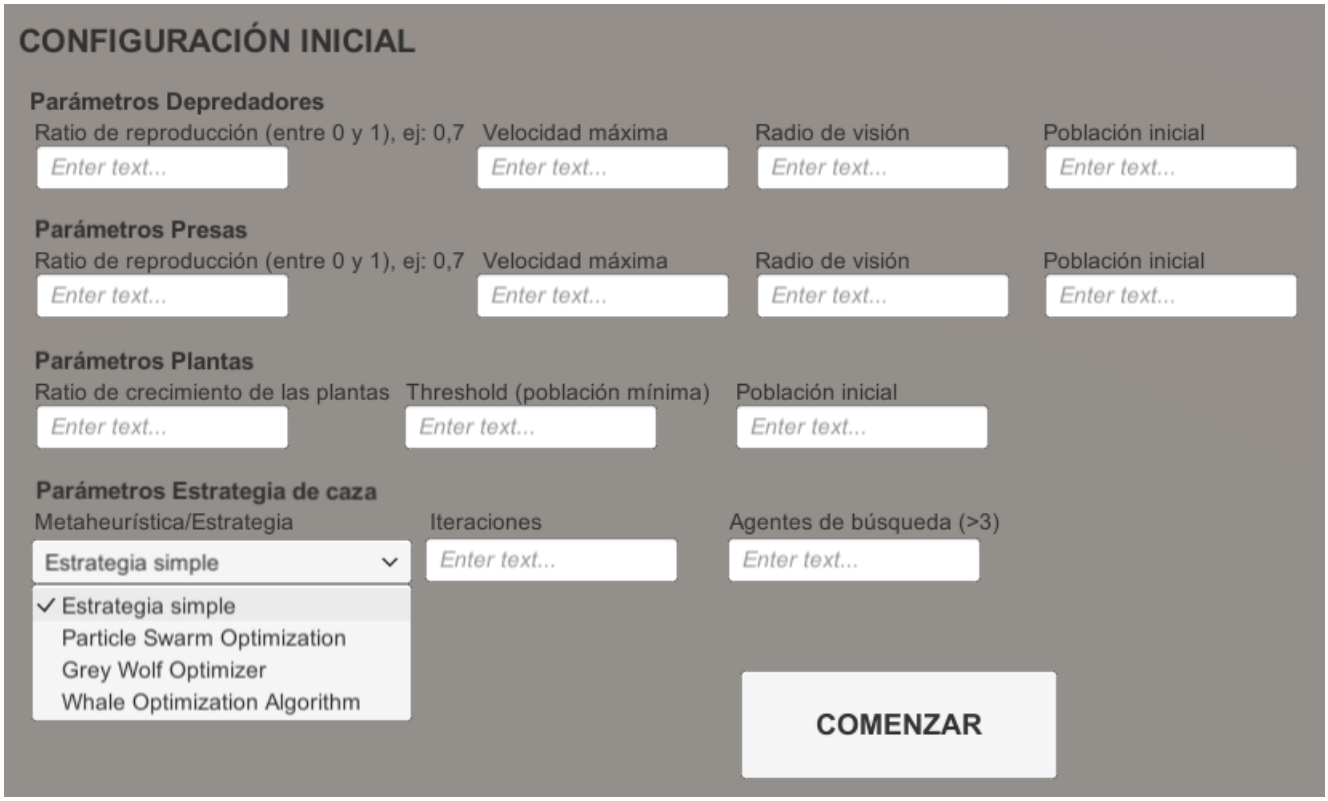

Figura B.3: Menú de configuración

Para cada parámetro relevante en la aplicación se ha añadido la posibilidad de definirlo manualmente por el usuario. Hay un campo de texto que rellenar. Si un campo se deja

en blanco se tomará un valor por defecto por lo que no es necesario rellenar todos los espacios. Es importante que los números decimales se escriban con comas en vez de con puntos, por ejemplo, 0,55.

Los parámetros de depredadores y presas funcionan de la misma forma.

- **Ratio de reproducción** es la probabilidad de que una pareja de supervivientes generen otro animal.
  - Valor por defecto para los **depredadores**: 0,3
  - Valor por defecto para las **presas**: 0,9
- **Velocidad máxima** es para limitar la velocidad que pueden alcanzar los animales. Es necesario tener en cuenta que 30 frames equivalen a 1 segundo, por lo tanto si se quiere hacer que los animales se muevan a 30 unidades por segundo, la velocidad máxima deberá ser 0,1.
  - Valor por defecto para los **depredadores**: 0,6
  - Valor por defecto para las **presas**: 0,55
- **Radio de visión** es el número de unidades de distancia a las que un animal puede detectar a otro, ya pertenezca o no a su misma especie.
  - Valor por defecto para los **depredadores**: 30
  - Valor por defecto para las **presas**: 8
- **Población inicial** es el número de individuos con el que empieza cada especie.
  - Valor por defecto para los **depredadores**: 200
  - Valor por defecto para las **presas**: 12

En cuanto a los parámetros que controlan el comportamiento de las plantas se tiene:

- **Ratio de crecimiento de las plantas** es un multiplicador que se aplica al final de cada etapa de evolución a la población de las plantas. Valor por defecto: 1,7
- **Treshold** es la cantidad mínima de plantas que puede haber, su población nunca será menor a este parámetro. Valor por defecto: 200

- **Población inicial** con la que se comienza la simulación. Valor por defecto: 1200

Por último tenemos la selección de estrategia:

- **Metaheurística/Estrategia** es el método que utilizarán los depredadores para calcular su posición en cada iteración. Por defecto se selecciona Estrategia simple.
  - **Estrategia simple**: los depredadores cazan de manera individual.
  - **Particle Swarm Optimization**: los depredadores cazan en manada actualizando su posición según el algoritmo *PSO*
  - **Grey Wolf Strategy**: los depredadores cazan en manada actualizando su posición según el algoritmo *GWO*
  - **Grey Wolf Strategy**: los depredadores cazan en manada actualizando su posición según el algoritmo *WOA*
- **Iteraciones** que se ejecutarán con cada algoritmo. Si se ha elegido la Estrategia Simple no es necesario indicar ningún valor aquí. Valor por defecto: 200
- **Agentes de búsqueda** que emplea la metaheurística. Si se ha elegido la Estrategia Simple no es necesario indicasr ningún valor aguí. Valor por defecto: 4

Al pulsar el botón *Comenzar* se ejecutará la simulación con los parámetros definidos. Los cubos azules representan a las presas, cuando llegan a la zona segura (cuando su posición de la coordenada x es mayor que 400 unidades) se detienen. Los cubos rojos representan a los depredadores y seguirán a las presas hasta la zona segura, si cazan a una presa esta se quedará parada.

Bibliografía

[1] C. V. de Lima, "¿Qué es la optimización?."
https://www.cassotis.com/insights/que-es-la-optimizacion.
Último acceso: 14/09/2021.

[2] D. G. González, "Diapositivas Simulación y Metaheurísticas, Master en Ingeniería de Sistemas de Decisión UCM."

[3] A. Duarte, "DAA,tema 1: Eficiencia algoritmica."

[4] M. G. d. L. Sergio Saugar, "Teoría de Autómatas, Tema 7: Máquinas de Turing."

[5] C. E. Maldonado, "Un problema fundamental en la investigación: Los problemas P vs NP."
https://www.redalyc.org/pdf/5177/517751544002.pdf.
Último acceso: 14/09/2021.

[6] F. S. Caparrini, "Metaheurísticas para búsqueda y optimización."
http://www.cs.us.es/~fsancho/?e=207.
Último acceso: 14/09/2021.

[7] C. Muro, "w*olf-pack hunting strategies emerge from simple rules in computational simulations.*"
https://www.sciencedirect.com/science/article/pii/S0376635711001884#!
Último acceso: 19/09/2021.

[8] A. Weitzenfeld, "a biologically-inspired wolf pack multiple robot hunting model."

https://www.researchgate.net/publication/238005968_A_Biologically-Inspired_Wolf_Pack_Multiple_Robot_Hunting_Model.
Último acceso: 19/09/2021.

[9] C. Reynolds, "Boids."
https://www.red3d.com/cwr/boids/.
Último acceso: 06/11/2020.

[10] A. Mirjalili, "Learn Particle Swarm Optimization."
https://www.youtube.com/watch?v=JhgDMAm-imI&t=949s.
Último acceso: 15/09/2021.

[11] A. L. Seyedali Mirjalili, Seyed Mohammad Mirjalili, "Grey Wolf Optimizer."
https://www.sciencedirect.com/science/article/pii/S0965997813001853.
Último acceso: 16/09/2021.

[12] A. L. Seyedali Mirjalili, "Whale Optimization Algorithm."
https://www.sciencedirect.com/science/article/pii/S0965997816300163.
Último acceso: 16/09/2021.