



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

*Algoritmos bioinspirados
para NetLogo*

Autor

José Antonio Martín Melguizo

Directora

Rocío Celeste Romero Zaliz



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, julio de 2021

Algoritmos bioinspirados para NetLogo

José Antonio Martín Melguizo

Palabras clave: NetLogo, Algoritmos Bioinspirados, Algoritmos Genéticos, Algoritmos de Colonia de Hormigas, Algoritmos de Enjambre de Partículas, Problemas de ingeniería.

Resumen

El ámbito de las disciplinas STEM es un contexto idóneo para el desarrollo del pensamiento lógico-matemático a través de la informática. El uso de herramientas digitales surgen como una forma de promover, implicar y acercar a los estudiantes el conocimiento y competencias necesarias para el desarrollo personal y profesional de éstos en nuestra sociedad contemporánea. Se propone utilizar NetLogo, un entorno de modelado programable para simular fenómenos naturales y sociales. Permite al estudiante el desarrollo, la interacción y visualización de modelos y objetos abstractos (tanto en 2D como 3D). Se han desarrollado una serie de modelos NetLogo como recurso para mejorar la enseñanza de algoritmos bioinspirados: Algoritmos genéticos (GA), Algoritmos de Colonia de Hormigas (ACO) y Algoritmos de Enjambre de Partículas (PSO). A través de estos modelos se fomenta otra forma de interactuar con este tipo de algoritmos aplicados a problemas de optimización básicos y propios de la ingeniería, mejorando la comprensión y el uso adecuado de los distintos parámetros de cada algoritmo por parte del estudiantado.

Bioinspired algorithms for NetLogo

José Antonio Martín Melguizo

Keywords: NetLogo, Bioinspired algorithms, Genetic Algorithms, Ant Colony Optimization, Particle Swarm Optimization, Engineering Problems.

Abstract

The field of STEM disciplines is an ideal context for the development of logical and mathematical thinking of students through computer science. The use of digital tools is a way to promote, involve and bring them closer to the knowledge and skills necessary for their personal and professional development in our modern society. We propose the use of NetLogo, a programmable modeling environment, to simulate natural and social phenomena. It allows a student to develop, interact and visualize abstract models and objects (both 2D and 3D). A series of NetLogo models have been developed as a resource to enhance the teaching of bio-inspired algorithms: Genetic Algorithms (GA), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). The application of these algorithms to basic optimization engineering problems promotes the improvement of students' understanding by using friendly interactions with both basic and advanced parameter configurations.

Yo, **José Antonio Martín Melguizo**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77561280J, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Antonio Martín Melguizo

Granada a 22 de junio de 2021.

D. Rocío Celeste Romero Zaliz, Profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Algoritmos bioinspirados para NetLogo*, ha sido realizado bajo su supervisión por **José Antonio Martín Melguizo**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 9 de julio de 2021.

La directora:

Rocío Celeste Romero Zaliz

ÍNDICE GENERAL

1. Introducción	21
1.1. Relevancia de las TIC en Educación STEM	21
1.2. Justificación del proyecto	22
1.3. Estructura de la memoria	23
2. Gestión y Planificación	25
2.1. Objetivos	25
2.2. Tareas	26
2.3. Metodología de desarrollo	27
2.3.1. SCRUM	29
2.4. Gestión del código y documentación	31
2.5. Gestión del tiempo	32
2.5.1. Planificación temporal	32
2.5.2. Planificación de los <i>Sprints</i>	32
2.6. Gestión de recursos	33
2.6.1. Recursos humanos	33
2.6.2. Recursos materiales para el desarrollo	33
2.6.3. Recursos software	34
2.6.4. Recursos de comunicación y documentación	34
2.7. Gestión de costes	35
2.7.1. Costes de recursos humanos	35
2.7.2. Costes de recursos materiales	35
2.7.3. Costes de recursos software	36
2.7.4. Costes adicionales	36

2.8. Presupuesto total	37
3. Análisis y Diseño	39
3.1. Especificación de requisitos de los modelos	39
3.2. Historias de usuario	39
3.2.1. Tarjetas de historias de usuario	40
3.3. Requisitos adicionales	43
3.4. Análisis de riesgos	43
4. Estado del Arte	45
4.1. Herramientas para el modelado de agentes	47
4.1.1. Swarm	47
4.1.2. Repast	47
4.1.3. MASON	48
4.1.4. StarLogo	49
4.1.5. NetLogo	50
4.1.6. AnyLogic	50
4.2. Tabla Comparativa	52
5. NetLogo	53
5.1. Características de NetLogo	53
5.2. Modelos de NetLogo	56
6. Algoritmos Bioinspirados	59
6.1. Algoritmos genéticos (GA)	59
6.2. Algoritmos de enjambre de partículas (PSO)	64
6.3. Algoritmos de colonias de hormigas (ACO)	67
6.3.1. Resolviendo el Problema del Viajante de Comercio . .	67
7. Problemas de optimización	73
7.1. Pressure Vessel Design	74
7.2. Tension/Compression Spring Design	75
7.3. 25-Bar Space Truss Design	77
7.3.1. Adaptación de ACO para Truss Design	79
8. Modelos implementados	85
8.1. Diagramas de Clases	86
8.2. Algoritmos Genéticos	88
8.2.1. Modelo Básico	88
8.2.2. Modelo Específico - Pressure Vessel Design Problem .	91
8.3. Colonia de hormigas	94
8.3.1. Modelo Básico	94
8.3.2. Modelo Específico - 25-Bar Space Truss Design	98
8.4. Enjambre de partículas	101
8.4.1. Modelo Básico	101

8.4.2. Modelo Específico - Tension/Compression Spring Design Problem	105
9. Pruebas	107
9.1. Sistema de tests específico	107
9.2. Resultados para modelos básicos	108
9.2.1. Algoritmos genéticos	108
9.2.2. Algoritmos de enjambre de partículas	111
9.2.3. Algoritmos de colonia de hormigas	113
9.3. Resultados para modelos específicos	114
9.3.1. Resultados para Pressure Vessel	115
9.3.2. Resultados para Tension/Compression Spring	116
9.3.3. Resultados para 25-Bar Space Truss	117
10. Conclusiones y Trabajo Futuro	119
10.1. Conclusiones	119
10.2. Trabajo Futuro	121
Bibliografía	127
A. Manual de usuario	131
A.1. Introducción a Netlogo	131
A.1.1. Instalación	131
A.1.2. Entorno de trabajo	132
A.1.3. Programando en Netlogo	137
A.2. Apertura y ejecución de modelos	138

ÍNDICE DE FIGURAS

2.1. Fases en una metodología ágil	28
2.2. Marco de trabajo SCRUM	29
2.3. Diagrama de Gantt	32
4.1. Logo de Swarm	47
4.2. Logo de Repast	47
4.3. Logo de Mason	48
4.4. Logo de StarLogo	49
4.5. Logo de NetLogo	50
4.6. Logo de AnyLogic	51
5.1. Taxonomía de los modelos de ejemplo de NetLogo	56
5.2. Clasificación de modelos de ejemplo de NetLogo	57
6.1. Diagrama de flujo para Algoritmo Genético general	60
6.2. Selección por torneo	61
6.3. Selección por ruleta	61
6.4. Tipos de cruce implementados	62
6.5. Operador de mutación	63
6.6. Atracción de una partícula en PSO	64
6.7. Diagrama de flujo para PSO	66
6.8. Esquema de funcionamiento ACO	67
6.9. Resolviendo TSP a través de ACO	70
6.10. Diagrama de flujo para ACO	72

7.1.	Representación del problema <i>Pressure Vessel Design</i>	74
7.2.	Representación del problema <i>Tension/Compression Spring</i> . .	76
7.3.	Representación del problema <i>25-Bar Space Truss</i>	78
7.4.	Representación de caminos virtuales	80
7.5.	Ejemplo de representación del problema	81
8.1.	Modelos implementados en NetLogo	85
8.2.	Diagrama de clases para modelos GA	86
8.3.	Diagrama de clases para modelos PSO	87
8.4.	Diagrama de clases para modelos ACO	87
8.5.	Interfaz modelo GA-Básico	88
8.6.	Configuración de operadores GA-Básico	89
8.7.	Configuración del esquema GA-Básico	90
8.8.	Interfaz modelo GA-Específico	91
8.9.	Configuración del dominio de <i>Pressure Vessel</i>	92
8.10.	Configuración de restricciones <i>Pressure Vessel</i>	93
8.11.	Interfaz modelo ACO-Básico	94
8.12.	Configuración de visualización ACO Básico	95
8.13.	Configuración de modelo ACO Básico	95
8.14.	Visualización de feromonas del modelo	97
8.15.	Interfaz modelo ACO-Específico	98
8.16.	Parámetros básicos de ACO Específico	99
8.17.	Parámetros particulares de ACO Específico	99
8.18.	Visualización del mundo en Truss Design	100
8.19.	Gráficos para ACO Específico	101
8.20.	Interfaz modelo PSO-Básico	102
8.21.	Configuración de visualización PSO Básico	102
8.22.	Espacio de búsqueda de función sencilla	103
8.23.	Espacio de búsqueda aleatorio	103
8.24.	Configuración de parámetros para PSO	104
8.25.	Gráficos de rendimiento para PSO	104
8.26.	Interfaz modelo PSO-Específico	105
8.27.	Configuración del dominio de <i>Spring Design</i>	106
9.1.	Comparativa esquema evolución para modelo GA Básico . .	109
9.2.	Comparativa parámetros para modelo GA Básico	109
9.3.	Comparativa atracción para modelo PSO Básico	111
9.4.	Comparativa inercia para modelo PSO Básico	112
9.5.	Comparativa parámetros para modelo ACO Básico	113
A.1.	Interfaz de Netlogo	132
A.2.	Pestañas principales en Netlogo	133
A.3.	Tipos de botones para la Interfaz	134
A.4.	Ajustes del mundo de Netlogo	135

A.5. Topología del mundo de Netlogo	135
A.6. Editor de agentes de Netlogo	136
A.7. Paleta de colores de Netlogo	136

ÍNDICE DE CUADROS

2.1. Diferencias entre metodologías tradicionales y ágiles	28
2.2. Costes asociado a los recursos humanos	35
2.3. Costes materiales	36
2.4. Presupuesto total del proyecto	37
3.1. Listado inicial de historias de usuario	40
3.2. Historia de Usuario - HU.1	41
3.3. Historia de Usuario - HU.2	41
3.4. Historia de Usuario - HU.3	42
3.5. Historia de Usuario - HU.4	42
3.6. Tabla con los riesgos del proyecto	44
3.7. Matriz <i>probabilidad-impacto</i> de riesgos	44
4.1. Tabla comparativa de herramientas de modelado de agentes .	52
7.1. Agrupamiento de elementos del problema	79
7.2. Caso de carga para el problema	79
9.1. Tabla comparativa para <i>Pressure Vessel Design</i> usando GA .	115
9.2. Tabla comparativa para <i>Tension/Compression Spring Design</i> usando PSO	116
9.3. Tabla comparativa para <i>25-Bar Space Truss Design</i> usando ACO	117

CAPÍTULO

1

INTRODUCCIÓN

Uno de los temas más interesantes y con más aplicaciones en el ámbito de las ingenierías son los algoritmos de optimización. La optimización consiste en la búsqueda sistemática de la solución óptima dentro de un espacio de búsqueda a través de algoritmos numéricos de optimización [28].

Son imprescindibles para lograr el diseño óptimo en un producto permitiendo reducir costes a la hora de su fabricación. Son diversas las técnicas computacionales disponibles para la aproximación a la solución óptima en un tiempo razonable.

Dentro de este tipo de algoritmos se pueden encontrar aquellos clasificados como *soft-computing*. Entre sus principales técnicas se encuentran los algoritmos bioinspirados, tales como los algoritmos genéticos, de colonias de hormigas y enjambres de partículas.

1.1. Relevancia de las TIC en Educación STEM

Toda persona necesita una formación base sólida en ciencias, tecnología y matemáticas básicas, útil para poder tomar decisiones, resolver retos y comprender mejor los fenómenos naturales y tecnológicos de nuestro entorno cotidiano [35].

Esta formación está ligada a la educación en disciplinas STEM (acrónimo de los términos en inglés *Science, Technology, Engineering and Mathematics*), que permite enriquecer no sólo al estudiante al que se le imparte, también

a la sociedad a la que pertenece.

Una vez que el aprendizaje se hace de manera coherente con el marco de la educación STEM, las prácticas en las que se implican los estudiantes deben ser análogas a las prácticas realizadas por expertos de este entorno.

Ya se propone con anterioridad recrear en el aula experimentos y problemas propios del mundo profesional, para crear junto a la teoría asociada a éstos, una didáctica más enriquecedora a la hora de impartir clase [24].

Por ejemplo, al crear una **simulación**, muchos docentes obtienen una nueva perspectiva del fenómeno que están tratando de explicar, lo que casi siempre aumenta el entusiasmo por el uso de esta tecnología de los alumnos [12]. El uso de herramientas TIC supone una serie de ventajas a la hora de la reproducibilidad y simulación de estos experimentos por parte del alumnado.

En la simulación de fenómenos naturales o sociales, estas herramientas nos permite superar las barreras de tiempo y distancia de dichos fenómenos, así como hacerlos más cercanos a los estudiantes ya que que por coste, complejidad o seguridad no podrían ser reproducibles en el aula [34].

1.2. Justificación del proyecto

Dada la importancia de estos algoritmos bioinspirados para la resolución de problemas de optimización, estos temas forman parte del contenido de la asignatura *Optimización y Computación Inteligente (OCI)*, asignatura obligatoria del Máster en Estructuras¹ de la Universidad de Granada.

Tras varios años de impartición de dicha asignatura, los profesores de la misma, incluyéndose la tutora de este proyecto, han notado la dificultad de los estudiantes para comprender el funcionamiento y potenciales aplicaciones de los algoritmos bioinspirados. La carencia, en muchos casos, de conocimientos de programación por parte del alumnado, hacen imposible abordar el problema desde ese punto de vista.

Si bien existen algunos recursos para la docencia de este tipo de algoritmos (mayoritariamente en inglés), tales como applets o vídeos, no se ha encontrado una herramienta que consiga adaptarse a OCI.

Es por ello que propongo crear contenido didáctico propio, novedoso, útil e interactivo que ayude a los estudiantes de dicha asignatura a comprender y utilizar de forma correcta estas técnicas bioinspiradas.

¹<https://masteres.ugr.es/iestructuras/>

1.3. Estructura de la memoria

La presente memoria está dividida en 10 capítulos y un apéndice:

- Capítulo 1. **Introducción:** En este primer capítulo se introduce el presente proyecto, se explica la motivación y justificación para el desarrollo del mismo.
- Capítulo 2. **Gestión y Planificación:** En este segundo capítulo se enumeran los objetivos, tareas y se habla de la metodología de desarrollo elegida, así como de la planificación temporal, gestión de recursos y costes del proyecto.
- Capítulo 3. **Análisis y Diseño:** En este tercer capítulo se analizan los requisitos del proyecto a través de historias de usuario. También se hace un análisis de riesgos del proyecto.
- Capítulo 4. **Estado del Arte:** En este cuarto capítulo se hace una revisión de las principales herramientas de modelado de agentes de la actualidad así como una comparativa entre ellas.
- Capítulo 5. **NetLogo:** En este quinto capítulo se habla del entorno de modelado de NetLogo, centrándonos en las principales características del lenguaje y haciendo una taxonomía de los modelos que nos ofrece.
- Capítulo 6. **Algoritmos Bioinspirados:** En este sexto capítulo se introducen los algoritmos bioinspirados que se desarrollarán en el proyecto: algoritmos genéticos, de enjambre de partículas y colonia de hormigas.
- Capítulo 7. **Problemas de optimización:** En este séptimo capítulo se hablan de los problemas de optimización que se contemplarán para ser resueltos a través de los modelos específicos de los algoritmos previamente nombrados.
- Capítulo 8. **Modelos implementados:** En este octavo capítulo se detalla la interfaz y funcionamiento de los modelos básicos y específicos implementados para la simulación de los algoritmos bioinspirados.
- Capítulo 9. **Pruebas:** En este noveno capítulo se comentan los resultados obtenidos en los modelos y se detallan los resultados para los problemas específicos comparándolos con los de otros autores.
- Capítulo 10. **Conclusiones y Trabajo Futuro:** En este décimo capítulo se sintetiza y concluye el proyecto. También se habla sobre posibles futuras líneas de trabajo.

Apéndice A. **Manual de usuario:** Manual de introducción al entorno de desarrollo de NetLogo así como a las sentencias del lenguaje más básicas.

CAPÍTULO

2

GESTIÓN Y PLANIFICACIÓN

2.1. Objetivos

Objetivo General: Desarrollo de modelos de simulación basados en algoritmos bioinspirados con capacidad de visualización y graficas para mostrar el uso de éstos en otros ámbitos distintos de la informática.

Objetivos Específicos:

- **OBJ 1** - Implementar **modelos básicos** para los **algoritmos bioinspirados** más significativos en NetLogo (algoritmos genéticos, colonia de hormigas, enjambre de partículas).
- **OBJ 2** - Implementar para cada uno de los algoritmos bioinspirados, un **modelo** más **específico** que **resuelva** un **problema** de optimización propio de la **ingeniería**.
- **OBJ 3** - Permitir la **visualización** y monitorización del **rendimiento** de los algoritmos en la búsqueda de la solución para cada uno de los modelos desarrollados.
- **OBJ 4** - Realizar **experimentos** a partir de los modelos específicos implementados y **comparar** los **resultados** con los de los autores de los **artículos** en los que se inspiran.

2.2. Tareas

Tareas alineadas con objetivos. Cada objetivo debe tener una o varias tareas que lo implementen.

- **T1** - Implementación de un modelo básico para algoritmos genéticos (OBJ 1)
 - **T1.1** - Lectura e investigación sobre algoritmos genéticos.
 - **T1.2** - Diseño y desarrollo del modelo GA-Basico en Netlogo (OBJ 3).
 - **T1.3** - Evaluación y pruebas del modelo GA-Basico.
- **T2** - Implementación de un modelo básico para algoritmos de enjambre de partículas (OBJ 1)
 - **T2.1** - Lectura e investigación sobre algoritmos de enjambre de partículas.
 - **T2.2** - Diseño y desarrollo del modelo PSO-Basico en Netlogo (OBJ 3).
 - **T2.3** - Evaluación y pruebas del modelo PSO-Basico.
- **T3** - Implementación de un modelo básico para algoritmos de colonia de hormigas (OBJ 1)
 - **T3.1** - Lectura e investigación sobre algoritmos de colonia de hormigas.
 - **T3.2** - Diseño y desarrollo del modelo ACO-Basico en Netlogo (OBJ 3).
 - **T3.3** - Evaluación y pruebas del modelo ACO-Basico.
- **T4** - Implementación de un modelo específico para algoritmos genéticos (OBJ 2)
 - **T4.1** - Investigación y búsqueda sobre problemas de ingeniería resueltos a través de algoritmos genéticos.
 - **T4.2** - Diseño y desarrollo del modelo GA-Basico en Netlogo.
- **T5** - Implementación de un modelo específico para algoritmos de enjambre de partículas (OBJ 2)
 - **T5.1** - Investigación y búsqueda sobre problemas de ingeniería resueltos a través de algoritmos de enjambre de partículas.
 - **T5.2** - Diseño y desarrollo del modelo PSO-Específico en Netlogo (OBJ 3)

- **T6** - Implementación de un modelo básico para algoritmos de colonia de hormigas (OBJ 2)
 - **T6.1** - Investigación y búsqueda sobre problemas de ingeniería resueltos a través de algoritmos de colonia de hormigas.
 - **T6.2** - Diseño y desarrollo del modelo ACO-Específico en NetLogo (OBJ 3)
- **T7** - Comparación de soluciones mostradas en artículos científicos con la solución alcanzada por los modelos específicos implementados. (OBJ 4)
 - **T7.1** - Comparación para el modelo GA-Específico.
 - **T7.2** - Comparación para el modelo PSO-Específico.
 - **T7.3** - Comparación para el modelo ACO-Específico.
- **T8** - Creación de un Manual de Usuario para facilitar el uso e introducción de la herramienta al alumnado.

2.3. Metodología de desarrollo

Hoy día, es muy importante la decisión sobre qué tipo de metodología se va a utilizar para el desarrollo de un proyecto, debe ser una metodología adecuada a éste.

El objetivo general del uso de una metodología es ser capaces de estandarizar, estructurar y organizar la forma de trabajar. Una metodología no es sino una gran herramienta para generar eficiencia a medida que se va utilizando. Se trata de establecer un plan de acción que ayude a reducir riesgos procedentes de posibles retrasos en el desarrollo o incluso un mal planteamiento que provoque el fracaso del proyecto.

Existe una gran variedad de metodologías que se han desarrollado a lo largo de los años, pero podemos diferenciarlas a grandes rasgos entre metodologías tradicionales o ágiles. En la Tabla 2.1 resumo las principales características que distinguen las metodologías tradicionales de las ágiles.

En este proyecto, se parte de la idea de desarrollar una serie de modelos de algoritmos bioinspirados: algoritmos genéticos, de colonia de hormigas y de enjambre de partículas.

Dichos modelos deben servir de herramienta para los profesores y alumnos de la asignatura de *Optimización y Computación Inteligente* (OCI) del Máster de Estructuras de la Universidad de Granada. Entre los profesores se encuentra la tutora del proyecto, que asumiría el papel de cliente.

Metodologías tradicionales	Metodologías ágiles
Proyectos de duración media o elevada	Proyectos de corta duración
Proceso mucho más controlado, con numerosas políticas y normas	Proceso menos controlado, con más flexibilidad
Respuesta lenta a los cambios	Respuesta rápida a los cambios
Gestión de equipos grandes de personas, generalmente distribuidas	Gestión de equipos pequeños
El cliente interactúa con el equipo de proyecto mediante reuniones	El cliente suele ser parte del equipo de proyecto
Curva de aprendizaje media o larga	Curva de aprendizaje corta

Cuadro 2.1: Diferencias entre metodologías tradicionales y ágiles

Se quiere hacer un modelo básico para cada uno de los algoritmos bioinspirados. Dicho modelo debe servir para ilustrar de una forma sencilla el funcionamiento de dichos algoritmos y el uso de los parámetros más representativos de éstos.

Además también se quiere hacer un modelo que resuelva un problema específico de ingeniería, como son los problemas de optimización, para cada uno de los algoritmos. En un principio no se sabe cual es el problema, debe hacerse una análisis y búsqueda de aquellos más representativos y adaptables a cada algoritmo.

Por lo tanto, teniendo en cuenta lo mencionado anteriormente, no podemos realizar una definición de requisitos inicial demasiado detallada, debe plantearse la metodología de desarrollo de forma que podamos agregar cambios de manera incremental y evolutiva. Es por ello que la **metodología** que mejor se adapta es la **ágil**.

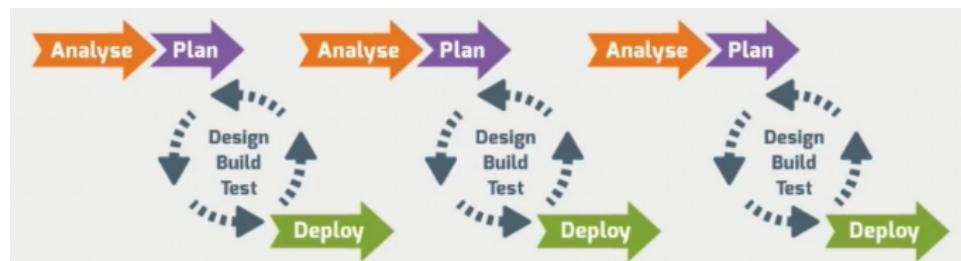


Figura 2.1: Fases en una metodología ágil

Gracias a sus ciclos, vamos a poder analizar y planear en primer lugar, después diseñar, construir y testear independientemente en cada iteración. Y así para cada uno de los modelos (ver Figura 2.1).

Hay una gran cantidad de distintos marcos de trabajo planteados para el desarrollo ágil, entre los más conocidos podemos destacar a Scrum, Kanban y Extreme Programming.

2.3.1. SCRUM

SCRUM es un marco de trabajo que usa la metodología ágil, con el cual las personas pueden abordar problemas complejos de adaptación, al tiempo que entregan productos de manera productiva creando el mayor valor posible del mismo.

Es un marco ligero que ayuda a las personas, los equipos y las organizaciones a generar valor a través de soluciones adaptables a cualquier problema. SCRUM implementa el método científico del empirismo. Su marco de trabajo está representado en la Figura 2.2.

Reemplaza un enfoque algorítmico programado por uno heurístico, con respeto por las personas y autoorganización para lidiar con la imprevisibilidad y la resolución de problemas complejos.

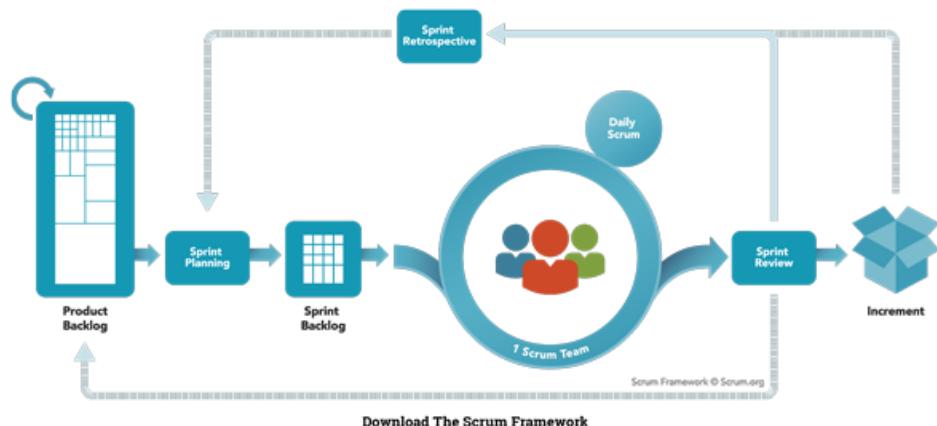


Figura 2.2: Marco de trabajo SCRUM

Características

La metodología SCRUM se centra en el **desarrollo incremental iterativo**. Está formado por un conjunto de buenas prácticas que van a permitir, conseguir una agilidad durante el desarrollo y obtener mejores resultados. Su principal prioridad es maximizar el retorno de la inversión (ROI).

A cada una de estas iteraciones se les conoce como **sprint**. Cada *sprint* suele tener una duración corta de entre 2 y 4 semanas, dando como resultado al final de éste una aportación o producto software de valor completo y

funcional.

Se definen todas las funcionalidades requeridas del producto a desarrollar en una lista conocida como ***Product Backlog***, durante la reunión de organización o *Sprint Planning*. En este informe, que evoluciona a lo largo del desarrollo, se determina la prioridad, riesgo y una estimación de la cantidad de trabajo que supone desarrollar dicha funcionalidad.

En SCRUM se trabaja con **roles**, en el que cada uno de ellos se encargará de realizar distintas tareas.

Los equipos serán organizados y dirigidos entre ellos, con reuniones diarias para comprobar y medir el grado de avance del proyecto. Servirá para comprobar que el desarrollo y la velocidad del equipo es correcta de forma que estarán llegando a tiempo a la entrega.

Roles

Un un equipo Scrum suele haber distintos roles, cada uno con diferentes responsabilidades:

- **Product Owner:** Es el encargado de optimizar y maximizar el valor del producto. Suele tener una labor de interlocutor con los *stakeholder* y sponsors del proyecto, así como de intermediario entre el cliente y el equipo de desarrollo.
- **Scrum Master:** Es el líder y mayor responsable del equipo. Tiene dos funciones principales: gestionar el proceso Scrum de forma que se siga la metodología y cumplimiento de valores y ayudar a eliminar impedimentos que puedan afectar a la entrega del producto.
- **Equipo de desarrollo:** Está formado por un grupo de profesionales que se encargan de desarrollar el producto. Ellos se organizan y gestionan de forma autónoma para conseguir un incremento de software al final del ciclo de desarrollo.

Ciclo de trabajo

El desarrollo del proyecto se hará a través de los *Sprints* o diferentes partes en las que éste se dividirá, lo que permitirá abordarlo de forma más rápida y eficiente.

Cada *Sprint* lleva asociada una serie de fases o etapas que permiten definir qué se va a desarrollar, cual va a ser su diseño y un plan flexible que guiará el trabajo y el producto final resultante. Las **5 fases** de los *Sprints* son:

1. **Reunión de planificación** (*Sprint planning*): Se prevee el trabajo a realizar para *sprint* determinado. En esta reunión se define el plan

para el desarrollo del *sprint*, así como qué va a entregarse al final del mismo.

2. **Scrum Diario** (*Daily meeting*): Tiene como objetivo evaluar el progreso del *sprint* y detectar si hay algún retraso en el desarrollo del mismo. Es una reunión diaria de no más de 15 minutos en la que cada miembro del equipo expone la situación de su trabajo y posteriormente se crea un plan de trabajo para ese día.
3. **Trabajo de desarrollo durante el Sprint** (*Sprint*): Durante el desarrollo de un *sprint* no se deben realizar cambios que afecten a los objetivos del mismo y no se disminuye los objetivos de calidad. Si es demasiado largo, se puede redefinir para evitar retrasos.
4. **Revisión del Sprint** (*Sprint review*): Se lleva a cabo al final del *sprint* con el objetivo de revisar lo que se hizo, detectar problemas y redefinir los items del *Product Backlog* en su caso.
5. **Retrospectiva del Sprint** (*Sprint retrospective*): Tiene com objetivo revisar el desarrollo del *sprint* en lo que respecta a personas, relaciones, procesos y herramientas, de forma que se detecten los puntos a mejorar.

Aplicación de SCRUM a este proyecto

Teniendo en cuenta que la realización de este proyecto se hará de forma individual, a diferencia de cómo está enfocada la metodología SCRUM, que es entorno a un grupo de trabajo, debemos realizar una serie de modificaciones para poder adaptarla al desarrollo del proyecto.

En lugar de tener un *daily meeting* para exponer lo realizado hasta el momento, al ser un único miembro en el grupo, prescindiremos de dicha reunión.

Como tenemos asignaturas que tendremos que sobrellevar durante el desarrollo del proyecto, vamos a realizar una reflexión semanal sobre todo aquello que hemos realizado y tenemos pendiente por hacer. Además, una **reunión semanal** con la **tutora** para informar el estado del proyecto y resolver dudas. Trataremos de hacer coincidir con las fecha de finalización de los *sprints* a modo de revisión de la iteración.

2.4. Gestión del código y documentación

Para la **gestión del código** de los modelos de NetLogo, se utilizará un repositorio local con un sistema de control de versiones que será sincronizado remóticamente con un repositorio privado en GitHub¹. Intentaremos prevenir el riesgo de pérdida del código en nuestro repositorio local a través de esta

¹<https://github.com/josemartin22/TFG>

copia remota.

Para la **gestión de la documentación** utilizaremos la plataforma online Overleaf² para edición de documentos escritos en LaTeX³. Además de estar almacenada en la nube y prever una posible pérdida en el caso de estar utilizando un programa de edición local (como podría ser Texmaker), permitirá el control de versiones y edición conjunta para escribir aclaraciones o dudas a la tutora.

2.5. Gestión del tiempo

En esta sección se desarrollará la planificación temporal final realizada para el proyecto. Previamente al inicio de los *sprints*, se hace una planificación, análisis y revisión del estado del arte.

2.5.1. Planificación temporal



Figura 2.3: Diagrama de Gantt

Los días que contienen una 'R' representan aquellos días en los que se tuvo una reunión con la tutora.

2.5.2. Planificación de los *Sprints*

Para la realización del proyecto tenemos un total de 4 meses aproximadamente, se ha estimado realizar un total de 8 *sprints* con una duración de entre 1 y 3 semanas cada uno.

²<https://www.overleaf.com/>

³<https://www.latex-project.org/>

- **Sprint 1:** En esta primera iteración se desarrollará un modelo básico para los algoritmos genéticos. Será necesaria una formación previa a su implementación y una posterior evaluación del mismo.
- **Sprint 2:** En esta segunda iteración se desarrollará un modelo específico de algoritmos genéticos para resolver un problema de optimización concreto. Será necesario una investigación y elección previa del problema. Posteriormente se comparará con los resultados de otros autores.
- **Sprint 3:** En esta tercera iteración se desarrollará un modelo básico para los algoritmos de enjambre de partículas. Será necesaria una formación previa a su implementación y una posterior evaluación del mismo.
- **Sprint 4:** En esta cuarta iteración se desarrollará un modelo específico de algoritmos de enjambre de partículas para resolver un problema de optimización concreto. Será necesario una investigación y elección previa del problema y una posterior comparación con otros autores.
- **Sprint 5:** En esta quinta iteración se desarrollará un modelo básico para los algoritmos de colonia de hormigas. Será necesaria una formación previa a su implementación y una posterior evaluación del mismo.
- **Sprint 6:** En esta última iteración se desarrollará un modelo específico de algoritmos de colonia de hormigas para resolver un problema de optimización concreto. Será necesario una investigación y elección previa del problema y una posterior comparación con otros autores.

2.6. Gestión de recursos

2.6.1. Recursos humanos

- Dña. Rocío Celeste Romero Zaliz, profesora del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, en calidad de tutora del proyecto presente.
- José Antonio Martín Melguizo, alumno en el grado de Ingeniería Informática en la Escuela Técnica Superior de Ingenierías Informática y Telecomunicación.

2.6.2. Recursos materiales para el desarrollo

A continuación listamos los recursos materiales o hardware empleados para la realización del proyecto:

- **Ordenador portátil:** Asus Zenbook con procesador Intel Core i5, arquitectura de 64 bits, con 8 GB de memoria RAM sobre el que se programará y documentará todo el proyecto.
- **Monitor:** Dell Ultrasharp 25 pulgadas, para ampliar la visibilidad a la hora de hacer tareas concurrentemente, como programar mientras se visualiza la simulación de los modelos.

2.6.3. Recursos software

Con el objetivo de reducir los costes el proyecto, trataremos de utilizar, en la medida de lo posible, programas y herramientas de software libre. A continuación hacemos un listado de los recursos software utilizados en el proyecto:

- **Sistema Operativo:** Utilizaremos Ubuntu Linux, concretamente la versión 20.04 LTS.
- **OpenJDK:** Gratuito y necesario para poder utilizar la máquina virtual de Java y por ende nuestro entorno de desarrollo. Concretamente, dispongo de la versión 11.0.11.
- **NetLogo:** Será nuestro IDE principal para la creación de los modelos. Como comentaremos en la revisión del estado del arte, NetLogo es *Open Source* y por no será necesario acarrear gastos por su utilización.

2.6.4. Recursos de comunicación y documentación

- **Google Meet:** plataforma para la comunicación vía video, voz y texto a través de Internet de manera gratuita.
- **Git:** sistema de control de versiones de código abierto.
- **Github:** plataforma para el alojamiento y control de versiones del repositorio del proyecto.
- **Texmaker:** herramienta opensource de escritorio para la creación y edición de documentos Latex.
- **Overleaf:** herramienta online para la creación, edición y revisión colaborativa de documentos en Latex.
- **Visual Paradigm:** herramienta de pago de escritorio para la creación de diagramas de desarrollo software.
- **Google Workspace:** herramientas online ofrecidas por Google como (Google Slides y Google Docs) para la creación de la presentación y documentación del trabajo respectivamente.

2.7. Gestión de costes

En este sección se hará una estimación de los costes del proyecto teniendo en cuenta todos los recursos anteriormente citados (humanos, materiales, software y de comunicación/documentación).

2.7.1. Costes de recursos humanos

La duración total para la realización del proyecto es de 130 días aproximadamente desde el inicio del segundo cuatrimestre hasta la entrega del trabajo, si no se tiene en cuenta fines de semana ni días festivos.

Una jornada laboral habitual tiene una duración de 8 horas, pero debido a la simultaneidad de asignaturas con las que debe realizarse, se verá reducida a unas 5 horas diarias para hacer un cálculo más realista. Por tanto tenemos un total de 650 horas de trabajo.

Concepto	Precio/Hora	Nº Horas	Importe
Trabajo autónomo	11,50€/hora	650	7475,00€
Total:			7475,00€

Cuadro 2.2: Costes asociado a los recursos humanos

El precio de cada hora asociado al trabajo autónomo por parte del alumno, se ha calculado de acuerdo al salario de un ingeniero informático recién egresado, que suele rondar los 22.000€ anuales según la Universidad Europera. Si dividimos entre 12 meses que tiene un año, una media de 20 días laborales al mes y una jornada completa de 8 horas diarias, el precio hora resultantes es de 11,5€.

El cálculo del coste por recursos humanos final de acuerdo a los precios hora establecidos lo podemos ver en la Tabla 2.2.

2.7.2. Costes de recursos materiales

Para calcular el precio de los recursos materiales, tendremos que tener en cuenta que el precio de dicho portatil ha ido devaluándose con el paso de los años. A esto último se le conoce como depreciación de un producto. Supongamos un valor residual de 80€ y un coeficiente de amortización lineal del 20 %, que sería el máximo permitido para equipos electrónicos según la Agencia Tributaria de España.

El **coste de amortización** viene dado entonces por la siguiente fórmula:

$$(P_c - V_r) \times C_a$$

Siendo: P_c el precio de compra del producto, V_r el valor residual y C_a el coeficiente lineal de amortización.

Por lo tanto, si sustituimos en la fórmula los datos de nuestro portátil nos queda: $(899\text{€} - 80\text{€}) \times 0,2 = 163,8\text{€}$. Ese sería el precio de nuestro portátil tras los años de uso.

Ahora tendríamos que estimar la proporción de horas de uso del mismo durante el proyecto respecto de las horas de uso anuales (12 meses, 5 días a la semana, 5 horas diarias) para obtener coste por el servicio que nos ofrece: $163,8\text{€} \times (650 / 12 \cdot 4 \cdot 5 \cdot 5) = 163,8\text{€} \times (650 / 1200) = 88,72\text{€}$.

Realizamos el mismo cálculo para el caso del monitor, teniendo en cuenta un precio de compra de 330€, un valor residual de 50€, un coeficiente lineal de amortización del 20 % y las mismas horas de uso que el portátil. El precio final sería de 30,33€

Recurso	Coste individual	Cantidad	Importe
Ordenador portátil	899,00€	1	88,72€
Monitor	330,00€	1	30,33€
Total:			119,05€

Cuadro 2.3: Costes materiales

El coste total material tras calcular la depreciación del mismo lo podemos ver en la Tabla 2.3.

2.7.3. Costes de recursos software

En nuestro caso, todas las herramientas de software utilizadas, desde el sistema operativo hasta los programas para la documentación y comunicación con la tutora, han sido herramientas gratuitas, a excepción de Visual Paradigm. Tenemos la suerte de que dicha herramienta incluye una licencia de prueba gratuita para su utilización durante 30 días. La utilizaremos para así poder beneficiarnos de ella y reducir costes.

Una de las razones por las que utilizamos NetLogo para desarrollar los modelos era por su característica de ser *Open Source*.

2.7.4. Costes adicionales

Se podría considerar como costes adicionales o indirectos aquellos gastos derivados del proyecto que son necesarios para la realización del mismo. Por

ejemplo se podría considerar como gasto adicional: la conexión a internet o factura de la luz entre otros.

Considerando un precio de conexión a Internet de 30€ mensuales, podríamos estimar un coste total de 120€. Si además incluimos los gastos de luz pertinentes ésta ascendería a 150€.

2.8. Presupuesto total

El presupuesto total estimado del proyecto queda reflejado y desglosado como muestra la Tabla 2.4.

Concepto	Importe
<i>Recursos humanos</i>	
Trabajo autónomo	7475,00€
<i>Recursos materiales</i>	
Ordenador portátil	88,72€
Monitor	30,33€
<i>Recursos software</i>	
Sistema Operativo	0,00€
OpenJDK	0,00€
Entorno de desarrollo (NetLogo)	0,00€
Herramientas de comunicación y documentación	0,00€
<i>Costes adicionales</i>	
Internet y consumo eléctrico	150,00€
Total: 7744,05€	

Cuadro 2.4: Presupuesto total del proyecto

CAPÍTULO

3

ANALISIS Y DISEÑO

En este capítulo se hará un análisis del proyecto, previo al diseño del mismo. Se analizará los requisitos del proyecto, así como de los posibles riesgos que puedan ocurrir durante el desarrollo de éste.

3.1. Especificación de requisitos de los modelos

Los requisitos son la descripción de las características y funcionalidades del sistema que debe de cumplir para satisfacer las necesidades del cliente.

La especificación de requisitos se describe en las metodologías tradicionales de acuerdo a tres subsecciones: requisitos funcionales, requisitos no funcionales y requisitos de la interfaz de usuario. En nuestro caso, al utilizar una metodología ágil como es SCRUM, vamos a listarlos a partir de **historias de usuario**.

3.2. Historias de usuario

Una historia de usuario no es más que una explicación general e informal de una función de software desde el punto de vista del usuario final. Ayudan a desarrollar el producto partiendo de un marco centrado en el usuario.

Las historias de usuario se describen en unas pocas frases en lenguaje coloquial, de forma que permitan describir el resultado deseado. No entran en detalle, ya que para ello están los requisitos que se añaden más tarde.

Cada una de las historias se expresará de acuerdo al siguiente patrón:

Como <rol>, quiero <evento> para <funcionalidad>

Además a cada una de ellas se le acompañará de una número del 1 al 3 que será el **nivel de prioridad** (alto, medio o bajo respectivamente), una **estimación** para su desarrollo y un **identificador** de historia de usuario.

Se hablará a continuación en las historias de usuario desde el punto de vista *o rol de alumno o profesor* de la asignatura de *Optimización y Computación Inteligente (OCI)*.

Identificador	Descripción	Prioridad
HU.1	Como profesor quiero poder iniciar, parar y ejecutar por pasos la simulación de dichos algoritmos para poder explicarlos con mayor claridad.	1
HU.2	Como alumno quiero que la interfaz sea sencilla y fácil de utilizar.	1
HU.3	Como profesor quiero la posibilidad de cambiar y editar la parametrización de todos los algoritmos para explicar mejor el uso de los mismos.	2
HU.4	Como alumno quiero visualizar gráficas sobre el rendimiento y principales parámetros de los algoritmos.	3

Cuadro 3.1: Listado inicial de historias de usuario

3.2.1. Tarjetas de historias de usuario

A continuación vamos a detallar cada una de las historias de usuario planificadas para el desarrollo del proyecto y priorizadas como se han listado en la Tabla 3.1.

Identificación	HU.1 - Control de la simulación de algoritmos
Descripción	<i>Como profesor quiero poder iniciar, parar y ejecutar por pasos la simulación de dichos algoritmos para poder explicarlos con mayor claridad.</i>
Aceptación	La interfaz de los modelos desarrollados debe incluir dichos botones. La detención de la simulación no debe interferir en el resultado de la misma.
Tareas	<ul style="list-style-type: none"> • Añadir a la interfaz un deslizador que permita cambiar el valor de cada parámetro. • Comprobar que funcionan correctamente.
Estimación	2 días

Cuadro 3.2: Historia de Usuario - HU.1

Identificación	HU.2 - Interfaz estructurada y sencilla
Descripción	<i>Como alumno quiero que la interfaz sea sencilla y fácil de utilizar.</i>
Aceptación	La interfaz de los modelos desarrollados debe estar bien estructurada y organizada. Debe ser manejable por cualquier persona ajena al ámbito de la informática.
Tareas	<ul style="list-style-type: none"> • Diseño y estructuración de la interfaz de los modelos. • Añadir información a modo de manual para facilitar el uso de la interfaz.
Estimación	6 días

Cuadro 3.3: Historia de Usuario - HU.2

Identificación	HU.3 - Parametrización de algoritmos
Descripción	<i>Como profesor quiero la posibilidad de cambiar y editar la parametrización de todos los algoritmos para explicar mejor el uso de los mismos.</i>
Aceptación	La interfaz de los modelos desarrollados permite la personalización de los parámetros más relevantes para cada uno de los algoritmos. Se puede modificar el valor de éstos en tiempo de ejecución.
Tareas	<ul style="list-style-type: none"> • Analizar y listar los principales parámetros de cada algoritmo bioinspirado. • Añadir a la interfaz un deslizador que permita cambiar el valor de cada parámetro.
Estimación	4 días

Cuadro 3.4: Historia de Usuario - HU.3

Identificación	HU.4 - Visualización de gráficas
Descripción	<i>Como alumno quiero visualizar gráficas sobre el rendimiento y principales parámetros de los algoritmos.</i>
Aceptación	La interfaz de los modelos desarrollados debe mostrar gráficas de la evolución del rendimiento de los algoritmos en función del tiempo. Debe monitorizar los valores de los parámetros en tiempo real.
Tareas	<ul style="list-style-type: none"> • Implementación de las gráficas de rendimiento. • Añadir a la interfaz la monitorización de los parámetros.
Estimación	4 días

Cuadro 3.5: Historia de Usuario - HU.4

3.3. Requisitos adicionales

- **Compatibilidad:** El código de los modelos desarrollado debe ser fácilmente portable y funcionar correctamente en cualquier sistema operativo.
- **Usabilidad:** La interfaz debe ser lo más intuitiva posible. El usuario no necesita saber sobre el funcionamiento del sistema para poder utilizarlo.
- **Rendimiento:** Se minimizará en la medida de lo posible, el tiempo de simulación.
- **Estabilidad:** Los modelos deben ser estables frente a cualquier configuración de parámetros. Los modelos siempre se ejecutarán y no aparecerá ningún error de cara al usuario.
- **Mantenimiento:** Las partes del código más complejas deberán estar bien comentadas.
- **Documentación:** Se incluirá información a modo de manual de usuario para facilitar la utilización del modelo y sus parámetros.
- **Extensibilidad:** Los modelos finalmente creados quedarán como código abierto para futuras mejoras por otras personas, siempre y cuando sea para uso no comercial.
- **Costo:** El costo final de implementación de los modelos debe ser bajo.

3.4. Análisis de riesgos

En esta sección se pretende **identificar y analizar los riesgos** que pueden aparecer y afectar a los objetivos del proyecto que estamos realizando. Un riesgo es una condición o evento que puede surgir y alterar el desarrollo natural del proyecto.

Con el objetivo de **actuar** de forma rápida **ante la aparición** de alguno de éstos, a continuación vamos a enumerar los posibles riesgos y a identificar su causa y el efecto que pueden producir. Además se definirá una estrategia o plan de actuación para minimizar el impacto de éste.

A continuación se presentan en la Tabla 3.6 los riesgos considerados, siguiendo el esquema *Riesgo-Causa-Actuación*. Además se clasifican dichos riesgos en una matriz de probabilidad-impacto (ver Tabla 3.7).

Riesgos del proyecto			
Nº	Riesgo	Causa	Plan de actuación
R1	Aparición de tareas imprevistas	Análisis pobre, planificación incorrecta	Replanificar tareas y actualizar backlog
R2	Fallo de las herramientas de comunicación	Caída de conexión, fallo del servidor de correo, etc.	Enviar correo al tutor y utilizar otra herramienta
R3	Cambios frecuentes en la organización	Trabajo demasiado ambicioso	Incremento del tiempo para replanificar
R4	Pérdida de documentos	Fallo hardware, robo de portátil	Utilizar una herramienta sincronizada en la nube a modo de backup.
R5	Ausencia de <i>feed-back</i> con el tutor	Enfermedad, problema tecnológico	Seguir trabajando en el proyecto.
R6	Interfaz no cumple con los requisitos	Lectura incorrecta de requisitos	Corregir requisitos y modificar la interfaz respectivamente
R7	Modelos no útiles para los usuarios	Modelos implementados demasiado complejos	Simplificar aún más las explicaciones e interfaz de los modelos.
R8	NetLogo no apruebe los modelos	Desconocida	Hacer los cambios pertinentes para que los apruebe.
R9	Problema de irreproducibilidad de problemas específicos	Ausencia de información, problemas muy complejos	Buscar adaptación del problema. Resolver otro problema distinto.
R10	Implementar un modelo que ya esté hecho	Revisión de modelos existentes insuficiente	Replantear el modelo a implementar.

Cuadro 3.6: Tabla con los riesgos del proyecto

Probabilidad → Impacto ↓	0,1	0,3	0,5	0,7	0,9
Muy Bajo		R10	R3,R8		
Bajo		R2			
Medio		R5	R9		
Alto	R6		R1		
Muy Alto	R4		R7		

Cuadro 3.7: Matriz *probabilidad-impacto* de riesgos

CAPÍTULO

4

ESTADO DEL ARTE

Es común en la enseñanza de técnicas bioinspiradas, y en general en la programación, centrar mayoritariamente la atención en la creación de los algoritmos o en la formulación matemática de heurísticas, descuidando el esfuerzo posterior de comprensión que necesitará alumnado para entenderlos en su totalidad [27].

Muchas veces es deseable acompañar las explicaciones de estos temas con alguna herramienta que permita la visualización directa de éstos, facilitando así la comprensión por parte del alumnado.

La visualización de modelos abstractos, algoritmos, y consecuentemente, el beneficio psicopedagógico de su aplicación en la enseñanza, constituyen una gran área de investigación que aún no ha sido completamente explorada.

Son diversas las técnicas que se pueden utilizar para llevar a cabo esta tarea. Desde **applets** de Java como *Falstad*¹, que permiten la simulación de pequeños modelos a través de un navegador que soporte Java, hasta **videos didácticos** de distintas plataformas, como podría ser Youtube².

Si aplicamos alguna de estas técnicas para motivar al alumnado a la creación de sus propios modelos, nos encontraremos con una serie de **inconvenientes**. Por ejemplo, partiendo de que el alumnado tiene una experiencia de programación previa escasa o nula, el principal problema de los *applets* es que necesitan conocimientos avanzados de Java para poder programarlos.

¹<https://www.falstad.com/mathphysics.html>

²<https://www.youtube.com/>

Además muchos de estos *applets* no tienen el código disponible para su modificación y plantearse desarrollarlos desde cero sería una tarea que no se contempla por escasez de tiempo. También desde que Adobe dejó de dar soporte a Flash Player muchos de ellos han dejado de funcionar en el navegador a menos que hayan migrado su código a tecnologías más modernas como HTML5.

Por otra parte, los videos didácticos están muy limitados a la visualización que los autores de los mismos nos ofrecen. No permiten por tanto una interacción causa-efecto directa con el usuario que los consume. Por ello, muchas veces pueden ser ilustrativos, pero no son la mejor metodología para ésto.

Sin embargo hay otras opciones como son las **herramientas** para el **modelado de agentes**. Este tipo de herramientas suelen ofrecer una interfaz amigable que facilita la creación de botones e interactuadores con el usuario, además de tener lenguaje de programación de alto nivel que no requiere demasiada experiencia de programación previa.

Muchas de las técnicas inspiradas en procesos físicos, químicos, o los propios algoritmos bioinspiradas suelen implementarse a través de *sistemas multi-agente* (SMA). Consisten en un conjunto de agentes que operan con el entorno, manteniendo información local del mismo, para tratar de alcanzar su propio objetivo cooperando con la ayuda o estorbo del resto de agentes. [32].

Son más apropiados en comparación a las técnicas clásicas de optimización gracias a la flexibilidad y modularidad que presentan, de ahí surge el concepto de modelado basado en agentes, del inglés *Agent-Based Modelling (ABM)*.

El comportamiento de una única entidad o agente suele seguir unas simples reglas, pero a medida que el sistema utiliza una mayor cantidad de agentes, el comportamiento del sistema en su totalidad se vuelve más complejo de entender.

Es por ello que tanto el diseño, como la simulación y tests de los modelos basados en agentes suele hacerse a través de herramientas informáticas que nos simplifiquen estas tareas. Estas plataformas suelen proporcionarnos un framework para programar, modelar y simular soluciones basadas en agentes de una forma más sencilla. El uso de este tipo de herramientas es muy útil para reducir tiempos a la hora de programar sin tener que construirlo todo desde cero.

4.1. Herramientas para el modelado de agentes

A día de hoy, hay una gran cantidad de entornos para el modelado y simulación de modelos basados en agentes. El objetivo de este trabajo no es hacer una comparación exhaustiva entre ellos, en la literatura ya podemos encontrar algunas muy detalladas [2] [30], sino hacer un repaso sobre aquellas más populares como *Swarm*, *Repast*, *MASON*, *StarLogo*, *NetLogo* y *AnyLogic*.

4.1.1. Swarm



Figura 4.1: Logo de Swarm

«Swarm Simulation System» es un software desarrollado por el Instituto de Santa Fe en Estados Unidos, enfocado a la simulación basada en sistemas de agentes con los que se pueden desarrollar sistemas de propósito general y complejos.

El lenguaje de programación que se utiliza para el modelado de sistemas fue Java a los inicios de la plataforma y posteriormente se desarrolló en Objective-C debido a los largos tiempos de ejecución. Está disponible para la gran mayoría de plataformas: Windows, Linux y Mac OS X.

El soporte para el usuario principalmente está basado en tutoriales y ejemplos a través de wikis, además de una páginas con un listado de publicaciones y preguntas más frecuentes (FAQ). Es software libre y está bajo la licencia GPL. Como inconvenientes, presenta la precondición de tener conocimientos avanzados ya sea en Java u Objective-C.

4.1.2. Repast



Figura 4.2: Logo de Repast

Son las siglas de *REecursive Porous Agent Simulation Toolkit*. Se trata de una plataforma para el modelado basado en agentes enfocado a modelos

sociales. Su interfaz permite visualizar durante el desarrollo y ejecución del modelo, permite conectividad con bases de datos y visualización de gráficas y resultados. Tiene 3 lenguajes de implementación: Java, Python y Microsoft.Net, los cuales ya han alcanzado una madurez y no se siguen desarrollando, pero sí se mantienen.

Su versión más conocida es la de Repast-HPC (Repast for High Performance Computing) que permite trabajar en entornos de altas prestaciones. Se requieren conocimientos de programación en C++ por parte de los usuarios. Requiere de una versión de Java 1.4 o superior y también tenemos documentación y páginas con FAQs y abundantes papers con tutoriales y ejemplos.

De las plataformas escritas en Java, Repast es la más completa. Implementa la mayoría de las características que ofrece Swarm y además agrega otras adicionales como la capacidad de reiniciar modelos desde la interfaz gráfica y un gestor de ejecución de experimentos.

Tiene la ventaja de que se puede integrar con el IDE de Eclipse, incluso con alguna herramienta matemática como Matlab.

Presenta inconvenientes como podría ser una documentación algo incompleta, incompatibilidad con distintos tipos de datos contenedores (solamente admite ArrayList), organización del software no muy clara con un etiquetado de paquetes y herramientas que no se distinguen si son de propósito general o específicos.

4.1.3. MASON



Figura 4.3: Logo de Mason

Son las siglas de *Multi Agent Simulation of Neighborhood*. Desarrollado por el Departamento de Informática en la Universidad George Mason en Estados Unidos, es un software de simulación también de propósito general pero más enfocado al modelado social, físico, abstracto y a la inteligencia artificial.

Fue desarrollado como una alternativa a Repast con objetivos muy claros: anteponer las simulaciones con un gran número de agentes sobre aquellas con gran número de iteraciones. Está escrito en Java y también es multiplataforma para aquellas que soporten una versión de Java 1.3 o superior. Tenemos documentación asociada a la API software así como tutoriales y papers con numerosos ejemplos.

Es una muy buena opción para expertos y programadores más experimentados que requieran de modelos con una carga computacional intensa (bien

porque tienen muchos agentes o el tiempo de ejecución será prolongado). Una de las ventajas es la de pausar una simulación y poder continuarla en cualquier otra máquina a partir de una copia de datos del estado de la simulación que el propio entorno nos proporciona.

Sin embargo tiene una serie de inconvenientes como podrían ser: una terminología confusa y no estándar a la hora de programar, ausencia de una ventana o terminal con la que se puedan escribir sentencias o instrucciones para interactuar con los modelos/gráficos. También presenta mayor complejidad a la hora de implementar ya que solo se pueden ejecutar métodos que se llamen "step" (lo cual facilita que sea rápido pero complica el desarrollo de los modelos).

4.1.4. StarLogo



Figura 4.4: Logo de StarLogo

StarLogo es un software de modelado de agentes desarrollado por el Instituto de Tecnología de Massachusetts (MIT). Sigue un modelo de distribución *shareware* (el usuario puede evaluar de forma gratuita el producto pero con limitaciones en las funciones o tiempo de uso). Sin embargo hay una versión de código abierto (OpenStarLogo³).

StarLogo, junto con su sucesor NetLogo, son lenguajes de programación procedurales, al contrario que el resto de herramientas que siguen un modelo de programación orientado a objetos. Esto no dificulta la flexibilidad, puesto que el paradigma procedural hace que estos lenguajes sean más fáciles de entender por personas que no tengan previa experiencia programando e incluso algo más rápidos a la hora de simular.

Carece de esa flexibilidad con respecto a las herramientas previas, ya que los modeladores están limitados a las funcionalidades que proporcione el sistema. Siempre tenemos la opción de modificar el código fuente y adaptarlas a nuestras necesidades pero es algo mucho más tedioso y complejo.

Como ventajas, nos proporciona la capacidad de graficar dinámicamente durante la simulación distintos parámetros de ésta. Así como una amplia documentación y biblioteca con modelos y ejemplos que podemos ejecutar.

³<http://web.mit.edu/mitstep/openstarlogo/index.html>

4.1.5. NetLogo



Figura 4.5: Logo de NetLogo

NetLogo viene de *Network Logo* y hereda particularidades del tanto del lenguaje de programación Logo, como de su predecesor StarLogo.

El lenguaje de programación NetLogo incluye una gran cantidad de estructuras de datos y primitivas de alto nivel que facilitan la programación. Es procedural como ya comentamos antes, está fundamentado en Logo (una variante de Lisp) y contiene muchas capacidades de control y estructuración de un lenguaje de programación estándar.

Comparte con StarLogo el diseño de la interfaz de usuario y el objetivo principal de éste: brindar una herramienta para el modelado fácil de usar por cualquier persona y muy enfocada en el uso del usuario. Sin embargo pese a su gran facilidad de uso, es una herramienta muy potente y utilizada por una gran cantidad de investigadores en ambientes de investigación a nivel profesional.

Es un entorno programable de modelado multiagente que sirve para simular fenómenos naturales y sociales. Comparada con el resto de herramientas, ésta es de mayor alto nivel ya que está diseñada para que el usuario aprenda rápido y fácilmente a utilizarla.

Es muy adecuada si queremos modelar y desplegar los sistemas sobre Internet. Permite la ejecución de estos a través del navegador, aunque es mucho más eficiente hacerlo desde la propia aplicación. Sirve para cualquier plataforma que soporte una máquina virtual con Java (versión 5 o posterior).

NetLogo es conocida por la profesionalidad de apariencia de su aplicación, así como por la gran cantidad de documentación detallada que tiene, la cual es muy accesible y fácil de consultar. También tiene foros donde consultar dudas.

4.1.6. AnyLogic

AnyLogic es un software propietario que ofrece una gran variedad de funcionalidades para el desarrollo de modelado de agentes.



Figura 4.6: Logo de AnyLogic

Entre las ventajas que tiene AnyLogic, podemos leer y escribir datos dinámicamente, durante la ejecución de un modelo, a partir de hojas de cálculo como Excel o bases de datos. Principalmente enfocado para sistemas operativos de Microsoft, pero ejecutable desde cualquier plataforma que soporte una máquina virtual de Java (versión SE 11.0 o superior).

Proporciona una web con una amplia gama de modelos, sobretodo del ámbito social, urbano y planificación de asistencia sanitaria. Sin embargo el código fuente no está disponible al ser este software propietario y por lo tanto debemos de pagar para poder acceder a él.

También hay a disposición del usuario videos explicativos y tutoriales del funcionamiento de algunos modelos (pero el código de éstos y la documentación no está disponible).

En resumen, hay una amplia **variedad de herramientas** para la simulación y por lo tanto no podemos decir que haya una mejor que el resto. Cada una de ellas ofrece una serie de **características y funciones distintas** que la hacen idónea para el propósito que fue creada. Es cierto que en algunas presentamos ciertas similitudes en cuanto a prestaciones que nos ofrecen y capacidad de ejecutarse en múltiples plataformas.

Entre ellas, podemos **destacar** a **NetLogo** como una herramienta enfocada para un uso docente, fácil de utilizar, potente y con una rica documentación muy detallada. Su velocidad de ejecución no es la mejor de todas ellas (recordemos que está implementada en mayoritariamente en Java y tenemos otras alternativas construidas en C que permiten una ejecución más veloz).

Sin embargo, es la **más adecuada** debido a que no requiere experiencia de programación previa, además de la gran cantidad de funcionalidades que nos ofrece.

4.2. Tabla Comparativa

Herramienta	MASON	Swarm	Repast	StarLogo	NetLogo	Propietario
Desarrolladores	George Mason University / SWARM Development Group, USA	Santa Fe Institute / SWARM Development Group, USA	University of Chicago, Department of Social Science Research Computing, USA	Massachusetts Institute of Technology (MIT), USA	Center for Connected Learning and Computer-Based Modelling Northwestern University, USA	XJ Technologies
Año aparición	2003	1996	2000	1990	1999	2000
Lenguaje implementado	Java	Objective-C/Java	Java, Python, Microsoft.NET	Java	Scala y Java	Scripting propietario
Plataformas (SO soportados)	Windows, UNIX, Linux, Mac OS X	Windows, UNIX, Linux, Mac OS X	Windows, UNIX, Linux, Mac OS X	Windows, UNIX, Linux, Mac OS X	Windows, UNIX, Linux, Mac OS X	Windows, UNIX, Linux, Mac OS X
Demostración de los modelos	Sí	Sí	Sí	Sí	Sí	Sí
Código abierto de los modelos	Sí	Sí	Sí	Sí	Sí	No
Capacidad de graficar integrada	No	Sí, usando paquetes de R	Sí, usando paquetes estadísticos de Colt	Sí	Sí	Sí
Tutoriales / Documentación	Sí / Muy extensa y muy detallada	Sí / Extensa y detallada	Sí / Extensa y con ejemplo	Sí / Extensa y detallada	Sí / Extensa, detallada y con ejemplos	Sí / Extensa y detallada
Requiere exp. programación	Avanzada	Avanzada	Avanzada	Básica	Básica	Intermedia
Funcionalidad GIS integrada	No	Sí, usando la librería apropiada	Sí, usando la extensión apropiada	No	Sí, usando la extensión apropiada	Sí
Más información	1	2	3	4	5	6

Cuadro 4.1: Tabla comparativa de herramientas de modelado de agentes

¹ <https://cs.gmu.edu/~eclab/projects/mason/>

² <http://www.swarm.org>

³ <http://repast.sourceforge.net>

⁴ <https://education.mit.edu/project/starlogo-tng>

⁵ <https://ccl.northwestern.edu/netlogo/>

⁶ <https://www.anylogic.com/>

CAPÍTULO

5

NETLOGO

“NetLogo es un ambiente para la programación de modelos en el cual se pueden simular fenómenos sociales y naturales”. Adicionalmente, se añade que “NetLogo es especialmente adecuado para modelar sistemas complejos que se desarrollan con el tiempo”. [29] “NetLogo es la siguiente generación de una serie de lenguajes de modelado de agentes múltiples, incluidos StarLogo y StarLogoT¹”.

5.1. Características de NetLogo

Como se menciona anteriormente, NetLogo no es solo un lenguaje de programación, sino que es un ambiente o entorno para el desarrollo de modelos cuyo núcleo central es un lenguaje de modelado de agentes fruto de anteriores lenguajes. Entre ellos, su predecesor StarLogo, que es un lenguaje de tipo *Lisp*, por tanto **procedural**, también diseñado para el modelado de agentes.

Tanto sus predecesores como el propio NetLogo, se ejecutan sobre una máquina virtual de Java, por lo tanto son lenguajes **multiplataforma**, que pueden ejecutarse como aplicación de escritorio en cualquiera de ellas (Windows, Linux, Mac, etc.).

NetLogo, como cualquier lenguaje de programación de alto nivel, debe ser traducido a código máquina para que el ordenador pueda entender y ejecu-

¹<https://ccl.northwestern.edu/cm/StarLogoT/>

tar esas instrucciones que el usuario ha codificado en el lenguaje NetLogo. Usualmente, un lenguaje de programación puede ser clasificado si es traducido a lenguaje máquina por un compilador o por un intérprete.

NetLogo es un lenguaje de programación **interpretado**, por lo que cada orden del programa es traducida y ejecutada individualmente. A diferencia de los lenguajes compilados, que son traducidos de forma completa a lenguaje máquina creando un *código objeto* que posteriormente se ejecutará.

Esto hecho de que NetLogo sea un lenguaje interpretado, tiene una serie de ventajas e inconvenientes:

■ **Ventajas:**

1. Es posible **probar órdenes** del lenguaje sin la necesidad de que éstas conformen un propio programa.
2. El **tamaño del programa** suele ser **más pequeño** que los compilados, ya que no se incluye el código de las referencias a otras bibliotecas o módulos en el propio programa.
El hecho de ser interpretado por una máquina virtual facilita el ser independiente de una plataforma u otra.
3. Permite **programar** de forma **incremental**, añadiendo nuevas órdenes a éste y probándolas, de manera que permita **detectar** de forma temprana los posibles **errores** de programación.

■ **Inconvenientes:**

1. Se ejecutan **más lento** que los compilados.
2. Son más **difíciles de depurar** o *debuggear* y esto favorece la aparición de errores lógicos, que son aquellos que producen resultados distintos a los que se esperan. A diferencia de los errores de sintaxis o de ejecución, en éstos no se recibe ningún tipo de aviso o alerta.

Además, el intérprete de NetLogo, antes de ejecutar un modelo, se realiza un **análisis léxico y sintáctico** sobre el programa codificado en el propio **editor** del IDE de NetLogo para detectar futuros errores por el uso de una sintaxis errónea del lenguaje.

Luego es posible que obtengamos errores *en tiempo de ejecución*, los cuales se pueden deber a diversas causas, como una incorrecta utilización de índices a la hora de acceder a posiciones de memoria no reservadas. Estos errores causan el detenimiento del programa y provocan la aparición de mensajes de error.

NetLogo, al ser una herramienta de modelado de agentes, está compuesta por diferentes tipos de agentes que siguen una serie de instrucciones las cuales les permiten interaccionar entre sí y con el entorno, aprender y adaptar su conocimiento en base a la experiencia con éstos.

Son 4 los tipos de agentes de NetLogo: tortugas, parcelas, enlaces y observador.

- Las **parcelas** o *patches* formarán el terreno donde se desplazarán y comunicarán las tortugas. Son estáticas y se distribuyen entorno a las coordenadas del mundo, por lo tanto hay un número determinado y fijo de éstas. Su representación visual es un cuadrado coloreado y son creados automáticamente al inicio.
- Las **tortugas** o *turtles* serán el agente principal que utilicemos para interactuar con las parcelas y con otras tortugas. Podemos definir diferentes tipos de tortugas, extendiendo su funcionalidad a modo de herencia. Podemos elegir y crear su representación visual a nuestro gusto.
- Los **enlaces** o *links* permitirán representar relaciones entre tortugas. Su posición vendrá determinada por el par de agentes (tortugas) que relaciona. Se representa visualmente por una línea.
- El **observador** o *observer* es un agente exterior al mundo, por lo tanto no tiene representación visual. Permitirá dar instrucciones al resto de agentes, será el encargado de crear y destruir a las tortugas.

La programación por tanto, se enfocará a la creación de procedimientos que puedan ser ejecutados por los distintos agentes. Este concepto de programación basada en agentes está ligada a la idea de que los agentes pueden realizar tareas de forma simultánea.

Podríamos preguntarnos entonces, **¿es realmente simultánea la ejecución de los agentes?**

La respuesta es rápida es que no. En los ordenadores para los que se desarrolló NetLogo, como la mayoría de software y otros entornos para desarrollo de modelos basados en agentes (MBA) **no es real el paralelismo**.

Las acciones ejecutadas por los agentes, se hacen de forma **concurrente**. Es decir, la ejecución final de acciones es **secuencial**, intercalándose las acciones de los agentes, pero **no es paralela**.

Son muy exclusivas las máquinas o supercomputadoras que soportan ese paralelismo real. Suelen estar reservadas para unas pocas instituciones o para

centros de investigación. Son bastante más complejas ya que requieren de un software especializado para la distribución del código en distintos nodos de cómputo.

Además la utilización del paralelismo real, no es algo imprescindible, pues hay una gran diversidad de modelos en los que la interacción entre agentes, independientemente de si hay paralelismo real o no, tras la ejecución del modelo se produce el mismo resultado.

5.2. Modelos de NetLogo

NetLogo dispone de una amplia biblioteca con modelos de diversos ámbitos que comprenden la mayoría de áreas de las ciencias, como son la física, matemática, química, medicina, biología e informática. También incluye modelos inspirados en comportamientos y teorías propias de las ciencias naturales, sociales, economía y psicología.

Podemos ver una taxonomía y clasificación de los modelos implementados para cada una de estas áreas en las figuras 5.1 y 5.2.

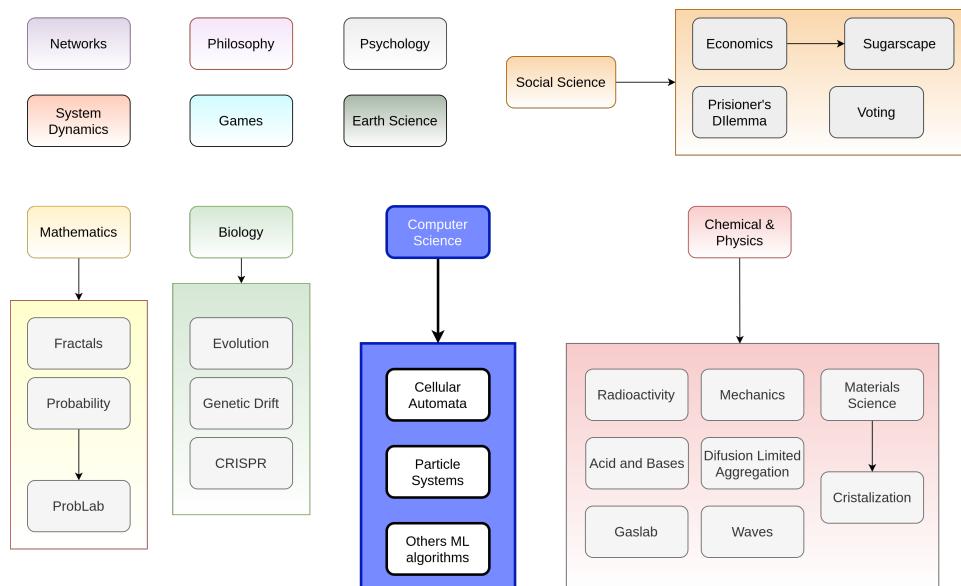


Figura 5.1: Taxonomía de los modelos de ejemplo de NetLogo

Además de los propios modelos incluidos en la plataforma de NetLogo, también hay una gran **comunidad**² de usuarios de NetLogo en la que se incluyen los modelos aportados por otros usuarios e investigadores.

²<https://ccl.northwestern.edu/netlogo/models/community/>

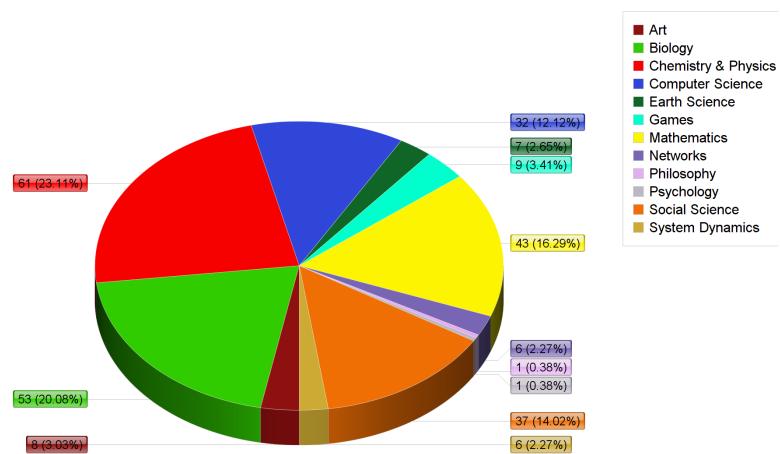


Figura 5.2: Clasificación de modelos de ejemplo de NetLogo

En este trabajo se desarrollarán 6 modelos, que se corresponden con los algoritmos bioinspirados más representativos de la literatura. En concreto, desarrollaremos modelos basados en algoritmos genéticos (GA), algoritmos de colonia de hormigas (ACO) y algoritmos de enjambre de partículas (PSO).

Por tanto tendremos dos modelos asociados a cada algoritmo, uno de ellos más básico y mucho más visual para ilustrar el funcionamiento de los mismos y otro más específico y complejo que tratará de resolver un problema propio de optimización de ingeniería.

Considerando la taxonomía propuesta en la Figura 5.1, los 6 modelos que se desarrollarán en este trabajo irían incluidos en al área de Informática o *Computer Science*.

El hecho de ser técnicas inspiradas en un comportamiento biológico, podría hacernos pensar que se clasificarían en el apartado *Biology* de la Figura 5.2. Sin embargo, son modelos computacionales y pertenecen al grupo *Computer Science*, ya que los individuos con los que se trata son una abstracción matemática incluida en un algoritmo propio de la informática.

CAPÍTULO

6

ALGORITMOS BIOINSPIRADOS

6.1. Algoritmos genéticos (GA)

Los algoritmos genéticos son una técnica informática con inspiración biológica que combina nociones de la genética mendeliana y la evolución darwiniana para buscar buenas soluciones en problemas de optimización incluidos problemas difíciles o *NP-hard* [17], por lo que no hay un algoritmo que encuentre una solución óptima en un tiempo polinómico.

Los algoritmos genéticos, que fueron introducidos J. H. Holland a inicios de los años setenta y más ampliamente desarrollados por distintos autores (véase Goldberg [18]), surgen para tratar de aproximar una buena solución a un problema en un tiempo razonable.

Se basan en el comportamiento y evolución de una población de individuos en busca de recursos como pueden ser la comida o el refugio. Aquellos individuos que tienen más éxito en sobrevivir y adaptarse al entorno, tendrán mayor probabilidad de producir descendientes. Por otro lado, aquellos individuos que no se adaptan bien tenderán a desaparecer o extinguirse.

Para medir cómo de bien se adaptará el individuo al entorno, o en otras palabras: determinar la bondad de una solución, recurriremos a una función de evaluación o *fitness*, que evaluará a cada individuo.

Por tanto, aquellos mejor dotados podrán reproducirse con mayor probabilidad y progarar sus características hacia posteriores generaciones que serán portadores de éstas. La combinación de dos individuos bien dotados puede

producir nuevos individuos con características aún mejores, heredadas por sus ancestros.

De esta manera las especies irán evolucionando siguiendo esa *selección natural* propuesta por Darwin [9].

La **representación** del problema vendrá dada por una población P de n individuos de la forma $P = (X_1, X_2, \dots, X_n)$ en la que cada individuo o **cromosoma** vendrá definido por un conjunto de características llamadas **genes**. Se codificará como un vector de la forma $X_i = [G_1, G_2, \dots, G_d]$

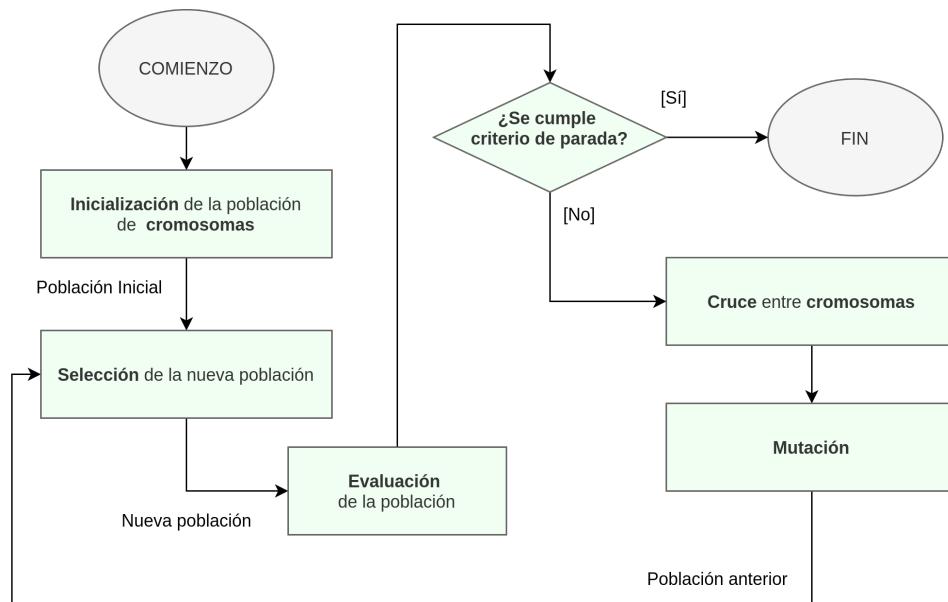


Figura 6.1: Diagrama de flujo para Algoritmo Genético general

La **evolución de la población** de cromosomas puede hacerse de forma general como muestra la figura 6.1, y además podemos destacar dos esquemas:

- **Modelo generacional:** En cada iteración se creará una nueva población con nuevos cromosomas. Podemos decidir si queremos que se produzca *elitismo*, es decir, en cada nueva población se mantenga el mejor individuo de la iteración anterior.
- **Modelo estacionario:** En cada iteración se escoge un par de padres de la población que se reproducirán para crear una serie de descendientes que reemplazarán a un número determinado de cromosomas de la población inicial.

Además, contamos con distintos **tipos de selección** a la hora de determinar cuales son las soluciones que van a cruzarse. Debemos ser cuidadosos a la

hora de diseñar el tipo de cruce, pues todos los individuos tienen derecho a reproducirse (no solo los mejores). Los más conocidos son:

- **Selección por torneo:** Escogemos aleatoriamente k individuos con reemplazamiento y seleccionamos al mejor de ellos. Habitualmente se utiliza un tamaño de torneo $k = 3$, como se puede ver en la Figura 6.2

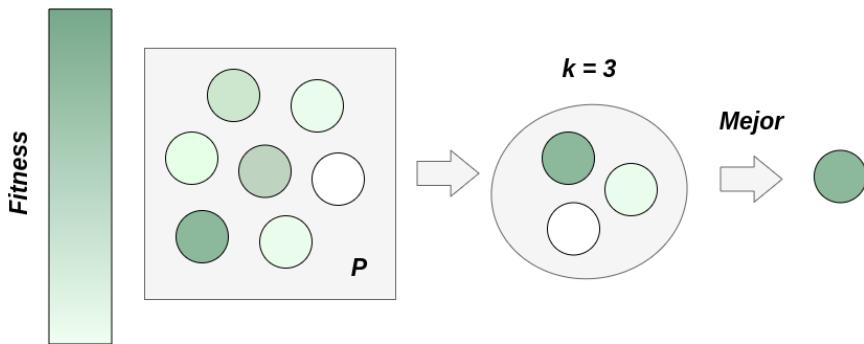


Figura 6.2: Selección por torneo, Fuente: Propia

- **Selección por ruleta:** Es el método clásico. Se asigna a cada cromosoma una probabilidad de ser elegido proporcional a su valor de *fitness*. En concreto, podemos definir la probabilidad de selección de un individuo como: $p_i = \frac{f_i}{\sum_{k=1}^n f_k}$. De forma gráfica se puede ver en la Figura 6.3.

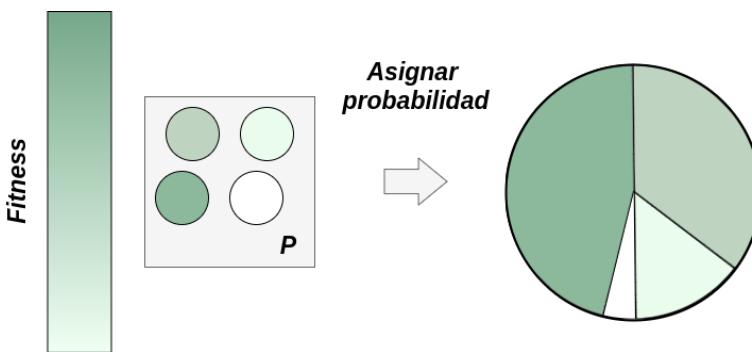


Figura 6.3: Selección por ruleta, Fuente: Propia

El operador de cruce, introducido por Holland en 1975 [22] (cruce en un punto), permite la generación de cromosomas descendientes a partir de un par de cromosomas progenitores.

De entre los tipos de cruce más conocidos para generar las soluciones descendientes tenemos:

- **Cruce en un punto:** Se define de forma aleatoria un punto de corte en los cromosomas progenitores de forma que se recombinen y se genere un nuevo par de cromosomas descendientes.
- **Cruce en dos puntos:** De forma análoga al cruce en un punto, en este caso se dividen los cromosomas a través de dos puntos de corte y se recombinarán para formar un par de cromosomas descendientes.
- **Cruce uniforme:** Se selecciona aleatoriamente la mitad de los genes de un cromosoma progenitor y la otra mitad del otro. No tienen por qué ser genes consecutivos, se eligen las posiciones aleatoriamente. Se recombinan y se formarán un par de cromosomas descendientes.

Podemos ver una representación de los mismos en la Figura 6.4.

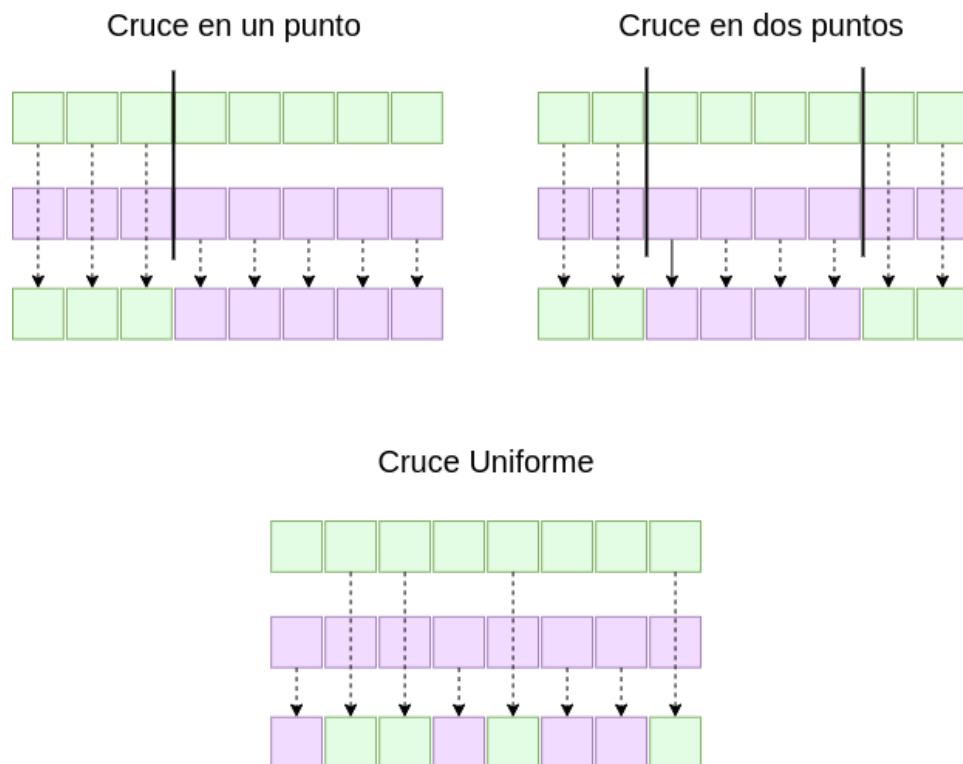


Figura 6.4: Tipos de cruce implementados, Fuente: Propia

En la naturaleza también se puede dar la posibilidad, aunque con una probabilidad muy baja, de que un nuevo descendiente aparezca con una característica particular que lo hace diferente del resto.

Esto se conoce como **mutación**, y consiste en la alteración del valor de uno

de los genes de uno o más individuos de la población. Dependiendo del fenotipo que se elija para representar cada cromosoma, se hará de una forma u otra, al igual que con el cruce entre cromosomas.

Para un ejemplo de codificación binaria, bastaría con generar un número aleatorio para elegir el gen que va a mutar y cambiar su valor como se muestra en la figura 6.5

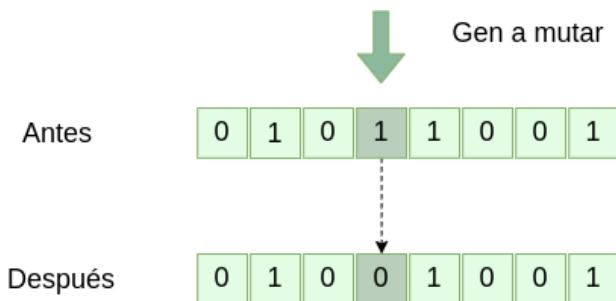


Figura 6.5: Operador de mutación, Fuente: Propia

El pseudocódigo de un algoritmo genético general viene representado por:

Algorithm 1 Genetic Algorithm (GA)

```


iteracion ← 0
Iniciar población
Evaluar población
while No se cumple la condición de parada do
    iteracion ← iteracion + 1
    Seleccionar nueva población
    Reproducción de la población
    Mutación de la población
    Evaluar población
end while


```

Normalmente, la parte más costosa computacionalmente es el cálculo del valor de *fitness* de cada uno de los cromosomas.

Es por ésto, que la **condición de parada** suele definirse como un número máximo de evaluaciones permitidas. También es común utilizar un número concreto de iteraciones o nuevas generaciones creadas a partir de la población inicial.

6.2. Algoritmos de enjambre de partículas (PSO)

También conocidos como algoritmos de *nube de partículas*, o de sus siglas en inglés: *Particle Swarm Optimization*.

Son un conjunto de algoritmos inspirados en el comportamiento de la inteligencia de bandadas de pájaros, peces y enjambres en particular. Los primeros desarrollos fueron introducidos en 1995 por Russ C. Eberhart y James Kennedy [25].

Se aplican para resolver problemas de optimización continua, tratando de mejorar de forma iterativa una solución candidata respecto de una bondad de medida.

Se parte de una población de soluciones candidatas, modeladas en forma de partícula y que se mueven por el espacio de búsqueda de las soluciones de acuerdo a una simples reglas matemáticas para cambiar su velocidad y posición.

La idea básica del algoritmo reside en que cada partícula guarda información de su mejor posición encontrada y cambiará su posición de acuerdo a ésta y a la mejor solución global como se puede ver en la Figura 6.6. Se asume por tanto un intercambio de información entre los agentes.

Además suelen llevar una aceleración adicional conocida como **inercia** que evitará en la medida de lo posible futuros estancamientos en óptimos locales.

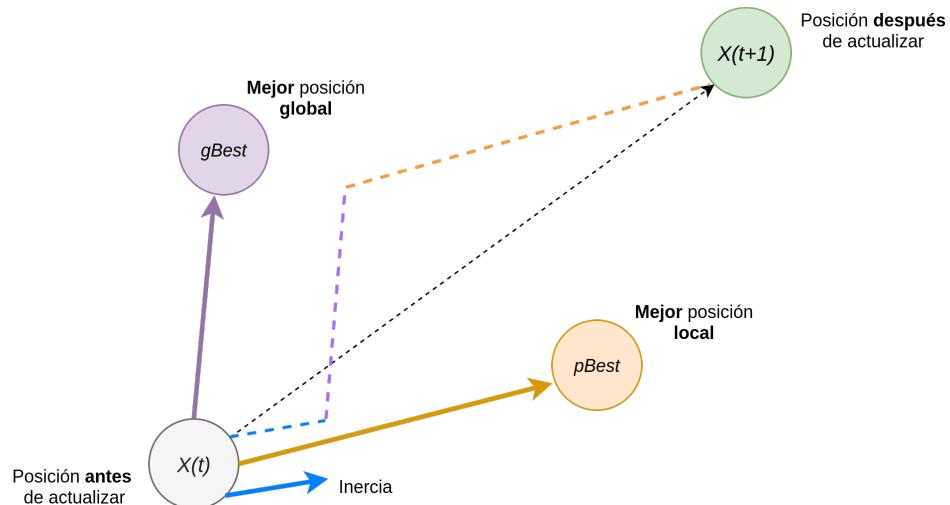


Figura 6.6: Atracción de una partícula en PSO
Fuente: Propia, Basado en Navid Delgarm

Cada partícula será por tanto una solución candidata, que estará en cons-

tante movimiento por el espacio de búsqueda y nunca desaparece. Tendrá asociados los siguientes atributos:

- Un vector solución X , que almacenará la posición de la partícula en el espacio de búsqueda en un instante dado. Dicha posición tendrá asociada un valor de *fitness* que medirá la bondad de la solución/posición actual.
- Otro vector solución $p-best$ que almacenará la mejor posición encontrada por la propia partícula hasta el momento.
- Un vector de velocidades V , que almacenará las dirección (gradiente) a la que se moverá la partícula en la siguiente iteración.

La regla matemática que sigue cada una de las partículas para calcular la nueva posición se calcula sumando el vector de velocidades V al vector de la posición actual X de la forma: $X_i(t+1) \leftarrow X_i(t) + V_i(t)$.

Para calcular el vector velocidad V se sigue la siguiente regla:

$$\nu_{ij} = \nu_{ij} + c_1 \cdot rand() \cdot (pBest_{ij} - X_{ij}) + c_2 \cdot rand() \cdot (gBest_{ij} - X_{ij}) \quad (6.1)$$

donde:

$i = 1, \dots, p$ es el número total de partículas que hay en el enjambre,
 $j = 1, \dots, d$ es el número de dimensiones del problema,
 c_1 y c_2 los coeficientes de aceleración,
 $rand()$ es una función que genera un número aleatorio en $[0, 1]$,
 $pBest$ es la mejor solución local encontrada para la propia partícula y
 $gBest$ es la mejor solución global encontrada.

Los **coeficientes de aceleración** c_1 y c_2 también son conocidos como los *ratios de aprendizaje*. Sirven para controlar la *aceleración* o inclinación hacia la mejor solución local propia o la mejor global. Son los componentes **cognitivo** y **social** respectivamente.

El funcionamiento general del algoritmo de enjambre de partículas viene representado por el diagrama de flujo de la Figura 6.7.

El pseudocódigo general es el siguiente:

Algorithm 2 Particle Swarm Optimization (PSO)

```

for Para cada partícula ( $i=1,\dots,n$ ) do
    Inicializar la posición de la partícula de forma uniforme distribuida
    Inicializar la mejor posición local encontrada de la partícula (pBest)
    if pBest es mejor que gBest then
        Actualizar la mejor solución global
    end if
    Inicializamos la velocidad de la partícula
    while num-evaluaciones  $\leq$  max-evaluaciones do
        for Para cada partícula ( $i=1,\dots,n$ ) do
            for Para cada dimensión ( $j=i,\dots,d$ ) do
                Actualizamos velocidad de la partícula
            end for
            Actualizamos posición de la partícula
            if posicion-actual es mejor que pBest then
                Actualizar la mejor solución local
            end if
        end for
        Actualizar la mejor solución global
    end while
end for

```

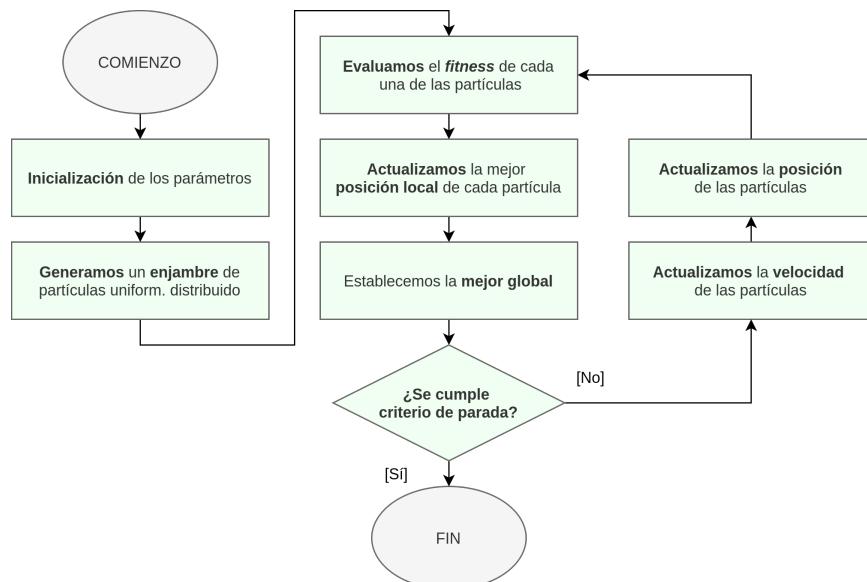


Figura 6.7: Diagrama de flujo para PSO

6.3. Algoritmos de colonias de hormigas (ACO)

La familia de algoritmos de colonia de hormigas u optimización por colonia de hormigas (Ant Colony Optimization, ACO) son un conjunto de técnicas o métodos inspiradas en el comportamiento de las hormigas en su búsqueda de comida. Esta técnica fue propuesta por Dorigo [11].

Las hormigas son capaces de encontrar la ruta óptima entre el hormiguero y su comida. Sin embargo éstas son ciegas. Lo hacen gracias a una sustancia llamada feromonas que todas ellas pueden oler. Esta sustancia se deposita de vuelta al hormiguero una vez se ha encontrado la comida.

La feromona depositada en el camino se evaporará con el tiempo, de modo que los trayectos menos prometedores pierden valor de feromona al ser visitados por menor número de hormigas.

Cuando una hormiga llega a una intersección de caminos elegirá aquel camino que contenga un mayor rastro de feromona. De esta forma, con el sucesivo paso de hormigas se acentuarán aquellos caminos más cortos entre la comida y el hormiguero y finalmente se podrá encontrar un camino óptimo entre ambos. Ver figura 6.8.

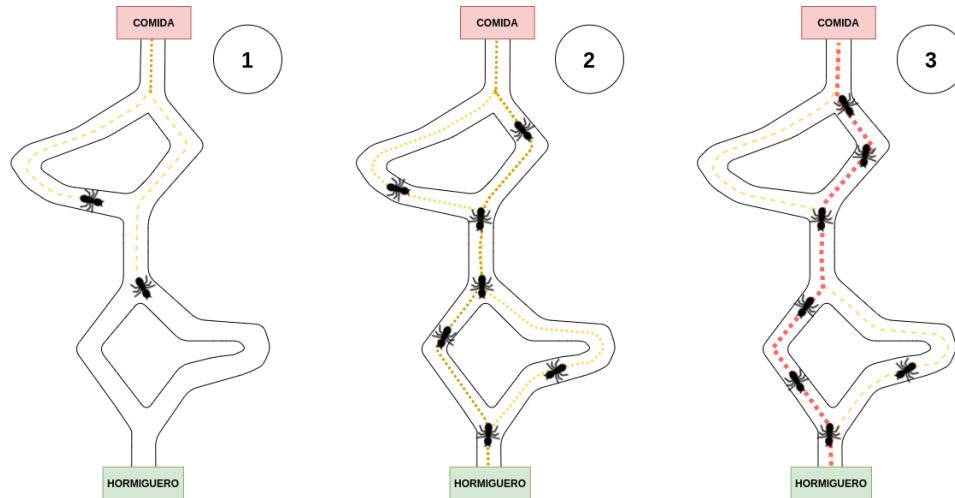


Figura 6.8: Esquema de funcionamiento ACO

Fuente: propia, Basado en Johann Dréo

6.3.1. Resolviendo el Problema del Viajante de Comercio

Los algoritmos de colonia de hormigas se utilizan ampliamente para resolver problemas de optimización de todo tipo. Estos problemas suelen reducirse a problemas de optimización sobre grafos, en los que hay determinar la ruta

óptima en un grafo.

Originalmente, se aplicó este algoritmo para resolver el clásico problema del viajante de comercio (Travelling Salesman Problem, TSP) [10]

A continuación, se enuncia la notación que se seguirá para resolver el problema del viajante de comercio de acuerdo a los operadores propios de la optimización por colonia de hormigas.

En la implementación del algoritmo, vamos a suponer que las N capitales de provincias están conectadas entre sí, por tanto las N ciudades forman un grafo completo. Por tanto definiremos en conceptos de teoría de grafos: las ciudades como nodos y las distancias (caminos que unen las ciudades) como aristas del grafo.

Denotaremos ese conjunto de ciudades como (C_1, C_2, \dots, C_n) . y denotaremos cada una de las aristas como d_{ij} siendo ésta la distancia o coste entre las ciudades C_i y C_j . Esta distancia, al tratarse de coordenadas en un espacio bidimensional, se calculará como la distancia euclídea entre ambas.

La distancia entre dos ciudades $C_i(x_i, y_i)$ y $C_j(x_j, y_j)$ será calculada como $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Al tratarse de un ejemplo básico, supondremos que las distancias son simétricas, pero ésto en la realidad no podría ser así, podrían modelarse en función del número de kilómetros entre ciudades, o podría considerarse que una ciudad no es accesible a otras (ausencia de completitud en el grafo), etc.

Cada una de las hormigas irá componiendo una solución al problema TSP de forma que al final tendrá que tener una secuencia de ciudades a visitar sin repetir ninguna de ellas y con la misma ciudad como origen y destino. Una representación del problema a lo largo de su resolución podría ser la mostrada en la Figura 6.9.

Para la construcción de ésta, se apoyará sobre una regla de transición que indica a la hormiga qué ciudad es más probable de visitar en cada momento.

Esta regla dependerá de:

- **Inversa de la distancia** hasta dicha ciudad: A esta medida se le conoce como *visibilidad*.

$$\nu_{ij} = \frac{1}{d_{ij}} \quad (6.2)$$

Aporta información local a modo de heurística ponderando más aque-

llas ciudades que están más próximas a la ciudad en la que se encuentra la hormiga actualmente.

Esta información suele ser estática puesto que las distancias entre ciudades también lo son.

- **Cantidad de feromona depositada** en la **arista** que une dos nodos. Lo denotaremos como $\tau_{ij}(t)$. Indica la cantidad de feromona entre la ciudad C_i y C_j para la iteración t .

Este valor nos da una idea de como de buena es esa arista. Su valor dependerá del número de hormigas que hayan pasado por dicha arista hasta el momento y se actualizará cada iteración del algoritmo.

Aporta la información global del hormiguero, servirá como traspaso de información entre hormigas.

Una vez definidas las métricas anteriores, denotaremos la **probabilidad de elección** de que una **hormiga k** situada en C_i **visite la ciudad C_j** como:

$$p_k(i, j) = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\nu_{ij}]^\beta}{\sum_{c \in J_k(i)} [\tau_{ic}]^\alpha \cdot [\nu_{ic}]^\beta}, & \text{si } j \in J_k(i) \\ 0, & \text{en otro caso} \end{cases}$$

donde $J_k(i)$ es el conjunto de ciudades o nodos alcanzables desde i no visitados aún por la hormiga k y α y β son parámetros que establecen los pesos que proporcionan un equilibrio entre la importancia de la información memorística y la heurística.

Como hemos mencionado, las hormigas comparten su información individual a través del medio (la feromona). Una vez todas las hormigas hayan construido su propia ruta o solución, se debe actualizar una cantidad $\Delta\tau_{ij}^k(t)$ el valor de la feromona en cada una de las aristas del grafo.

Se utiliza una *retroalimentación positiva* en el valor de feromona de una arista cuando ésta ha sido incluida en una solución de una hormiga, además cuanto mejor sea la solución que compone, mayor cantidad de feromona se aportará a dicha arista.

Por el contrario, se utiliza la **evaporación** o *retroalimentación negativa* del valor de feromona para evitar las malas decisiones tomadas en un futuro. La fórmula de evaporación es de la forma:

$$\tau_{ij}(t+1) \leftarrow (1 - \rho) \cdot \tau_{ij}(t) \quad (6.3)$$

Este mecanismo de evaporación suele ser más recurrente para evitar que las hormigas se estanquen en óptimos locales por la prolongación innecesaria de feromonas que no aportan información de calidad en las aristas del grafo.

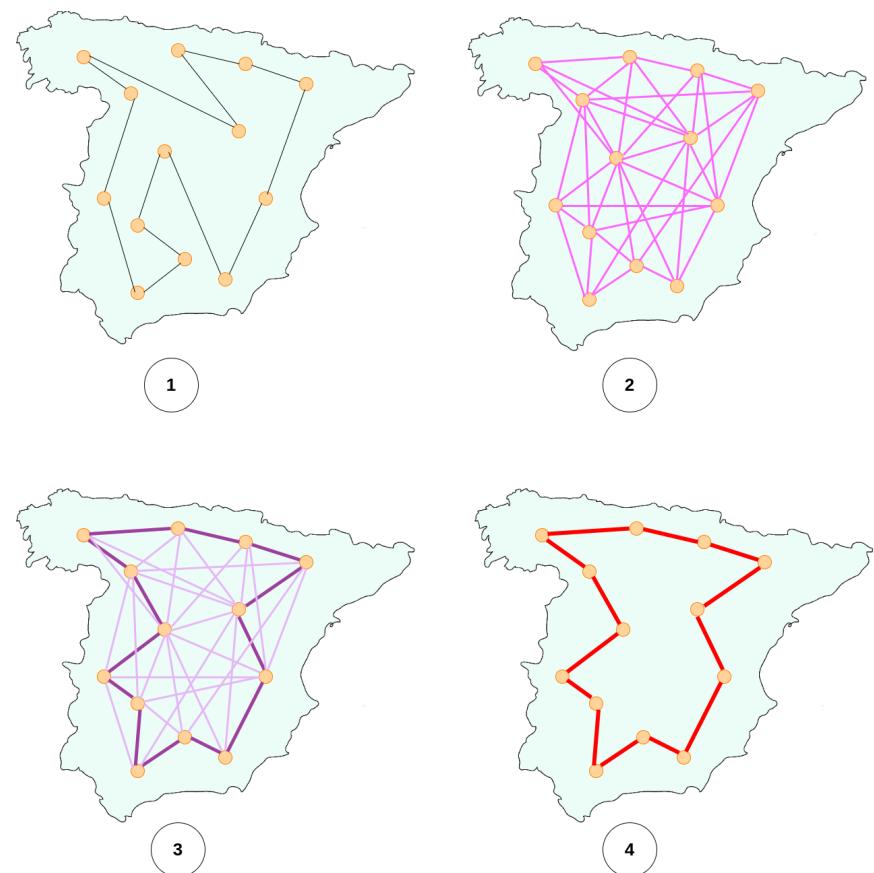


Figura 6.9: Resolviendo TSP a través de ACO
Fuente: Propia, Basado en Johann Dréo

El **esquema global de actualización** de la **feromonas** para una arista compuesta por los nodos i y j puede seguir la siguiente fórmula:

$$\tau_{ij}(t+1) \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \sum_{k=1}^m \Delta\tau_{ij}^k(t)$$

donde ρ es un parámetro que indica el coeficiente de evaporación de las feromonas.

El incremento de feromona que deposita la hormiga k en el instante t para el arco que conecta los nodos i y j se nota como $\Delta\tau_{ij}^k$. Tomará un valor dependiendo de cómo de bueno haya sido el recorrido que compone dicha arista. En concreto:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L(C_k)} & \text{si la hormiga } k \text{ visita el arco } rs \\ 0, & \text{en otro caso} \end{cases}$$

donde $L()$ es una función que calcula la longitud o coste del circuito C_k generado por la hormiga k .

Q es un parámetros que se puede ajustar para determinar como de compensada sea la influencia de la retroalimentación positiva respecto de la evaporación. Normalmente en la práctica se adopta un valor $Q = 1$.

Existen muchas variantes para mejorar la eficiencia del algoritmo ACO inicialmente propuesto. Las más populares son:

- **Versión elitista:** Únicamente la mejor/es soluciones depositan una cantidad de feromonas en sus caminos en cada iteración
- **Rank Ant System:** Las soluciones se clasifican según su coste o longitud. Cada hormiga deposita una cantidad de feromona proporcional a la bondad de su solución.
- **Max-Min Ant System:** Controla el máximo y el mínimo de feromonas en cada arco. Solo la mejor solución global puede agregar feromonas.
- **Colony System:** La ejecución del algoritmo no está sincronizada. Las hormigas que ya han construido su solución pueden poner en marca otra sin tener que esperar al resto de hormigas.

El funcionamiento general del algoritmo de colonia de hormigas viene representado por el diagrama de flujo de la Figura 6.10.

El pseudocódigo general es el siguiente:

Algorithm 3 Ant Colony Optimization (ACO)

Inicializamos cada arco con una cantidad de feromonas
Creamos una población de hormigas
while No se cumpla criterio de parada **do**
 for cada una de las hormigas **do**
 Creamos una solución usando las feromonas y el coste de los arcos
 Depositamos feromonas en aquellos arcos incluidos en la solución
 end for
end while
Esquema de evaporación global de la feromona
Devolvemos la mejor solución encontrada

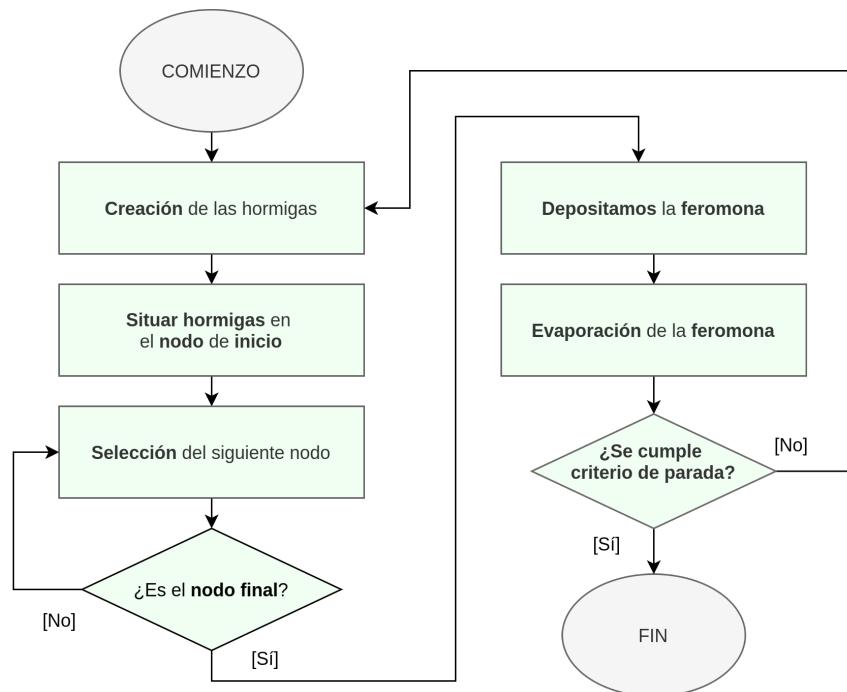


Figura 6.10: Diagrama de flujo para ACO

CAPÍTULO

7

PROBLEMAS DE OPTIMIZACIÓN

En este capítulo describiremos los **problemas de optimización** considerados en la implementación de los modelos específicos.

Un problema de optimización general se puede reducir a **minimizar una función objetivo**, que podemos expresar de la forma:

$$\min f(x)$$

donde x representa los valores de entrada de la función objetivo. Además, sobretodo en problemas de ingeniería, la función objetivo suele estar sujeta a una serie de **restricciones**. Hay dos tipos de restricciones:

Restricciones de **igualdad**:

$$h_i(x) = 0, i = 1, \dots, n_h \quad (7.1)$$

y/o sujeto a unas restricciones de **desigualdad**:

$$g_i(x) \leq 0, i = 1, \dots, n_g \quad (7.2)$$

siendo n_h y n_g el número de restricciones de igualdad y desigualdad respectivamente.

La función objetivo inicial $f(x)$ se modifica entonces para tener en cuenta dichas restricciones, de la siguiente manera:

$$f'(x) = f(x) + p \cdot \sum_{i=1}^{n_h} [h_i(x)]^2 + r \cdot \sum_{i=1}^{n_g} [\max(0, g_i(x))]^2 \quad (7.3)$$

donde p y r son **factores de penalización** y habitualmente toman números positivos grandes del orden de los millones.

Se observa que todas las restricciones de igualdad (7.1) y desigualdad (7.2) se normalizan a un rango de valores menores o iguales a cero. Ésto nos permitirá penalizar positivamente el valor de la función objetivo si alguna de las restricciones no se cumpliese.

7.1. Pressure Vessel Design

El recipiente a presión (*pressure vessel*)[33] es un objeto que se utiliza a día de hoy en una amplia variedad de áreas, tales como la ingeniería química, tratamiento médico, aviación y aeronáutica, incluso en ingeniería nuclear.

Actualmente los recipientes a presión tienden a desarrollarse a gran escala y con un alto número de parámetros, especialmente en la industria química. Estos recipientes se someten a un complejo entorno, en el que aparecen las altas presiones y temperaturas.

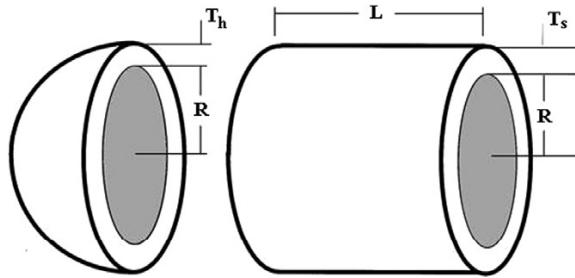


Figura 7.1: Representación del problema *Pressure Vessel Design* [14]

Esto significa no solo un **gran desafío** respecto al rendimiento del material y la estructura, sino también con respecto al diseño del propio recipiente. Será fundamental por tanto, conseguir una **combinación perfecta** de excelente **rendimiento y bajo coste** en el **diseño** de éstos, que además esté **bajo ciertas condiciones** o restricciones.

En este problema tenemos 4 variables de diseño, que se representan como indica la Figura 7.1:

- $x_1 = \text{Radio } (R)$

- x2 = Longitud (L)
- x3 = Espesor del caparazón (T_s)
- x4 = Espesor del recipiente (T_h)

Aquí nuestro principal objetivo será minimizar el coste de fabricación de dicho recipiente. Éste está directamente relacionado con el peso del mismo.

La función objetivo está definida como:

$$f(x) = 0,6224x_1x_2x_3 + 1,7781x_1^2x_3 + 3,1661x_2x_4^2 + 19,84x_4x_1^2$$

Las restricciones se establecen de acuerdo con los códigos de diseño ASME. Las cuatro restricciones que se están considerando son:

$$g_1 = -T_s + 0,0193R \leq 0 \quad (7.4)$$

$$g_2 = -T_h + 0,00954R \leq 0 \quad (7.5)$$

$$g_3 = -\pi R^2 L - \frac{4}{3}\pi R^3 + 750 \cdot 1728 \leq 0 \quad (7.6)$$

$$g_4 = L - 240 \leq 0 \quad (7.7)$$

donde $1 \times 0,0625 \leq T_s$, $T_h \leq 99 \times 0,0625$ y $10 \leq R$. Podemos observar que todas ellas cumplen con la forma descrita en la Ecuación (7.2). Inicialmente en este problema se tomó como límite superior en la variable L el valor de 200.

Más tarde, muchos investigadores la aumentaron hasta 240 para ampliar el espacio de búsqueda y obtener mejores resultados. En la implementación he considerado esta segunda aproximación y los compararemos con dichos autores.

Otra restricción considerada por muchos autores es que los valores de las variables de diseño T_s y T_h únicamente puedan ser múltiplos de 0,0625.

Para tratar las restricciones, seguiremos del modelo general de optimización, formulado anteriormente en la Ecuación (7.3).

7.2. Tension/Compression Spring Design

Diseñar un resorte que pueda cumplir con una tarea mecánica es una parte crucial en un proceso de diseño mecánico. Se necesitan algunas variables

de diseño, como el diámetro del alambre, diámetro medio de la bobina y el número de pliegues activos en ésta.

EL objetivo principal es **minimizar el peso** de estos resortes, el cual está directamente relacionado con su precio de fabricación.

Además se deben de cumplir ciertas restricciones o condiciones en el funcionamiento de estos resortes. Entre ellos se debe cumplir con: una deflexión mínima, frecuencia y esfuerzo de sobretensión, así como con límites en las variables de diseño.

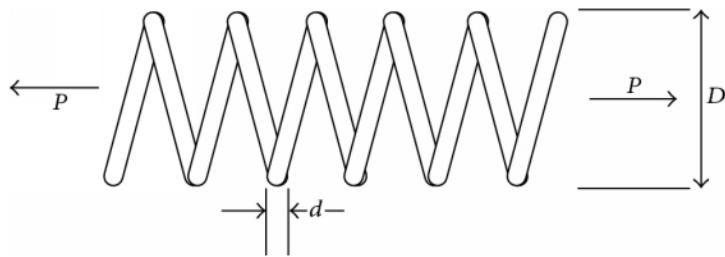


Figura 7.2: Representación del problema *Tension/Compression Spring* [14]

En este problema tenemos 3 variables de diseño, representadas como muestra la Figura 7.2:

- x_1 = Diámetro del alambre (d).
- x_2 = Diámetro medio de la bobina (D).
- x_3 = Número de pliegues activos en la bobina (N).

La función objetivo a minimizar es:

$$f(x) = (x_3 + 2)x_2x_1^2$$

la cual está sujeta a las siguientes cuatro restricciones:

$$g_1(x) = 1 - \frac{x_2^3 x_3}{71785 x_1^4} \leq 0 \quad (7.8)$$

$$g_2(x) = \frac{4x_2^3 x_3}{1256 \cdot (x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} - 1 \leq 0 \quad (7.9)$$

$$g_3(x) = 1 - \frac{140,45 x_1}{x_2^2 x_3} \leq 0 \quad (7.10)$$

$$g_4(x) = \frac{x_1 + x_2}{1,5} - 1 \leq 0 \quad (7.11)$$

donde $0,05 \leq x_1 \leq 2,0$, $0,025 \leq x_2 \leq 1,3$ y $2,0 \leq x_3 \leq 15,0$.

Podemos observar que todas ellas cumplen con la forma descrita en la Ecuación (7.2). Para tratar las restricciones, seguiremos del modelo general de optimización, formulado anteriormente en la Ecuación (7.3).

7.3. 25-Bar Space Truss Design

La optimización estructural es una de las áreas más activas de la ingeniería estructural en los últimos años. Los problemas que la incluye se caracterizan por tener una gran cantidad de variables de diseño con una gran cantidad de restricciones en el mismo [1]. Además éstas llevan un gran espacio de búsqueda que requiere un tiempo razonable para encontrar buenas soluciones.

En el diseño de una estructura de tipo *truss*, el objetivo es minimizar el coste de construcción de la misma, el cual está relacionado directamente con el peso de la estructura.

Además dicha estructura tiene que soportar una carga dada y se deben de satisfacer ciertas restricciones de diseño, como máxima longitud de deflexión en las juntas, un límite de tensión y de compresión permitida en las vigas.

Hay diversas variantes de este problema en la literatura, nosotros proponemos resolver aquella conocida como *25-Bar Truss*, que tiene una representación similar a la de la Figura 7.3.

El peso de la estructura, depende primordialmente de las áreas de sección transversal de cada miembro de la estructura. El problema se reduce por tanto a, seleccionar el área de la sección transversal de cada miembro de forma que el peso de la estructura se minimice siempre y cuando se cumplan todas las restricciones.

La función que calcula el peso de la estructura es definida como:

$$W = \sum_{i=1}^{nm} \gamma_i L_i A_i \quad (7.12)$$

donde W es el peso de la estructura, nm el número de miembros de la estructura, γ a densidad del material del miembro i , L_i la longitud del miembro i y A_i el área de la sección transversal del miembro i .

y sujeta a las restricciones:

$$\sigma^l \leq \sigma \leq \sigma^u \quad (7.13)$$

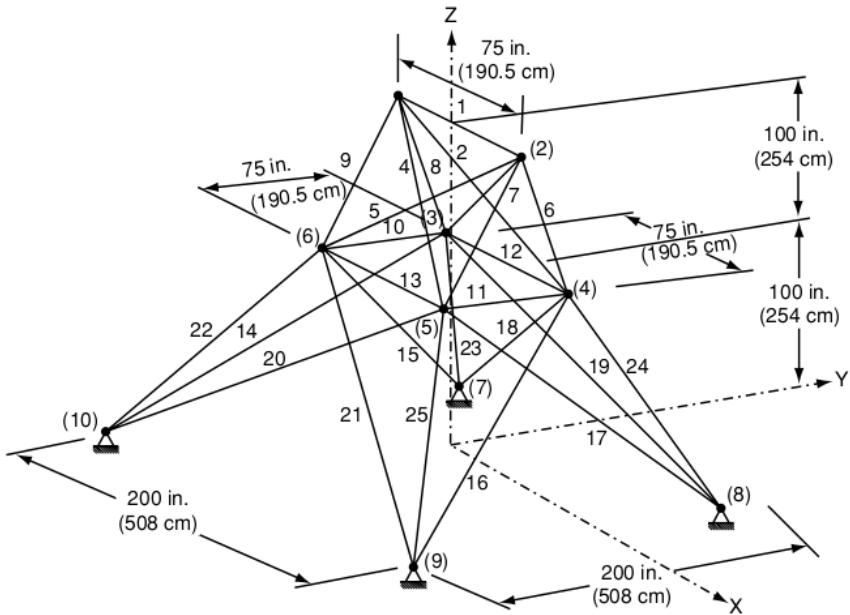


Figura 7.3: Representación del problema 25-Bar Space Truss [14]

$$\delta^l \leq \delta \leq \delta^u \quad (7.14)$$

$$A^l \leq A \leq A^u \quad (7.15)$$

donde las variables σ y δ representarán la tensión y deflexión de la estructura respectivamente, contenida entre los límites inferiores y superiores dados por las cotas con los superíndices l y u respectivamente.

Muchos de los métodos de optimización clásicos requieren información de gradiente para ir mejorando la solución al diseño. Además también dependen de los puntos seleccionados inicialmente.

La complejidad aumenta a medida que las variables de diseño aumentan, es por ello que se han desarrollado diferentes métodos basados en diversas heurísticas (incluidos algoritmos genéticos, inteligencia de enjambre, etc) para resolver estos problemas.

Como veremos en el siguiente capítulo de Implementación 8, nosotros lo resolvemos a través de algoritmos de colonia de hormigas. Éstos poseen ventajas similares a los algoritmos evolutivos porque es un método de búsqueda aleatorizado y de múltiples agentes. Además no necesita que la función dada sea diferenciable.

En el problema hay un solo caso de carga y se proporciona la siguiente

información:

- El módulo de elasticidad del material es de 10^7 psi .
- La densidad de masa del material es de $\gamma = 0,1 \text{ lb/in}^3$.
- La tensión permitida para cada miembro de la armadura es $+/-40 \text{ ksi}$.
- El límite de desplazamiento permitido para cada junta o articulación es de $0,35 \text{ in}$ sobre cada uno de los ejes x,y,z .

El rango de áreas discretas de sección transversal para los miembros de la estructura es de $0,1$ a $3,4 \text{ in}^2$ con un incremento de $0,1 \text{ in}^2$.

Los 25 miembros están organizados en 8 grupos donde todos los miembros pertenecientes a un mismo grupo comparten la misma área transversal. Dicha información se muestra a través de la Tabla 7.1.

El caso de carga considerado se muestra a través de la Tabla 7.2.

Grupos de elementos							
1	2	3	4	5	6	7	8
1:(1,2)	2:(1,4)	6:(2,4)	10:(6,3)	12:(3,4)	14:(3,10)	18:(4,7)	22:(8,6)
	3:(2,3)	7:(2,5)	11:(5,4)	13:(6,5)	15:(6,7)	19:(3,8)	23:(3,7)
	4:(1,5)	8:(1,3)			16:(4,9)	20:(5,10)	24:(4,8)
	5:(2,6)	9:(1,6)			17:(5,8)	21:(6,9)	25:(5,9)

Cuadro 7.1: Agrupamiento de elementos del problema

Nodo	$F_x \text{ (kips)}$	$F_y \text{ (kips)}$	$F_z \text{ (kips)}$
1	1.0	-10.0	-10.0
2	0.0	-10.0	-10.0
3	0.5	0.0	0.0
6	0.6	0.0	0.0

Nota: $1\text{kip} = 4,45\text{kN}$

Cuadro 7.2: Caso de carga para el problema

7.3.1. Adaptación de ACO para Truss Design

Las ventajas de aplicar ACO en el diseño de estructuras es similar a las ventajas de otros algoritmos inspirados en evolución, por ejemplo algoritmos genéticos, los cuales no requieren una relación explícita entre la función

objetivo y las restricciones.

En optimización de estructuras de tipo *truss*, el objetivo suele ser minimizar el coste de fabricación de la estructura, mientras se satisfacen una serie de restricciones como son el estrés permitido y límites de deflexión en las juntas de la estructura.

Para aplicar un **algoritmo de colonia de hormigas** a este tipo de problemas, el concepto de *tour* o ruta planteado para la resolución de TSP como vimos en la Sección 6.3.1 es necesario modificarlo para **adaptarlo** a este nuevo problema [4].

Recordemos que la función objetivo en el problema del viajante de comercio era encontrar la ruta más corta que pasaba por todas las ciudades. En el diseño de este tipo de estructuras el objetivo será **seleccionar las secciones de área** de cada uno de los miembros de la estructura.

Se hace por tanto, una **reducción** al problema del viajante de comercio para poder aplicar el algoritmo de colonia de hormigas, que sabemos que nos sirve para encontrar caminos óptimos sobre un grafo. Son necesarias una serie de modificaciones:

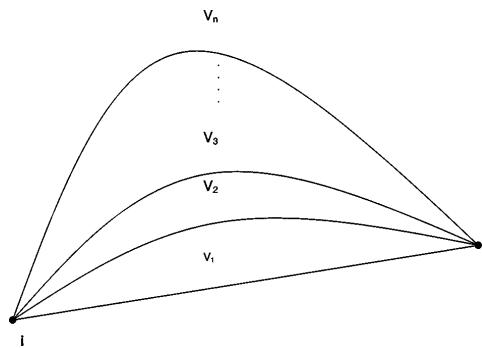


Figura 7.4: Representación de caminos virtuales [4]

- Hay múltiples caminos o aristas de un nodo de la estructura a otro, que se conocen como *caminos virtuales* (ver Figura 7.4). En el tradicional TSP hay un único camino entre dos ciudades i y j .
- El orden en el que se visitan los miembros de la estructura por las hormigas no es importante. A diferencia del TSP que el orden de visita era lo más importante para determinar el coste de la ruta.
- Un diseño desarrollado por una hormiga no es necesariamente factible. Algo que en TSP siempre ocurría ya que bastaba con elegir un nodo una única vez.

Como podemos observar en la Figura 7.4, entre un par de nodos existen **tantos caminos virtuales** como número **áreas de secciones** de corte haya disponibles para unir dicho par de nodos.

La **longitud** de cada *caminos virtual* representa un volumen desde V_1 (el menor) hasta V_n (el mayor) donde n es un número discreto que indica el número de áreas de secciones de corte disponibles para la estructura.

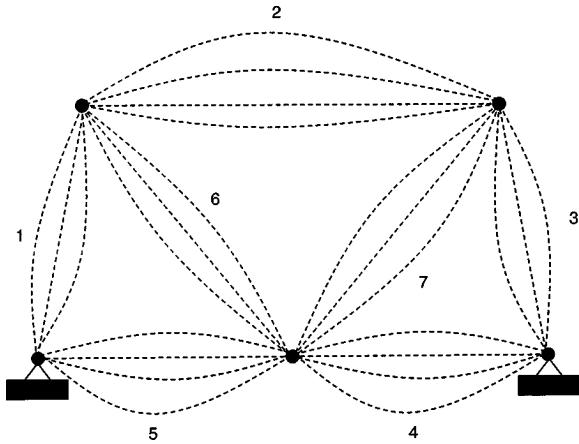


Figura 7.5: Ejemplo de representación del problema [4]

La selección de un conjunto de áreas determinado para cada uno de los miembros de la estructura, definirá una ruta posible en el cual habrá que comprobar si cumple o no las restricciones del diseño de la estructura.

De esta forma, es fácil darse cuenta de que planteando el problema como una búsqueda de caminos en grafos, es decir reduciendo el problema de diseño de estructuras al problema del viajante de comercio, será posible buscar la ruta más corta posible que lo resuelva.

Un ejemplo de representación para una estructura que tenga 5 nodos y un máximo de 3 áreas de sección sería la mostrada en la Figura 7.5.

A continuación, comentaremos de forma general la definición de los operadores de los algoritmos de colonia de hormigas adaptados a la resolución de diseño de estructuras.

Construiremos una **matriz de feromonas**, que llamaremos *trail-matrix*, en la que por cada una de variables de diseño N_s se asumirá un número de distintos *caminos virtuales* o areas de sección posibles N_d [19] (ver Ecuación

7.16).

$$T = \begin{bmatrix} T_{11} & T_{12} & \dots & \dots & T_{1N_d} \\ T_{21} & T_{22} & \dots & \dots & T_{2N_d} \\ \dots & \dots & T_{ij} & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ T_{N_s 1} & T_{N_s 2} & \dots & \dots & T_{N_s N_d} \end{bmatrix} \quad (7.16)$$

Inicializamos el valor de cada posición de la matriz al valor de **feromona inicial** τ_0 , que se define de la forma:

$$\tau_0 = \frac{1}{W_{min}} \quad (7.17)$$

donde W_{min} es el peso de la estructura resultante de asignar el menor área posible a cada uno de los miembros de la estructura, calculados por la Ecuación 7.12. Al ser el mínimo peso posible no cumplirá con las restricciones de las Ecuaciones 7.13 a 7.15.

Para la visibilidad ν_{ij} desde un nodo i a otro j , de forma similar al problema de viajante de comercio (ver Ecuación 6.2), la definiremos como:

$$\nu_{ij} = \frac{1}{A_{ij}} \quad (7.18)$$

siendo A_{ij} el área de sección del miembro que une a dicho par de nodos.

El proceso de selección de uno de los caminos virtuales para un miembro de la estructura se lleva a cabo por cada una de las hormigas, de forma que no se empiezan a elegir los del siguiente miembro hasta que todas las hormigas han seleccionado uno.

Cuando una hormiga hace la elección de un camino virtual, la feromona de éste se reduce. De esta forma se obliga a que no todas las hormigas no sigan los caminos más prometedores, aumentando por tanto la capacidad exploración de la colonia en el espacio de búsqueda.

Para ello, además de la esquema global de evaporación de las feromonas, se introduce un **esquema local de evaporación** definido como:

$$T_{ij}^{new} = \varepsilon \cdot T_{ij}^{old} \quad (7.19)$$

donde ε es el parámetro de tasa de evaporación local de la feromona, que toma valores entre 0 y 1.

Manejo de restricciones [4]

Cada una de las estructuras generadas por las hormigas es analizada para **determinar el estrés** de cada miembro y la **deflección** de cada nodo. Estos valores se comparan con las restricciones de diseño para determinar si la estructura las satisface y es una **solución factible**.

El estrés σ_i en cada miembro i es comparado con los máximos estreses permitidos σ^\pm donde (-) indica el límite de estrés de compresión y (+) el límite de estrés de tensión. La deflección δ_c de cada nodo c es comparada con la máxima deflección permitida δ .

La penalización por el estrés en cada miembro i , Φ_σ^i es calculada como:

$$\text{Si } \sigma^- \leq \sigma_i \leq \sigma^+ \text{ entonces } \Phi_\sigma^i = 0 \quad (7.20)$$

$$\text{Si } \sigma_i > \sigma^+ \text{ o } \sigma_i < \sigma^- \text{ entonces } \Phi_\sigma^i = \frac{\sigma_i - \sigma^\pm}{\sigma^\pm} \quad (7.21)$$

La penalización por estrés total Φ_σ^k generada por la hormiga k es:

$$\Phi_\sigma^k = \sum_{i=1}^{nm} \Phi_\sigma^i \quad (7.22)$$

La penalización por deflección $\Phi_\delta^{cx}, \Phi_\delta^{cy}, \Phi_\delta^{cz}$ en la dirección x, y, z respectivamente para cada uno de los nodos c es determinada de la siguiente forma:

$$\text{Si } \delta_{x,y,z} \leq \delta \text{ entonces } \Phi_\delta^{x,y,z} = 0 \quad (7.23)$$

$$\text{Si } \delta_{x,y,z} > \delta \text{ entonces } \Phi_\delta^{x,y,z} = \frac{\delta_{x,y,z} - \delta}{\delta} \quad (7.24)$$

De forma similar al estrés, la penalización por deflección total generada por la hormiga k es:

$$\Phi_\delta^k = \sum_{c=1}^{nn} (\Phi_\delta^{cx} + \Phi_\delta^{cy} + \Phi_\delta^{cz}) \quad (7.25)$$

donde nn es el número de nodos de la estructura. La **penalización total** Φ^k para la estructura generada por la hormiga k es una función de suma de las anteriores penalizaciones:

$$\Phi^k = (1 + \Phi_\sigma^k + \Phi_\delta^k)^\epsilon \quad (7.26)$$

donde ϵ es un parámetro de penalización positivo. Esta penalización final se agrega multiplicándose a la función objetivo de la forma $W^k = \Phi^k w^k$, donde w^k es el peso actual de la estructura. De esta forma aquellas soluciones que **incumplan más restricciones** tendrán asignado un peso final mayor, y por lo tanto de **menor calidad**.

CAPÍTULO

8

MODELOS IMPLEMENTADOS

En esta sección describo cada uno de los 6 modelos creados (ver Figura 8.1).

Hay dos modelos, uno básico y otro específico, por cada uno de los algoritmos bioinspirados explicados anteriormente en el Capítulo 6.

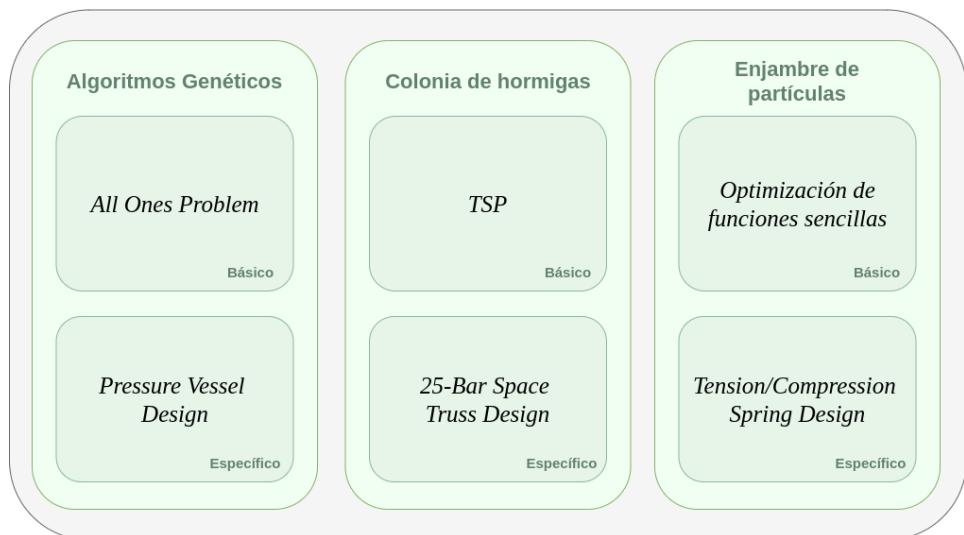


Figura 8.1: Modelos implementados en NetLogo

8.1. Diagramas de Clases

Los diagramas de clases son uno de los tipos de diagramas más útiles en UML. Trazan la estructura de un sistema concreto al modelar sus clases, atributos, operaciones y relaciones entre objetos.

A continuación, presentamos los diagramas de clases diseñados para cada uno de los algoritmos bioinspirados: algoritmos genéticos (ver Figura 8.2), algoritmos de enjambre de partículas (ver Figura 8.3) y algoritmos de colonia de hormigas (ver Figura 8.4).

Se mostrará en color **verde** aquellos atributos primitivos que aportan los agentes de NetLogo por defecto. Para el resto de atributos y operaciones se marcará en **morado** los respectivos al **modelo básico**, en **azul** los respectivos al **modelo específico** y en **negrita** los **comunes** en ambos modelos.

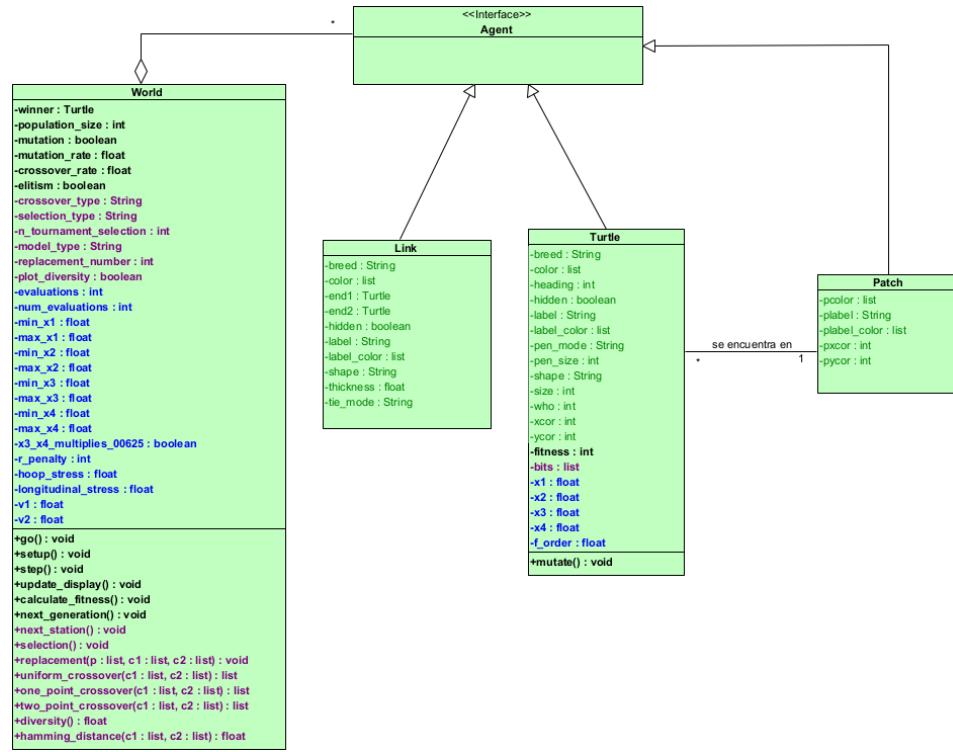


Figura 8.2: Diagrama de clases para modelos GA

Nota: Hay muchas más operaciones y atributos primitivos propios del mundo y del resto de agentes de NetLogo que pueden consultarse en su **diccionario**, y que no están incluidos en los presentes diagramas de clases para que sean autocontenidos. Únicamente se muestran aquellos que son más relevantes.

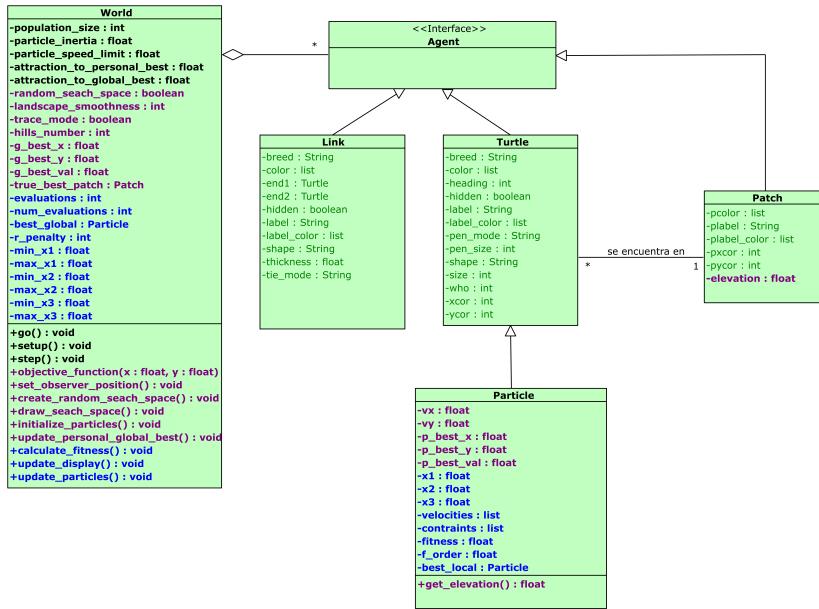


Figura 8.3: Diagrama de clases para modelos PSO

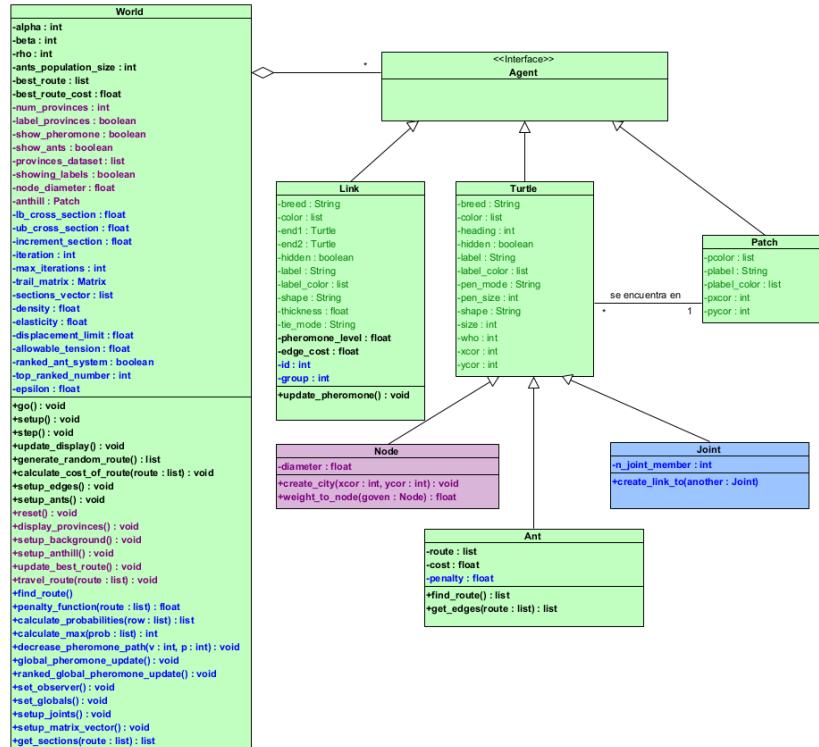


Figura 8.4: Diagrama de clases para modelos ACO

8.2. Algoritmos Genéticos

8.2.1. Modelo Básico

Este modelo pretende demostrar el uso de los algoritmos genéticos en un problema sencillo: encontrar el cromosoma que tiene todos los valores de genes iguales a 1.

Se partirá de una cadena con ceros y unos, inicializada con un mayor proporción de ceros. Se pretende alcanzar una solución que sea una secuencia de 1s y no contenga 0s. Por tanto, la solución que mejor resuelve el problema es "11111 ... 11".

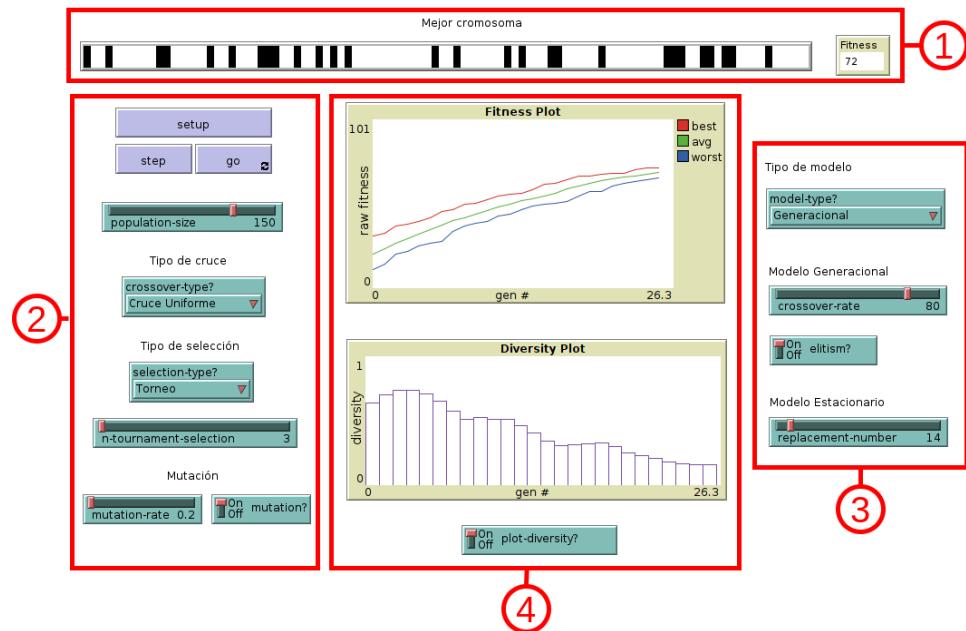


Figura 8.5: Interfaz modelo GA-Básico

En la interfaz para este modelo sencillo de algoritmos genéticos, se pueden diferenciar distintas partes como muestra la Figura 8.5.

A continuación describiremos cada una de esas **cuatro partes**, así como la función de los selectores, deslizadores y botones de la interfaz:

1. Mundo. En la ventana del mundo, se mostrará el cromosoma con el mejor valor de *fitness* de toda la población. Al lado hay un pequeño **monitor** con dicho valor de *fitness*.

Los valores ceros o unos de los genes serán interpretados como una parcela del mundo de color negro o blanco respectivamente como muestra la Figura

8.5-1.

2. Configuración de operadores genéticos. En esta parte de la interfaz, además de los botones principales para la ejecución del modelo, tendremos los selectores y deslizadores para elegir el tipo de cruce, selección y si queremos que haya mutación o no.

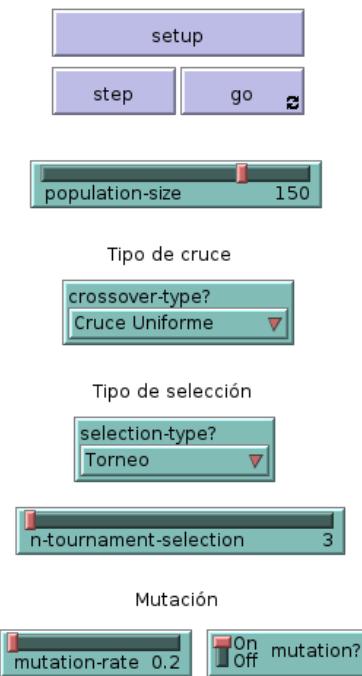


Figura 8.6: Configuración de operadores GA-Básico

- El deslizador **population-size** controla el número de soluciones o cromosomas que están presentes en cada generación.
- El selector **crossover-type** controla qué tipo de cruce se va a utilizar el algoritmo genético: cruce uniforme o cruce por segmento.
- El selector **selection-type** controla qué tipo de método de selección preferimos para seleccionar a los padres: selección por torneo o selección por ruleta.
- El deslizador **n-tournament-selection** controla cuantos posibles padres vas a estar involucrados cuando se haya seleccionado la selección por torneo. Normalmente se utiliza un valor de 3.
- El deslizador **mutation-rate** controla la probabilidad de mutación. Esta posibilidad se aplica a cada gen en la cadena de bits de un nuevo individuo.

Por ejemplo, si la cadena tiene 100 bits de longitud y la tasa de mutación se establece en 1 %, entonces, en promedio, se cambiará un bit durante la creación de cada nuevo individuo.

- Además contamos con un interruptor **mutation** que permite activar y desactivar este operador de mutación típico de los algoritmos genéticos.

3. Configuración del esquema evolutivo. En esta parte de la interfaz se podrá ajustar si preferimos un esquema de evolución generacional o estacionario. Además se dará la opción de elegir el uso de elitismo.

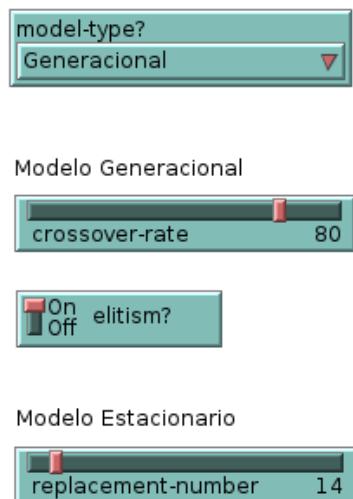


Figura 8.7: Configuración del esquema GA-Básico

- El selector **model-type** controla qué esquema (forma de generar la siguiente población) será utilizada por nuestro algoritmo genético.

- El deslizador **crossover-rate** controla qué porcentaje de cada nueva generación se crea a través de la reproducción sexual (recombinación o cruce entre el material genético de dos padres) y qué porcentaje ($100 - \text{crossover rate}$) se crea a través de la reproducción asexual (clonación del material genético de uno de los padres).

- El interruptor **elitism** controla si queremos garantizar que la mejor solución encontrada (el cromosoma con el mejor valor de fitness) se mantenga en la siguiente generación o no.

- El deslizador **replacement-number**

controla cuantos pares de cromosomas queremos que sean reemplazados antes de concluir con una estación y pasar a la siguiente generación. Si tenemos un valor de 10 para esta variable, 20 nuevos individuos serán insertados en la generación actual para generar una generación nueva.

Estos deslizadores, selectores e interruptores serán **variables globales** del modelo básico como se muestra en el diagrama de clases de la Figura 8.2.

4. Estadísticas de rendimiento. En esta parte de la interfaz se podrá visualizar en varios gráficos cómo es la evolución de la calidad de los cromosomas a lo largo de las generaciones, así como la diversidad genética entre ellos en función del tiempo.

- Esto primero se hace puede ver, tanto para el mejor, peor y media de cromosomas a través del gráfico **fitness plot** (ver Figura 8.5).
- El interruptor **plot-diversity** controla si queremos dibujar o no la cantidad de diversidad dentro de la población de soluciones en cada generación. Se muestra en el “Gráfico de diversidad”. Si no está activado, se aumenta significativamente la velocidad del modelo porque calcular la diversidad requiere bastante cómputo.

8.2.2. Modelo Específico - Pressure Vessel Design Problem

Este modelo demuestra el uso de algoritmos genéticos sobre un problema específico de optimización: *Pressure Vessel Design* o en español, problema del recipiente a presión.

Como hemos comentado anteriormente, el problema tiene 4 variables de diseño, por lo tanto el vector de genes de cada uno de los cromosomas se podrá representar como $X = [x_1, x_2, x_3, x_4]$.

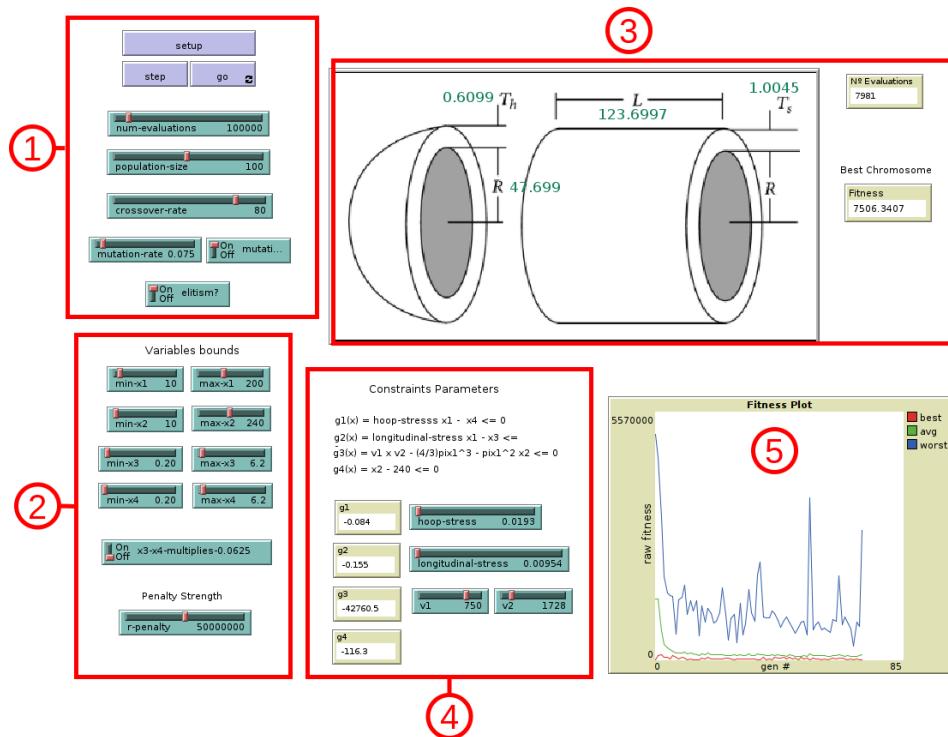


Figura 8.8: Interfaz modelo GA-Específico

En la interfaz para este modelo específico de algoritmos genéticos, se pueden diferenciar distintas partes como muestra la Figura 8.8.

A continuación describiremos cada una de esas **cinco partes**, así como la función de los gráficos, monitores, selectores, deslizadores y botones de la interfaz:

1. Configuración del algoritmo genético. En esta parte de la interfaz, además de los botones para controlar la ejecución del modelo, tendremos los selectores y deslizadores para elegir los principales parámetros de nuestro algoritmo genético al igual que en el modelo básico. Concretamente son los deslizadores de tamaño de la población y probabilidades de cruce y mutación,

así como un interruptor para habilitar la mutación y el elitismo.

En este caso, al situarnos sobre un problema específico no daremos tanta variedad a la hora de elegir el tipo de selección y tipo cruce entre cromosomas. El esquema evolutivo que implementa es el generacional, con cruce uniforme y selección por torneo ($k=3$).

A diferencia del modelo básico, tendremos un deslizador adicional:

- El deslizador **num-evaluations** controla el número de evaluaciones máximo que se harán como máximo para finalizar la ejecución del algoritmo. Este será por tanto nuestro **criterio de parada**.

2. Configuración del problema. En esta parte nos encontraremos los deslizadores asociados al dominio de las variables de diseño de nuestro problema.

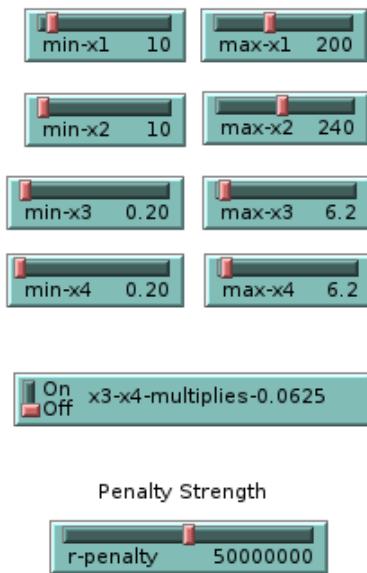


Figura 8.9: Configuración del dominio de *Pressure Vessel*

más alto que otras soluciones que no rompan tantas restricciones.

Podemos ver con más detalle como afecta dicho parámetros en la función de *fitness* modificada $f'(x)$ (ver Ecuación (7.3)).

Todos estos deslizadores e interruptores serán **variables globales** del modelo específico como se muestra en el diagrama de clases de la Figura 8.2.

3. Mundo. En esta parte de la interfaz, se verá el esquema del problema (ver Figura 7.1) y junto a cada una de sus **variables de diseño**, su valor

- Los controles deslizantes **min-x** y **max-x** son los que controlan los límites superior e inferior que definen el dominio de cada una de las variable de diseño. Podemos intentar establecer diferentes límites superiores a la longitud L , para comprobar que aumentando el espacio de búsqueda alcanzamos mejores soluciones.
- El interruptor **x3-x4-multiplies-0.0625** controla si los valores considerados en las variables de diseño x_3 y x_4 son valores en el dominio y múltiplos de 0,0625 (es una restricción adicional que muchos autores consideran).

- El control deslizante **r-penalty** controla cuánta penalización vamos a agregar al valor de fitness de cada solución. Si una solución rompe muchas restricciones, este parámetro tendrá un valor

que cambiará en **tiempo de ejecución**.

Además a la derecha de éste habrá un par de **monitores** que indicarán el valor de *fitness* asociado al valor de las variables en dicho momento y un contador con el número de evaluaciones hasta el momento.

4. Visualización de restricciones. En esta parte de la interfaz se puede visualizar a través de los monitores g_1 a g_4 en tiempo de ejecución del algoritmo, el valor que están tomando cada una de las restricciones enumeradas en el problema (ver Ecuaciones 7.4 a 7.7).

Además podremos modificar algunos parámetros que según el enunciado del problema, podrían considerarse con otros valores y que están implícitos en el cálculo de dichas restricciones.

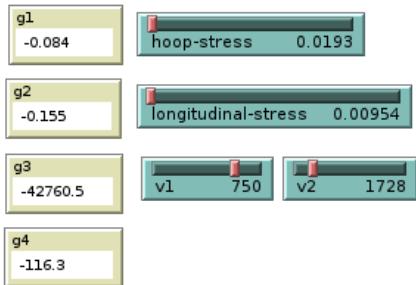


Figura 8.10: Configuración de restricciones *Pressure Vessel*

- El deslizador **hoop-stress** indicará el valor de estrés permitido para el cabezal de nuestro recipiente.
- El deslizador **longitudinal-stress** indicará el valor de estrés permitido para la otra parte con forma de cilindro de nuestro recipiente.
- Los deslizadores **v1** y **v2** son parámetros que restriccionarán el volumen permitido de nuestro recipiente.

El valor por defecto de estos parámetros son los elegidos por los autores (incluir referencia) y además hemos permitido que no puedan establecerse valores negativos tanto en volumen como en medidas longitudinales, ya que no son valores permitidos.

Depende de la presión o tensión soportada por cada material. En nuestro caso el material utilizado es acero al carbono ASME SA 203 grado B.

Por supuesto también se **comprobará** a la hora de generar los cromosomas, ya sea de forma aleatoria al inicio del algoritmo o como resultado de cruces entre éstos, que los **valores** de los **genes** están dentro del **dominio permitido**.

5. Gráfica de rendimiento. Por último, se incluirá un gráfico llamado **Fitness Plot** que nos permitirá al igual que en el modelo básico, ver el tránscurso del mejor y peor cromosoma (además de un valor medio de éstos) de nuestra población a lo largo del tiempo (ver Figura 8.8).

8.3. Colonia de hormigas

8.3.1. Modelo Básico

En este modelo, se implementa el algoritmo de colonia de hormigas para resolver el problema clásico del viajante de comercio, del inglés *Travelling Salesman Problem* (TSP).

Para facilitar la futura comprensión de la explicación del modelo, se tratará de minimizar una ruta que pase por las capitales de provincia de España. Para ello, hemos hecho uso de la extensión *GIS*¹ de NetLogo.

De esta forma, cada una de las capitales de provincia será un nodo en el grafo. Sobre dicho grafo, las hormigas tendrán que encontrar el circuito hamiltoniano de menor longitud.

Los **datos** de los límites provinciales y autonómicos para poder representar el mapa y las ciudades los hemos descargado directamente del *Centro Nacional de Información Geográfica (CNIG)*.

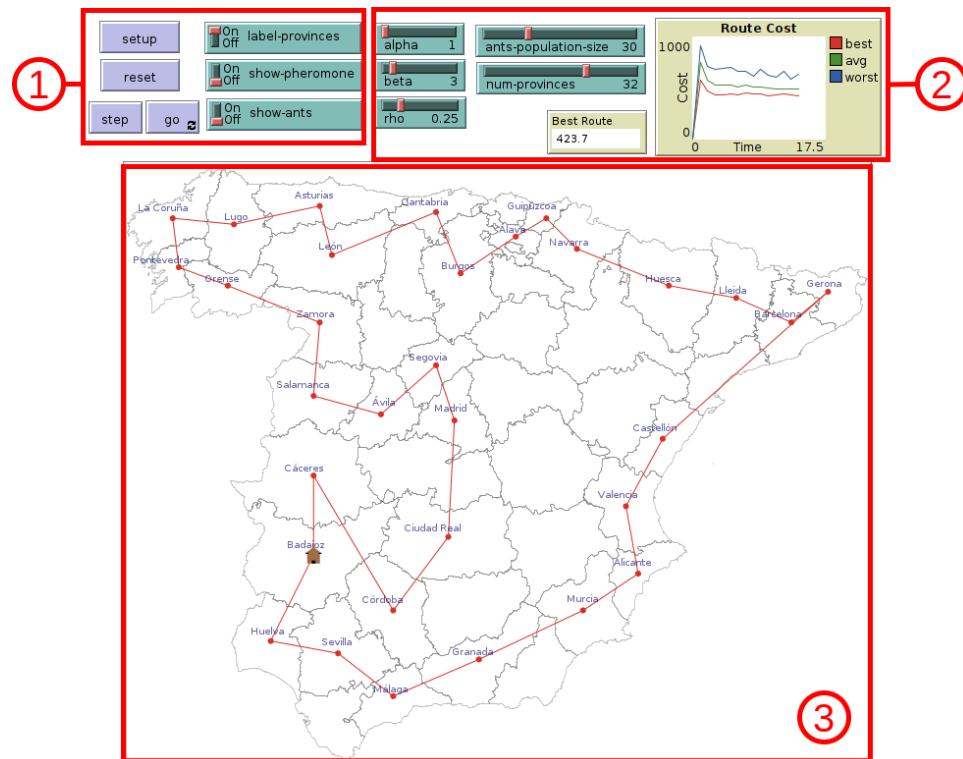


Figura 8.11: Interfaz modelo ACO-Básico

En la interfaz para este modelo básico de algoritmos de colonia de hormigas,

¹<https://ccl.northwestern.edu/netlogo/docs/gis.html>

se pueden diferenciar distintas partes como muestra la Figura 8.11.

A continuación describiremos cada una de esas **tres partes**, así como la función de los selectores, interruptores, deslizadores y botones de la interfaz:

1. Control de ejecución y visualización. En esta parte de la interfaz tendremos los botones principales para la ejecución del modelo.

Contamos con los botones estándar para el control de la ejecución además de uno llamado **reset**. Su función será volver a inicializar tanto la ruta inicial de nuestro problema TSP (ver Sección 6.3.1), como los valores iniciales de feromonas de cada una de las aristas del grafo.

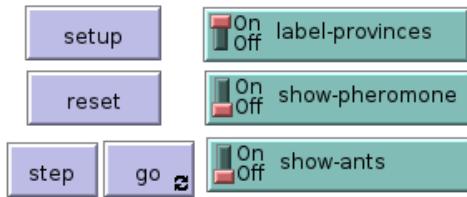


Figura 8.12: Configuración de visualización ACO Básico

- El interruptor **label-provinces** permite visualizar el nombre de cada capital de provincia (nodo).
- El interruptor **show-pheromone** permite visualizar la cantidad/nivel de feromonas que las hormigas depositan en cada arista. Habrá más feromonas en una arista que es más gruesa y menos feromonas en una arista que sea más translúcida que otra.

- El interruptor **show-ants** permite visualizar las hormigas recorriendo las rutas encontradas por cada una de ellas. Cuando todas las hormigas llegan al hormiguero (todas han construido su solución) se aplica el esquema de actualización global de las feromonas.

2. Configuración de la colonia de hormigas. En esta parte de la interfaz se proporcionan deslizadores para controlar los parámetros básicos utilizados en este tipo de algoritmos. Se puede ver una explicación más detallada de ellos en la Sección 6.3.

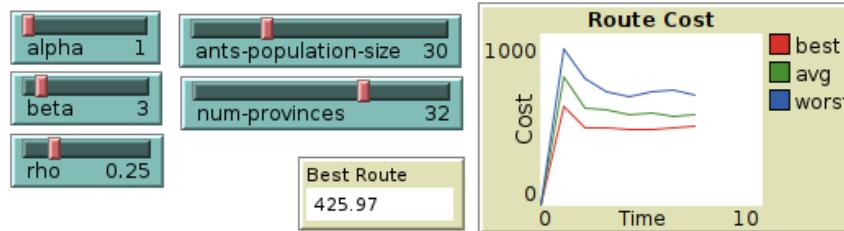


Figura 8.13: Configuración de modelo ACO Básico

- El deslizador **ants-population-size** controla el número de hormigas de nuestra población.

- El deslizador **num-provinces** controla el número de nodos (capitales de provincias) que vamos a considerar para construir la solución. Observe que la ubicación del nodo capital es el centroide de la provincia y no la ubicación real.
- El deslizador **alpha** controla la influencia de la feromonía sobre las hormigas a la hora de seleccionar una nueva componente para la solución.
- El deslizador **beta** controla la influencia de la heurística de visibilidad sobre las hormigas a la hora de seleccionar una nueva componente para la solución (como de greedy queremos que sean las hormigas).
- El deslizador **rho** controla el valor del parámetro de tasa de evaporación de la feromonía.

Todos estos deslizadores serán **variables globales** del modelo básico y específico para algoritmos de colonias de hormigas como se muestra en el diagrama de clases de la Figura 8.4. Los interruptores pertenecerán únicamente al modelo básico.

En la parte derecha podemos ver que hay un **grafico** con el valor de la mejor y peor ruta encontrada por la colonia de hormigas a través del tiempo. Además se incluye un **monitor** que muestra el valor de la mejor ruta encontrada hasta el momento (será la misma que se visualizará en el mundo).

3. Mundo. Es la parte principal de nuestro modelo. A través de él podremos **visualizar las hormigas** y la **feromonía** que éstas depositan, así como la mejor ruta encontrada para nuestro conjunto de ciudades establecido.

Dependiendo de la configuración de visualización que elijamos así como de la velocidad de ejecución (ajustes del propio NetLogo), podremos ver con mejor detalle cómo las hormigas van mejorando la solución al problema cada iteración del algoritmo.

En la Figura 8.14, de forma análogo a como se explicó para la Figura 6.9, se muestra un ejemplo real de la **visualización de la feromonía** durante el tránscurso de la **ejecución** del modelo básico para colonia de hormigas.

Podemos ver que aquellas aristas del grafo por las han pasado un mayor número de hormigas tendrá un color de azul más intenso y grueso. Seguramente estas aristas serán las que formen parte del circuito final construido.

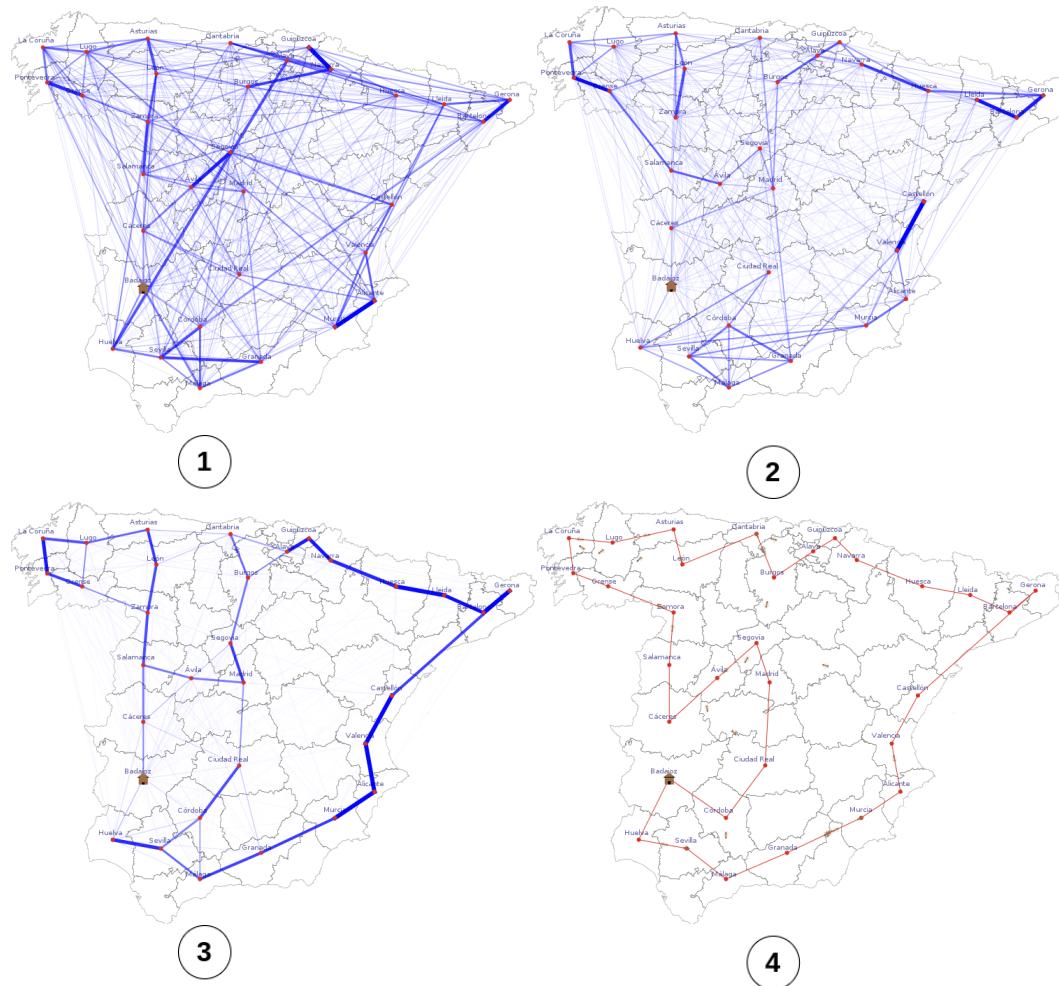


Figura 8.14: Visualización de feromona del modelo

8.3.2. Modelo Específico - 25-Bar Space Truss Design

En este modelo 3D se pretende mostrar el uso de algoritmos de colonia de hormigas en la resolución de un problema típico de ingeniería de estructuras, como es el *25-Bar Space Truss Design*, explicado con mayor detalle en la Sección 7.3.

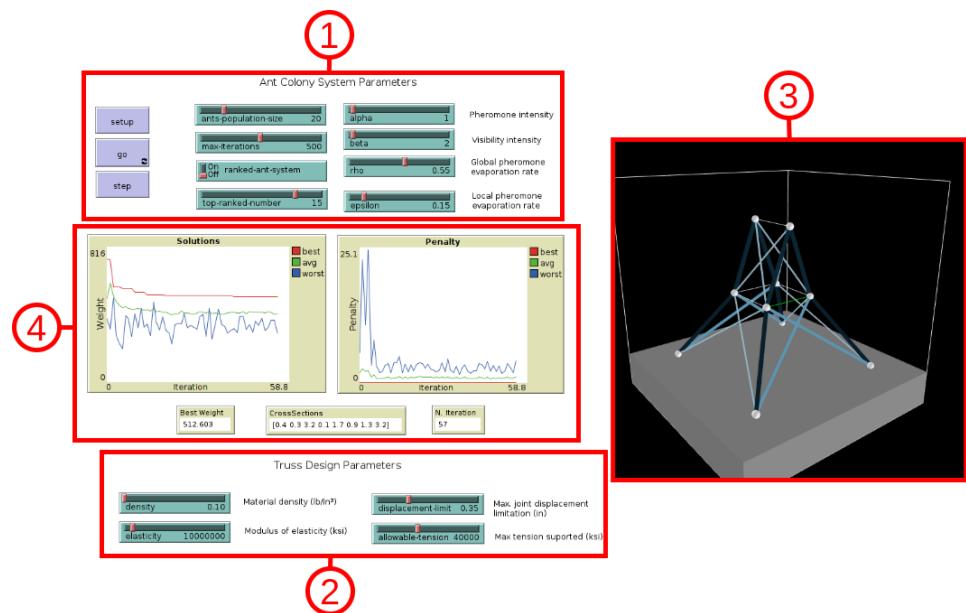


Figura 8.15: Interfaz modelo ACO-Específico

En la interfaz para este modelo específico de algoritmos de colonia de hormigas, se pueden diferenciar distintas partes como muestra la Figura 8.15.

A continuación describiremos cada una de esas **cuatro partes**, así como la función de los selectores, interruptores, deslizadores, botones de la interfaz y el tipo de información que muestran los gráficos:

1. Configuración del algoritmo y parámetros básicos. En esta parte de la interfaz tendremos los botones principales para la ejecución del modelo, así como los parámetros principales para este tipo de algoritmos. Además un interruptor para habilitar otra variante del algoritmo como comentaremos a continuación.

- El control deslizante **ants-population-size** controla el número de hormigas de nuestra colonia.
- El control deslizante **max-iterations** controla cuántas iteraciones deseamos que se ejecute nuestro algoritmo de colonia de hormigas.

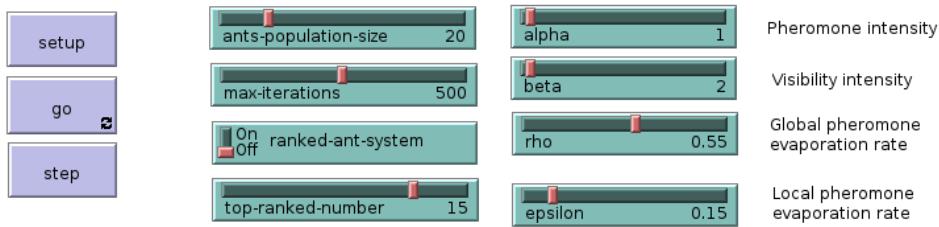


Figura 8.16: Parámetros básicos de ACO Específico

- El interruptor **ranked-ant-system** permite habilitar la variante *Rank Ant System* del algoritmo de optimización de colonias de hormigas. En esta variante, las soluciones se clasifican según su idoneidad y cada hormiga deposita una cantidad de feromonas proporcional a la bondad de su solución.

La principal diferencia respecto al algoritmo de colonia de hormigas clásico reside en el esquema de actualización de la feromonas (referencia al artículo):

- El control deslizante **top-ranked-number**, asociado al anterior interruptor, controla cuántas hormigas queremos tener en cuenta a la hora de construir la clasificación.

Los parámetros **alpha**, **beta** y **rho** son los mismos que los comentados anteriormente en el modelo básico. Sirven para controlar el funcionamiento del algoritmo de colonia de hormigas general.

Adicionalmente, esta implementación del problema usando colonia de hormigas, lleva un parámetro adicional que es **epsilon**: determina la frecuencia de la tasa de **evaporación local**.

2. Configuración de los parámetros del problema. El resto de parámetros particulares para el problema *25-Bar Space Truss Design* son los siguientes:



Figura 8.17: Parámetros particulares de ACO Específico

- El deslizador **density** permite establecer el valor de densidad del material utilizado en la estructura.

- El deslizador **elasticity** permite establecer el valor de densidad asociado al material utilizado en la estructura.
- El deslizador **displacement-limit** permite establecer el valor máximo de desplazamiento permitido en las juntas o nodos de la estructura
- El deslizador **allowable-tension** permite establecer el valor máximo de tensión soportada por cada uno de los miembros que compone la estructura.

Al lado de cada uno de ellos, se añade una pequeña nota informativa con el nombre de dicho parámetro y las unidades en las que se expresa. Estos deslizadores e interruptores serán **variables globales** del modelo como se muestra en el diagrama de clases de la Figura 8.4.

3. Mundo. Al ser un modelo en 3D, la visualización del mundo se hará en una pestaña adicional a la que se encuentran los botones principales de la interfaz. En este caso, veremos la **estructura** de nuestro problema, construida con unas **vigas** con un **grosor** y **color** en función de las variables de la **mejor solución** encontrada por el algoritmo en dicho momento.

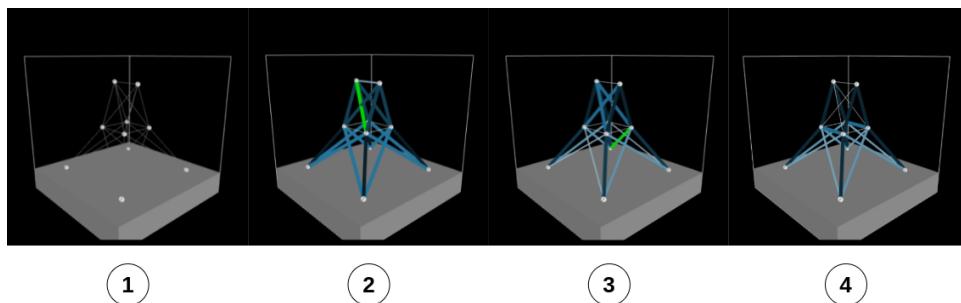


Figura 8.18: Visualización del mundo en Truss Design

En el momento en el que una **nueva solución** se haya encontrado por nuestra colonia de hormigas, se hará una transición de **transformación de la estructura**, en la que se establecerá el nuevo grosor y color de aquellas secciones de áreas encontradas (ver Figura 8.18).

4. Visualización del rendimiento del modelo

En la Figura 8.19 podemos ver en la parte izquierda el valor de *fitness* para la mejor, peor y media de soluciones encontradas por las hormigas.

En la parte derecha, de la misma forma podremos ver el **valor de penalización** asociado a dichas soluciones. De esta forma, la mejor solución siempre deberá tener un valor de penalización 0. Cuanto peor sea la solución, mayor será su valor de penalización.

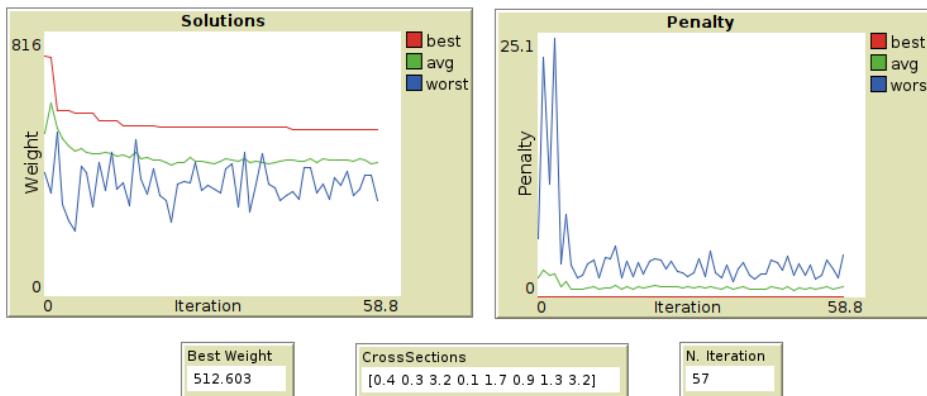


Figura 8.19: Gráficos para ACO Específico

En la parte inferior podemos ver varios **monitores**, de izquierda a derecha, estos indican: el valor de *fitness* o peso de la estructura, la secuencia de áreas de secciones transversales para los distintos grupos de vigas y la iteración del algoritmo.

8.4. Enjambre de partículas

8.4.1. Modelo Básico

En este modelo básico, el enjambre de partículas puede tratar de optimizar dos opciones:

1. Una **función sencilla**, hemos elegido la función que define a una *esfera* $f(x, y) = x^2 + y^2$ (ver Figura 8.22).
2. Se crea un **espacio de búsqueda aleatorio**, en el que habrá uno o dos varios óptimos globales y las partículas tratarán de alcanzarlo (ver Figura 8.23).

En la interfaz para este modelo básico de algoritmos de enjamnbro de partículas, se pueden diferenciar distintas partes como muestra la Figura 8.20.

A continuación describiremos cada una de esas **cuatro partes**, así como la función de los interruptores, deslizadores y botones de la interfaz:

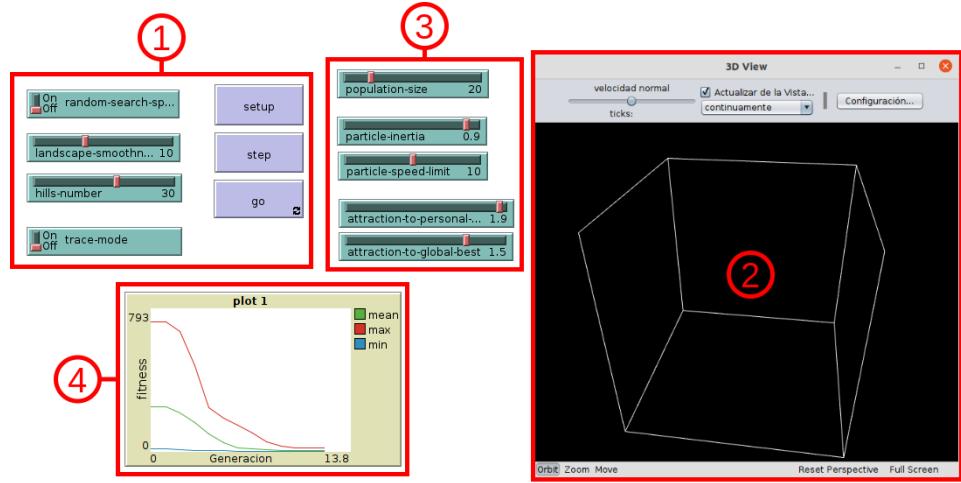


Figura 8.20: Interfaz modelo PSO-Básico

1. Control de ejecución y visualización.

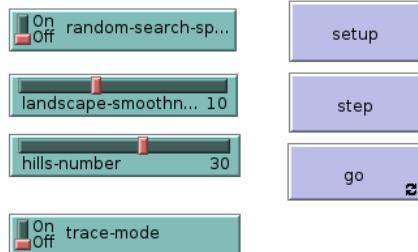


Figura 8.21: Configuración de visualización PSO Básico

- Presione el botón **setup** para inicializar el paisaje de fitness y colocar las partículas al azar en el espacio de la función de esfera.
- Si se selecciona el interruptor **random-search-space**, cada vez que presione **setup**, se creará un paisaje aleatorio diferente.
- Presione el botón **step** (para un paso) o **go** para ejecutar el algoritmo de optimización del enjambre de partículas.

- El deslizador **landscape-smoothness** determina cómo de suave queremos que sea el paisaje que se cree (cuanto más bajo más apreciables son los picos) cuando se presione el botón **SETUP**.
- El deslizador **hilss-number** determina cuántas casillas (parcelas) recibirán una gran cantidad de elevación para simular colinas en nuestro espacio de búsqueda aleatorio.
- El interruptor **trace-mode** permitirá la visualización de la trayectoria de cada una de las partículas a modo de historial gráfico.

Estos deslizadores e interruptores serán **variables globales** del modelo básico, como se muestra en el diagrama de clases de la Figura 8.3.

2. Mundo. En esta parte de la interfaz podremos visualizar en 3D la función que se estará optimizando a través del enjambre de partículas. A continuación mostramos un ejemplo de visualización para ambos casos de nuestro modelo (ver Figuras 8.22 y 8.23).

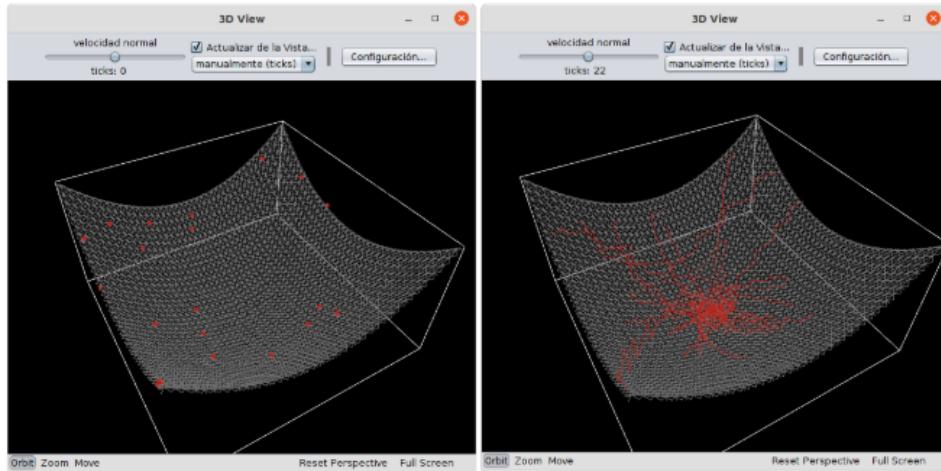


Figura 8.22: Espacio de búsqueda de función sencilla

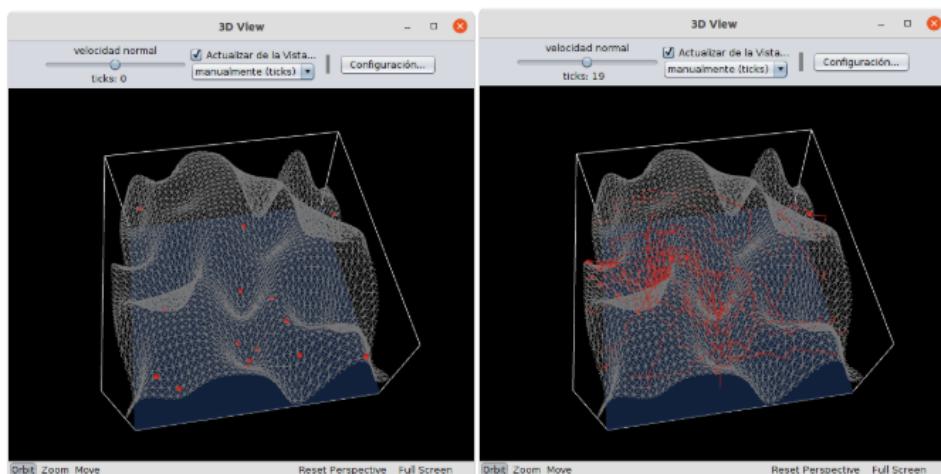


Figura 8.23: Espacio de búsqueda aleatorio

Podemos observar en la imagen derecha de las figuras anteriores, una captura de la ejecución del algoritmo habiendo activado la opción **trace-mode**. De esta forma podemos visualizar la trayectoria seguida por las partículas.

3. Parámetros para PSO

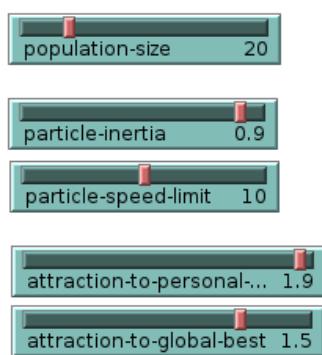


Figura 8.24: Configuración de parámetros para PSO

donde anteriormente había encontrado el valor más alto (en su propia historia).

- El deslizador **attraction-to-global-best** determina la fuerza de atracción de cada partícula hacia la mejor ubicación jamás descubierta por cualquier miembro del enjambre.

Estos deslizadores serán **variables globales**, utilizadas en el modelo básico y en el específico, como se muestra en el diagrama de clases de la Figura 8.3.

4. Gráfico de fitness. Al igual que en los otros modelos, se graficará a través del tiempo el valor de *fitness* para la mejor, peor y media de soluciones encontradas por las partículas. En la Figura 8.25 podemos ver los gráficos de rendimiento para la función sencilla (izquierda) y para el espacio de búsqueda aleatorio (derecha).

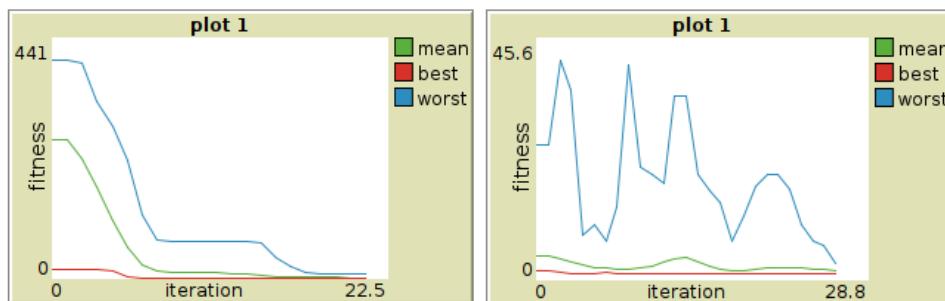


Figura 8.25: Gráficos de rendimiento para PSO

8.4.2. Modelo Específico - Tension/Compression Spring Design Problem

Este modelo demuestra el uso de algoritmos de enjambre de partículas sobre un problema específico de optimización: *Tension/Compression Spring Design* o en español, problema del diseño de resortes. Este problema se explica con más detalle en la Sección 7.2.

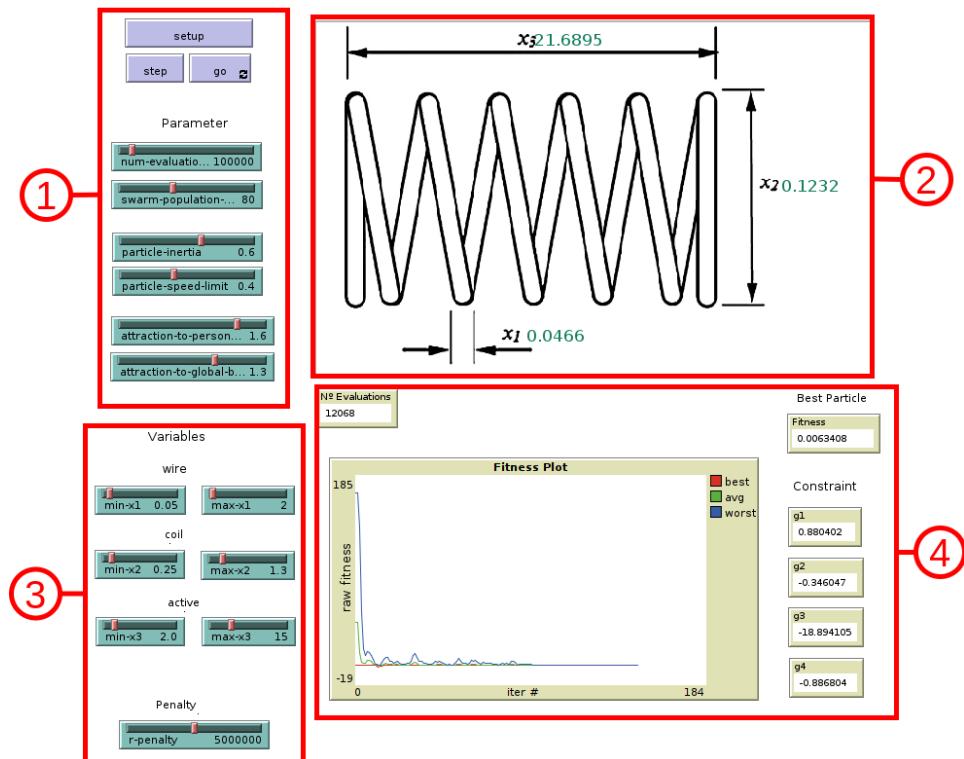


Figura 8.26: Interfaz modelo PSO-Específico

En la interfaz para este modelo específico de algoritmos de enjambre de partículas, se pueden diferenciar distintas partes como muestra la Figura 8.26.

A continuación describiremos cada una de esas **cuatro partes**, así como la función de los gráficos, monitores, selectores, deslizadores y botones de la interfaz:

1. Configuración de parámetros del algoritmo. En esta parte de la interfaz aparecen los deslizadores para establecer los parámetros típicos de un algoritmo de enjambre de partículas.

En el modelo básico ya explicamos el funcionamiento de cada uno de ellos (ver Sección 8.4.1). Además se añade el deslizador **num-evaluations** que

permitirá establecer un número máximo de evaluaciones de la función objetivo, a modo de criterio de parada.

2. Mundo. En esta parte de la interfaz, se verá el esquema del problema (ver Figura 7.2) y junto a cada una de sus **variables de diseño**, su valor que cambiará en **tiempo de ejecución**.

Además a la derecha de éste habrá un par de **monitores** que indicarán el valor de *fitness* asociado al valor de las variables en dicho momento y un contador con el número de evaluaciones hasta el momento.

3. Configuración del problema. En esta parte nos encontraremos los deslizadores asociados al dominio de las variables de diseño de nuestro problema.

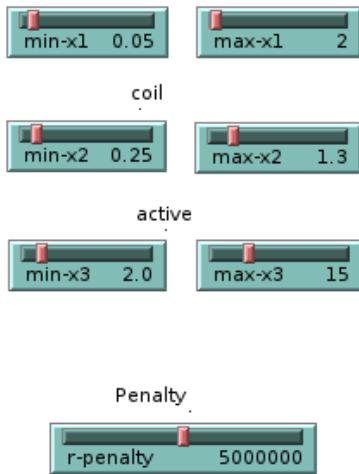


Figura 8.27: Configuración del dominio de *Spring Design*

- Los controles deslizantes **min-x** y **max-x** son los que controlan los límites inferior y superior que definen el dominio de las variables de diseño.

Controlan cómo vamos a restringir la optimización del diseño del recipiente a presión. Depende de la presión o tensión soportada por cada material.

- El control deslizante **r-penalty** controla cuánta penalización vamos a agregar al valor de fitness de cada solución. Si una solución rompe muchas restricciones, este parámetro tendrá un valor más alto que otras soluciones que no rompan tantas restricciones.

Más detalles de como afecta dicho parámetros en la función de *fitness* modificada (ver Ecuación (7.3)).

4. Visualización y restricciones. En esta parte de la interfaz se puede visualizar a través de los monitores g_1 a g_4 en tiempo de ejecución el valor que están tomando cada una de las restricciones enumeradas en el problema (ver Ecuaciones 7.8 a 7.11).

Además, como ya acostumbrados en los modelos anteriores, un gráfico para la evolución del valor de *fitness* de nuestro modelo, así como un monitor para la mejor solución encontrada y un contador de evaluaciones.

CAPÍTULO

9

PRUEBAS

9.1. Sistema de tests específico

Será necesario comprobar que los modelos desarrollados son lo suficientemente robustos, para que independientemente de la configuración de parámetros que elija el usuario desde la interfaz, no tengamos ningún problema de ejecución con ellos.

No es lo mismo la ausencia de aparición de errores que la ausencia de aparición de buenas soluciones. Estas últimas se darán si los parámetros asociados a un algoritmo determinado son apropiados, pero no impiden la ejecución y búsqueda de soluciones de forma menos eficiente a través del mismo.

Además de las comprobaciones que se mostrarán a continuación, recordamos que se han **tenido en cuenta** los siguientes **aspectos** a la hora de **diseñar** los deslizadores y botones de la interfaz de usuario en los modelos:

- Para los deslizadores asociados al **dominio** de las variables de diseño, el mínimo valor y máximo valor no podían exceder los propios indicados por el autor en el caso de los problemas o por la propia naturaleza del parámetros. Por ejemplo no se permite que una distancia sea negativa.
- Para los deslizadores asociados a los **parámetros** de los algoritmos, hemos respetado y utilizado como valor mínimo y máximo los propios recomendados por los autores. De la misma forma para establecer el valor inicial por defecto de éstos.

- Hay parámetros que hacen referencia a tasas o probabilidades dentro del algoritmo y por sentido común tendrán valores asociados entre cero y uno.

También se ha probado la ejecución de los modelos en **diferentes plataformas**, como son Windows, Linux y Mac. Al ser NetLogo una herramienta multiplataforma, no tenemos ningún problema en ello.

Respecto a los modelos desarrollados en 3D, también se ha comprobado que el ángulo de visualización del observador es indiferente de la ejecución del mismo. La propia herramienta de NetLogo nos da esa garantía.

Adicionalmente NetLogo, cuenta con una **herramienta** llamada *BehaviourSpace*, que permite la **realización de experimentos** sobre un modelo determinado. Estos experimentos consisten en ejecutar dicho modelo muchas veces, modificando sistemáticamente los parámetros del modelo y grabando los resultados de cada ejecución.

Es común que los modelos tengan muchos parámetros de los que dependan, cada uno con un dominio de valores diferente. En la mayoría de ellos, sobretodo en sistemas complejos, la modificación de dichos parámetros puede provocar un cambio drástico en el comportamiento del sistema que está siendo modelado.

Es aquí donde aparece esta herramienta que podrá permitirnos conocer a priori los valores de los parámetros para los que el modelo va a seguir un comportamiento interesante. Habrá una gran mayoría de combinaciones posibles en los que el comportamiento del modelo no será interesante.

Tendremos que extremar cuidado con el criterio de parada que se esté utilizando, pues los tiempos de ejecución para todas estas combinaciones puede dispasarse considerablemente.

9.2. Resultados para modelos básicos

9.2.1. Algoritmos genéticos

Realizaremos una serie de comprobaciones y pruebas para ver cómo afectan los parámetros del algoritmo al desempeño del mismo. Comparamos a continuación el **esquema de evolución** generacional respecto del esquema estacionario (ver Figura 9.1).

Si comparamos las gráficas de *fitness* y diversidad para el esquema generacional (Figura 9.1-(a)) respecto a las del esquema estacionario (Figura 9.1-(b)) podemos observar como ésta requiere **menos iteraciones** para converger a la solución óptima. Esto se debe a que el modelo estacionario, a diferencia del generacional, introduce un pequeño número de nuevos indivi-

duos a la población anterior cada iteración.

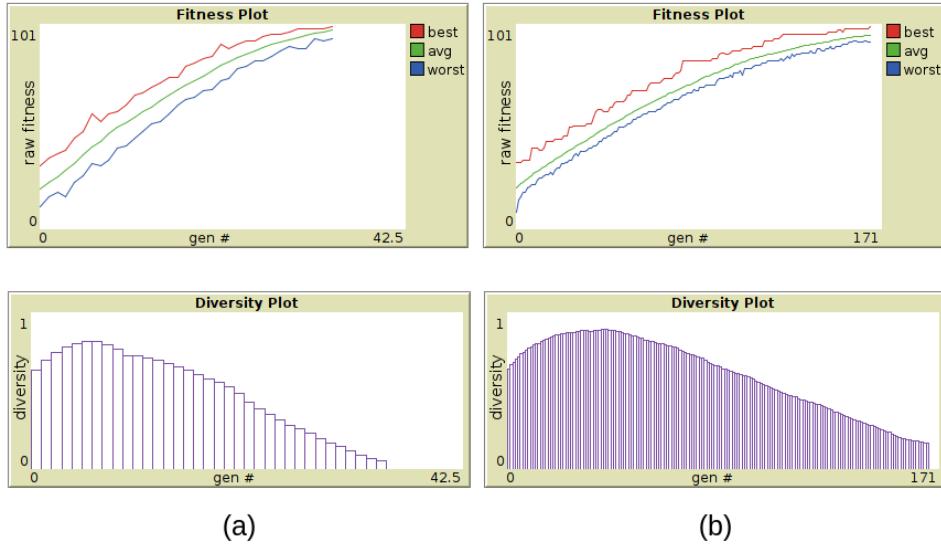


Figura 9.1: Comparativa esquema evolución para modelo GA Básico

En cambio, el esquema generacional reemplaza en cada iteración la generación anterior. La calidad de ésta aumenta, aunque de forma similar al esquema estacionario, con mayor velocidad. Esto se debe a la introducción en la nueva población de más cromosomas con mayor calidad.

Sin embargo, podemos conseguir un rendimiento similar a partir del esquema estacionario si incrementamos el número de cromosomas a reemplazar en la siguiente iteración. Esto hará converger rápidamente a la solución óptima, pero como inconveniente reducirá la diversidad de la población.

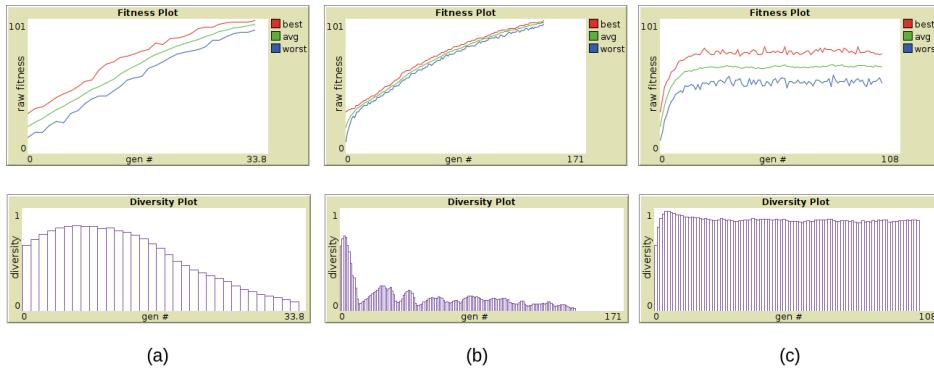


Figura 9.2: Comparativa parámetros para modelo GA Básico

Comparamos a continuación las gráficas obtenidas de la ejecución del modelo básico de algoritmos genéticos con los **parámetros** por defecto y un uso

extremista (con algunos valores al máximo o mínimo) de éstos.

A la vista de la Figura 9.2, podemos observar las gráficas de rendimiento para 3 ejecuciones bastante diferentes.

- (a) En esta ejecución, utilizamos los parámetros **por defecto** de nuestro modelo. En concreto una probabilidad de cruce de un 80 % y una probabilidad de mutación de un 0.02 %. Podemos observar como en la gráfica de fitness va creciendo de forma continua hasta alcanzar el máximo valor de fitness. En cuanto a la diversidad de la población, pese a no tener activado el elitismo obtenemos un descenso de la diversidad conforme se va alcanzando la mejor solución.
- (b) En esta ejecución, utilizamos el **mínimo** valor de **probabilidad de cruce** de 0 %. En este caso es necesario un mayor número de generaciones para poder alcanzar la solución óptima, ya que la única forma de mejorar es mediante la mutación. Cada generación la nueva población se forma por reproducción asexual o clonación, es por ello que apenas hay diversidad durante la ejecución. Los picos de diversidad son introducidos cuando una mutación más fuerte aparece.
- (c) En esta ejecución, utilizamos un valor de **probabilidad de mutación** del 10 %, que es el **máximo** que consideraremos. Es un valor bastante alto, ya que introducirá una gran diversidad en los cromosomas de nuestra población. Si observamos la gráfica de diversidad, vemos que prácticamente no hay similitud entre ningún cromosoma y las soluciones son en su mayoría de baja calidad. Es por ello que independientemente de la probabilidad de cruce, nunca llegaremos a alcanzar la solución óptima.

Será necesario establecer los parámetros de forma acorde, para que haya un cierto **equilibrio** entre **exploración** y **explotación** en la búsqueda de soluciones como podemos ver en la Figura 9.2-(a).

De forma general, si activamos el **elitismo** estamos introduciendo **presión selectiva** ya que el mejor individuo permanecerá en la siguiente generación y tendrá mayor probabilidad de generar descendientes similares a él. De esta forma, podemos llegar a conseguir una población que sea genéticamente similar o más diversa en función de la probabilidad de cruce que utilicemos.

El uso de los distintos **operadores de selección**, provocan diferencias más sutiles que se traducen en diferente número de iteraciones para alcanzar la solución óptima. Por ejemplo, en el uso del operador de selección por ruleta, respecto del operador de selección por torneo, se introduce una mayor diversidad en los cromosomas de la población pero necesita unas pocas más de iteraciones.

El uso de los distintos **operadores de cruce** también difiere en el rendimiento del algoritmo. Por norma general, el cruce uniforme es el que mejor resultado nos ha dado ya que recomienda mucho más los genes de los cromosomas progenitores y facilita la búsqueda de la solución óptima para el problema considerado, que es hallar la cadena que contiene únicamente unos.

9.2.2. Algoritmos de enjambre de partículas

Realizaremos una serie de comprobaciones y pruebas para ver cómo afectan los parámetros del algoritmo al desempeño del mismo. Para que las **comparaciones sean justas**, utilizaremos la opción de optimización de la función sencilla (ver Figura 8.23) y evitaremos el uso del espacio de búsqueda aleatorio ya que no permitirían comparaciones equivalentes.

Podemos apreciar en la Figura 9.3 una comparación del rendimiento del algoritmo en función del **grado de atracción** a la mejor solución **local** de cada partícula y en función de la atracción a la mejor solución **global** encontrada del enjambre.

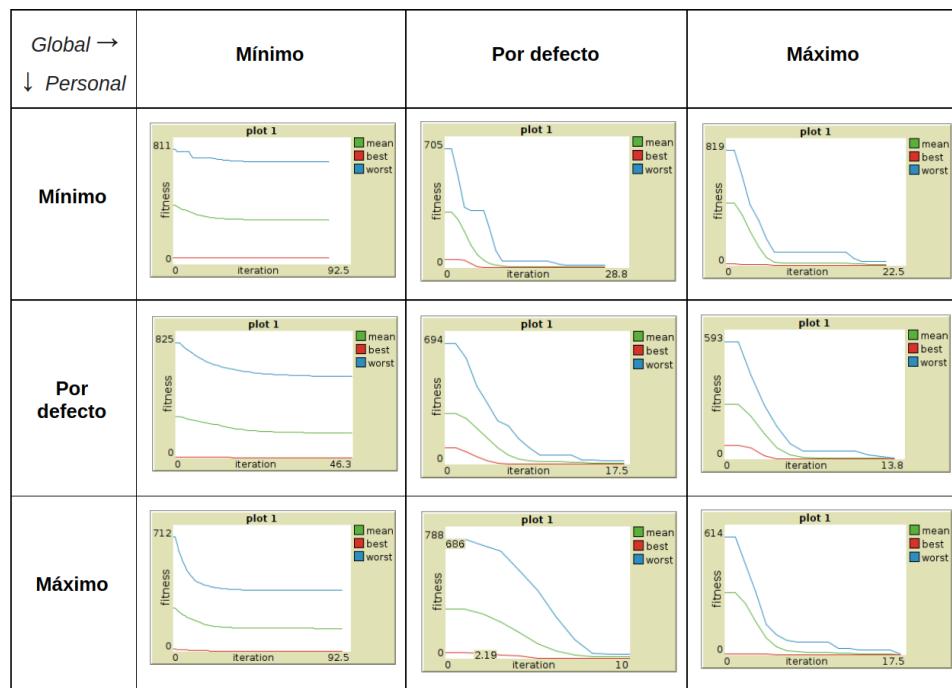


Figura 9.3: Comparativa atracción para modelo PSO Básico

A la vista de los resultados, podemos ver que en **todos los casos** se alcanza la **solución óptima**, aunque no en todos es alcanzada por todas las partículas. Al ser una función de optimización sencilla, será interesante observar cuando **todas** las **partículas** consiguen alcanzarla.

Se puede apreciar que cuando la **atracción al global** tiene un valor **mínimo**, será imposible que todas las partículas alcancen la mejor solución. Apenas tiene un valor por defecto, la convergencia a la solución óptima es instantánea.

La **atracción** a la mejor **posición local** encontrada por una partícula, influye en el número de iteraciones totales que les llevará que todas las partículas puedan alcanzar la solución óptima. Conforme este parámetro **aumenta de valor**, obtendremos una **convergencia más lenta** hacia la mejor solución pues no será tan fuerte la atracción hacia la mejor solución encontrada.

A continuación, compararemos el efecto del parámetro de la **inercia** en el comportamiento de nuestro enjambre de partículas. Podemos ver las gráficas de rendimiento obtenidas en la Figura 9.4 para un valor mínimo (a), para el valor por defecto (b) y el máximo valor (c).

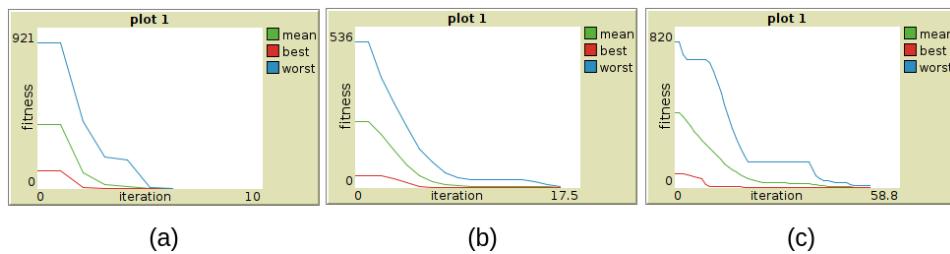


Figura 9.4: Comparativa inercia para modelo PSO Básico

La inercia, como comenté anteriormente en el Capítulo 6, es un parámetro que aportará un extra de movimiento a la partícula (ver Figura 6.6 y que servirá para evitar posibles estancamientos en potenciales óptimos locales).

A medida que aumenta el valor de ésta, se interferirá en mayor grado la información aportada por la mejor solución encontrada y la mejor posición local y por lo tanto, necesitaremos más iteraciones para conseguir que todas las partículas convergan a la posición óptima.

En cuanto al **tamaño del enjambre** o población, para este problema será insignificante, puesto que se resuelve con todos los valores posibles para dicho parámetro. Hay que tener en cuenta que si el número de partículas es muy grande, el tiempo de ejecución y visualización aumentará debido al mayor coste de cómputo.

Por último, el parámetro de **límite de velocidad** de la partícula no sirve más que para limitar la velocidad en la que se acerca la solución hacia su siguiente posición. Aumentar este límite a su máximo valor no interferirá a menos que la diferencia entre la mejor y peor solución sea muy grande. Si disminuimos el valor de ésta, necesitaremos más iteraciones para alcanzar la mejor solución por todas las partículas.

9.2.3. Algoritmos de colonia de hormigas

Realizaremos una serie de comprobaciones y pruebas para ver cómo afectan los parámetros del algoritmo al desempeño del mismo. Los parámetros para la configuración del algoritmo de colonia de hormigas están representados en la configuración del modelo en la Figura 8.13. Éstos son conocidos, de forma general, por ***alpha***, ***beta*** y ***rho***.

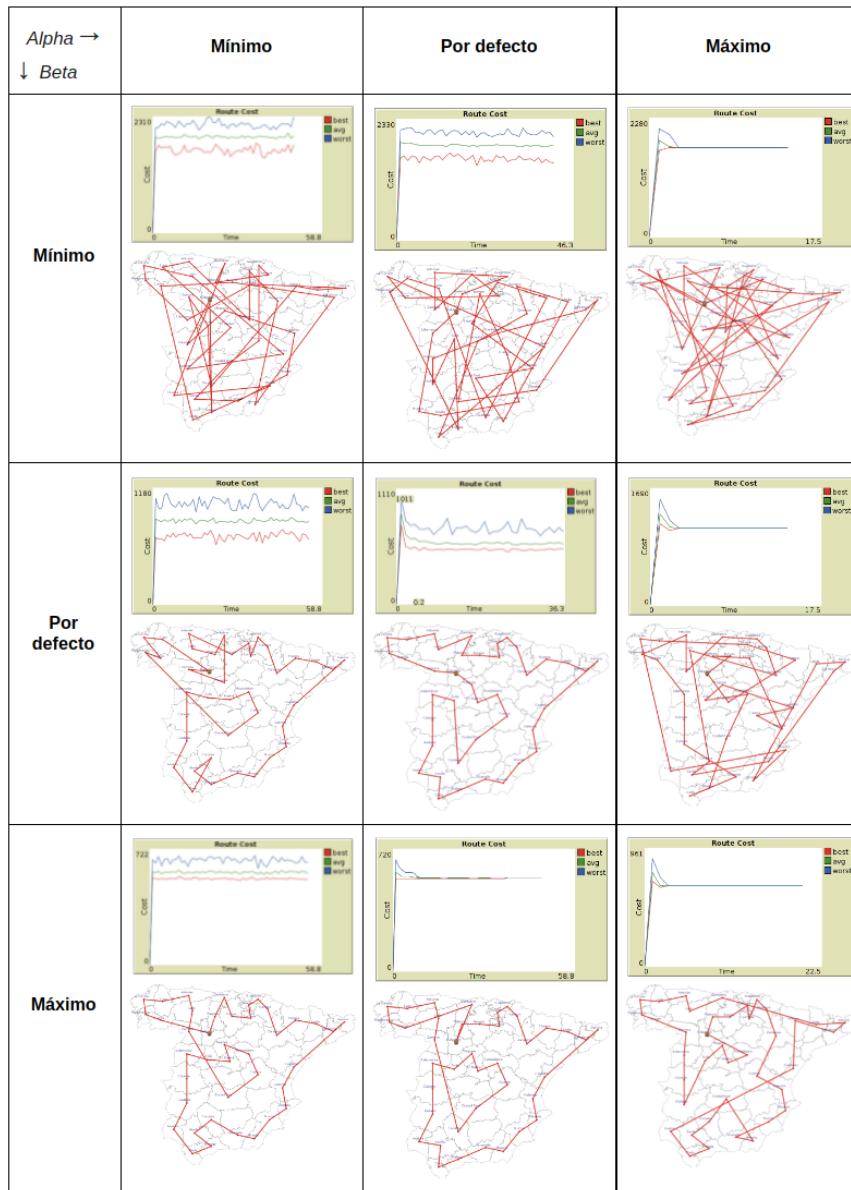


Figura 9.5: Comparativa parámetros para modelo ACO Básico

Los dos primeros, ***alpha*** y ***beta***, servirán para controlar bien: la influencia

de la feromona a la hora de seleccionar una nueva componente para la solución o la heurística de la visibilidad de las hormigas, respectivamente. Se muestra una **tabla comparativa** (ver Figura) para el valor mínimo, valor por defecto y valor máximo de estos dos parámetros, con la solución obtenida y la gráfica de *fitness* asociada.

Podemos observar que para el **mínimo** valor de **beta**, el algoritmo se estanca y no consigue mejorar en la búsqueda de la solución. El factor heurístico de la distancia, será por tanto determinante para encontrar buenas soluciones.

Sin embargo, un **alto valor** de éste no significa que la solución será mejor, ya que el algoritmo pasaría a comportarse como si fuera más bien de tipo *greedy* y obtendríamos una solución que no es óptima, pero que trata de conectar las ciudades que están más cercanas entre sí sin tener en cuenta la longitud total del recorrido.

En cuanto a los **valores** del parámetro **alpha**: Si el valor es **máximo**, podemos ver que independientemente del valor de beta, todas las hormigas convergen a la misma solución. Esto ocurre porque todas tomarían la misma decisión en función de la feromona de los arcos ya que el valor de la heurística de la distancia apenas influiría.

Lo ideal será por tanto, como podemos ver en la casilla central de la Figura 9.5, valores de alpha y beta por defecto, se **ajusten** de forma que tanto la **información aportada** por la visibilidad como la información que proporcionan el resto de hormigas a través de la feromona influyan de forma **equitativa** para aprovechar ambas.

Esto es algo que se suele **determinar** de forma **experimental** ejecutando combinaciones de los parámetros para intentar adecuarlos lo mejor posible.

El tercer parámetro, **rho**, servirá para controlar la **tasa de evaporación** de la feromona. A medida que el valor de dicho parámetro aumenta, las aristas del grafo que han sido visitadas por menos hormigas, que tienen un valor de feromona más pequeño, tenderán a desaparecer ya que se evaporarán rápidamente.

9.3. Resultados para modelos específicos

A continuación se detallarán los resultados obtenidos para cada una de las variables de diseño así como la función objetivo en los problemas considerados por los modelos específicos.

Como **comprobación de reproducibilidad**, se comprobará que los **resultados** obtenidos por otros **autores** en las variables de diseño, al ser asignados en nuestro modelo, producen el **mismo valor** de *fitness* como

solución. De esta forma nos aseguramos de que la implementación de la función objetivo y cálculo de restricciones son correctas.

Recordar que siempre tenemos un **criterio de parada** en estos modelos. Normalmente es el número de evaluaciones o generaciones/iteraciones del algoritmo.

Dependiendo de los parámetros seleccionados podremos alcanzar una solución mejor o peor. Si éstos están configurados con sentido, obtendremos un mejor rendimiento en la búsqueda y por tanto una mejor solución.

9.3.1. Resultados para Pressure Vessel

Los resultados obtenidos para el diseño del recipiente a presión usando algoritmos genéticos a través del modelo específico implementado se detallan en la Tabla 9.1.

Una comparativa mucho más exhaustiva para la resolución de este problema con resultados de más de 40 autores la podemos ver en el Artículo [15].

	FA ¹ [16]	SA-DS ² [23]	PSO ³ [21]	GA ⁴ [6]	Trabajo Actual
Variables de diseño					
$x_1(R)$	38.860	39.809	42.098	40.097	38.826
$x_2(L)$	221.365	207.225	176.636	176.654	221.881
$x_3(T_s)$	0.75	0.7683	0.8125	0.8125	0.75
$x_4(T_h)$	0.375	0.3797	0.4375	0.4375	0.375
Restricciones					
g_1	-0.000	-0.000	0	-0.00002	-0.00065
g_2	-0.004	-0.000	-0.035	-0.035	-0.005
g_3	-0.013	-10.706	0	-27.886	-0.000
g_4	-18.634	-32.774	-63.363	-63.345	-18.119
Función Objetivo					
$f(x)$	5850.383	5868.764	6059.714	6059.946	5855.084

Cuadro 9.1: Tabla comparativa para *Pressure Vessel Design* usando GA

¹ FA: Firefly Algorithm

² SA-DA: Direct Search

³ PSO: Particle Swarm Optimization

⁴ GA: Genetic Algorithm

Podemos observar que a pesar de no superar el mejor resultado de la literatura, si nos acercamos bastante a él mejorando otras muchas aproximaciones de otros autores.

9.3.2. Resultados para Tension/Compression Spring

Los resultados obtenidos para el diseño del resorte usando algoritmos de enjambre de partículas a través del modelo específico implementado se detallan en la Tabla 9.2

En este caso, las diferencias entre los resultados obtenidos por los distintos autores son más sutiles. Conseguimos acercarnos a la mejor solución encontrada con una precisión de 5 decimales, siendo la nuestra un poco inferior.

	CPSO ¹ [20]	ES ² [8]	GA [7]	CC ³ [3]	Trabajo Actual
Variables de diseño					
$x_1(d)$	0.0517	0.0519	0.0514	0.0533	0.0525
$x_2(D)$	0.3576	0.3639	0.3516	0.3991	0.3773
$x_3(N)$	11.2445	10.8905	11.6322	9.1854	10.1778
Restricciones					
g_1	-0.00084	-0.00001	-0.00208	-0.00001	-0.000021
g_2	-0.00001	-0.00002	-0.00011	-0.00001	-0.000017
g_3	-4.05130	-4.06133	-4.02631	-4.12383	-4.092399
g_4	-0.72709	-0.72269	-4.02631	-0.69828	-0.71345
Función Objetivo					
$f(x)$	0.0126747	0.0126810	0.0127048	0.0127303	0.0126786

Cuadro 9.2: Tabla comparativa para *Tension/Compression Spring Design* usando PSO

¹ CPSO: Co-evolutionary PSO

² ES: Evolution Strategy

³ CC: Constraint Correction

9.3.3. Resultados para 25-Bar Space Truss

Los resultados obtenidos para la elección de las distintas áreas de secciones de corte de nuestra estructura usando algoritmos de colonia de hormigas a través del modelo específico implementado se detallan en la Tabla 9.3

El resultado obtenido en este caso, no se acerca tanto al óptimo como conseguimos en los otros problemas. No obstante la solución consigue mejorar alguna aproximación propuesta en la literatura como las de [31] y [26].

Grupo	Miembros	GA [31]	TA ¹ [26]	GA [5]	ACO ² [4]	Trabajo Actual
Variables de diseño						
1	1	0.10	0.10	0.10	0.10	0.10
2	2,3,4,5	1.80	1.90	0.50	0.30	0.30
3	6,7,8,9	2.30	2.60	3.40	3.40	3.40
4	10,11	0.20	0.10	0.10	0.10	0.10
5	12,13	0.10	0.10	1.90	2.10	1.90
6	14,15,16,17	0.80	0.80	0.90	1.00	1.00
7	18,19,20,21	1.80	2.10	0.50	0.50	0.70
8	22,23,24,25	3.00	2.60	3.40	3.40	3.30
Función objetivo						
$Weight(lb)$		546.01	562.93	485.05	484.85	491.007

Cuadro 9.3: Tabla comparativa para *25-Bar Space Truss Design* usando ACO

¹ TA: Templeman's Algorithm

² ACO: Ant Colony Optimization

CAPÍTULO

10

CONCLUSIONES Y TRABAJO FUTURO

10.1. Conclusiones

En este trabajo se presentan varios modelos de algoritmos bioinspirados implementados en NetLogo, una herramienta para el modelado basado en agentes. Se han desarrollado, conforme a los objetivos del proyecto, seis modelos diferentes, dos por cada tipo de algoritmo bionspirado en estudio: *algoritmos genéticos, colonias de hormigas y enjambres de partículas*.

El público objetivo de este proyecto es, en primera instancia, estudiantes de la asignatura *Optimización y Computación Inteligente (OCI)* del Máster en Estructuras, de la Universidad de Granada. No obstante podrían resultar interesantes estos y otros modelos a estudiantes de ámbitos tales como la biología, química, farmacia, ciencias sociales, etc.

Para cada cada tipo de algoritmo se plantea un **problema básico** que permita su entendimiento mucho más enfocado en la visualización del propio algoritmo y un **problema avanzado** dentro del ámbito de la ingeniería, cumpliendo con los objetivos OBJ-1 y OBJ-2 respectivamente

Además, se ha proporcionado una **visualización en 2D** de la evolución de cromosomas en la resolución de un problema sencillo para los algoritmos genéticos, así como de hormigas en la búsqueda de una ruta minimal.

También una **visualización en 3D** de un espacio de búsqueda para el caso

básico de los algoritmos de enjambres de partículas, así como en la optimización de una estructura para el caso específico de algoritmos de colonias de hormigas. Es por ello que también se cumple con el tercer objetivo (OBJ-3).

Además se hace una **comparativa** de los **resultados** obtenidos en cada uno de los problemas específicos con los aquellos obtenidos por los **autores** que lo proponen, así como con los mejores resultados existentes en la literatura para dicho problema. Cumpliendo así con el cuarto objetivo (OBJ-4).

Por lo tanto, se han **cumplido** con todos y cada uno de los **objetivos inicialmente propuestos**.

Para el desarrollo del proyecto se han utilizado conceptos e información adquirida en asignaturas del *Plan de Estudios* del GII¹ como son *Fundamentos y Metodología de la Programación* para la implementación de los modelos, *Cálculo, Álgebra Lineal y Estructuras Matemáticas* para el entendimiento y desarrollo de los problemas específicos, *Fundamentos de Ingeniería del Software* para describir toda la parte asociada a la planificación y metodología de trabajo, de forma esencial las asignaturas de *Metaheurísticas* en las que se adquirió un conocimiento previo a los algoritmos bioinspirados, así como *Informática Gráfica* para la representación tanto bidimensional como tridimensional de los modelos implementados. También han sido muy útiles conocimientos de concurrencia adquiridos en *Sistemas Concurrentes y Distribuidos* para entender el mecanismo de funcionamiento de nuestra herramienta de modelado de sistema de agentes.

Como reflexión final, no hay que olvidar la **importancia del docente** a la hora de seleccionar y crear la necesidad de uso de estas herramientas sobre el alumnado. La inclusión de herramientas digitales ‘per se’ no implica una mejora en la calidad docente, es conveniente aplicar un **enfoque correcto** al uso de las mismas. Es decir, el hecho de usarlas no tiene por qué suponer una mejora para la calidad de la docencia. Será eficaz didácticamente si contribuye a mejorar la enseñanza y el aprendizaje del alumnado. Para ello deben estar bien enfocadas y alineadas con el temario que esté tratando en clase. Como objetivo final se espera que su uso mejore los resultados académicos [13].

También se han enviado los modelos desarrollados a la página oficial de NetLogo y se ha recibido un correo con la aprobación para ser publicados en la página de **Modelos de la Comunidad** de usuarios de NetLogo.

Adicionalmente se ha elaborado un póster que sintetiza el presente proyecto, que ha sido presentado al **Congreso Universitario de Innovación Educativa en las Enseñanzas Técnicas (CUIET)** de este año 2021 (en espera de aceptación).

¹Grado en Ingeniería Informática

10.2. Trabajo Futuro

Como trabajo futuro se prevé el uso de *HubNet*² (una extensión de Netlogo) para el desarrollo de experimentos que permitan ejecutar simulaciones de modelos más complejos con **participación directa** de los estudiantes en clase de forma **simultánea**.

En una simulación participativa, toda la clase participa en la representación del comportamiento de un sistema, ya que cada estudiante controla una parte del sistema mediante el uso de un dispositivo individual como es un ordenador conectado a la red.

De esta forma, la respuesta del sistema simulado dependerá de las **decisiones tomadas en tiempo real** por los propios alumnos. Un ejemplo podría ser el de *Gridlock*: cada estudiante controla un semáforo de forma individual en una ciudad simulada. La clase en su conjunto trata de hacer que el tráfico fluya de manera eficiente por la ciudad.

También se tiene pensado acceder al *Máster en Profesorado de Enseñanza Secundaria Obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomas*³ (MAES) de la Universidad de Granada. No se descarta un posible futuro trabajo final de máster relacionado con ésta u otra herramienta como propuesta para la mejora de la calidad docente en el ámbito de la educación en STEM.

²<https://ccl.northwestern.edu/netlogo/2.0/docs/hubnet.html>

³<https://masteres.ugr.es/profesorado/>

BIBLIOGRAFIA

- [1] Hojjat Adeli y Osama Kamal. “Efficient optimization of space trusses”. En: *Computers & structures* 24.3 (1986), págs. 501-511.
- [2] Robert John Allan y col. *Survey of agent based modelling and simulation tools*. Science & Technology Facilities Council New York, 2010.
- [3] Jasbir Singh Arora. *Introduction to optimum design*. Elsevier, 2004.
- [4] Charles V Camp y Barron J Bichon. “Design of space trusses using ant colony optimization”. En: *Journal of structural engineering* 130.5 (2004), págs. 741-751.
- [5] Guozhong Cao. *Optimized design of framed structures using a genetic algorithm*. The University of Memphis, 1996.
- [6] CA Coello Coello y E Mezura Montes. “Use of dominance-based tournament selection to handle constraints in genetic algorithms”. En: *Intelligent Engineering Systems through Artificial Neural Networks* 11.1 (2001), págs. 177-182.
- [7] Carlos A Coello Coello. “Use of a self-adaptive penalty approach for engineering optimization problems”. En: *Computers in Industry* 41.2 (2000), págs. 113-127.
- [8] Carlos A Coello Coello y Efrén Mezura Montes. “Constraint-handling in genetic algorithms through the use of dominance-based tournament selection”. En: *Advanced Engineering Informatics* 16.3 (2002), págs. 193-203.

- [9] Charles Darwin y William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt New York, 2009.
- [10] Marco Dorigo y Luca Maria Gambardella. “Ant colonies for the travelling salesman problem”. En: *Biosystems* 43.2 (1997), págs. 73-81. ISSN: 0303-2647. DOI: [https://doi.org/10.1016/S0303-2647\(97\)01708-5](https://doi.org/10.1016/S0303-2647(97)01708-5).
- [11] Marco Dorigo, Vittorio Maniezzo y Alberto Colorni. “Ant system: optimization by a colony of cooperating agents”. En: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), págs. 29-41.
- [12] Francisco Esquembre. “Easy Java Simulations: A software tool to create scientific simulations in Java”. En: *Computer physics communications* 156.2 (2004), págs. 199-204.
- [13] Ana María Fernández-Pampillón Cesteros, Elena Domínguez Romero e Isabel de Armas Ranero. “Diez criterios para mejorar la calidad de los materiales didácticos digitales”. En: (2012).
- [15] Amir Hossein Gandomi, Xin-She Yang y Amir Hossein Alavi. “Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems”. En: *Engineering with computers* 29.1 (2013), págs. 17-35.
- [16] Amir Hossein Gandomi, Xin-She Yang y Amir Hossein Alavi. “Mixed variable structural optimization using firefly algorithm”. En: *Computers & Structures* 89.23-24 (2011), págs. 2325-2336.
- [17] Michael R Garey y David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979.
- [18] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.
- [19] Oğuzhan Hasانçebi y Serdar Çarbaş. “Ant colony search method in practical structural optimization”. En: 1.1 (2011), págs. 91-105.
- [20] Qie He y Ling Wang. “An effective co-evolutionary particle swarm optimization for constrained engineering design problems”. En: *Engineering applications of artificial intelligence* 20.1 (2007), págs. 89-99.
- [21] S He, E Prempain y QH Wu. “An improved particle swarm optimizer for mechanical design optimization problems”. En: *Engineering optimization* 36.5 (2004), págs. 585-605.
- [22] John Henry Holland y col. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

- [23] Shun-Fa Hwang y Rong-Song He. “A hybrid real-parameter genetic algorithm for function optimization”. En: *Advanced Engineering Informatics* 20.1 (2006), págs. 7-21.
- [24] Mercè Izquierdo y col. “Caracterización y fundamentación de la ciencia escolar”. En: *Enseñanza de las Ciencias* 17.1 (1999), págs. 45-59.
- [26] Duan Ming-Zhu. “An improved Templeman’s algorithm for the optimum design of trusses with discrete member sizes”. En: *Engineering Optimization* 9.4 (1986), págs. 303-312.
- [28] Emilio Carlos Nelly Silva y Luis Iván Negrín Hernández. “Optimización en ingeniería”. En: (2018).
- [29] “QUE ES NETLOGO”. En: (). (*Accedido 10-Jun-2021*). URL: <https://ccl.northwestern.edu/netlogo/resources/Que%5C%20es%5C%20NetLogo.pdf>.
- [30] Steven F. Railsback, Steven L. Lytinen y Stephen K. Jackson. “Agent-based Simulation Platforms: Review and Development Recommendations”. En: *SIMULATION* 82.9 (2006), págs. 609-623. DOI: 10.1177/0037549706073695.
- [31] Srijith Rajeev y CS Krishnamoorthy. “Discrete optimization of structures using genetic algorithms”. En: *Journal of structural engineering* 118.5 (1992), págs. 1233-1250.
- [32] Stuart Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3.^a ed. Prentice Hall, 2010.
- [33] E Sandgren. “Nonlinear integer and discrete programming in mechanical design optimization”. En: (1990).
- [34] Víctor López Simó, Digna Couso Lagarón y Cristina Simarro Rodríguez. “Educación STEM en y para el mundo digital. Cómo y por qué llevar las herramientas digitales a las aulas deficiencias, matemáticas y tecnologías”. En: () .
- [35] Hersh Salganik L. Simona Rychen D. “Competencias clave para el bienestar personal social y económico”. En: (2006).

BIBLIOGRAFÍA

- [1] Hojjat Adeli y Osama Kamal. “Efficient optimization of space trusses”. En: *Computers & structures* 24.3 (1986), págs. 501-511.
- [2] Robert John Allan y col. *Survey of agent based modelling and simulation tools*. Science & Technology Facilities Council New York, 2010.
- [3] Jasbir Singh Arora. *Introduction to optimum design*. Elsevier, 2004.
- [4] Charles V Camp y Barron J Bichon. “Design of space trusses using ant colony optimization”. En: *Journal of structural engineering* 130.5 (2004), págs. 741-751.
- [5] Guozhong Cao. *Optimized design of framed structures using a genetic algorithm*. The University of Memphis, 1996.
- [6] CA Coello Coello y E Mezura Montes. “Use of dominance-based tournament selection to handle constraints in genetic algorithms”. En: *Intelligent Engineering Systems through Artificial Neural Networks* 11.1 (2001), págs. 177-182.
- [7] Carlos A Coello Coello. “Use of a self-adaptive penalty approach for engineering optimization problems”. En: *Computers in Industry* 41.2 (2000), págs. 113-127.
- [8] Carlos A Coello Coello y Efrén Mezura Montes. “Constraint-handling in genetic algorithms through the use of dominance-based tournament selection”. En: *Advanced Engineering Informatics* 16.3 (2002), págs. 193-203.

- [9] Charles Darwin y William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt New York, 2009.
- [10] Marco Dorigo y Luca Maria Gambardella. “Ant colonies for the travelling salesman problem”. En: *Biosystems* 43.2 (1997), págs. 73-81. ISSN: 0303-2647. DOI: [https://doi.org/10.1016/S0303-2647\(97\)01708-5](https://doi.org/10.1016/S0303-2647(97)01708-5).
- [11] Marco Dorigo, Vittorio Maniezzo y Alberto Colorni. “Ant system: optimization by a colony of cooperating agents”. En: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26.1 (1996), págs. 29-41.
- [12] Francisco Esquembre. “Easy Java Simulations: A software tool to create scientific simulations in Java”. En: *Computer physics communications* 156.2 (2004), págs. 199-204.
- [13] Ana María Fernández-Pampillón Cesteros, Elena Domínguez Romero e Isabel de Armas Ranero. “Diez criterios para mejorar la calidad de los materiales didácticos digitales”. En: (2012).
- [14] Amir Hossein Gandomi y Xin-She Yang. “Benchmark problems in structural optimization”. En: *Computational optimization, methods and algorithms*. Springer, 2011, págs. 259-281.
- [15] Amir Hossein Gandomi, Xin-She Yang y Amir Hossein Alavi. “Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems”. En: *Engineering with computers* 29.1 (2013), págs. 17-35.
- [16] Amir Hossein Gandomi, Xin-She Yang y Amir Hossein Alavi. “Mixed variable structural optimization using firefly algorithm”. En: *Computers & Structures* 89.23-24 (2011), págs. 2325-2336.
- [17] Michael R Garey y David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979.
- [18] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.
- [19] Oğuzhan Hasانçebi y Serdar Çarbaş. “Ant colony search method in practical structural optimization”. En: 1.1 (2011), págs. 91-105.
- [20] Qie He y Ling Wang. “An effective co-evolutionary particle swarm optimization for constrained engineering design problems”. En: *Engineering applications of artificial intelligence* 20.1 (2007), págs. 89-99.
- [21] S He, E Prempain y QH Wu. “An improved particle swarm optimizer for mechanical design optimization problems”. En: *Engineering optimization* 36.5 (2004), págs. 585-605.

- [22] John Henry Holland y col. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [23] Shun-Fa Hwang y Rong-Song He. “A hybrid real-parameter genetic algorithm for function optimization”. En: *Advanced Engineering Informatics* 20.1 (2006), págs. 7-21.
- [24] Mercè Izquierdo y col. “Caracterización y fundamentación de la ciencia escolar”. En: *Enseñanza de las Ciencias* 17.1 (1999), págs. 45-59.
- [25] J. Kennedy y R. Eberhart. “Particle swarm optimization”. En: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942-1948 vol.4. DOI: 10.1109/ICNN.1995.488968.
- [26] Duan Ming-Zhu. “An improved Templeman’s algorithm for the optimum design of trusses with discrete member sizes”. En: *Engineering Optimization* 9.4 (1986), págs. 303-312.
- [27] Norma Moroni y Perla Señas. “La Visualización de Algoritmos como Recurso para la Enseñanza de la Programación”. En: *IV Workshop de Investigadores en Ciencias de la Computación*. 2002.
- [28] Emilio Carlos Nelly Silva y Luis Iván Negrín Hernández. “Optimización en ingeniería”. En: (2018).
- [29] “QUE ES NETLOGO”. En: (). (Accedido 10-Jun-2021). URL: <https://ccl.northwestern.edu/netlogo/resources/Que%5C%20es%5C%20NetLogo.pdf>.
- [30] Steven F. Railsback, Steven L. Lytinen y Stephen K. Jackson. “Agent-based Simulation Platforms: Review and Development Recommendations”. En: *SIMULATION* 82.9 (2006), págs. 609-623. DOI: 10.1177/0037549706073695.
- [31] Srijith Rajeev y CS Krishnamoorthy. “Discrete optimization of structures using genetic algorithms”. En: *Journal of structural engineering* 118.5 (1992), págs. 1233-1250.
- [32] Stuart Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3.^a ed. Prentice Hall, 2010.
- [33] E Sandgren. “Nonlinear integer and discrete programming in mechanical design optimization”. En: (1990).
- [34] Víctor López Simó, Digna Couso Lagarón y Cristina Simarro Rodríguez. “Educación STEM en y para el mundo digital. Cómo y por qué llevar las herramientas digitales a las aulas deficiencias, matemáticas y tecnologías”. En: () .
- [35] Hersh Salganik L. Simona Rychen D. “Competencias clave para el bienestar personal social y económico”. En: (2006).

APÉNDICE

A

MANUAL DE USUARIO

A.1. Introducción a Netlogo

A.1.1. Instalación

(hacer un manual de usuario y explicar como se instala) Aquí que es, historia, evolución.. Lo categorizamos en qué es...

A continuación, vamos a explicar los conceptos básicos para introducirse en la programación de Netlogo, así como para familiarizarnos con el uso de su interfaz.

Lo primero será decidir la forma en la que ejecutar los modelos, puede hacerse de dos formas distintas:

- Descargando el programa desde la página oficial de Netlogo e instalándolo en nuestro ordenador (necesitaremos una máquina virtual de Java). Podemos descargar la última versión (6.2) desde el siguiente enlace: <https://ccl.northwestern.edu/netlogo/6.2.0/>
- La otra opción es a través de Netlogo Web. Es un servicio que permite ejecutar a través Javascript el código de los modelos en lugar de usar la JVM de la aplicación de escritorio. De esta forma perdemos funcionalidad ya que no podemos editar y crear nuestros propios modelos, únicamente podemos ejecutarlos

Se recomienda la versión de escritorio para poder obtener todas las prestaciones que Netlogo nos ofrece. Además será la opción que nosotros utilizaremos

y la que se explicará a continuación.

Una vez lo hemos descargado y descomprimido, podremos ver que hay dos ficheros ejecutables: Netlogo y Netlogo3D. EL primero de ellos permitirá la simulación de ficheros con extensión .nlogo y el segundo ficheros con extensión n.logo3d.

La única diferencia entre las interfaces de ambos programas será que en la versión 3D tendremos otra ventana adicional en la que visualizar el modelo o *mundo* a simular.

A.1.2. Entorno de trabajo

En esta subsección daremos una base o visión general de la herramienta de trabajo Netlogo. Explicaremos un poco las posibilidades que nos ofrece a la hora de crear una interfaz para un modelo.

Comenzaremos explicando las distintas partes que componen la interfaz general de la aplicación. Una vez la hemos ejecutado, nos aparecerá una pestaña similar a la de la figura A.1.

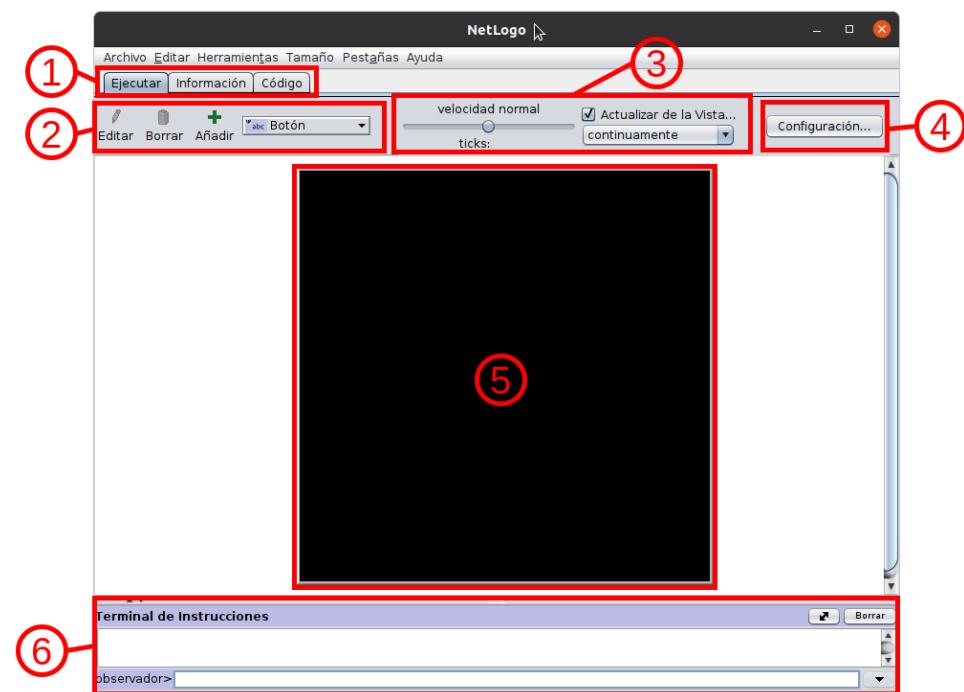


Figura A.1: Interfaz de Netlogo

En ella, hemos agrupado y enumerado del 1 al 6 las partes más relevantes y que definiremos a continuación:

1. **Pestañas principales.** A través de ellas se visualizará, documentará

e implementará el modelo como puede verse en la figura A.2. Consta de 3 pestañas principales:

- **Interfaz o Interface:** En esta pestaña podremos visualizar el mundo y la simulación de nuestro modelo. En ella se definen los botones, seleccionadores, deslizadores y gráficas que permiten al usuario interactuar con el modelo.
- **Información o Info:** En esta pestaña podremos añadir información relevante a nuestro modelo, explicar el funcionamiento de éste, cómo utilizar los botones de la interfaz adecuadamente, destacar alguna parte del código relevante, etc.
- **Código o Code:** En esta pestaña se escribe la implementación del modelo. Todos los procedimientos asociados a éste van en el mismo fichero.

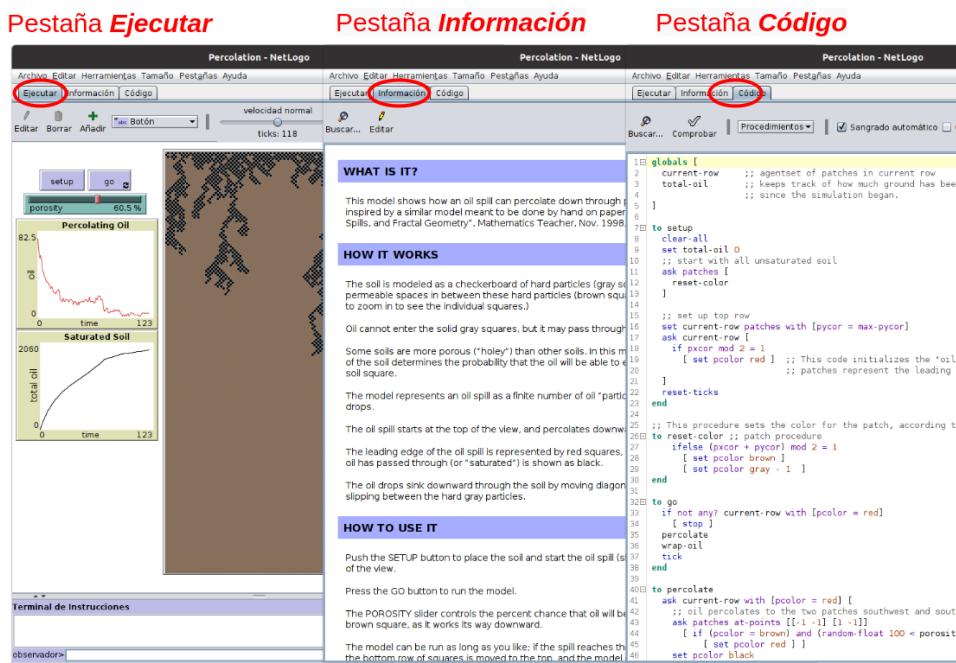


Figura A.2: Pestañas principales en Netlogo

2. **Elementos de interacción con el usuario:** Nos permitirán el control y la modificación de la ejecución. También servirán para establecer valores de los parámetros de nuestro modelo así como alterarlos de forma dinámica.

Tenemos la opción de añadir, editar y eliminar los elementos de nuestra interfaz del modelo. Los **distintos tipos de elementos** que podemos incluir en nuestra **interfaz** son los que aparecen en la figura A.3

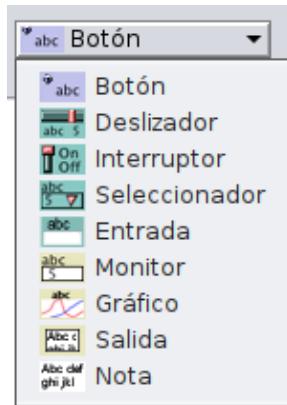


Figura A.3: Tipos de botones para la Interfaz

Disponemos de una gran variedad de elementos como botones, deslizadores, interruptores para establecer y cambiar valores de posibles parámetros del modelo. También de monitores y gráficos de salida que permitirán visualizar de forma gráfica los valores que consideremos oportunos en nuestro modelo.

3. **Control de la simulación.** El deslizador central nos permitirá regular la velocidad de *ticks* con la que queremos que nuestro modelo se ejecute.

Si seleccionamos la casilla *Actualizar de la vista* podremos ver los cambios en el modelo paso a paso. Si está desactivada solamente veremos el mundo en su estado final tras finalizar la ejecución.

4. **Configuración del mundo.** Cuando pulsamos sobre dicha opción, podremos establecer las dimensiones y topología del mundo sobre que el que actuarán nuestros agentes. Ver figura A.4.

A priori es un tablero de dos dimensiones, pero podemos hacer que no tenga fronteras o límites de forma horizontal (tendríamos un cilindro) y que tampoco tenga fronteras verticalmente (obtendríamos una figura matemática conocida como *toro*).

Las distintas formas que puede adoptar el mundo las podemos ver de forma más clara en la figura A.5.

5. **Mundo:** En esta ventana veremos la simulación de nuestro modelo, así como el comportamiento de nuestros agentes con el propio mundo.
6. **Terminal de instrucciones:** Permitirá ejecutar instrucciones desde el punto de vista del agente *observador*. Además, en ella visualizaremos cualquier mensaje de salida del modelo.



Figura A.4: Ajustes del mundo de Netlogo

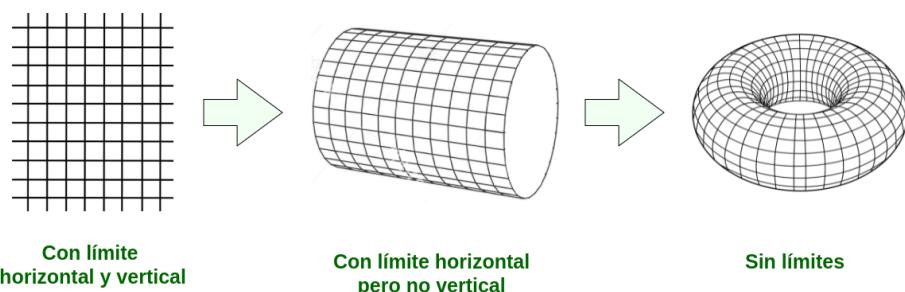


Figura A.5: Topología del mundo de Netlogo

Adicionalmente, como cualquier otro programa, tenemos una barra de opciones con la típica información para Abrir, Guardar o Crear un modelo nuevo.

También tenemos una gran cantidad de *Herramientas* para visualizar cada uno de nuestros agentes con sus atributos y de personalizar nuestros propios agentes. Esta función viene ya integrada en el propio NetLogo (Ver figura A.6).

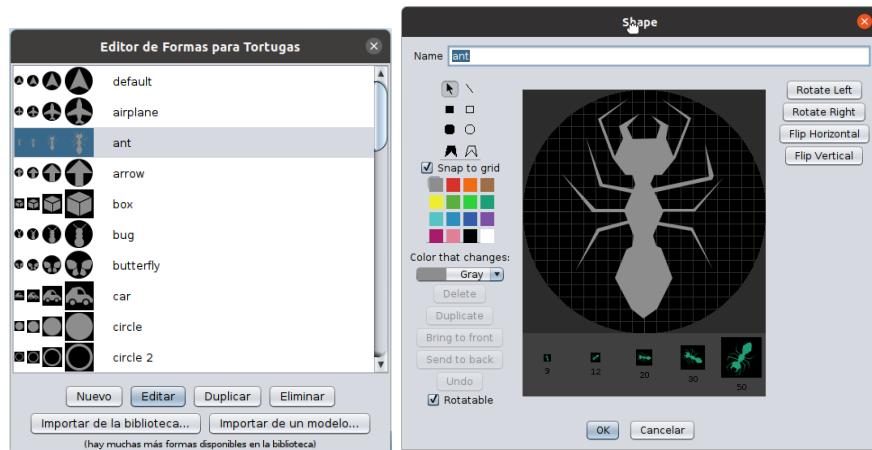


Figura A.6: Editor de agentes de Netlogo

También disponemos de una paleta de colores para obtener el código RGB o hexadecimal que más nos guste para visualizar de una forma más personalizada nuestros modelos (Ver figura A.7).



Figura A.7: Paleta de colores de Netlogo

A.1.3. Programando en Netlogo

Netlogo, al ser una herramienta de modelado de agentes, está compuesta por diferentes tipos de agentes que siguen una serie de instrucciones las cuales les permiten interaccionar entre sí y con el entorno, aprender y adaptar su conocimiento en base a la experiencia con éstos.

Son 4 los tipos de agentes de Netlogo: tortugas, parcelas, enlaces y observador.

- Las **parcelas** o *patches* formarán el terreno donde se desplazarán y comunicarán las tortugas. Son estáticas y se distribuyen entorno a las coordenadas del mundo, por lo tanto hay un número determinado y fijo de éstas. Su representación visual es un cuadrado coloreado y son creados automáticamente al inicio.
- Las **tortugas** o *turtles* serán el agente principal que utilicemos para interactuar con las parcelas y con otras tortugas. Podemos definir diferentes tipos de tortugas, extendiendo su funcionalidad a modo de herencia. Podemos elegir y crear su representación visual a nuestro gusto.
- Los **enlaces** o *links* permitirán representar relaciones entre tortugas. Su posición vendrá determinada por el par de agentes (tortugas) que relaciona. Se representa visualmente por una línea.
- El **observador** o *observer* es un agente exterior al mundo, por lo tanto no tiene representación visual. Permitirá dar instrucciones al resto de agentes, será el encargado de crear y destruir a las tortugas.

Netlogo no es un lenguaje orientado a objetos, es un **lenguaje procedural**. La programación se enfocará a la creación de procedimientos que puedan ser ejecutados por los distintos agentes.

Las palabras reservadas para la creación de un procedimiento son:

```
to <nombre_procedimiento> [parametros] :  
...  
end
```

Tendremos que tener presente qué contexto (qué agente) ejecutará cada procedimiento, pues no todos los agentes pueden ejecutar todas las instrucciones posibles. Cada tipo de agente tendrá un conjunto de atributos y métodos que lo define.

Si queremos que la función devuelva algún valor, se declarará con **to-report** y al final del método debe retornarse un valor con **report**.

Las variables locales y globales se declaran de la forma:

```
let <nombre-variable> [valor]
```

Si una variable no ha sido inicializada, tendrá un valor de 0 por defecto. Tanto variables como argumentos de entrada a los procedimientos se verifican de forma léxica (no dinámica), antes de la ejecución.

Podemos hacer uso de las típicas estructuras de control utilizando las palabras reservadas: **if**, **while**, **foreach** y utilizar tipos de datos ya predefinidos como: *list* (lista) y *word* (string).

En cuanto a las tortugas, disponemos de una gran cantidad de instrucciones de alto nivel para interactuar con ellas:

- Podemos **moverlas por el mundo** con **forward** (fd), **right** (rt), **left** (lt), **back** (bk), **move-to-patch** xcor ycor, etc.
- Seleccionar a un **subconjunto** de ellas que **cumplan** una determinada **condición** con la instrucción: **turtles with [condición]**.
- Pedirles que **ejecuten** algún procedimiento: **ask turtles [instrucciones]**.

Y una larga lista más de primitivas y atributos que nos facilitarán información sobre la posición, orientación, tamaño, color y otras características del agente.

A.2. Apertura y ejecución de modelos

Bastará con abrir la aplicación correspondiente al modelo que tengamos pensado crear/abrir. Si hemos construido un modelo en 3D (extensión .nlogo3d), abriremos **NetLogo3D**. En caso de tratarse de un modelo en 2D (extensión .nlogo), abriremos la aplicación **NetLogo**.

A continuación pulsamos sobre **Archivo > Abrir** y seleccionar el modelo deseado entre nuestro árbol de directorios.

Para ejecutar el modelo, pulsaremos sobre el botón **setup** para inicializarlo y **go** para comenzar la simulación.