Universidad Camilo José Cela

COMPARACIÓN DE TÉCNICAS DE *REINFORCEMENT LEARNING* Y DE *EVOLUTIVE COMPUTING* EN CONTROL DE ENJAMBRES DE DRONES

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

2023 - 2024

Nombre y Apellidos del alumno

JAIME PARADA LÓPEZ

Nombre y Apellidos del tutor

FRANCISCO LAMAS LÓPEZ

Resumen

Las técnicas de *Reinforcement Learning* y de *Evolutive Computing* han demostrado ser herramientas poderosas en la resolución de una amplia gama de problemas en campos como la robótica, la logística y la planificación de rutas. En particular, se han utilizado con éxito en la optimización de trayectorias para vehículos aéreos no tripulados (UAVs). Sin embargo, a pesar de su potencial, su aplicación en el control de enjambres de drones ha sido notablemente escasa.

El uso de enjambres de drones implica la agilización del tiempo de vuelo y la reducción de los costos operativos, en donde, además de optimizar las trayectorias individuales de cada dron, es crucial garantizar la cohesión del enjambre, evitar colisiones y adaptarse a cambios en las condiciones ambientales y en las demandas de la misión. Estos desafíos requieren enfoques innovadores que puedan aprender y adaptarse en tiempo real, lo que hace que las técnicas de *Reinforcement Learning* y *Evolutive Computing* sean opciones prometedoras.

Actualmente, la mayoría de los estudios en el campo del control de enjambres de drones se han centrado en enfoques con *Reinforcement Learning*, mientras que, la aplicación de algoritmos de *Evolutive Computing* es limitada. Dicha escasez de casos puede atribuirse a la dificultad para diseñar una representación adecuada del problema de control de enjambres de drones que sea compatible con dichos algoritmos o incluso, a que el enfoque en los últimos años ha estado, sobre todo, en la utilización de otro tipo de algoritmos relacionados con el *Reinforcement Learning*.

En este estudio se evaluaron la eficacia de algoritmos de *Deep Q-Learning*, Algoritmo Genético (AG) y *Ant Colony Optimization* (ACO) para la planificación de rutas en enjambres de drones en dimensiones de 3x3, 5x5 y 7x7. Los resultados revelaron que ACO superó significativamente a las demás técnicas en reducir acciones y tiempos de ejecución, mostrando su valor en contextos donde la eficiencia operativa es esencial. Aunque *Deep Q-Learning* y AG también presentaron mejoras, no lograron igualar la eficiencia de ACO. Estos hallazgos sugieren que ACO es más adecuado para la implementación práctica en misiones complejas, proporcionando una base sólida para la elección de algoritmos en función de las demandas operativas y específicas de cada tarea. Es fundamental destacar la importancia de seleccionar el algoritmo adecuado y la configuración óptima del número de drones para maximizar la eficiencia operativa de los enjambres en aplicaciones reales.

Palabras clave: *Reinforcement Learning*, *Evolutive Computing*, vehículos aéreos no tripulados, UAVs, planificación de rutas

Abstract

Reinforcement Learning and Evolutionary Computing techniques have proven to be powerful tools in solving a wide range of problems in fields such as robotics, logistics and path planning. In particular, they have been successfully used in trajectory optimization for unmanned aerial vehicles (UAVs). However, despite their potential, their application in drone swarm control has been remarkably scarce.

The use of drone swarms involves streamlining flight time and reducing operational costs, where, in addition to optimizing individual drone trajectories, it is crucial to ensure swarm cohesion, avoid collisions, and adapt to changing environmental conditions and mission demands. These challenges require innovative approaches that can learn and adapt in real time, which makes Reinforcement Learning and Evolutionary Computing techniques promising options.

Currently, most of the studies in the field of drone swarm control have focused on approaches with Reinforcement Learning, while the application of Evolutionary Computing algorithms is limited. This scarcity of cases can be attributed to the difficulty in designing a suitable representation of the drone swarm control problem that is compatible with these algorithms, or even to the fact that the focus in recent years has been mainly on the use of other types of algorithms related to Reinforcement Learning.

This study evaluated the effectiveness of Deep Q-Learning, Genetic Algorithm (GA) and Ant Colony Optimization (ACO) algorithms for route planning in drone swarms in 3x3, 5x5 and 7x7 dimensions. The results revealed that ACO significantly outperformed the other techniques in reducing actions and execution times, showing its value in contexts where operational efficiency is essential. Although Deep Q-Learning and AG also showed improvements, they failed to match the efficiency of ACO. These findings suggest that ACO is more suitable for practical implementation in complex missions, providing a solid basis for algorithm choice based on operational and task-specific demands. It is critical to highlight the importance of selecting the right algorithm and the optimal configuration of the number of drones to maximize the operational efficiency of swarms in real applications.

Keywords: Reinforcement Learning, Evolutive Computing, unmanned aerial vehicles, UAVs, path planning

Agradecimientos

Quisiera comenzar estos agradecimientos expresando mi profunda gratitud a mis padres. Gracias por todas las oportunidades que me habéis brindado y por siempre tener confianza en mí en todo lo que hago. Vuestro apoyo incondicional ha sido un pilar fundamental en mi vida. Sin vuestro amor y respaldo, no habría llegado hasta aquí.

A mi tutor, Francisco Lamas López, quiero agradecerle su guía y apoyo durante todo este proceso. Su conocimiento, paciencia y disposición para ayudarme en cada etapa del proyecto han sido esenciales para el desarrollo de este trabajo. Gracias por creer en mí y por proporcionarme las herramientas necesarias para crecer tanto académica como profesionalmente.

A mis amigos, gracias por estar siempre, tanto en los momentos felices, como en los momentos difíciles y, darme esos pequeños momentos de felicidad donde he conseguido desconectar de lo rutinario. En especial, quiero mencionar a Pablo Palacios, quien ha sido mi compañero incondicional durante los últimos cinco años. No solo me has ayudado a ser mejor en lo académico, sino también en lo personal. Hemos compartido tantos momentos juntos, y tenerte a mi lado en este camino ha sido una de las mejores partes de esta experiencia.

Finalmente, quiero tomar un momento para agradecerme a mí mismo el trabajo duro y la dedicación que he puesto, no solo en este proyecto, sino en todo lo que hago. Este logro es el resultado de mi esfuerzo y determinación, y me siento orgulloso de todo lo que he alcanzado. Siguiendo siempre mi lema, ”*Strive for greatness*”, he logrado superar los desafíos y alcanzar mis metas.

Gracias a todos los que de alguna manera han contribuido a la realización de este Trabajo Fin de Máster. Vuestro apoyo y confianza han sido cruciales para alcanzar esta meta.

Índice de Contenidos

Índice de figuras

Índice de tablas

Índice de algoritmos

1. Introducción

El rápido avance de la tecnología de drones ha generado un gran interés en su aplicación en una variedad de campos, como la agricultura (Pederi and Cheporniuk, 2015), la vigilancia (Pinto et al., 2017), la logística (Ni et al., 2018), la búsqueda y rescate (Haddad et al., 2023) o incluso el monitoreo ambiental (Abdelkader et al., 2021). En particular, los enjambres de drones, compuestos por múltiples vehículos aéreos no tripulados que trabajan de manera colaborativa, han emergido como una herramienta prometedora para abordar una amplia gama de desafíos prácticos.

El control efectivo de estos enjambres de drones es fundamental para maximizar su utilidad en aplicaciones del mundo real. Coordinar múltiples drones en tiempo real, optimizando sus trayectorias y tareas, mientras se evitan colisiones y se adapta a los cambios en el entorno, representa un desafío complejo que requiere enfoques computacionales avanzados.

Uno de los objetivos clave en la ampliación de los enjambres de drones es la eficiencia operacional. El estudio realizado por Puente-Castro et al. (2023a) ha demostrado que al incrementar el número de drones en operación, se pueden reducir significativamente los tiempos de misión y el número de acciones individuales de cada dron, lo que resulta en una ejecución más rápida y eficiente de tareas complejas.

Las técnicas de *Reinforcement Learning* y *Evolutive Computing* han demostrado ser herramientas poderosas en la resolución de problemas de control y planificación en una variedad de dominios. Sin embargo, su aplicación en el control de enjambres de drones ha sido limitada hasta la fecha.

A pesar de los avances significativos utilizando técnicas de *Reinforcement Learning* como *Q-Learning* (Puente-Castro et al., 2022b) o *Deep Q-Learning* (Puente-Castro et al., 2023a), la aplicación de algoritmos de *Evolutive Computing* ha sido escasa.

Por lo tanto, es fundamental continuar investigando y desarrollando modelos pertenecientes al campo del *Reinforcement Learning* y, además, contribuir a la aplicación de algoritmos de *Evolutive Computing* para así, poder determinar si es de mayor utilidad utilizar un tipo de algoritmo u otro dependiendo del entorno en el que nos encontremos. Además, es fundamental demostrar por qué es de mayor utilidad los enjambres de drones que el uso de un solo dron.

1.1. Motivación

La motivación principal detrás de incrementar el número de drones radica en la mejora significativa en eficiencia y efectividad que estos sistemas colaborativos pueden ofrecer en comparación con operaciones de drones individuales.

El uso de enjambres de drones permite una cobertura más rápida y extensa del área de operación, reduciendo así los tiempos de misión. Investigaciones como la de Hayat et al. (2020) han demostrado que la coordinación y control de múltiples drones, mediante algoritmos sofisticados de planificación de trayectorias, puede disminuir considerablemente el tiempo necesario para completar tareas de planificación de rutas.

Introducir más drones en operaciones también incrementa la capacidad de carga y la redundancia del sistema. Esto es especialmente crucial en tareas críticas donde el fallo de un solo dron no debe comprometer la misión completa. La capacidad de carga distribuida entre varios drones facilita la realización de tareas que serían imposibles para un solo dron (Hayat et al., 2020).

Las técnicas de *Reinforcement Learning* (RL) y *Evolutive Computing* (EC) ofrecen métodos prometedores para optimizar el control y la coordinación de enjambres de drones. Estas técnicas permiten que los sistemas de drones aprendan y se adapten a condiciones dinámicas en tiempo real, mejorando continuamente su eficiencia operativa. La investigación en este campo busca explorar diversos algoritmos y estrategias para determinar cuál ofrece mejor rendimiento en diferentes escenarios operacionales.

La comprensión de las fortalezas y limitaciones de ambas técnicas en este contexto no solo puede impulsar la investigación en el campo de los enjambres de drones, sino también mejorar la implementación práctica de estos sistemas en diversas aplicaciones del mundo real.

En este Trabajo de Fin de Máster, se desarrolló y evaluó dos algoritmos de *Evolutive Computing* y uno de *Reinforcement Learning*, diseñados para optimizar la coordinación y eficiencia de enjambres de drones en entornos representados a través de mallas. Estos algoritmos facilitan la simulación precisa de decisiones estratégicas en tiempo real y se adaptan a las complejidades dinámicas de los entornos operativos, representando un avance significativo en las técnicas de control autónomo para aplicaciones prácticas.

La contribución incluye una comparativa exhaustiva del rendimiento de los algoritmos desarrollados frente a métodos existentes, utilizando diversas configuraciones de drones. Este análisis no solo valida la efectividad de los algoritmos propuestos, sino que también ofrece una perspectiva crítica sobre su comportamiento en diferentes escenarios, estableciendo un sólido precedente para futuras investigaciones en el campo del control autónomo.

Aunque haya artículos que hablen de técnicas de *Reinforcement Learning* como es el caso del artículo de Puente-Castro et al. (2023a) o que hablen de técnicas de *Evolutive Computing*

como puede ser el artículo de Deng et al. (2023), en este trabajo se ha querido hacer una comparación entre ambos tipos de técnicas y se ha enfocado a optimizar el número de drones utilizando dichas técnicas.

Además, se demostró la aplicabilidad práctica de estos algoritmos, resaltando cómo pueden ser efectivamente implementados en el mundo real. Esto no solo ayuda a cerrar la brecha entre la teoría y la práctica, sino que también enriquece la literatura existente sobre la optimización de enjambres de drones, proporcionando un recurso valioso para futuros investigadores interesados en la robótica y los sistemas autónomos.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Máster es optimizar el uso del número de drones en enjambres mediante la comparación y aplicación de técnicas avanzadas de *Reinforcement Learning* y *Evolutive Computing*. A continuación, se detallan los objetivos específicos del proyecto:

1. **Investigación y Análisis del Estado del Arte:** Se realizará una revisión exhaustiva de las aplicaciones actuales de RL y EC en el control de enjambres de drones, identificando las metodologías predominantes y los resultados significativos en este campo emergente.
2. **Desarrollo e Implementación de Algoritmos:** Se adaptarán y evaluarán algoritmos de RL y EC en entornos de simulación diseñados para probar su eficacia en la optimización del número de drones en diversas condiciones operativas.
3. **Evaluación Comparativa y Análisis de Resultados:** Los resultados obtenidos serán analizados para comparar la eficiencia de los distintos algoritmos. Se determinarán las fortalezas y limitaciones de cada técnica, estableciendo en qué condiciones una técnica puede ser más adecuada que otra para la optimización en el uso de enjambres de drones.
4. **Digitalización y Simulación de Cartas Reales:** Un objetivo adicional será digitalizar una carta real topográfica en cuadrícula con obstáculos y simular su desarrollo en un entorno realista. Esto incluirá la implementación de algoritmos de RL y EC para evaluar cómo los enjambres de drones pueden navegar y completar misiones en este entorno detalladamente modelado, proporcionando una valoración crítica de su capacidad para operar en condiciones que mimetizan escenarios del mundo real.

A través de la comparación directa de técnicas de *Reinforcement Learning* y *Evolutive Computing*, se espera identificar cuál es más adecuada para mejorar la eficiencia y efectividad en el uso de drones en enjambres, adaptándose a diferentes situaciones y condiciones operativas. Este enfoque no solo contribuirá al avance del conocimiento en el campo del control de enjambres de drones, sino que también proporcionará nuevas perspectivas sobre su aplicación práctica.

1.3. Estructura del trabajo

El presente trabajo se va a dividir en cinco capítulos:

1. **Introducción**. En este capítulo se establece el marco teórico y práctico para la optimización del uso de enjambres de drones mediante técnicas de *Reinforcement Learning* y *Evolutive Computing*. Se destaca la importancia de los enjambres de drones en la mejora de la eficiencia operativa.

   Además, se expone la motivación principal del trabajo: aumentar el número de drones para mejorar tanto la cobertura operacional como la eficacia en la ejecución de tareas mediante algoritmos que permitan una simulación precisa y adaptativa de decisiones estratégicas en tiempo real. Se enfatiza la importancia de comparar las técnicas de *Reinforcement Learning* y *Evolutive Computing* para identificar cuál es más efectiva para optimizar el uso de enjambres de drones en diferentes condiciones operativas, apuntando a un avance significativo en las técnicas de control autónomo para aplicaciones prácticas (Capítulo 1).

2. **Estado del arte**. En este segundo capítulo, se lleva a cabo un exhaustivo análisis del estado del arte en las áreas de *Reinforcement Learning* y *Evolutive Computing*, enfocadas específicamente en su aplicación al control de enjambres de drones. Este capítulo se inicia con una revisión detallada de los conceptos fundamentales de los enjambres de drones, destacando su relevancia en la planificación de rutas y la optimización de trayectorias.

   El análisis profundiza en los avances recientes y las aplicaciones de *Reinforcement Learning* y *Evolutive Computing*, discutiendo cómo estos métodos pueden facilitar el aprendizaje y la adaptación en tiempo real para mejorar la eficiencia operativa de los enjambres de drones. El capítulo concluye con un llamado a la investigación futura para desarrollar y probar algoritmos que integren estos enfoques, con el objetivo de superar los desafíos actuales y optimizar aún más el funcionamiento de los enjambres de drones en aplicaciones prácticas y reales (Capítulo 2).

3. **Experimentación y metodología**. En este tercer capítulo se presenta la metodología utilizada para desarrollar y evaluar modelos de *Reinforcement Learning* y *Evolutive Computing* aplicados al control de enjambres de drones. Este capítulo inicia con una revisión de trabajos previos y métodos aplicados recientemente, para contextualizar las bases sobre las cuales se construyen los nuevos modelos propuestos en este estudio. Se hace especial énfasis en la adaptación de algoritmos existentes y la comparación de su rendimiento a través de distintas métricas, estableciendo así un marco comparativo robusto.

   Además, se detalla la creación del entorno de simulación que replica situaciones realistas para los drones, configurado como mallas bidimensionales con obstáculos. Este diseño permite evaluar de manera precisa y eficiente los algoritmos de control propuestos, proporcionando una representación simplificada pero efectiva del espacio operativo de los drones. La introducción de obstáculos en la malla añade complejidad al

modelo, lo que desafía a los algoritmos a optimizar la planificación de rutas y evitar colisiones, reflejando así los desafíos reales que enfrentan los sistemas de enjambres de drones en entornos operativos (Capítulo 3).

4. **Resultados**. En el capítulo 4, se presentan los resultados obtenidos a partir de la aplicación de diferentes algoritmos de optimización sobre enjambres de drones en entornos simulados. Se detalla cómo las métricas de rendimiento, como el porcentaje de acciones válidas y la variabilidad del desempeño, se ven afectadas por el entorno, el número de drones, y el tipo de algoritmo utilizado.

   Además, se destaca que los enjambres de drones, en comparación con el uso de un solo dron, mejoran significativamente el porcentaje de acciones válidas al incrementar la cobertura, la redundancia, y la capacidad de adaptación del sistema. Este capítulo subraya la importancia de seleccionar el algoritmo adecuado según el tamaño del espacio operativo y el número de drones para optimizar la efectividad de los enjambres en aplicaciones reales (Capítulo 4).

5. **Simulación en entorno real**. En este quinto capítulo se explora la simulación en entornos reales digitalizados, donde se ha conseguido transcribir una carta topográfica real en una cuadrícula con obstáculos para evaluar la operación de múltiples drones. Utilizando herramientas avanzadas como PX4 Autopilot, Unreal Engine 4.27, AirSim y QGroundControl, se crea un entorno de simulación altamente realista que permite a los drones navegar y completar misiones en escenarios que imitan condiciones del mundo real.

   La configuración de este entorno de simulación incluye la integración de diversas tecnologías que trabajan en conjunto para simular la dinámica y física de vuelo de los drones, proporcionando una plataforma robusta para la validación de estrategias de control y navegación. A lo largo del capítulo, se detalla cómo estos sistemas permiten la ejecución de simulaciones complejas, donde se monitorean y gestionan las rutas de vuelo, se visualiza en tiempo real la posición y el estado de los drones, y se evalúa la eficacia de los algoritmos en operar de manera segura y eficiente dentro de entornos digitales que reflejan desafíos operativos reales (Capítulo 5).

6. **Conclusiones del proyecto**. En este sexto capítulo, se detallan las conclusiones finales del estudio comparativo entre técnicas de *Reinforcement Learning* y algoritmos de *Evolutive Computing* aplicados al control de enjambres de drones. A lo largo del proyecto, se evaluaron varios algoritmos como *Deep Q-Learning*, *Ant Colony Optimization* y Algoritmo Genético.

   Además, se resalta la importancia de elegir el algoritmo adecuado en función del tamaño del espacio operativo y el número de drones, subrayando la variabilidad en el rendimiento según estas condiciones. Para futuras investigaciones, se recomienda la exploración de enfoques híbridos que combinen las fortalezas de ambos tipos de algoritmos y el desarrollo de algoritmos adaptativos que puedan ajustarse a cambios dinámicos del entorno (Capítulo 6).

2. Estado del Arte

En este capítulo, se lleva a cabo una revisión detallada de las técnicas actuales de *Reinforcement Learning* y *Evolutive Computing*, especialmente en su aplicación al control de enjambres de drones. Este capítulo aborda la historia y la evolución de estas técnicas, destacando cómo se han aplicado en diversos contextos y qué se ha logrado hasta la fecha. También se analizan los enfoques predominantes y los resultados significativos, identificando brechas y oportunidades para futuras investigaciones.

Además, el capítulo tiene como objetivo detallar cómo estas tecnologías pueden ser implementadas para mejorar la eficiencia y efectividad de los enjambres de drones, enfocándose en optimizar el número de drones en operación mediante una aplicación práctica de estas técnicas en entornos simulados y reales. Se discute la posibilidad de integrar y comparar directamente *Reinforcement Learning* y *Evolutive Computing* para determinar cuál ofrece mejor rendimiento en escenarios operativos específicos, contribuyendo así al avance del conocimiento y ofreciendo nuevas perspectivas para la implementación práctica en el campo del control de enjambres de drones.

2.1. Conceptos generales de los enjambres de drones

A lo largo de esta sección se van a explicar los aspectos técnicos relacionados con los enjambres de drones para, así, entender de mejor manera el problema de los enjambres de drones en un problema de planificación de rutas.

Un dron es una aeronave semiautónoma que puede controlarse y manejarse a distancia, sin tripulación a bordo, mediante un sistema electrónico de inteligencia y control (Puente-Castro et al., 2022a). Los drones, hasta el comienzo de los años 2000, eran utilizados en el ámbito militar y no eran accesible para los civiles, sin embargo, a lo largo de los últimos años, los drones han empezado a ser utilizados por los civiles debido a los avances en *hardware* y *software* que permiten un desarrollo de sistemas más pequeños, fáciles de controlar y más económicos. Dichos drones han empezado a utilizarse para múltiples actividades como la agricultura, la fotografía o los deportes extremos entre muchos otros (Tezza and Andujar, 2019).

A continuación, se van a nombrar y detallar los tipos de vehículos no tripulados, los cuales se clasifican según el entorno en el que operan y sus aplicaciones específicas:

1. **UAV (Unmanned Aerial Vehicle)**: Vehículo aéreo sin operador humano, el cual utiliza fuerzas aerodinámicas para sustentarse, puede volar de forma autónoma o ser pilotado a distancia, puede ser recuperable, y puede transportar cargas letales o no letales (Chen et al., 2009).
2. **UGV (Unmanned Ground Vehicle)**: Vehículos terrestres no tripulados, los cuales se definen como cualquier pieza de equipo mecanizado que se desplaza sobre la superficie del terreno y sirve como medio para llevar o transportar algo, pero que explícitamente no transporta a un ser humano (Tran, 2007).
3. **USV (Unmanned Surface Vehicle)**: Vehículos que operan en la superficie del agua, basados en innovaciones en el diseño de pequeñas embarcaciones de alta velocidad y gran capacidad de carga y en la tecnología de sistemas no tripulados (Yan et al., 2010).
4. **UUV (Unmanned Underwater Vehicle)**: Los UUV se utilizan para describir vehículos submarinos que pueden ser operados remotamente o funcionar de manera autónoma (Vervoort, 2009).

    - **AUV (Autonomous Underwater Vehicle)**: Subcategoría de los UUVs que operan independientemente con baterías y sonar para cumplir misiones sin intervención directa del operador (Vervoort, 2009).
    - **ROV (Remotely Operated Vehicle)**: Subcategoría de los UUVs que está conectado a una plataforma de comando mediante un cable o un enlace acústico que permite el control constante (Vervoort, 2009).

Una vez definido qué son los drones y los diferentes tipos de vehículos no tripulados que existen, tenemos que indicar en qué consisten los enjambres de drones. Los cuales son un conjunto de drones que pueden ejecutar una tarea de manera más efectiva y económica, ya que, con el desarrollo de ligeros componentes electrónicos y sensores, el tamaño y el coste de los drones se ha visto altamente reducido, permitiendo a los enjambres estar compuestos de pequeños drones. Sin embargo, aunque los enjambres de drones presentan diversas ventajas, también hay algunos temas desafiantes que necesitan ser tratados, los cuales podrían ser, el hecho de no tener ningún problema de conexión para comunicarse y, la limitación de energía y almacenamiento, entre otros (Chen et al., 2020).

Debido al alto coste económico que conlleva realizar experimentos en entornos reales y a la necesidad de conocer la legislación de cada país, este tipo de problemas se suele probar mediante simulaciones que simulan las distintas condiciones que hay en un entorno real.

2.2. *Path Planning*

La planificación de rutas, comúnmente conocido como *path planning*, es el proceso de usar los datos acumulados proveniente de los sensores y una información inicial para que un robot autónomo encuentre la mejor ruta para alcanzar un objetivo. Este tipo de problemas es muy común, no solo en problemas relacionados con los UAVs, sino con cualquier tipo de robot móvil. Además, la planificación de rutas está compuesta por dos pasos principales: el primero, es recopilar toda la información disponible en un espacio de configuración; y segundo, utilizar un algoritmo de búsqueda para encontrar la mejor ruta en ese espacio (Giesbrecht, 2004). Asimismo, hay que destacar la presencia de distintos desafíos en este tipo de problemas como puede ser la presencia de obstáculos, la tolerancia a los posibles fallos que puedan ocurrir en ciertos casos, la completitud de acuerdo a ciertos criterios o el descubrimiento de un camino más corto que los anteriores, por lo tanto, este sería el óptimo hasta el momento (Puente-Castro et al., 2022a).

La planificación de trayectorias es esencial en múltiples campos de la robótica y sistemas autónomos. A continuación, se presentan algunas aplicaciones clave:

- **Robótica móvil:** Es fundamental para la navegación y la evitación de obstáculos en robots industriales. En el artículo de Kuffner and LaValle (2000), se discute en profundidad estos aspectos en su estudio sobre el algoritmo RRT (*Rapidly-exploring Random Tree*), que es ampliamente utilizado para la planificación de trayectorias en robots móviles.
- **Vehículos autónomos:** Los métodos de planificación son cruciales para la seguridad y eficiencia en vehículos autónomos. En el estudio de Yoshida and Kanehiro (2011), se proponen métodos para la planificación de trayectorias que consideran tanto la seguridad como la eficiencia operativa en vehículos autónomos.
- **Exploración espacial:** La navegación de *rovers* en planetas desconocidos es un área prominente de aplicación, tal y como se expone en el artículo de Biesiadecki et al. (2007), en donde, se discute cómo estos métodos se han aplicado en *rovers* de la NASA en Marte.

Estas aplicaciones muestran la diversidad y la importancia de la planificación de trayectorias en múltiples dominios, cada uno con sus propios desafíos y requisitos. Los estudios citados proporcionan un marco robusto para entender cómo el *path planning* contribuye significativamente a la eficiencia y la funcionalidad en sistemas autónomos y robóticos.

En el contexto de los enjambres de drones, el *path planning* adquiere una dimensión adicional, ya que no se trata solo de planificar la trayectoria de un solo dron, sino coordinar múltiples trayectorias simultáneamente, optimizando la ruta global del enjambre mientras se evitan colisiones entre los drones y con el entorno. Esta coordinación debe considerar los movimientos dinámicos de cada dron y las incertidumbres del entorno, lo que hace significativamente más complejo el problema (McLain, 1999).

El estudio realizado por Bortoff (2000) propone un enfoque novedoso para el *path planning* de UAVs, donde se emplea un método de dos pasos que incluye la generación de una ruta inicial mediante la búsqueda en un grafo basado en polígonos, seguido por una optimización mediante la simulación de ecuaciones diferenciales que representan fuerzas virtuales actuando sobre masas virtuales. Este enfoque permite una planificación que equilibra efectivamente la sigilosidad y la longitud de la ruta, proporcionando soluciones que son viables para la implementación en tiempo real.

Las técnicas de *Reinforcement Learning* y *Evolutive Computing* han comenzado a implementarse para mejorar la planificación de trayectorias en enjambres de drones. Estos métodos permiten que los enjambres de drones no solo sigan rutas predefinidas, sino también que se adapten de manera inteligente a los cambios del entorno, optimizando sus rutas en tiempo real. Esto resulta en una mayor flexibilidad y robustez del sistema de navegación del enjambre (Vásárhelyi et al., 2018).

2.3. *Reinforcement Learning*

El *Reinforcement Learning* es una técnica que consiste en aprender a asignar acciones a situaciones específicas con el fin de maximizar una señal numérica de recompensa, en donde, al agente no se le dice qué acciones tiene que tomar, sino que debe descubrir qué acciones producen la mayor recompensa probándolas. En los casos más interesantes y desafiantes, las acciones puede afectar no solo a una recompensa inmediata y, sino que también a la situación siguiente y, a través de ella, a todas las recompensas posteriores (Sutton and Barto, 2018).

El *Reinforcement Learning* es particularmente apropiado para la planificación de trayectorias en enjambres de drones debido a su capacidad para manejar entornos complejos y dinámicos a través del aprendizaje cooperativo y distribuido. Tal y como se expone en el artículo de Pham et al. (2018), los drones equipados con *Reinforcement Learning* pueden mejorar significativamente la cobertura de campos agrícolas, optimizando sus rutas de manera colaborativa para maximizar la eficiencia y minimizar el tiempo de operación. Este enfoque permite que cada dron en el enjambre ajuste su comportamiento en función de las recompensas recibidas, aprendiendo a navegar el entorno de manera más efectiva mientras coordina con otros drones para cubrir el área de manera integral.

Uno de los ejemplos en donde se aplica *Reinforcement Learning* puede ser un robot móvil, el cual decida si debe entrar en una habitación en busca de más basura que recoger o empezar a tratar de encontrar su camino de regreso a su estación de recarga de baterías. Su decisión se toma basándose en el nivel de carga actual de su batería y, en lo rápido y fácil que ha sido capaz de encontrar el cargador en el pasado.

El agente debe explotar lo que ya sabe para beneficiarse de las recompensas, pero también debe explorar para elegir mejor sus acciones futuras. El problema es que perseguir únicamente la exploración o la explotación conduciría al fracaso. El agente debe probar varias opciones diferentes y favorecer gradualmente la que parezca funcionar mejor (Puente-Castro et al., 2022b).

A continuación, se van a definir los elementos principales que debemos de conocer pertenecientes a un sistema de *Reinforcement Learning* (Sutton et al., 1999):

- **Función de recompensa**: Define el objetivo del agente en una situación determinada, por lo que, el único objetivo de este es maximizar la recompensa neta a largo plazo. La función de recompensa define lo que es objetivamente bueno y malo para el agente y, este no puede modificar la función de recompensa.
- **Modelo del entorno**: Este modelo imita el comportamiento del entorno, permitiendo hacer inferencias sobre cómo se comportará el entorno en el futuro. Además, hay que destacar que su presencia es opcional, es decir, podemos tenerlo o no en un problema de *Reinforcement Learning*.
- **Política**: Define la forma en que el agente se comporta en un momento dado, mapeando estados percibidos del entorno a acciones a tomar. Corresponde a lo que en psicología se llamaría un conjunto de reglas o asociaciones estímulo-respuesta. Cabe destacar que, la política es el núcleo de un agente de aprendizaje por refuerzo, es decir, solo ella es suficiente para determinar el comportamiento.
- **Función de valor**: Especifica lo que es bueno a largo plazo, indicando el valor total de una secuencia de estados desde un estado inicial. Es la cantidad total de recompensa que un agente puede esperar acumular en el futuro, comenzando desde ese estado. Esta función tendrá en cuenta los estados que probablemente seguirán y las recompensas disponibles en esos estados.

Los problemas de *Reinforcement Learning* normalmente son modelados utilizando el proceso de decisión de Markov (MDP), el cual forma parte de la base teórica del aprendizaje por refuerzo. Este proceso puede describirse mediante una tupla de cinco elementos (S, A, P, R, γ) (Jia and Wang, 2020):

1. **S** es un conjunto limitado de estados. Todos los estados del entorno están incluidos en el conjunto S.
2. **A** es un conjunto limitado de acciones. Todas las acciones que el agente puede tomar están incluidas en el conjunto A.
3. **P** representa una matriz de transición de estados, es decir, representa el estado del agente en el momento t y la probabilidad de alcanzar el siguiente estado s' después de tomar la acción a en el estado s. Por lo tanto, dicha probabilidad será P(s'|s, a).
4. **R** es la recompensa inmediata en el estado donde nos encontraríamos.

5. γ es el factor de descuento, el cual es un valor comprendido entre 0 y 1 que determina la importancia relativa de las recompensas futuras en comparación con las recompensas actuales.

El modelo de decisión de Markov establece que la distribución de probabilidad de los estados futuros depende únicamente del estado actual y de la acción tomada, independientemente de cómo se llegó a ese estado. Además, algunos de los algoritmos de aprendizaje por refuerzo que veremos próximamente, utilizan los MDP como base para aprender políticas óptimas o funciones de valor que guían al agente hacia el logro de sus objetivos en el entorno.

A continuación, vamos a empezar a definir uno de esos algoritmos, comenzando por el de *Q-Learning*.

2.3.1. *Q-Learning*

Q-Learning es una forma de aprendizaje por refuerzo sin modelos y proporciona a los agentes la capacidad de aprender a actuar de forma óptima en dominios markovianos, lo que quiere decir que nuestro mundo es sin memoria y, experimentando las consecuencias de las acciones, sin necesidad de que construyan mapas de los dominios (Watkins and Dayan, 1992).

Los agentes tienen que usar su experiencia para aprender los valores de todas las políticas en paralelo, incluso cuando ellos solo puedan seguir una política a la vez (Puente-Castro et al., 2023a). Sigue una estrategia sin modelo, donde el agente adquiere conocimiento siguiendo una política solo mediante ensayo y error. De esta manera, la convergencia de *Q-Learning* hacia la solución óptima es codiciosa, lo que permite alcanzar la solución óptima sin depender de la política de toma de decisiones. En otras palabras, toma decisiones basadas puramente en el entorno que rodea al agente y sus interacciones con él. De esta manera, se garantiza que el sistema pueda funcionar con diferentes tipos de entornos sin tener que buscar la política óptima que funcione en todos ellos. La ”Q” en *Q-Learning* significa calidad, que intenta representar qué tan útil es una acción dada para obtener alguna recompensa futura (Puente-Castro et al., 2022b).

El beneficio más conocido de *Q-Learning* sobre otras técnicas de *Reinforcement Learning* es que permite la comparación de la utilidad predicha entre diferentes acciones sin necesidad de un modelo de entorno. Estos algoritmos se destacan de otros enfoques debido a que toman decisiones basadas en valores almacenados dentro de una tabla. Estos valores se conocen como valores Q, y la tabla se denomina *Q-Table*. Los valores Q representan esencialmente la recompensa anticipada de una acción dentro del contexto específico del entorno. A partir de estos valores, se elige la acción con el valor más alto para cada estado.

Otro concepto muy importante del que hay que hablar es la ecuación de Bellman, la cual combina con las predicciones previas del sistema para entrenarlo.

La ecuación de Bellman es:

$$Q(s,a) = r + \gamma \cdot \max_{a'} Q(s',a') \tag{2.1}$$

donde $Q(s,a)$ es la función que calcula el valor Q para el estado actual s, del conjunto de estados, y para la acción dada a, del conjunto de acciones A, r es la recompensa de la acción tomada en ese estado y se calcula mediante la función de recompensa R(s, a), γ es el factor de descuento y $\max_{a'} Q(s',a')$ es el valor Q máximo calculado del par (s', a') representado como Q(s', a'). El par (s', a') es un par estado-acción potencial siguiente, s' es el próximo estado y se da mediante la función de transición T(s, a) que devuelve el estado resultante de la ejecución de la acción seleccionada y a' es cada una de las acciones disponibles.

Con probabilidad ε, una parte de las acciones en un problema de Q-Learning se realizan al azar, y con probabilidad 1−ε, se adopta la acción con el mayor valor Q para ese estado. Un episodio es la serie de acciones que un agente realiza para un cierto ε hasta que alcanza una condición de finalización. La operación se reinicia al comienzo de cada episodio. Durante las pruebas, los episodios reducen el valor de ε por un factor de reducción (Puente-Castro et al., 2023a). En el tipo de problemas que vamos a tratar en este Trabajo Fin de Máster, la decisión de qué hacer está influenciada más por los valores Q calculados y menos por el azar. Al considerar el valor mínimo de ε, se evita que este se acerque demasiado a cero y se evita el sobreajuste.

Asimismo, la actualización del valor Q se puede expresar de la siguiente manera (Phan and Manjanna, 2012):

$$Q_n(x,a) = \begin{cases} (1-\alpha_n)Q_{n-1}(x,a) + \alpha_n[r_n + \gamma V_{n-1}(y_n)], & \text{if } x = x_n \text{ and } a = a_n \\ Q_{n-1}(x,a), & \text{otherwise} \end{cases} \tag{2.2}$$

donde $V_n(.)$ es la función de valor actualizada en el estado n.

La convergencia del Q-Learning consiste en demostrar que la función de valor de acción aprendida, Q, se aproxima directamente a Q^*, la función de valor de acción óptima, independientemente de la política que se esté siguiendo. La clave para la prueba de convergencia es un proceso de Markov controlado artificial llamado Proceso de Repetición de Acciones (ARP) (Phan and Manjanna, 2012).

El Proceso de Repetición de Acciones (ARP) es un proceso de decisión de Markov ficticio que se utiliza como dispositivo de prueba en el Q-Learning. Se construye progresivamente a partir de la secuencia de episodios y la secuencia de tasa de aprendizaje. Se puede pensar en el ARP como un juego de cartas donde cada carta representa un episodio, con información como el estado actual, la acción tomada, el estado resultante, la recompensa inmediata y la

tasa de aprendizaje.

Cada estado en el ARP corresponde a un estado en el proceso real. Cuando se ejecuta una acción en un estado determinado en el ARP, se selecciona aleatoriamente un episodio previo que coincida con ese estado y acción, y se decide si repetir ese episodio basándose en una moneda sesgada según la tasa de aprendizaje. El proceso continúa hasta que se alcanza un estado especial de finalización (Watkins and Dayan, 1992). En resumen, el ARP es una herramienta conceptual que permite probar la convergencia del *Q-Learning*. A continuación, en el algoritmo 2.1, podremos observar el algoritmo necesario para realizar la acción a en el estado <x, k> (Phan and Manjanna, 2012).

Algoritmo 2.1: Pseudocódigo del procedimiento para realizar una acción en el Proceso de Repetición de Acciones (ARP)

procedimiento PerformAction(a, x, k)
  si $k = 0$ **entonces** ▷ Para realizar a en $< x, 0 >$
    Terminar el ARP con la recompensa inmediata de $Q_0(x, a)$ y detenerse.
  si $k > 0$ **entonces** ▷ Para realizar a en $< x, k >$ para $k > 0$
    si $x = x_n$ y $a = a_n$ **entonces**
      begin
      Con probabilidad α_k:
        Ir a $< y_k, k - 1 >$ con una recompensa inmediata de r_k y detenerse.
      or con probabilidad $1 - \alpha_k$:
        Realizar a en $< x, k - 1 >$.
      end
    si no
      Realizar a en $< x, k - 1 >$.

2.3.2. *Deep Q-Learning*

Recientemente, ha surgido una variante conocida como *Deep Q-learning* (DQN) como una alternativa, la cual tiene como objetivo mejorar el cálculo de la tabla Q utilizando aprendizaje profundo(Puente-Castro et al., 2023a).

Deep Q-learning combina el aprendizaje por refuerzo con redes neuronales profundas, ya que, el aprendizaje por refuerzo tradicional está limitado a entradas de baja dimensionalidad y las redes neuronales profundas pueden extraer representaciones abstractas de entradas de alta dimensionalidad (Mnih et al., 2015), es decir, como los espacios de estado y acción pueden ser demasiado grandes entonces no es factible almacenarlos todos en una tabla y, por lo tanto, aquí es donde entra el *Deep Q-learning*, el cual utiliza una Red Neural (NN) profunda para aproximar una función de valor Q en lugar de guardarlos en una tabla.

Es destacable que, la red neuronal puede generalizar sobre estados y acciones similares, eligiendo un movimiento deseable incluso si no ha sido entrenada en la situación exacta, eliminando la necesidad de una tabla grande.

En la imagen 2.1 podemos observar cómo sería un problema de aprendizaje por refuerzo, donde están los elementos básicos como son la observación, el entorno, el agente y las acciones y que, a la hora de tener una observación, ahí es donde pasa a tomar acción las redes neuronales, ya que, a partir de la red neuronal conseguimos ciertas salidas, las cuales nos indicarán qué acción es más recomendable tomar.

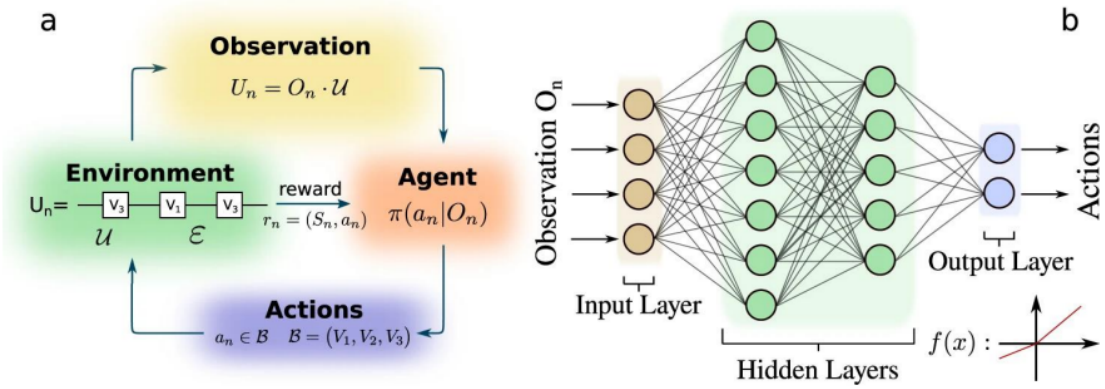


Figura 2.1: Funcionamiento de *Deep Q-learning* (Castillo, 2024).

En el ámbito más avanzado, los agentes aprenden de su entorno mediante la técnica conocida como Reproducción de Memoria. Esta técnica consiste en entrenar un modelo con un conjunto de observaciones previamente almacenadas, llamadas memoria, que contienen detalles como las acciones realizadas y las recompensas obtenidas. Al hacer esto, se logra una mejora en la eficiencia del aprendizaje al reutilizar experiencias pasadas de manera repetida, lo que contribuye a estabilizar el proceso de entrenamiento del modelo. Es crucial que esta memoria contenga la mayor cantidad posible de observaciones recientes, pero se gestiona su tamaño para optimizar los recursos computacionales. Por ello, se adopta un enfoque de eliminación de observaciones antiguas basado en el principio de ”Primero en Entrar, Primero en Salir” (Puente-Castro et al., 2022b).

A continuación, en el algoritmo 2.2, se va a mostrar el algoritmo que implementa la técnica de experiencia de repetición para mejorar el aprendizaje del agente en el contexto del *Deep Q-Learning* (Mnih et al., 2015).

Algoritmo 2.2: Pseudocódigo de *Deep Q-learning* con experiencia de repetición

Inicializar memoria de repetición D a capacidad N
Inicializar función de valor de acción Q con pesos aleatorios θ
Inicializar función de valor de acción objetivo $\hat{Q}$ con pesos $\theta^{-} = \theta$
para episodio = 1, M **hacer**
    Inicializar secuencia $s_1 = x_1$ y secuencia preprocesada $\phi_1 = \phi(s_1)$
    para t = 1, T **hacer**
        Con probabilidad ε seleccionar una acción aleatoria a_t
        de lo contrario seleccionar $a_t = argmax_a Q(\phi(s_t); a; \theta)$
        Ejecutar acción a_t en emulador y observar recompensa r_t e imagen x_{t+1}
        Establecer $s_{t+1} = s_t, a_t, x_{t+1}$ y preprocesar $\phi_{t+1} = \phi(s_{t+1})$
        Almacenar transición $(\phi_t, a_t, r_t, \phi_{t+1})$ en D
        Muestrear minibatch aleatorio de transiciones $(\phi_j, a_j, r_j, \phi_{j+})$ de D
        Establecer

$$y_j = \begin{cases} r_j & \text{si el episodio termina en el paso } j+1 \\ r_j + max_{a'} \hat{Q}(\phi_{j+1}; a'; ^{-}) & \text{de lo contrario} \end{cases}$$

        Realizar un paso de descenso de gradiente en $(y_j - Q(\phi_j; a_j;))^2$ con respecto a los parámetros de la red
        Cada C pasos resetar $\widehat{Q} = Q$

2.3.3. *SARSA (State-Action-Reward-State-Action)*

SARSA es un algoritmo de aprendizaje por refuerzo en política que estima $\widehat{Q}_\pi(s, a)$ para un par dado de estado-acción (s, a), donde el agente está siguiendo alguna política π. SARSA recibe tuplas (a, a, r, s', a') y actualiza nuestras estimaciones para $Q_\pi(s, a)$ de la siguiente manera:

$$\widehat{Q}_\pi(s, a) = (1 - \eta)\widehat{Q}_\pi(s, a) + \eta(r + \gamma \widehat{Q}_\pi(s', a')) \tag{2.3}$$

donde r es la recompensa por tomar la acción a en el estado s, γ es algún valor de descuento (en este caso, $\gamma = 1$) y η es una elección de algún valor. SARSA crea una estimación *bootstrap* $\widehat{Q}_\pi(s, a)$ minimizando la diferencia entre esa estimación y el "valor real" de nuestra política $r + \gamma \widehat{Q}_\pi(s_0, a_0)$ a lo largo de muchas iteraciones (Vu and Tran, 2020).

El proceso de SARSA se visualiza en el gráfico 2.2, donde se ilustran los estados, las acciones y las recompensas en una secuencia de tiempo. La actualización de los valores Q en SARSA incorpora tanto la recompensa inmediata como la estimación de la recompensa futura.

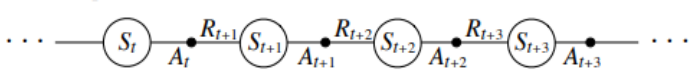


Figura 2.2: Proceso de SARSA (Sutton and Barto, 2018).

A continuación, en el algoritmo 2.3 se puede observar el pseudocódigo de SARSA (Sutton and Barto, 2018).

Algoritmo 2.3: Pseudocódigo del algoritmo de SARSA

Inicializar $Q(s,a)$ arbitrariamente y $\varepsilon(s,a) = 0$, para todo s, a
repetir (para cada episodio)
    Inicializar s, a
    repetir (para cada paso del episodio)
        Tomar acción a, observar r, s'
        Elegir a' de s' usando la política derivada de Q (por ejemplo, ε-greedy)
        $\delta = r + \gamma Q(s', a') - Q(s, a)$
        $\varepsilon(s, a) = \varepsilon(s, a) + 1$
        para cada s, a **hacer**
            $Q(s, a) = Q(s, a) + \varepsilon(s, a)\delta$
            $\varepsilon(s, a) = \gamma\lambda\varepsilon(s, a)$
        s = s'; a = a'
    hasta recorrer todos los pasos de cada episodio
hasta recorrer todos los episodios

Por lo tanto, se ha podido observar que SARSA es similar al *Q-Learning*, pero actualiza los valores de Q basados en la política actual y es un modelo *On-Policy*, ya que, el agente aprende y mejora su política de toma de decisiones mientras interactúa directamente con el entorno utilizando la misma política.

2.4. *Evolutive Computing*

Los Algoritmos Evolutivos son parte de un área de estudio llamada Computación Evolutiva, que se enfoca en métodos computacionales inspirados en cómo funciona la evolución en la naturaleza. Darwin propuso la idea de que la evolución ocurre a través de la selección natural, donde las especies cambian y se adaptan a su entorno con el tiempo. Los Algoritmos Evolutivos investigan cómo recrear estos procesos de evolución en sistemas computacionales, con el objetivo de desarrollar sistemas que puedan adaptarse y mejorar con el tiempo (Brownlee, 2011).

Generalmente, los algoritmos en el campo del *Evolutive Computing* funcionan de la siguiente manera: Primero se inicializa la población de manera aleatoria y luego se somete a un proceso de selección, combinación y mutación a lo largo de varias generaciones, lo cual hace que las generaciones posteriores evolucionen hacia regiones más favorables del espacio de búsqueda. El avance en la búsqueda se logra al evaluar la aptitud de todos los individuos en la población, seleccionar aquellos con una aptitud mejor y combinarlos para crear nuevos individuos con una mayor probabilidad de mejorar su aptitud. Después de algunas generaciones, el programa llega a un punto de convergencia, donde el mejor individuo representa la solución óptima o casi óptima. Aunque hay varios algoritmos evolutivos, la estructura básica de todos es bastante similar (Reddy and Kumar, 2020).

A continuación, en el algoritmo 2.4 se muestra el típico proceso de un algoritmo de *Evolutive Computing* (Wong, 2015).

El diseño del algoritmo evolutivo se puede dividir en varios elementos, los cuales son importante conocer para poder comprender de mejor manera los algoritmos de *Evolutive Computing*. A continuación, vamos a definir dichos componentes (Wong, 2015).

- **Representación**. Este aspecto implica cómo representamos la información genética y cómo la convertimos en características observables.
- **Selección de padres**. Aquí nos centramos en elegir individuos que servirán como progenitores en el proceso de reproducción.
- **Operadores de cruce**. Estos operadores imitan el proceso de reproducción en la naturaleza. Su función es combinar la información genética de dos individuos para crear uno nuevo.
- **Operadores de mutación**. Aquí simulamos cambios aleatorios en partes de los genomas, como los que ocurren naturalmente debido a la mutación genética. Las mutaciones pueden ser una forma de explorar nuevas posibilidades y equilibrar la influencia de los operadores de cruce en el proceso evolutivo.
- **Selección de supervivencia**. Este proceso se encarga de elegir qué individuos de una generación pasada sobrevivirán para reproducirse en la próxima generación. Normalmente, los individuos con mejores características tienen más posibilidades de sobrevivir, lo que ayuda a mantener y mejorar las características deseables en la población.

- **Condición de terminación**. Aquí definimos el criterio que indica cuándo el algoritmo evolutivo debe detenerse.

Algoritmo 2.4: Pseudocódigo del algoritmo Evolutivo típico

```
Elegir métodos de representación adecuados

P(t) : Población de Padres en el tiempo t;
O(t) : Población de Descendientes en el tiempo t;

t ← 0;
Inicializar P(t);
repetir (hasta cumplir con el criterio de terminación)
    temp = Selección de Padres de P(t);
    O(t + 1) = Cruce en temp;
    O(t + 1) = Mutar O(t + 1);
    si superposición entonces
        P(t + 1) = Selección de Supervivencia de O(t + 1) ∪ P(t)
    si no
        P(t + 1) = Selección de Supervivencia de O(t + 1)
    t ← t + 1;
hasta se alcance el número máximo de generaciones o se satisfaga algún criterio de convergencia

Los buenos individuos se pueden encontrar en P(t)
```

El artículo de Lamont et al. (2007), destaca la utilidad de los algoritmos de *Evolutive Computing* en la planificación de rutas para enjambres de drones. *Evolutive Computing* proporciona soluciones eficaces al permitir la exploración de diversas configuraciones y ajustes en la planificación de rutas, lo que es esencial en escenarios donde las rutas deben adaptarse a múltiples variables y restricciones operativas.

Además, dicho artículo ilustra cómo *Evolutive Computing* integra comportamientos de enjambre y planificación de rutas en una arquitectura jerárquica, lo que permite gestionar la complejidad del control de múltiples UAVs en misiones simultáneas. Esta capacidad para manejar la complejidad y proporcionar soluciones optimizadas a problemas multifacéticos subraya la adecuación del *Evolutive Computing* para la planificación de rutas en enjambres de drones (Lamont et al., 2007).

2.4.1. Algoritmos Genéticos

Los algoritmos genéticos son técnicas de búsqueda y optimización que imitan la selección natural de Darwin. En estos algoritmos, las variables se representan como genes en cromosomas y se busca encontrar soluciones óptimas mediante procesos de selección y operaciones genéticas como la recombinación y la mutación. Esto permite identificar cromosomas con características más adaptadas (Tu and Yang, 2003).

Se puede afirmar que, un algoritmo genético se ejecuta con cinco componentes, los cuales son: una representación genética adecuada para cada individuo (cromosomas), un método para generar la población inicial, una función de adaptación para evaluar la calidad de cada solución potencial, operadores genéticos que modifican la composición genética de los padres para producir una nueva descendencia y, finalmente, la elección de los valores de los diversos parámetros, como puede ser el tamaño de la población, la tasa de cruzamiento, la tasa de mutación, los criterios de detención, etc (Lamini et al., 2018). Estos componentes se pudieron visualizar en el algoritmo 2.4, esto es debido a que este pseudocódigo corresponde también con el funcionamiento de los algoritmos genéticos.

Los algoritmos genéticos puedes aplicarse a multitud de problemas y, en la figura 2.3 podemos ver un ejemplo, que proporciona Mitchell (1995), de un tipo de problema al que se puede aplicar, como es el de planificación de rutas. Dicho problema consiste en encontrar una secuencia de pasos que moverán el robot desde la entrada hasta la salida. En el ejemplo de la imagen podemos observar que, cada movimiento puede ser representado por dos bits. La aptitud de una secuencia particular puede calcularse dejando que el robot siga la secuencia y luego midiendo el número de pasos entre su posición final y la salida; cuanto menor sea la distancia, mayor será la aptitud de la secuencia. La máxima aptitud se alcanza si el robot llega a la salida en N pasos o menos.

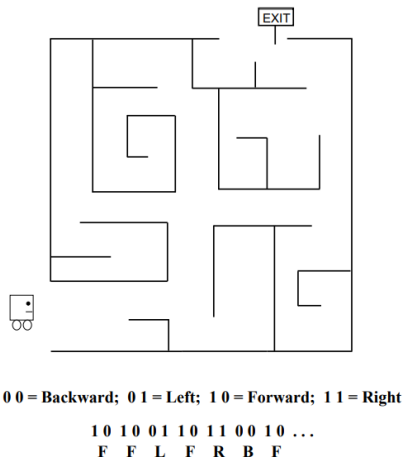


Figura 2.3: Ejemplo para algoritmos genéticos (Mitchell, 1995).

El proceso que se seguiría en dicho ejemplo sería el siguiente: Supongamos que tenemos una población inicial de 5 individuos (rutas posibles), cada uno representado por una cadena binaria.

- **Generación 0**:
  - Individuo 1: 101001
  - Individuo 2: 110010
  - Individuo 3: 011011
  - Individuo 4: 100101
  - Individuo 5: 111000

Evaluamos cada individuo en el laberinto y asignamos una puntuación basada en qué tan cerca llegaron a la salida. Supongamos que las puntuaciones son las siguientes:

- Individuo 1: 3
- Individuo 2: 2
- Individuo 3: 4
- Individuo 4: 1
- Individuo 5: 1

Selección: Elegimos los individuos con las mejores puntuaciones para ser padres. En este caso, serían los individuos 1 y 3.

Cruce: Creamos una nueva generación cruzando los padres. Por ejemplo:

- **Padre**: 101001
- **Madre**: 011011
- **Hijo**: 101011

También podríamos introducir mutaciones aleatorias para aumentar la diversidad genética. Por ejemplo, el hijo podría mutar a 101010.

Continuamos este proceso durante varias generaciones hasta que encontramos una solución satisfactoria o alcanzamos un número máximo de generaciones. Este es un ejemplo simplificado y en la práctica, los detalles específicos del algoritmo genético pueden variar. Por ejemplo, podríamos usar diferentes métodos de selección, cruce y mutación, o ajustar la tasa de mutación y otros parámetros. Además, la representación de los individuos podría ser diferente dependiendo del problema específico que estamos tratando de resolver.

Hay que destacar que, se pueden elegir más de 2 padres, es decir, se pueden realizar múltiples cruces en pares para generar la próxima generación. Por ejemplo, si seleccionamos cuatro padres, podríamos cruzar el primer y segundo padre para producir un hijo, el tercer y

cuarto padre para producir otro hijo, y así sucesivamente.

Los algoritmos tienen diversas aplicaciones en diversos campos, como en el de la medicina (Ghaheri et al., 2015), en el de los problemas de planificación de rutas (Lamini et al., 2018), en el de las aplicaciones financieras (Aguilar-Rivera et al., 2015), entre muchos otros.

2.4.2. *Ant Colony Optimization* (ACO)

Ant Colony Optimization es un algoritmo que se inspira en cómo las hormigas reales buscan comida y encuentran los caminos más cortos. Es una técnica de búsqueda general que utiliza poblaciones para resolver problemas de optimización combinatoria difíciles. El primer algoritmo de sistema de hormigas se basó en cómo las colonias de hormigas reales buscan alimentos. Este algoritmo se activa por el rastro de feromonas dejado por las hormigas y su comportamiento de seguimiento (Janga Reddy and Kumar, 2012). Aunque, *Ant Colony Optimization* no es un algoritmo evolutivo tradicional, comparte algunos elementos que podrían asociarse con el *Evolutive Computing*, como puede ser, la búsqueda de soluciones óptimas, la adaptación al entorno y el uso de heurísticas para guiar la búsqueda.

Seguidamente, mediante la figura 2.4, se va a explicar con un ejemplo el funcionamiento de un algoritmo de colonia de hormigas. En dicho ejemplo, se considera que H es el hogar, F es la fuente de alimento y A-B representa un obstáculo en el camino. Al inicio, las hormigas eligen de manera uniforme entre los caminos de la izquierda y la derecha mientras buscan la fuente de alimento. Luego, las hormigas que tomaron el camino F-B-H llegan a la fuente de alimento primero y regresan a casa, mientras que aquellas que tomaron el camino H-A-F aún están a mitad de camino hacia la fuente de alimento. Más tarde, dado que las hormigas se mueven a una velocidad constante, las que eligieron el camino más corto, a la derecha (H-B-F), regresan a casa más rápido, dejando más feromona en ese camino. Finalmente, la feromona se acumula más rápidamente en el camino más corto (H-B-F), por lo que automáticamente es preferido por las hormigas y, por lo tanto, todas las hormigas seguirán ese camino más corto. La intensidad del sombreado es proporcional a la cantidad de feromona que depositan las hormigas.

En la imagen 3.10 se va a representar mediante un diagrama de flujo, el funcionamiento de un algoritmo de *Ant Colony Optimization* (Brand et al., 2010), el cual podemos observar qué sigue el mismo procedimiento que el ejemplo que se ha explicado anteriormente.

Además, hay que destacar que *Ant Colony Optimization* es una técnica utilizada para resolver una amplia gama de problemas difíciles. Ha sido aplicada con éxito en problemas de enrutamiento, asignación, programación y otros, incluso en campos como el aprendizaje automático y la bioinformática. A menudo, los mejores resultados se obtienen cuando se combina con una búsqueda local. Asimismo, *Ant Colony Optimization* ha demostrado ser eficaz en redes de telecomunicaciones y en aplicaciones industriales, como la optimización de rutas de vehículos y la gestión de la distribución de recursos.

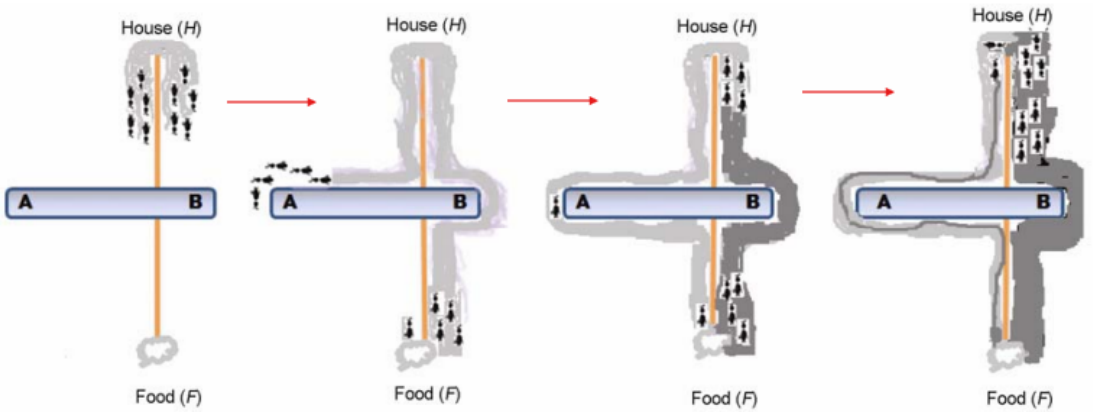


Figura 2.4: Algoritmo de *Ant Colony Optimization* (Janga Reddy and Kumar, 2012).

Actualmente, una parte importante de la investigación sobre *Ant Colony Optimization* se enfoca en aplicaciones prácticas, pero hay un creciente interés en abordar problemas aún más desafiantes, como aquellos con múltiples objetivos, cambios dinámicos en los datos y naturaleza estocástica en las funciones objetivo y restricciones. Además, se están desarrollando extensiones para aplicar los algoritmos de ACO en problemas de optimización continua y se están explorando implementaciones paralelas (Dorigo et al., 2006).

Para concluir, en el algoritmo 2.5 se ha decidido mostrar cómo es el pseudocódigo de un algoritmo de *Ant Colony Optimization* (Janga Reddy and Kumar, 2012).

Algoritmo 2.5: Pseudocódigo de *Ant Colony Optimization* (ACO)

```
Inicializar población
Evaluar la aptitud de la población
mientras criterio de parada no satisfecho hacer
    Posicionar cada hormiga en un nodo de inicio
    repetir
        para cada hormiga hacer
            Elegir el siguiente nodo aplicando la regla de transición de estado
            Aplicar actualización de feromona paso a paso
    hasta cada hormiga haya construido una solución
    Evaluar la aptitud de la población
    Actualizar la mejor solución
    Aplicar actualización de feromona fuera de línea
```

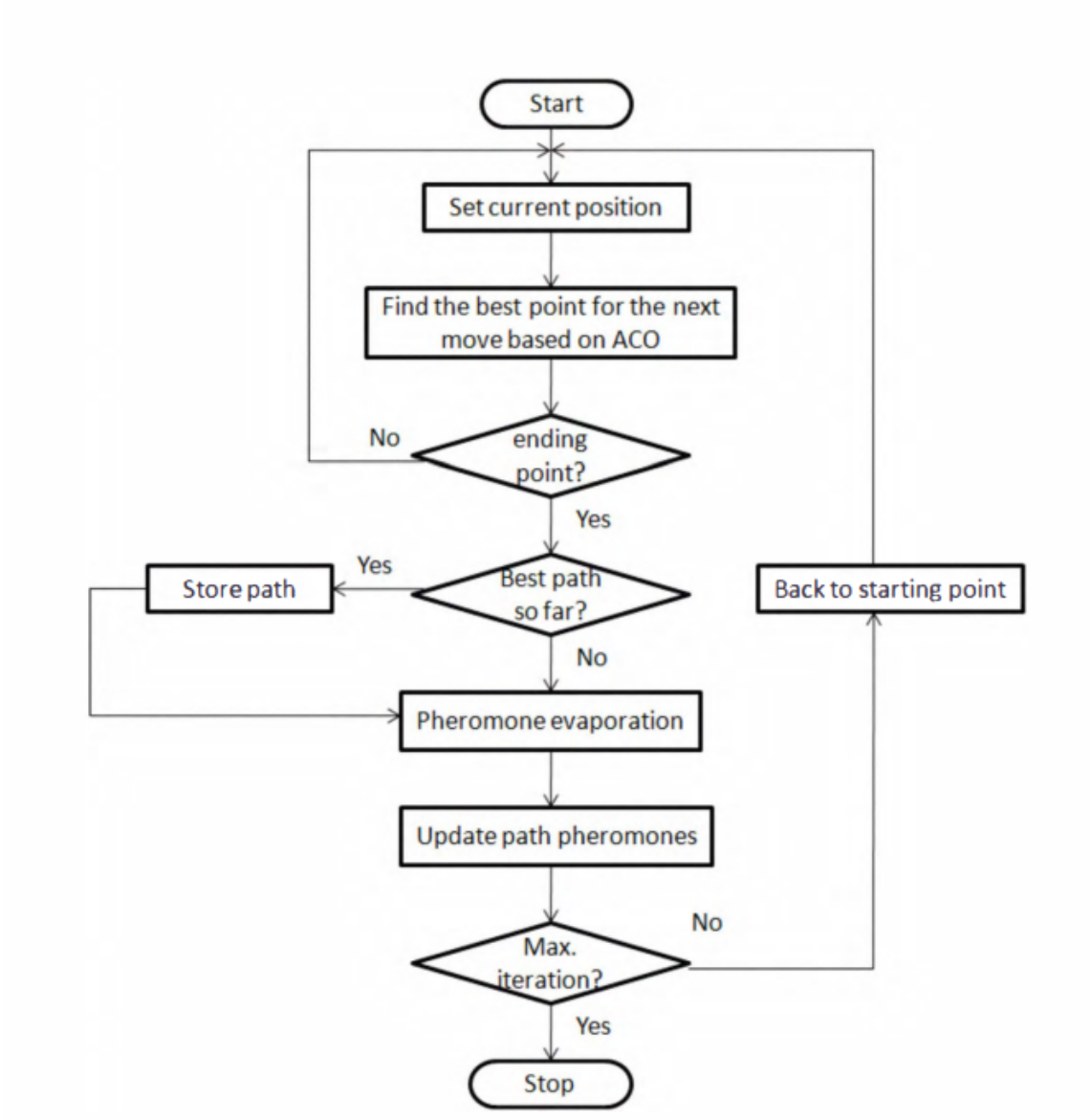


Figura 2.5: Diagrama de flujo de *Ant Colony Optimization* (Brand et al., 2010).

2.4.3. *Particle Swarm Optimization* (PSO)

La *Particle Swarm Optimization* (PSO) es un algoritmo de optimización estocástica basado en poblaciones, motivado por el comportamiento colectivo inteligente de algunos animales como bandadas de aves o bancos de peces (Wang et al., 2018). Al igual que el anterior algoritmo, *Ant Colony Optimization*, no es un algoritmo evolutivo tradicional, pero comparte algunos elementos que podrían asociarse con el *Evolutive Computing*, como el modo de enjambre que le permite buscar simultáneamente en una gran región del espacio de soluciones de la función objetivo optimizada.

Al estudiar el comportamiento de los animales sociales con la teoría de la vida artificial, para cómo construir sistemas de vida artificial de enjambre con comportamiento cooperativo mediante computadora, Millonas propuso cinco principios básicos (Millonas, 1993):

1. **Proximidad**: El enjambre debe ser capaz de realizar cálculos simples tanto en el espacio como en el tiempo.

2. **Calidad**: El enjambre debe tener la capacidad de detectar cambios en la calidad del entorno y reaccionar ante ellos.

3. **Respuesta diversa**: El enjambre no debe restringir su búsqueda de recursos a un ámbito limitado.

4. **Estabilidad**: El enjambre debe mantener un comportamiento consistente incluso ante cambios en el entorno.

5. **Adaptabilidad**: El enjambre debe ajustar su comportamiento cuando el cambio lo justifique.

En la figura 2.6 podemos observar el diagrama de flujo para el algoritmo de *Particle Swarm Optimization*, el cual podemos observar que empieza en la inicialización del enjambre, después sigue con la evaluación de la aptitud de las partículas, seguidamente calculará las posiciones óptimas históricas individuales y del enjambre, sigue con la actualización de la velocidad y posición de las partículas según la ecuación de actualización de velocidad y posición y, finalmente, se pregunta si se cumple la condición de finalización, si se cumple termina y si no, se vuelve al segundo paso.

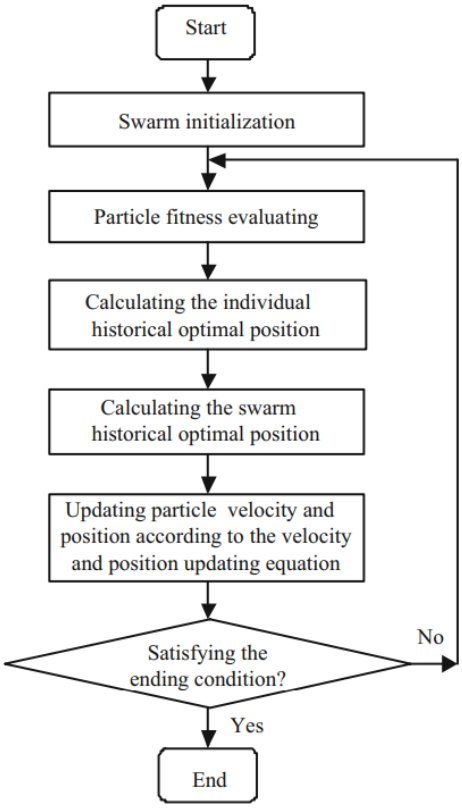


Figura 2.6: Diagrama de flujo de *Particle Swarm Optimization* (Wang et al., 2018).

A continuación, se van a definir algunas de las ecuaciones más importantes durante el proceso de un algoritmo de *Particle Swarm Optimization* (Gad, 2022).

La primera ecuación que se va a definir es aquella que se refiere a la actualización de la mejor posición individual de cada partícula, la cual representa la mejor solución encontrada por una partícula individual hasta el momento actual en el espacio de búsqueda del problema.

Las ecuaciones correspondientes son:

$$p^i_{best_i} = x^*_i | f(x^*_i) = \min_{k=1,2,(\ldots,t} \{f(x^k_i)\}) \tag{2.4}$$

donde $i \in \{1, 2, \ldots, N\}$, y

$$g^i_{best} = x^t_* | f(*^t_i) = \min_{\substack{i=1,2,\ldots,N \\ i=1,2,\ldots,K}} \{f(x^k_i)\}) \tag{2.5}$$

donde i denota el índice de la partícula, t es el número de la iteración actual, f es la función

objetivo a optimizar, x es el vector de posición, y N es el número total de partículas en el enjambre. Las siguientes ecuaciones actualizan, en cada iteración actual $t + 1$, la velocidad v y la posición x de cada partícula i como:

$$v_i^{t+1} = \omega v_i^t + c_1 r_1 (p_{best_i}^t - x_i^t) + c_2 r_2 (g_{best}^t - x_i^t), \tag{2.6}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1}, \tag{2.7}$$

donde v representa el vector de velocidad, ω es el peso de inercia utilizado para equilibrar la explotación local y la exploración global,r_1 y r_2 son vectores aleatorios distribuidos uniformemente dentro del rango [0, 1] en una dimensión D (siendo D la dimensionalidad del espacio de búsqueda o el tamaño del problema en cuestión), y c1 y c2, llamados ”coeficientes de aceleración”, son constantes positivas.

A continuación, en el código 2.6, se va a mostrar el pseudocódigo del algoritmo de *Particle Swarm Optimization*, en donde van a ser muy importantes, en ciertos pasos, las ecuaciones que se han descrito anteriormente.

Algoritmo 2.6: Pseudocódigo del *Particle Swarm Optimization* (PSO)

Require: N - Tamaño del enjambre, D - Dimensionalidad del problema, T - Número máximo de iteraciones, LB - Límite inferior del espacio de búsqueda, UB - Límite superior del espacio de búsqueda
Ensure: g^t_{best} - la mejor posición (solución) encontrada hasta ahora

Inicio
Inicializar el enjambre aleatoriamente;
para $i = 1$ hasta N **hacer** ▷ Iterar a través del enjambre
    $v_i^0 \leftarrow$ un vector aleatorio dentro de $[LB, UB]^D$; ▷ Inicializar la velocidad de las partículas usando una distribución uniforme
    $x_i^0 \leftarrow$ un vector aleatorio dentro de $[LB, UB]^D$; ▷ Inicializar las posiciones de las partículas usando una distribución uniforme
    $p^0_{best_i} \leftarrow x_{0,i}$; ▷ Inicializar p_{best} a su posición inicial
Aplicar la ecuación 2.5 para encontrar g^0_{best}; ▷ Inicializar g_{best} a la posición con el valor de aptitud mínimo
$t \leftarrow 1$; ▷ Inicializar el número de iteración
mientras $t \leq T$ **hacer** ▷ Se alcanza el número máximo de iteraciones o se cumple el criterio de terminación
    para $i = 1$ hasta N **hacer** ▷ Iterar a través del enjambre
        $r_1, r_2 \leftarrow$ dos vectores independientes aleatoriamente generados de $[0, 1]^D$
        Aplicar la ecuación 2.6; ▷ Actualizar la velocidad de la partícula
        Aplicar la ecuación 2.7; ▷ Actualizar la posición de la partícula
        si $f(x_i^t) < f(p^{t-1}_{best_i})$ **entonces** ▷ Si la nueva solución es mejor que la mejor personal actual
            $f(p^t_{best_i}) \leftarrow f(x_i^t)$; ▷ Actualizar la mejor posición conocida de la partícula
    Aplicar la ecuación 2.5 para encontrar g^t_{best}; ▷ Actualizar la mejor posición conocida globalmente del enjambre
    $t \leftarrow t + 1$;
Fin

En resumen, el algoritmo de *Particle Swarm Optimization* (PSO) comienza inicializando una población inicial de partículas distribuidas aleatoriamente, cada una con una posición y velocidad iniciales, después se evalúa su aptitud, seguidamente, se actualiza la mejor posición personal encontrada por cada partícula y la posición global encontrada por todas las partículas, posteriormente se actualiza la velocidad y posición y, se repiten los pasos anteriores hasta cumplir un criterio de terminación.

Para concluir, hay que mencionar en qué campos se puede aplicar este tipo de algoritmos, ya que, se puede aplicar a proyectos donde se traten problemas de planificación de rutas (Qin et al., 2004), también, a proyectos relacionados con la medicina (Pervaiz et al., 2021) y, a proyectos pertenecientes al campo de la logística y el transporte (Qi, 2011), entre otras muchas posibles aplicaciones.

2.5. Pros y Contras

A lo largo de esta sección se van a detallar los pros y contras correspondientes a los algoritmos que se han explicado anteriormente en este capítulo.

1. ***Q-Learning***

    - **Pros**
        - Fáciles de implementar: El *Q-Learning* es uno de los algoritmos de aprendizaje por refuerzo más básicos y es relativamente fácil de entender e implementar.
        - Convergencia garantizada: Bajo ciertas condiciones, el *Q-Learning* garantiza la convergencia a una política óptima.
        - Aplicabilidad en múltiples dominios: Puede ser aplicado en una variedad de problemas de toma de decisiones secuenciales.
    - **Contras**
        - Limitado por el tamaño del espacio de estados: El *Q-Learning* puede ser ineficiente o incluso impracticable en problemas con un espacio de estados muy grande debido a la tabla Q.
        - No considera la generalización: Cada estado es tratado de manera individual sin considerar similitudes entre estados.

2. ***Deep Q-Learning***

    - **Pros**
        - Manejo de grandes espacios de estado: Utiliza redes neuronales para aproximar la función Q, permitiendo trabajar con espacios de estado grandes y continuos.
        - Capacidad de generalización: Puede generalizar a estados no vistos previamente gracias a la naturaleza de las redes neuronales.
        - Aprendizaje eficiente: Utiliza técnicas para mejorar la estabilidad y eficiencia del aprendizaje.
    - **Contras**
        - Complejidad computacional: Requiere una gran cantidad de recursos computacionales y tiempo para entrenar.
        - Sensibilidad a hiperparámetros: El rendimiento puede ser muy sensible a la elección de hiperparámetros y arquitectura de la red.

3. **SARSA**

   - **Pros**
     - Convergencia más segura en entornos ruidosos: Puede ser más estable en entornos con ruido debido a su naturaleza *on-policy*.
     - Simple implementación: Comparte muchas similitudes con Q-Learning, manteniendo la simplicidad de implementación.
   - **Contras**
     - Convergencia más lenta: Tiende a converger más lentamente comparado con *Q-Learning* en ciertos problemas.
     - Dependencia de la política: El desempeño depende en gran medida de la política seguida durante el aprendizaje.

4. ***Ant Colony Optimization (ACO)***

   - **Pros**
     - Capacidad de encontrar soluciones óptimas: ACO es efectivo en la búsqueda de soluciones óptimas en problemas de optimización combinatoria.
     - Adaptabilidad: Puede adaptarse y mejorar con el tiempo gracias a la actualización continua de feromonas.
     - Aplicabilidad en problemas complejos: Es particularmente útil en problemas de optimización de rutas.
   - **Contras**
     - Requiere ajuste de parámetros: El rendimiento de ACO puede ser sensible a la configuración de parámetros como la tasa de evaporación de feromonas.

5. **Algoritmos Genéticos (GA)**

   - **Pros**
     - Exploración global del espacio de búsqueda: Los GAs son efectivos para explorar ampliamente el espacio de búsqueda y evitar óptimos locales.
     - Flexibilidad: Pueden ser aplicados a una amplia gama de problemas de optimización y búsqueda.
     - Paralelización: Fácilmente paralelizables, lo que puede acelerar el proceso de búsqueda.
   - **Contras**
     - Convergencia lenta: Pueden requerir muchas generaciones para converger a una solución óptima.
     - Selección de operadores: El rendimiento depende en gran medida de la selección de operadores genéticos como cruce y mutación.

6. **Particle Swarm Optimization (PSO)**

   - **Pros**
     - Simplicidad y facilidad de implementación: PSO es relativamente fácil de entender e implementar.
     - Rápida convergencia: Puede converger rápidamente a soluciones aceptables en muchos problemas.
     - Menos parámetros a ajustar: Tiene menos parámetros que otros algoritmos evolutivos, lo que facilita su ajuste.
   - **Contras**
     - Problemas de optimización multimodal: PSO puede quedar atrapado en óptimos locales, especialmente en problemas con múltiples picos.
     - Sensibilidad a la parametrización: Aunque tiene menos parámetros, el rendimiento sigue siendo sensible a la configuración de los mismos.

3. Experimentación y metodología

Primeramente, se va a comenzar mencionando que se ha realizado un trabajo aplicado, en donde se han desarrollado modelos relacionados con el *Reinforcement Learning* y el *Evolutive Computing* para, posteriormente, poder compararlos mediante distintas métricas. Además, en este capítulo, también se van a mencionar, a modo de investigación, otros modelos y algoritmos que se han aplicado durante los últimos años, para así, poder hacernos una idea de los resultados que se están obteniendo en los últimos años.

Además, me gustaría mencionar que, la base de nuestros programas desarrollados, ha sido sobre el proyecto de Puente-Castro et al. (2023a), el cual hemos adaptado la mayoría de sus hiperparámetros acorde a nuestras necesidades y, asimismo, hemos modificado parte de su algoritmo, pero hemos mantenido en cierta medida su estructura.

A continuación, se va a tratar de explicar todos los aspectos necesarios para comprender el desarrollo del proyecto de la mejor manera. Primeramente, vamos a ir explicando los aspectos generales como el diseño del entorno, la descripción de los agentes y, así, sucesivamente, hasta llegar a los aspectos más concretos de cada tipo de algoritmo.

3.1. Diseño del Entorno

El entorno de simulación para el control de enjambres de drones se establecerá como mallas bidimensionales con obstáculos. Este diseño del entorno proporciona una representación simplificada, pero realista del espacio en el que operarán los drones, lo que permite la implementación y evaluación de algoritmos de control de manera eficiente y precisa.

Este entorno se modelará como una malla bidimensional, en donde, cada celda de la malla representa una posición en el espacio. Esto permite una representación clara y estructurada del área de operación de los drones, lo que facilita la navegación y el análisis del entorno. Además, se tendrán obstáculos en la malla para simular entornos complejos y realistas. La presencia de obstáculos añade un nivel de complejidad al problema, desafiando a los algoritmos de control a encontrar rutas óptimas y evitar colisiones.

Se ha decidido, probar los modelos correspondientes en mallas de dimensiones 3x3 (Figura 3.1), 5x5 (Figura 3.2) y 7x7 (Figura 3.3), cada una con sus correspondientes obstáculos.

Se ha concluido que el tamaño de dichas mallas no sea muy elevado, por eso el espacio dimensional con más superficie es el de 7x7. Esto es debido a la gran capacidad computacional que se debe tener para operar en superficies de vuelo superiores y, en este caso no se tiene una alta capacidad de procesamiento.

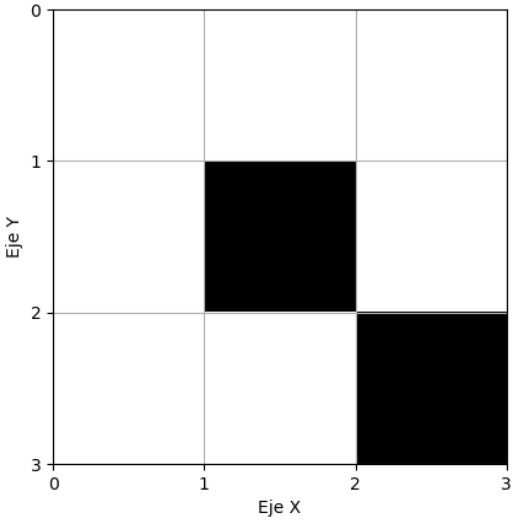


Figura 3.1: Espacio dimensional 3x3.

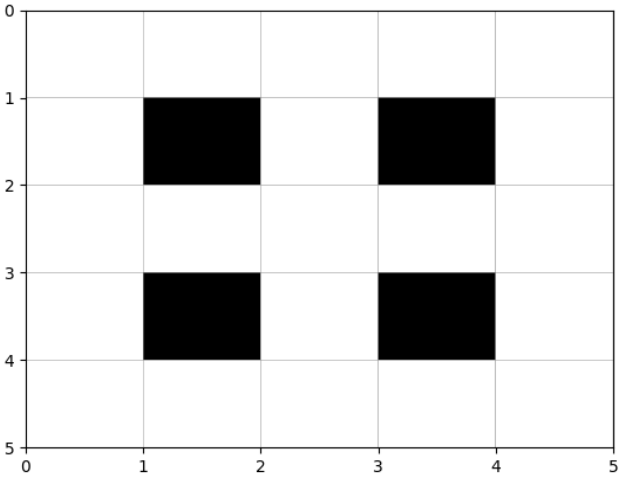


Figura 3.2: Espacio dimensional 5x5.

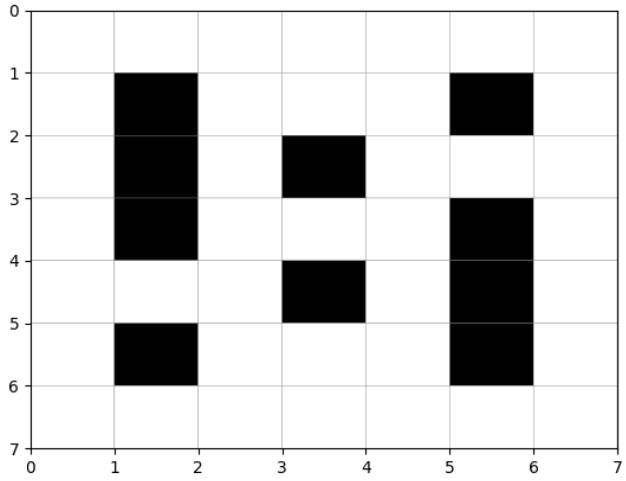


Figura 3.3: Espacio dimensional 7x7.

Se ha decidido utilizar esta distribución de obstáculos para así, poder comparar de manera justa posteriormente con los resultados que obtuvo Puente-Castro et al. (2023a), ya que, en el caso de los entornos 5x5 y 7x7 son exactamente iguales.

Tal y como podemos observar en dichas imágenes, hemos ubicado ciertos obstáculos en ciertos puntos, los cuales siempre estarán en esas posiciones. Dichos obstáculos incrementan la dificultad del problema y desafía la precisión que puedan tener los modelos correspondientes.

3.2. Descripción de los agentes

Los agentes en nuestro problema serán los drones que interaccionarán en el entorno. Estos drones tendrán unas características y atributos similares, las cuales son:

- **Tiempo de batería**. Representa en minutos el tiempo de autonomía del agente.
- **Velocidad**. Se tendrá en cuenta la velocidad a la que el dron puede desplazarse. Generalmente, se asociará un valor de 18 para la variable de velocidad de los drones, tal y como lo hicieron Puente-Castro et al. (2023a), suponiendo así que, que el dron pueda pertenecer a la subcategoría A1 de drones, ya que, estos solo pueden ir a una velocidad máxima de 19 m/s, tal y como se cita en el Reglamento de Ejecución (UE) 2019/947 de la Comisión (The European Commission, 2019).
- **Altura**. Representa la altura máxima a la que el dron puede volar.

Por lo tanto, estas serán las variables más significativas que tendrán que contener nuestros drones, las cuales describen las características y el comportamiento del agente en el entorno.

Para concluir, es importante mencionar que, en todos los algoritmos se van a probar los 3 espacios dimensionales que se describieron anteriormente con 1 y 2 drones y, en el caso de los espacios 5x5 y 7x7, se probarán también configuraciones con 3 y 5 drones para ambos casos. Sin embargo, no todos los drones comenzarán en la misma posición. Los modelos que tengan solamente 1 o 2 drones interaccionando con el entorno comenzarán en la posición (0,0), es decir, en la esquina izquierda, tal y como podemos observar en la figura 3.4.

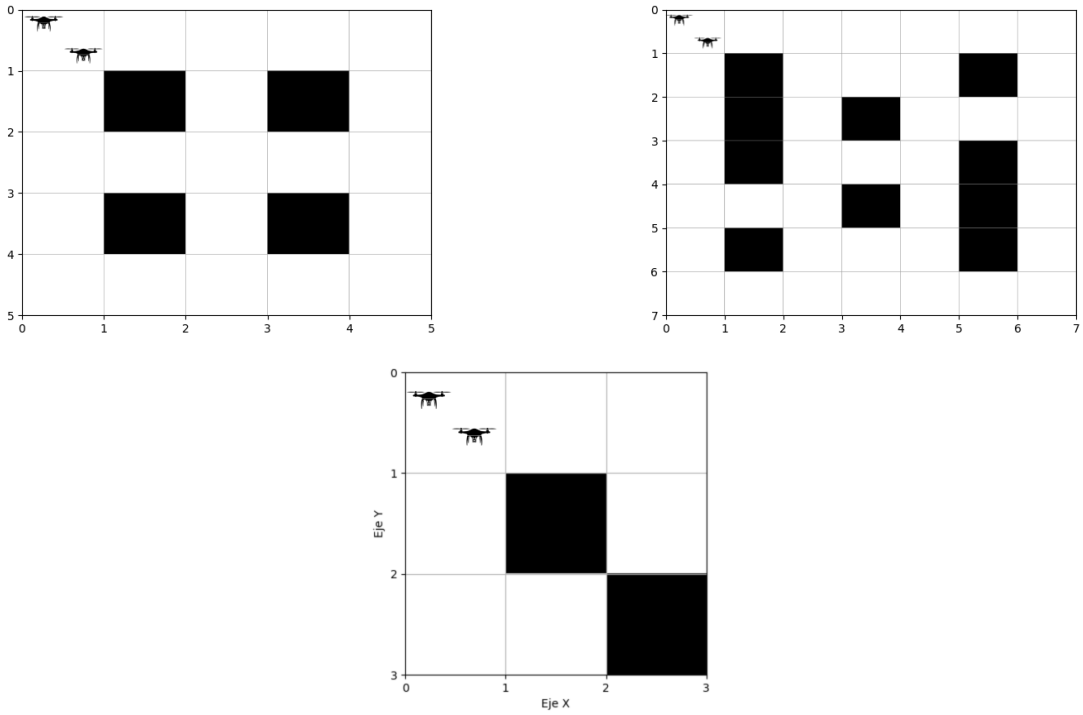


Figura 3.4: Posiciones de agentes en situaciones con 2 drones.

Seguidamente, en la figura 3.5 podemos observar la situación para 3 drones, en donde, comenzarán 2 drones en la esquina superior izquierda y 1 dron en la esquina inferior derecha.

Finalmente, en la figura 3.6 podemos observar la situación para 5 drones, en donde, comenzarán 2 drones en la esquina superior izquierda, 2 drones en la esquina inferior derecha y 1 dron en la esquina superior derecha.

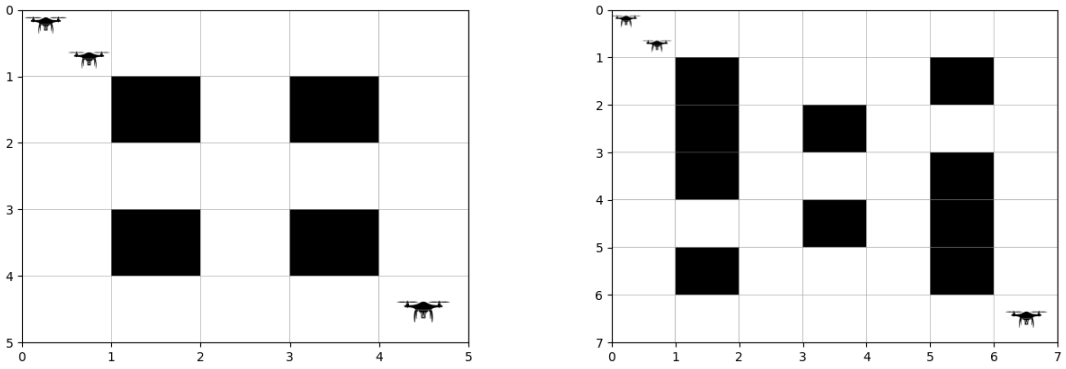


Figura 3.5: Posiciones de agentes en situaciones con 3 drones.

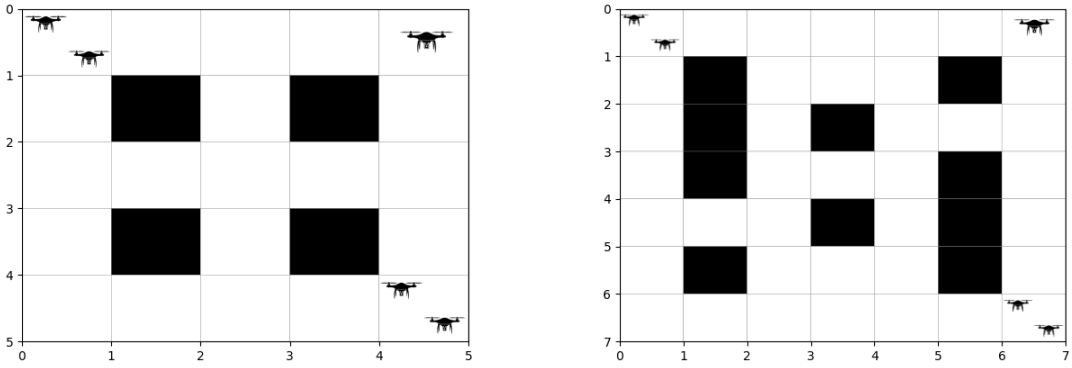


Figura 3.6: Posiciones de agentes en situaciones con 5 drones.

Además, hay que destacar, que la distribución de estos drones es debida al correcto funcionamiento de los drones posteriormente, ya que, en el algoritmo de *Ant Colony Optimization* se enfocará en buscar celdas que no se hayan visitado y, por lo tanto, está distribución era la más adecuada. Además, se pudo observar que en los demás algoritmos también funcionaban de mejor manera.

A continuación, en la tabla 3.1, se van a representar los ensayos que se van a realizar posteriormente. Se puede observar que probaremos los entornos dimensionales 5x5 y 7x7 con 1, 2, 3 y 5 drones y, que el entorno 3x3 solo con 1 y 2 drones debido a que este último es un espacio dimensional más pequeño que los otros dos.

Se puede observar que se podrá comparar el rendimiento obtenido con un solo dron en comparación con 2, 3 o incluso 5 drones.

Además, se probarán para todas esas configuraciones tres algoritmos: *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*. Asimismo, para todas las configuraciones se ejecutarán 10 episodios y, solo en algunas de ellas 100 episodios, como son en los entornos 5x5 y 7x7 con 2 y 3 drones. Esto último se hará para poder comparar posteriormente los resultados con los que obtuvo Puente-Castro et al. (2023a) en su trabajo.

Espacio dimensional	Número de drones	Número de Episodios	Algoritmos
3x3	1	10	*Deep Q-Learning*, *Ant Colony Optimization*, Algoritmo Genético
	2		
5x5	1	10	
	2	10 y 100	
	3	10 y 100	
	5	10	
7x7	1	10	
	2	10 y 100	
	3	10 y 100	
	5	10	

Tabla 3.1: Ensayos.

3.3. Acciones de los agentes

En esta sección se van a definir los diferentes movimientos que el agente puede tomar para desplazarse por el entorno correspondiente.

El agente solo podrá tomar cuatro acciones posibles: arriba, abajo, izquierda y derecha. Además, el agente solo podrá tomar esas acciones siempre y cuando sean acciones válidas, es decir, siempre y cuando, no se salga del entorno ni impacte con alguno de los obstáculos, lo que se garantizará que el agente tome decisiones seguras y viables sobre su movimiento dentro del entorno.

Asimismo, es de alta importancia mencionar que, mediante el uso de enjambres de drones, se coordina simultáneamente numerosos agentes, de tal manera, que no llegan a colisionar

entre ellos y se optimiza el proceso de exploración, tal y como se puede observar en la imagen 3.7.

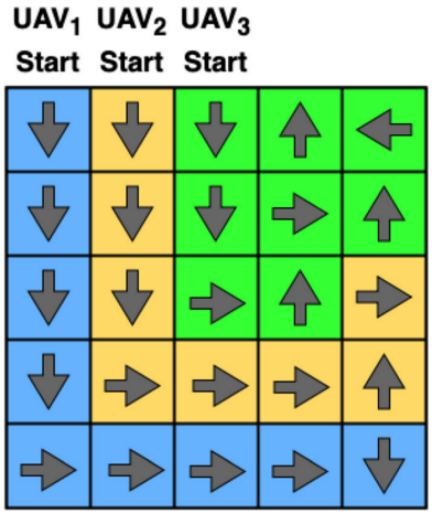


Figura 3.7: Ruta para 3 UAVs (Puente-Castro et al., 2022b).

3.4. Creación de los modelos

Una vez se haya concluido con los aspectos más generales de nuestro problema, se tiene que continuar explicando qué algoritmos se han implementado y por qué. Este capítulo se va a orientar solamente a la descripción de los correspondientes algoritmos y, en la siguiente sección, se expondrán los resultados obtenidos.

Los algoritmos que se han desarrollado, ejecutado y evaluado en nuestro entorno han sido los tres siguientes: *Deep Reinforcement Learning*, *Ant Colony Optimization* y un algoritmo genético. Además, concluiremos esta sección, a modo de investigación, comentando los recientes proyectos que se han desarrollado, tratando de enfocarnos tanto en los resultados que se han obtenido como en la propia implementación de dichos algoritmos.

Por lo tanto, a continuación, se va a comenzar tratando de explicar el algoritmo de *Deep Reinforcement Learning*.

3.4.1. *Deep Reinforcement Learning*

A lo largo de este capítulo se va a tratar de explicar en qué consiste el funcionamiento de este algoritmo y cuáles son las configuraciones que se han probado. Antes de comenzar, es crucial mencionar que la base de este proyecto ha sido el repositorio implementado por Puente-Castro et al. (2023a), en donde se ha creado un modelo de *Deep Reinforcement Learning* para problemas de planificación de rutas con enjambres de drones en entornos con obstáculos (Puente-Castro et al., 2023b). Aun así, no se ha implementado ni se han probado todas las funcionalidades de dicho repositorio, sino que se ha adaptado a nuestras necesidades, utilizando ciertas funcionalidades y modificando la configuración de los episodios en función de nuestros objetivos. Por lo tanto, no se van a describir todos los aspectos del repositorio, solamente se explicará lo que se ha implementado y, a su vez, se detallará que modificaciones hemos implementado.

Tal y como se ha visto anteriormente, *Deep Reinforcement Learning* es un algoritmo de aprendizaje por refuerzo que utiliza redes neuronales para aprender políticas de toma de decisiones en entornos complejos y de gran dimensionalidad, el cual consiste en entrenar un agente para que aprenda a interactuar con un ambiente desconocido y maximice una recompensa numérica a lo largo del tiempo. Como resultado, cada experimento de Q-Learning sigue los siguientes pasos (Puente-Castro et al., 2023a):

1. Se construye los modelos redes neuronales utilizando la configuración seleccionada.
2. Se determina los valores de la tabla Q y el curso óptimo de acción para cada UAV en el enjambre utilizando los modelos de redes neuronales
3. Se entrenan los modelos basados en los resultados de cada acción elegida.
4. Se seleccionan los casos donde el número de movimientos requeridos para explorar todo el mapa sea menor.

En la figura 3.9 se puede observar un diagrama de flujo, el cual define de manera detallada y secuencial cómo funciona un episodio en el algoritmo propuesto.

El modelo que se ha propuesto en el repositorio de Puente-Castro et al. (2023b) fue desarrollado mediante una búsqueda aleatoria de hiperparámetros, para así, conseguir el mejor modelo posible. La arquitectura que se propuso está compuesta por dos capas densas, una con 1013 neuronas y una función de activación *ReLU*, y la otra con 4 neuronas y una función de salida *softmax*. Además, se configuró el modelo para el entrenamiento con optimizador de Descenso de Gradiente Estocástico y una función de pérdida del Error Cuadrático Medio. La red neuronal utiliza datos del entorno existentes sin necesidad de información adicional. Las salidas son los valores Q para acciones en un estado, ajustados usando la función softmax. Hay cuatro valores Q distintos para cada dirección de movimiento: Norte, Este, Sur y Oeste.

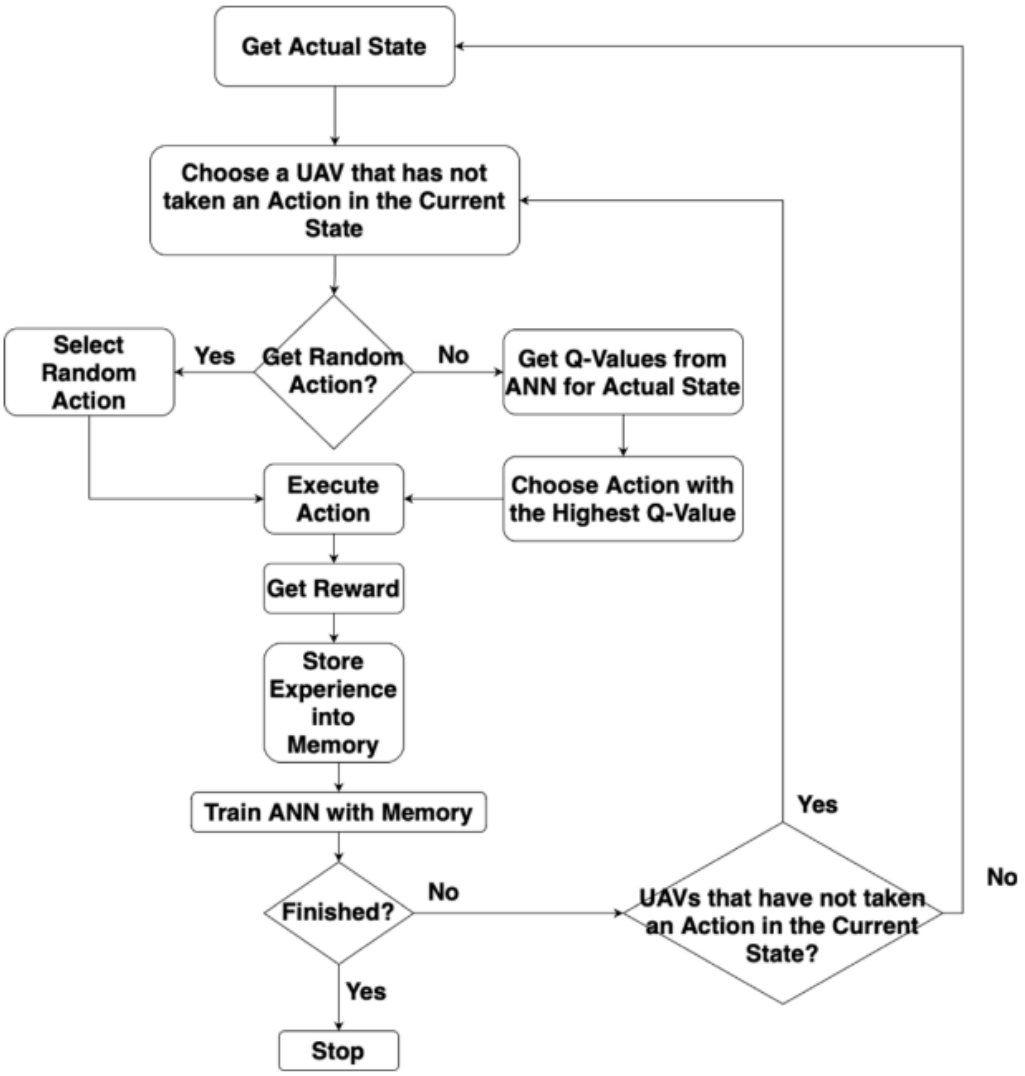


Figura 3.8: Diagrama de flujo para algoritmo de *Deep Reinforcement Learning* (Puente-Castro et al., 2023a).

Mediante la figura 3.9, proporcionada por Puente-Castro et al. (2023a), se puede observar de manera visual la arquitectura de la red neuronal y cómo interactúa con el entorno.

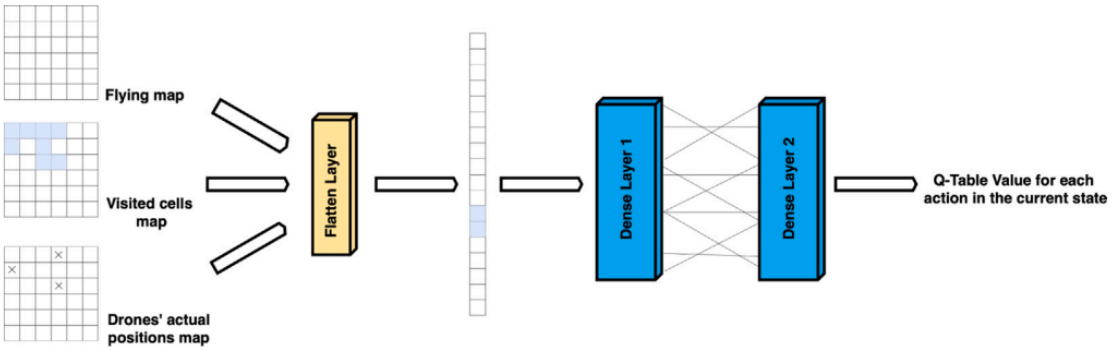


Figura 3.9: Diagrama de la red neuronal propuesta (Puente-Castro et al., 2023a).

Otra de las variables a destacar, es épsilon (ϵ), cuyo parámetro se encarga de decidir entre si explorar o explotar, es decir, si probar acciones desconocidas para descubrir su efecto en el entorno y aprender más sobre el mismo o tomar las acciones que parecen ser las mejores según el conocimiento actual para maximizar las recompensas a corto plazo. A dicha variable se le dio el valor de 0.468, por lo que, con una probabilidad del 46.8% se comenzará probando acciones aleatorias y con un 51% se comenzará eligiendo la mejor acción conocida. Además, el factor de descuento (γ) será de 0.90.

Otra cuestión muy importante a tratar es la entrega de recompensas dependiendo de si decide ir a un lugar u otro. Si se descubre una celda se le otorga una recompensa de 358.73, mientras que si se acaba en una celda ya visitada se le penaliza con una recompensa de -31.13, por lo que se le incita a los agentes a explorar nuevas áreas en lugar de revisitar las conocidas. Asimismo, se le penalizará con una recompensa de -225.17 en el caso de que se explore una celda prohibida. Por lo que, se puede observar que se penaliza de manera significativa el hecho de explorar una celda prohibida.

Igualmente, se tienen ciertos criterios de parada para la detención temprana de experimentos, los cuales son los siguientes:

- Un **tiempo máximo de ejecución**, mediante el cual podremos definir el tiempo máximo que se permitirá que un experimento se ejecute antes de detenerlo. Este parámetro está inicializado en 30 minutos.
- Un parámetro para conocer la **cobertura de completitud deseada**, la cual está definida como 1, lo que quiere decir que cuando se haya alcanzado 100% de la completitud se detendrá.

- Un **número máximo de episodios consecutivos en los que el entorno no ha cambiado**, mediante el cual se pretende que si el entorno no ha cambiado durante este número de episodios, se considera que el agente ha convergido y el experimento se detiene. En nuestro problema le daremos un valor de 99999, por lo que, no lo tendremos en cuenta, ya que, hemos definido con un valor de 10 el número de episodios totales, pero es importante conocerlo para futuros experimentos.

De igual manera, hay que destacar la presencia de una técnica utilizada de reproducción de memoria, el cual consiste en que el entorno se somete a entrenamiento utilizando un conjunto almacenado de observaciones pasadas y, dichas observaciones abarcan una variedad de información, incluyendo las acciones tomadas por el agente y las recompensas correspondientes recibidas. Por lo que, se aprovecha las experiencias pasadas para enriquecer el proceso de aprendizaje, ayudando a los agentes a comprender y adaptarse mejor a su entorno.

3.4.2. *Ant Colony Optimization*

En todo el transcurso de este capítulo, se van a tratar los principales aspectos y características del modelo que se ha realizado utilizando un algoritmo de optimización de colonia de hormigas.

Tal y como se describió anteriormente, el algoritmo *Ant Colony Optimization* se basa en el comportamiento colectivo de las hormigas al buscar caminos óptimos entre una fuente de alimentos y su colonia. Las hormigas depositan feromonas en el suelo mientras exploran, y estas feromonas atraen a otras hormigas a seguir el mismo camino. Con el tiempo, las rutas con más feromonas se refuerzan, mientras que las rutas menos favorables tienden a desaparecer.

Los pasos que se siguen durante un episodio de este modelo son muy similares a los que se describieron en el algoritmo 2.5, sin embargo, se han realizado algunas modificaciones para que se adapte correctamente a nuestro problema y obtengamos los mejores resultados posibles. Para empezar, hay que destacar que el modelo seguirá el algoritmo visto, pero adaptado a un problema con múltiples UAVs, entonces ahora ya no saldrá solo una hormiga, sino que saldrán varias, en nuestro caso 2, 3 o 5 a la vez. Además, se priorizará el hecho de que la hormiga elija una celda por la que no hayan pasado las demás. Así se explorarán nuevas áreas del espacio de búsqueda que aún no han sido exploradas y, se evitarán ciclos en donde los agentes se puedan quedar atrapados.

Asimismo, en el algoritmo clásico de *Ant Colony Optimization* las hormigas, una vez llegaban a su destino, estas se volvían al lugar de inicio, pero en nuestro problema se ha definido que el algoritmo se detenga en cuanto se haya explorado todo el entorno. Esto es debido a que este tipo de algoritmos de optimización están diseñados para ir de un punto de inicio a un destino en el menor tiempo o el menor número de pasos posibles, sin embargo, en nuestro

problema nuestro objetivo es recorrer todo el espacio dimensional en el menor número de pasos posibles. Por lo tanto, en cuanto hayan explorado todo el entorno, automáticamente se detendrá el algoritmo y comenzará un nuevo episodio.

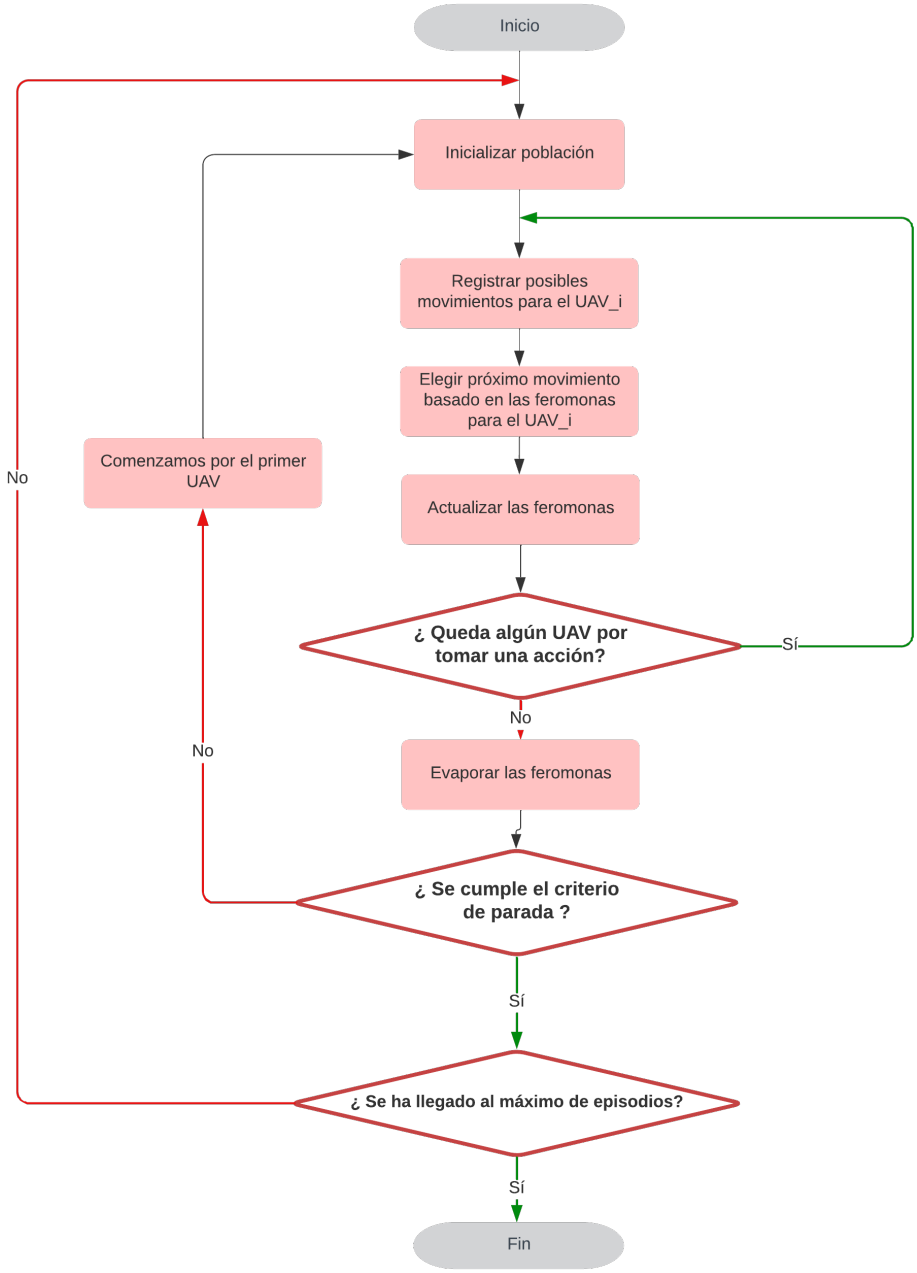


Figura 3.10: Diagrama de flujo del algoritmo *Ant Colony Optimization* adaptado.

En la figura 3.10 se ha podido observar el diagrama de flujo que representa los pasos que se han seguido en el modelo que se ha desarrollado.

Además, hay que destacar que en la etapa de elegir el próximo movimiento basado en las feromonas se ha introducido una estrategia específica en el primer 25% de los episodios, la cual es la elección de las nuevas posiciones basándose en las probabilidades asignadas a los posibles movimientos con el objetivo de fomentar la exploración durante las primeras etapas del proceso de optimización. Esta estrategia permite que el algoritmo evite la convergencia prematura hacia soluciones subóptimas y mejore la calidad de las rutas encontradas en etapas posteriores del proceso. En el 75% de los episodios restantes se ha decidido basarse puramente en las feromonas en donde se fomenta más la explotación, ya que, anteriormente se ha decidido darle más importancia a la exploración y así intentar encontrar soluciones óptimas.

Asimismo, se ha diseñado una estrategia para gestionar las feromonas de manera que se penalicen los episodios que no encuentren el óptimo durante las primeras iteraciones y se refuerce la búsqueda de soluciones óptimas en etapas posteriores. Al final de cada episodio, se realiza una evaluación basada en el rendimiento comparativo con el mínimo número de acciones registrado. Si el número total de acciones es mayor o igual al mínimo registrado, las feromonas en las celdas visitadas se reducen, penalizando así las rutas menos eficientes. En contraste, si el número de acciones es menor o igual a óptimo, las feromonas en las celdas visitadas se incrementan significativamente, incentivando las rutas más prometedoras. Este enfoque balanceado permite una exploración eficiente al inicio y una explotación más enfocada en soluciones óptimas en etapas posteriores.

Para concluir, hay que mencionar que los criterios de parada para este algoritmo son los mismos que los del algoritmo anterior y que, esta vez se ha decidido priorizar la visita de las celdas que no se haya explorado para realizar una exploración exhaustiva y prevenir sobre posibles ciclos y, además, para posteriormente poder comparar con otros algoritmos donde se ha tenido contemplado el tomar acciones aleatoriamente.

3.4.3. Algoritmo Genético

A lo largo de este episodio se va a proceder a explicar en qué consiste el algoritmo genético que se ha implementado y cuáles son las características más destacables de este.

Tal como se detalló anteriormente, los algoritmos genéticos son aquellos que son un tipo de algoritmo de optimización y búsqueda inspirado en el proceso de evolución natural. Estos algoritmos se utilizan para encontrar soluciones aproximadas a problemas de optimización, donde se busca encontrar la mejor solución posible dentro de un espacio de búsqueda amplio y complejo.

El proceso que se ha seguido ha sido muy parecido al de un algoritmo genético tradicional, pero también se han implementado algunos cambios para que se adaptase correctamente al problema de múltiples drones.

Primeramente, hay que destacar que en la inicialización de los caminos iniciales se va a variar entre explotación y exploración, ya que, mediante esta técnica se puede mejorar significativamente la eficiencia y la efectividad del modelo. Inicialmente, en el pseudocódigo clásico se indicaba que todos los caminos se inicializaban aleatoriamente, pero en nuestro modelo se ha optado por darle importancia a que explore también aquellas celdas que no se ha visitado anteriormente. Al balancear estos dos aspectos, se logra una mejor cobertura del espacio de búsqueda y se reduce la probabilidad de quedar atrapado en óptimos locales, lo que puede llevar a encontrar soluciones de mayor calidad de manera más eficiente.

Las dos situaciones en donde el algoritmo clásico sufre más modificaciones son en el cruzamiento y en la mutación, ya que, al tener caminos y no tener otra estructura, como por ejemplo los cromosomas, no es posible realizar esos dos procedimientos de manera convencional.

Respecto al cruzamiento, es importante destacar que las soluciones serán una lista compuesta de n listas, una por cada camino del dron correspondiente, lo que añade complejidad al modelo. En el artículo publicado por Singh et al. (2020), se consiguió realizar un algoritmo genético de tres padres con aplicación al enrutamiento en redes de mallas inalámbricas y, el cual nos indica que, aunque no es biológicamente posible llegar a obtener un hijo a partir de 3 padres, realizarlo en este tipo de problemas puede ser beneficioso debido a que puede aumentar la diversidad genética de la población, lo que potencialmente puede llevar a una mejor exploración del espacio de búsqueda y, por lo tanto, mejorar la convergencia del algoritmo hacia soluciones óptimas. En nuestro caso, hemos decidido realizar el cruzamiento según el número de drones que haya en dicho modelo, lo que nos ha proporcionado resultados beneficiosos que se verán próximamente. En la imagen 3.11 podemos observar la función de cruzamiento para cuando se tienen 2 padres, de los cuales se ha considerado que se tendrá solamente un hijo y, en la imagen 3.12 se puede visualizar en qué consiste el cruzamiento entre 3 padres. En dichos cruzamientos nos quedamos con un solo hijo y seleccionamos aleatoriamente caminos que se combinarán en una sola población.

En lo que respecta al proceso de mutación, hay que destacar que este es una operación difícil de adaptar y desarrollar en este tipo de problemas, considerando que las poblaciones están compuestas de caminos. Es complicado debido a que en el proceso se intercambian aleatoriamente celdas de un camino de un UAV, por lo que si, por ejemplo, se tiene el camino [(0,0),(0,1),(0,2),...] y se cambia mediante mutación al [(0,0),(0,1),(0,3),...], se estaría suponiendo que el dron pasa de la celda (0,1) a la (0,3), lo cual no es posible. Entonces, se ha decidido que para aprovechar el proceso de mutación, se evalúen y se completen dichos caminos mediante una búsqueda en anchura y, por lo tanto, encontraríamos el camino más corto, obteniendo la siguiente lista para el ejemplo anterior: [(0,0),(0,1),(0,2),(0,3),...]. Cabe destacar que, se ha decidido no prescindir del proceso de mutación debido a que introduce

variabilidad en la población de soluciones, permitiendo explorar nuevas áreas del espacio de búsqueda y ayudando a prevenir la convergencia prematura hacia óptimos locales.

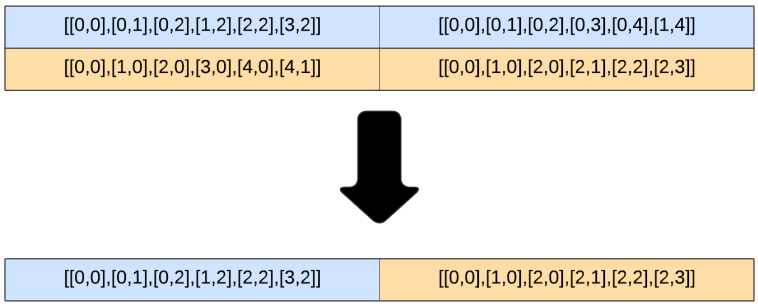


Figura 3.11: Cruzamiento entre dos padres.

En la figura 3.13 se puede observar el proceso de mutación, en donde se cambia aleatoriamente una celda y se corrige la ruta para que sea posible.

Finalmente, mediante las acciones de todos los drones, se consigue recorrer todo el espacio dimensional y, de igual manera que en los algoritmos anteriores, realizando una acción por cada dron en cada iteración.

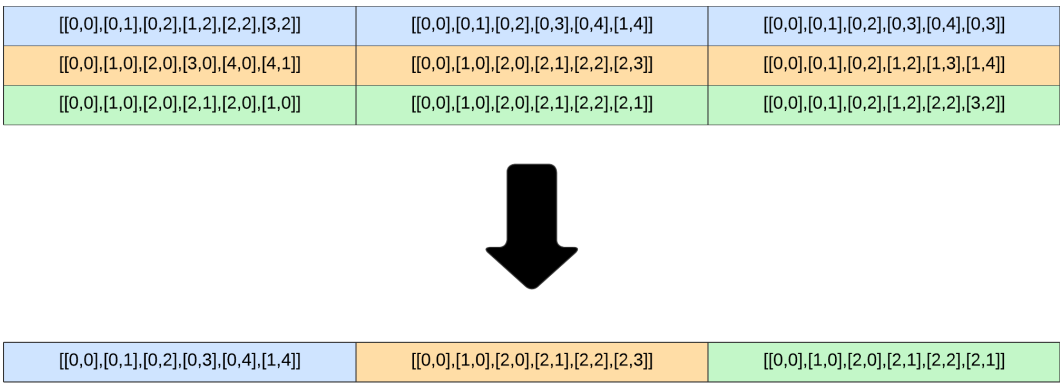


Figura 3.12: Cruzamiento entre tres padres.

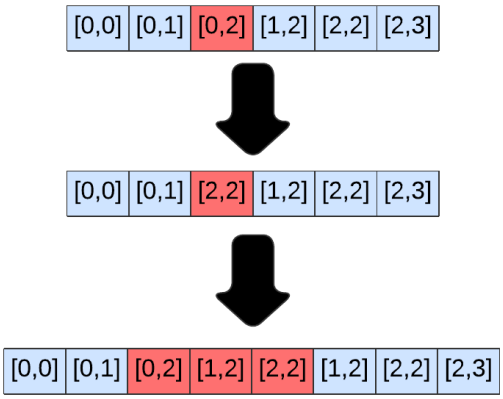


Figura 3.13: Mutación adaptada.

En último término, el criterio de parada de este algoritmo será que se haya explorado toda la malla, es decir, que se hayan explorado todas las celdas con al menos un dron o, también se podrá detener si el algoritmo supera el tiempo máximo de ejecución, tal y como se consideraba en las anteriores técnicas.

4. Resultados

En esta sección, se presentan los resultados obtenidos a partir de la investigación llevada a cabo, centrándose en la explicación de dichos resultados mediante el uso de tablas y figuras. Los resultados que se van a presentar para todos los modelos van a ser las medias con sus respectivas desviaciones estándares. Se van a comparar dichas métricas según las distintas configuraciones, llegando a tratar los modelos con distintos espacios dimensionales y con diversos números de drones.

Primeramente, se va a comenzar mostrando los resultados que se han obtenido respecto a los porcentajes de acciones válidas, los cuales se pueden representados en la tabla 4.1 y, en donde se puede visualizar cómo afecta en gran medida el entorno, el número de drones y el tipo de algoritmo que se aplique.

Se considerarán como acciones válidas aquellas acciones que acaban en una celda que no ha sido explorada previamente.

Los resultados indican que ACO supera consistentemente a los otros dos algoritmos en todas las configuraciones, especialmente en el espacio más pequeño (3x3) con una desviación estándar mínima, reflejando su robustez. Sin embargo, a medida que aumenta el tamaño del espacio y el número de drones, la variabilidad del rendimiento de ACO incrementa. *Deep Q-Learning* y el Algoritmo Genético muestran un rendimiento más variable y generalmente inferior al de ACO. Además, el rendimiento de todos los algoritmos tiende a degradarse con el aumento del tamaño del espacio y el número de drones, destacando la mayor complejidad de gestionar entornos más grandes y coordinaciones de múltiples drones.

Por otra parte, es importante destacar que se han obtenido, en general, mejores resultados mediante el uso de enjambres de drones que mediante el uso de un solo dron. Por lo tanto, se puede decir que el uso de enjambres de drones puede mejorar el porcentaje de acciones válidas al aumentar la cobertura, la redundancia y la capacidad de adaptación del sistema en comparación con un solo dron.

Esto hace que los enjambres de drones sean una opción más efectiva para muchas aplicaciones donde se requiere un alto nivel de precisión y fiabilidad en la ejecución de acciones.

Espacio dimensional	Número de drones	*Deep Q-Learning* ($\mu \pm \sigma$)	Algoritmo genético ($\mu \pm \sigma$)	*Ant Colony Optimization* ($\mu \pm \sigma$)
3x3	1	9.54 ± 5.52	16.55 ± 15.26	58.17 ± 5.5
	2	12.26 ± 6.34	25.88 ± 19.36	100 ± 0
5x5	1	7.50 ± 3.19	4.93 ± 2.70	52.05 ± 8.41
	2	9.99 ± 6.31	20.23 ± 12.39	60.6 ± 8.64
	3	8.89 ± 6.30	18.05 ± 9.66	60.8 ± 8.11
	5	13.99 ± 10.15	17.17 ± 3.02	61.29 ± 4.16
7x7	1	6.78 ± 2.1	14.90 ± 7.20	54.81 ± 13.17
	2	7.82 ± 2.69	20.57 ± 17.25	68.62 ± 8.46
	3	7.26 ± 2.04	20.95 ± 10.04	53.76 ± 14.17
	5	8.89 ± 5.45	18.99 ± 5.84	65.48 ± 11.21

Tabla 4.1: Resultados de los porcentajes de acciones válidas.

A continuación, en la tabla 4.2 se puede visualizar el número de acciones totales para todos los casos propuestos, en donde comparamos, al igual que en la tabla anterior, con distintos espacios dimensionales, número de UAVs y diversos algoritmos. En todos los algoritmos y configuraciones diferentes, se obtiene mejores resultados con enjambres de drones que con un solo dron, lo cual indica cómo con múltiples drones trabajando de manera colaborativa, se pueden abordar tareas de manera más eficiente y efectiva.

En esta tabla, cuando se dice acciones totales, nos estamos refiriendo al número total de acciones entre todos los drones que ha hecho falta para poder recorrer toda la malla.

Para esta métrica podemos observar cómo el algoritmo de *Ant Colony Optimization* obtiene mejores resultados, ya que, consigue una solución en una media menor de número de acciones totales, en comparación con los otros dos algoritmos. En esta tabla, al igual que en la anterior, se tiene que nuestros algoritmos evolutivos adaptados han obtenido mejores resultados en comparación con el algoritmo de aprendizaje por refuerzo. Por ejemplo, en el caso de un entorno 7x7 con 3 drones, se ha obtenido la mitad de acciones en el algoritmo genético en comparación con el de aprendizaje por refuerzo y, a su vez, tres veces menos acciones totales del algoritmo de colonia de hormigas respecto al algoritmo genético.

Espacio dimensional	Número de drones	*Deep Q-Learning* ($\mu \pm \sigma$)	Algoritmo genético ($\mu \pm \sigma$)	*Ant Colony Optimization* ($\mu \pm \sigma$)
3x3	1	93.14 ± 57.39	89.0 ± 69.29	10.4 ± 0.91
	2	92.44 ± 94.55	39.0 ± 24.20	6.0 ± 0.0
5x5	1	343.66 ± 200.28	530.6 ± 245.01	39.5 ± 6.74
	2	301.8 ± 180.15	152.7 ± 116.36	33.75 ± 5.33
	3	326.11 ± 197.88	140.1 ± 73.97	31.87 ± 6.74
	5	240.00 ± 205.56	111.3 ± 36.9	29.50 ± 5.33
7x7	1	615.0 ± 176.36	329.2 ± 156.06	73.2 ± 16.49
	2	538.22 ± 161.42	369.11 ± 271.50	57.6 ± 6.70
	3	542.11 ± 124.48	247.4 ± 184.8	74.2 ± 16.49
	5	518.5 ± 196.87	225.2 ± 135.8	56.2 ± 6.70

Tabla 4.2: Resultados de número acciones totales.

Finalmente, en la tabla 4.3, se han presentado los resultados que se han obtenido respecto a los tiempos de ejecución en las distintas configuraciones que se han probado. Para comenzar, hay que destacar que, los tiempos obtenido en los algoritmos de *Deep Q-Learning* son muy superiores a los obtenidos con los algoritmos genéticos, esto es debido principalmente a la complejidad computacional asociada con el entrenamiento de redes neuronales profundas y la mayor dependencia del procesamiento paralelo para acelerar el proceso de entrenamiento.

En esta tabla, el tiempo de ejecución se mide en segundos y se refiere al tiempo que tarda un episodio en completar su tarea, en este caso la de explorar todas las celdas de la malla correspondiente.

Por otro lado, se tiene un bajo tiempo de ejecución en los modelos con algoritmos evolutivos debido a que no dependen del uso de GPU para su ejecución, lo que puede hacer que sean más rápidos en entornos pequeños en comparación con los algoritmos de *Deep Q-Learning* que sí dependen de GPU.

Espacio dimensional	Número de drones	*Deep Q-Learning* ($\mu \pm \sigma$)	Algoritmo genético ($\mu \pm \sigma$)	*Ant Colony Optimization* ($\mu \pm \sigma$)
3x3	1	213.05 ± 120.90	70.0 ± 54.90	3.25 ± 0.39
	2	333.90 ± 373.59	30.26 ± 19.46	2.47 ± 0.09
5x5	1	1162.8 ± 688.2	377.4 ± 170.4	11.41 ± 1.28
	2	779.5 ± 458.4	119.4 ± 90.6	11.82 ± 0.30
	3	690.0 ± 506.4	83.17 ± 37.99	10.86 ± 0.98
	5	510.0 ± 496.8	83.10 ± 24.13	10.16 ± 0.63
7x7	1	1944.1 ± 616.34	246.58 ± 127.33	26.42 ± 5.62
	2	1430.8 ± 444.45	304.6 ± 214.8	23.68 ± 3.23
	3	1657.5 ± 249.28	181.2 ± 139.8	21.38 ± 2.78
	5	1406.5 ± 392.97	165.49 ± 89.61	20.35 ± 3.46

Tabla 4.3: Resultados de tiempos de ejecuciones en segundos.

Se ha decidido elegir, cuál de los algoritmos es el mejor para nuestro problema. Dicho algoritmo es *Ant Colony Optimization*, ya que, tal y como hemos comprobado anteriormente mediante la representación de los resultados, es el que ha obtenido mejores resultados en menor tiempo, por lo que se consigue optimizar tanto las soluciones obtenidas como el tiempo de ejecución. Si bien los algoritmos de *Deep Q-Learning* y los algoritmos genéticos también pueden ser aplicables a la planificación de rutas en ciertos contextos, las colonias de hormigas destacan por su capacidad para adaptarse a entornos dinámicos, explorar eficientemente el espacio de búsqueda y encontrar soluciones óptimas en problemas de optimización combinatoria y, sobre todo, en un problema como el nuestro en donde el espacio dimensional no es muy grande, *Ant Colony Optimization* es el que mejores resultados ofrece.

Es importante destacar que, generalmente, conforme mayor número de drones tengamos mejores serán nuestros resultados. Sin embargo, hay ciertos casos vistos en las anteriores tablas, en las que esto no ocurre, esto es debido a que se han ejecutado 10 episodios de cada algoritmo, por lo que, en muchos casos es posible que no converjan los algoritmos y que en las primeras etapas obtenga peores resultados aunque tenga mayor número de drones. No obstante, para poder comprobar de mejor manera que conforme utilicemos más drones, mejores resultados obtendremos, vamos a realizar a continuación 100 ejecuciones del algoritmo *Ant Colony Optimization*.

Por lo tanto, debido a que *Ant Colony Optimization* es el algoritmo que tiene mayor número de diferencias significativas, en tiempo y en acciones totales, tiene respecto a los demás

algoritmos, habiendo ejecutado 10 episodios, se va a proceder a ejecutar 100 episodios para ciertas configuraciones. Además, utilizaremos estas configuraciones para, posteriormente, comparar dichos resultados con los que obtuvo Puente-Castro et al. (2023a) mediante técnicas de *Deep Q-Learning* en la realización de su trabajo.

En la tabla 4.4 se pueden observar los resultados que hemos obtenido para las configuraciones planteadas con el algoritmo de *Ant Colony Optimization* con 100 episodios.

Espacio dimensional	**Número de drones**	**Acciones totales** ($\mu \pm \sigma$)	**Acciones válidas** ($\mu \pm \sigma$)	**Tiempo de ejecución** ($\mu \pm \sigma$)
5x5	2	29.76 ± 5.10	0.68 ± 0.10	11.60 ± 1.65
	3	28.35 ± 6.32	0.69 ± 0.11	11.38 ± 2.79
7x7	2	53.55 ± 15.79	0.76 ± 0.20	21.20 ± 6.85
	3	50.06 ± 18.58	0.81 ± 0.20	18.87 ± 6.84

Tabla 4.4: Resultados de *Ant Colony Optimization* con 100 episodios.

En el experimento se utilizaron diferentes configuraciones de espacio dimensional y número de drones, evaluando el desempeño del algoritmo en términos de acciones totales, acciones válidas y tiempo de ejecución. Los resultados obtenidos en 100 iteraciones con un algoritmo de Optimización por Colonia de Hormigas (ACO) muestran variaciones en función del tamaño del espacio y el número de drones.

Para un espacio dimensional de 5x5 con 2 drones, el número promedio de acciones totales requeridas fue de aproximadamente 29.76 con una desviación estándar de 5.10, indicando una variabilidad moderada. La proporción de acciones válidas fue de 0.68, con una desviación estándar de 0.10, sugiriendo que alrededor del 68% de las acciones realizadas fueron válidas. El tiempo promedio de ejecución fue de 11.60 segundos, con una desviación estándar de 1.65 segundos, lo que indica una ejecución relativamente consistente. Al aumentar el número de drones a 3, el número promedio de acciones requeridas disminuyó ligeramente a 28.35 con una desviación estándar de 6.32, mostrando una variabilidad ligeramente mayor. La proporción de acciones válidas mejoró a 0.69, con una desviación estándar de 0.11, y el tiempo promedio de ejecución fue de 11.38 segundos, con una desviación estándar de 2.79 segundos, mostrando una mayor variabilidad en el tiempo de ejecución.

En un espacio dimensional de 7x7 con 2 drones, el número promedio de acciones requeridas fue de 53.55 con una desviación estándar de 15.79, mostrando una variabilidad considerablemente alta. La proporción de acciones válidas mejoró a 0.76, con una desviación estándar de 0.20, y el tiempo promedio de ejecución fue de 21.20 segundos, con una

desviación estándar de 6.85 segundos, lo que indica una mayor variabilidad y un tiempo de ejecución más largo comparado con la configuración 5x5. Al aumentar el número de drones a 3, el número promedio de acciones requeridas disminuyó a 50.06 con una desviación estándar de 18.58, manteniendo una alta variabilidad. La proporción de acciones válidas mejoró aún más a 0.81, con una desviación estándar de 0.20, y el tiempo promedio de ejecución se redujo a 18.87 segundos, con una desviación estándar de 6.84 segundos, mostrando una reducción en el tiempo de ejecución comparado con el caso de 2 drones en la configuración 7x7.

En general, a medida que el espacio dimensional aumenta, el número de acciones totales y el tiempo de ejecución también aumentan, lo cual es esperable debido a la mayor complejidad del entorno. La proporción de acciones válidas tiende a ser mayor en espacios más grandes y con más drones, lo que sugiere que el algoritmo ACO puede gestionar mejor la planificación de rutas en escenarios más complejos con un mayor número de agentes. Sin embargo, la variabilidad en las métricas también aumenta con el tamaño del espacio, indicando que el rendimiento del algoritmo es menos predecible en entornos más grandes.

Estos resultados proporcionan una visión detallada del comportamiento del algoritmo ACO en diferentes configuraciones de *path planning*, resaltando tanto sus fortalezas como áreas potenciales de mejora.

4.1. Discusión de resultados

A lo largo de esta sección, se va a realizar un análisis y discusión de los resultados. Se van a presentar los anteriores resultados en forma de gráficas para poder comentarlas y destacar lo más importante de cada una de ellas.

Para comenzar, en la figura 4.1 se puede observar dos diagramas de cajas, también conocidos como *boxplots*, los cuales representan las acciones totales para los algoritmos de *Deep Q-Learning* y el Algoritmo Genético.

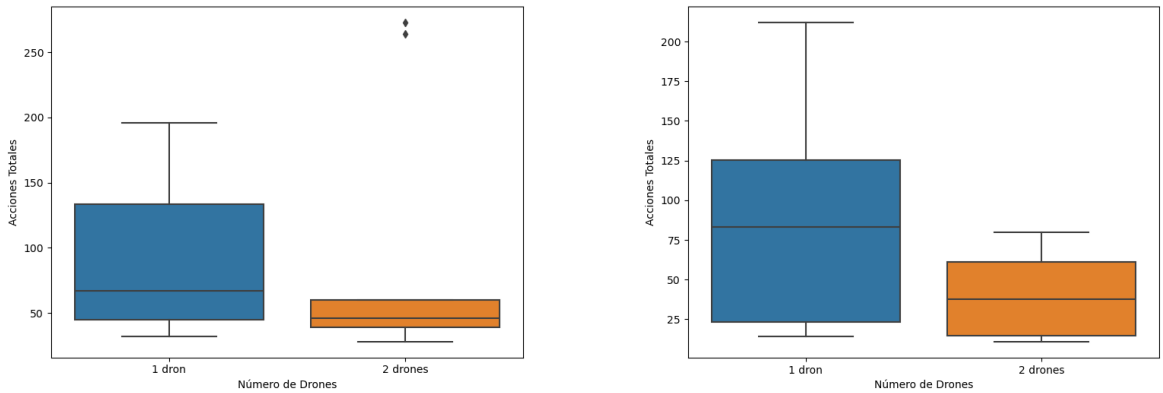
En la presentación de los resultados obtenidos mediante el uso de *boxplots*, se ha decidido omitir algunos gráficos cuya visualización no era clara, con el fin de evitar confusiones y garantizar una interpretación precisa de los datos. Esto se debe a que algunos boxplots resultaban ser visualmente indistinguibles, principalmente aquellos en los que la variabilidad de los datos era extremadamente baja o los datos estaban altamente concentrados en un rango muy estrecho, haciendo que las representaciones gráficas no añadieran valor significativo al análisis.

Además, es importante explicar que, en algunos casos, los boxplots no muestran bigotes. Esto ocurre cuando no existen valores de datos más allá de 1.5 veces el rango intercuartílico desde los cuartiles superior e inferior. Esencialmente, esto significa que todos los valores se concentran cerca de la mediana, o que los valores extremos están muy cerca de los cuartiles.

Los *boxplots* presentados en la figura 4.1 muestran la distribución de las acciones totales para los algoritmos *Deep Q-Learning* y Algoritmo Genético en un espacio de 3x3, analizando el rendimiento con 1 y 2 drones. En el caso del *Deep Q-Learning*, el uso de un solo dron muestra una mediana cercana a 100 acciones totales, con una variabilidad significativa en los resultados, y un valor atípico extremadamente alto que sugiere posibles desafíos o ineficiencias. Sin embargo, al aumentar a dos drones, la mediana se reduce considerablemente, aproximadamente a la mitad, con menos variabilidad y sin valores atípicos, indicando una mejora en la eficiencia y consistencia.

Por otro lado, el Algoritmo Genético también muestra una mediana de acciones totales similar para un dron, pero con menos variabilidad que el *Deep Q-Learning*. Al usar dos drones, este algoritmo demuestra una reducción en la mediana de acciones necesarias, aunque la mejora no es tan pronunciada como en el *Deep Q-Learning*. La consistencia y eficiencia mejoran también con dos drones, aunque la diferencia no es tan marcada como con el primer algoritmo.

Ambos algoritmos reflejan una mejora clara en el rendimiento al incrementar el número de drones de uno a dos, sugiriendo que la colaboración adicional puede ayudar significativamente a mejorar la eficiencia en tareas complejas. La menor variabilidad observada en los resultados con dos drones para ambos algoritmos sugiere que las ejecuciones son más predecibles y posiblemente más robustas en comparación con el uso de un solo dron.



(a) Diagrama de caja de acciones totales para entorno 3x3 con *Deep Q-Learning*

(b) Diagrama de caja de acciones totales para entorno 3x3 con Algoritmo Genético

Figura 4.1: Diagramas de cajas de acciones totales para entorno 3x3 con algoritmos *Deep Q-Learning* y el Algoritmo Genético.

A continuación, en la figura 4.2 se puede observar los *boxplots* para las distintas configuraciones en el entorno 5x5. Cada algoritmo muestra una tendencia única en la reducción de acciones totales a medida que se incrementa el número de drones.

Para el *Deep Q-Learning*, se observa una tendencia decreciente en las acciones totales con un incremento en el número de drones, alcanzando el punto más bajo con tres drones. Sin embargo, con cinco drones, la mediana de las acciones aumenta ligeramente, lo cual podría indicar una disminución en la eficiencia marginal al añadir más drones más allá de un cierto punto. La presencia de valores atípicos altos sugiere que hay situaciones en las que el algoritmo podría no manejar eficientemente la coordinación entre múltiples drones.

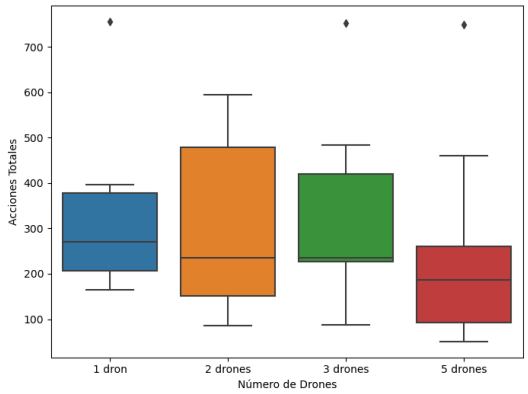
En el Algoritmo Genético, el número de acciones disminuye significativamente de uno a dos drones, estabilizándose después con tres drones. Este patrón indica que el algoritmo logra una optimización considerable con un pequeño aumento en el número de drones, pero no gana beneficios adicionales significativos al incrementar más allá de dos drones, como lo demuestra la variabilidad reducida y las medianas consistentes para dos y tres drones.

Finalmente, el *Ant Colony Optimization* muestra una reducción drástica y continua en las acciones totales a medida que se incrementa el número de drones de uno a cinco. Este algoritmo parece ser extremadamente efectivo en aprovechar un mayor número de drones, reduciendo consistentemente las acciones necesarias para completar las tareas. La reducción sustancial en acciones con cada adición de drones sugiere una alta eficiencia y capacidad para escalar bien con números mayores de drones.

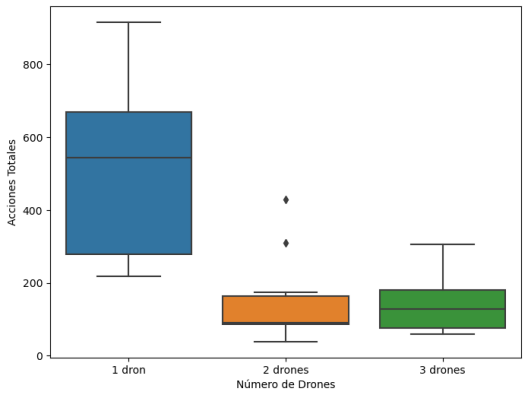
Cada gráfico refleja diferencias clave en cómo cada algoritmo gestiona la escalabilidad y la colaboración entre múltiples drones, resaltando la importancia de seleccionar el algoritmo adecuado según el número de drones disponibles y las características específicas del entorno de operación.

En resumen, los análisis de los boxplots revelan que, aunque cada algoritmo mejora con el aumento del número de drones, sus rendimientos varían significativamente. *Deep Q-Learning* muestra una eficiencia optimizada con tres drones, pero con una disminución marginal al aumentar a cinco. El Algoritmo Genético alcanza una eficiencia óptima con dos drones, sin beneficios significativos al añadir más. Por su parte, *Ant Colony Optimization* exhibe una mejora continua y destacada en la eficiencia con cada dron adicional, demostrando ser altamente escalable y efectivo en entornos complejos.

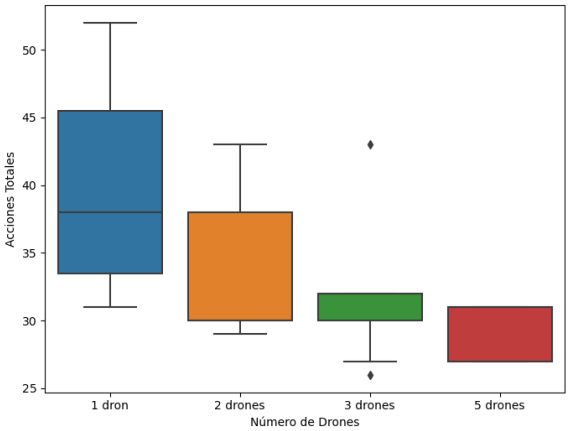
Estos hallazgos subrayan la importancia de seleccionar la configuración adecuada del algoritmo y el número de drones para maximizar la eficiencia operativa.



(a) Diagrama de caja de acciones totales para entorno 5x5 con *Deep Q-Learning*



(b) Diagrama de caja de acciones totales para entorno 5x5 con Algoritmo Genético



(c) Diagrama de caja de acciones totales para entorno 5x5 con *Ant Colony Optimization*

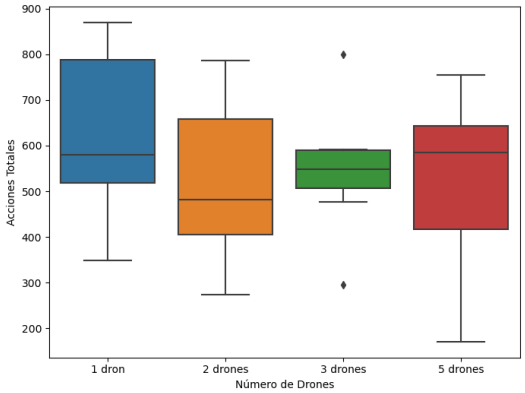
Figura 4.2: Diagramas de cajas de acciones totales para entorno 5x5 con algoritmos *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*.

Finalmente, se van a realizar los *boxplots* correspondientes al entorno 7x7, los cuales podemos observar en la figura 4.3.

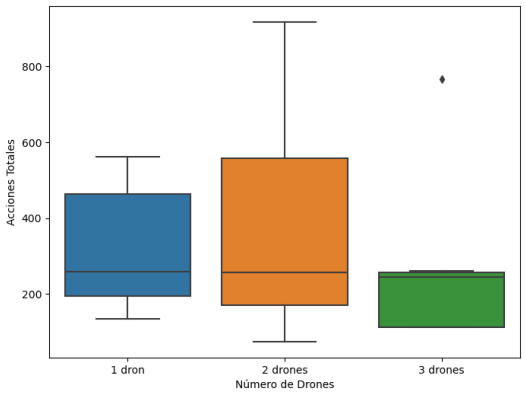
En el *Deep Q-Learning*, el número de acciones totales disminuye significativamente al aumentar de uno a dos drones, pero luego se observa una variabilidad en el rendimiento con tres y cinco drones. La presencia de valores atípicos con tres drones sugiere que, aunque el algoritmo puede ser altamente eficiente bajo ciertas configuraciones, también puede experimentar fluctuaciones significativas en su desempeño, debido a la naturaleza del espacio de búsqueda y la dinámica de interacción entre los drones.

El Algoritmo Genético muestra una notable eficiencia con dos drones, donde la mediana y la variabilidad de las acciones totales se reducen drásticamente comparadas con un solo dron. Con tres drones, la eficiencia se mantiene, indicando que este algoritmo puede optimizar eficazmente las acciones en configuraciones de múltiples drones.

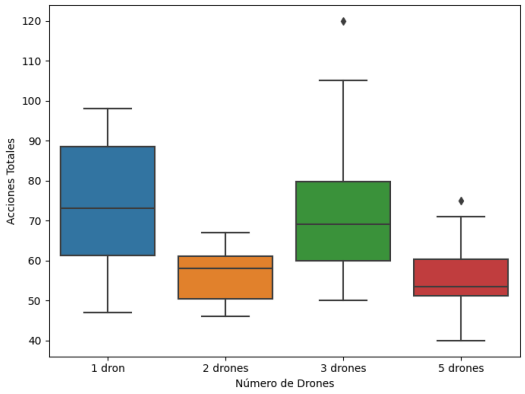
Para el *Ant Colony Optimization*, hay una tendencia clara de reducción en las acciones totales conforme aumenta el número de drones, con el menor número de acciones y la menor variabilidad observadas con cinco drones. Este algoritmo demuestra una habilidad excepcional para escalar con el número de drones, mejorando la eficiencia operativa a medida que más drones participan en la tarea. Estos boxplots indican cómo la selección del algoritmo adecuado y la configuración del número de drones son cruciales en entornos más complejos.



(a) Diagrama de caja de acciones totales para entorno 7x7 con *Deep Q-Learning*



(b) Diagrama de caja de acciones totales para entorno 7x7 con Algoritmo Genético



(c) Diagrama de caja de acciones totales para entorno 7x7 con *Ant Colony Optimization*

Figura 4.3: Diagramas de cajas de acciones totales para entorno 7x7 con algoritmos *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*.

Asimismo, a continuación, se van a representar mediante *violin plots*, los tiempos de ejecución de las distintas configuraciones.

Los *violin plots* son visualizaciones que combinan las características de un diagrama de caja y un gráfico de densidad de kernel, mostrando tanto la distribución de los datos como su probabilidad de densidad, lo cual permite visualizar la forma completa de la distribución de los datos (Hintze and Nelson, 1998).

Las gráficas representadas en la figura 4.4 ilustran las distribuciones de tiempos de ejecución para diferentes algoritmos y configuraciones de drones en un entorno simulado 5x5.

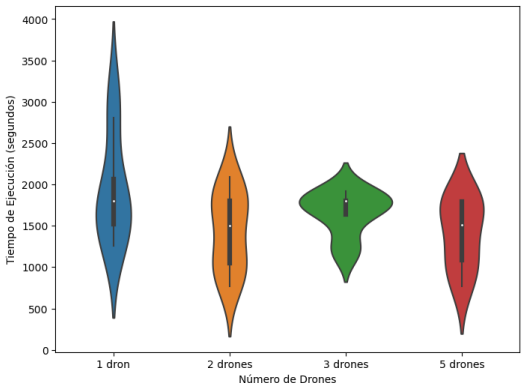
Cada gráfico ofrece una comparación visual de cómo los tiempos de ejecución varían con el número de drones empleados, proporcionando una representación detallada de la densidad de los datos y su rango.

Para *Deep Q-Learning*, observamos que los tiempos de ejecución son generalmente altos, pero tienden a ser más consistentes con tres drones, aunque se dispersan considerablemente con cinco drones, indicando una posible ineficiencia o variabilidad en la coordinación cuando se aumenta el número de drones.

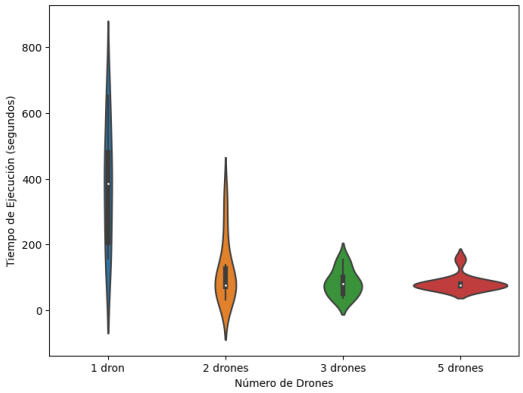
En el caso del Algoritmo Genético, los tiempos de ejecución muestran una notable disminución al pasar de un dron a dos, pero con tres drones, los tiempos se incrementan levemente y se reducen drásticamente con cinco drones. Esto sugiere que el algoritmo puede ser muy eficiente con configuraciones específicas, pero también puede ser susceptible a inconsistencias dependiendo de la configuración y el entorno.

Finalmente, los resultados del *Ant Colony Optimization* muestran que este algoritmo logra tiempos de ejecución relativamente bajos y consistentes, especialmente con un mayor número de drones. Esto indica que *Ant Colony Optimization* puede ser particularmente eficaz en la optimización del rendimiento con incrementos en el número de drones, logrando una mejora continua en la eficiencia.

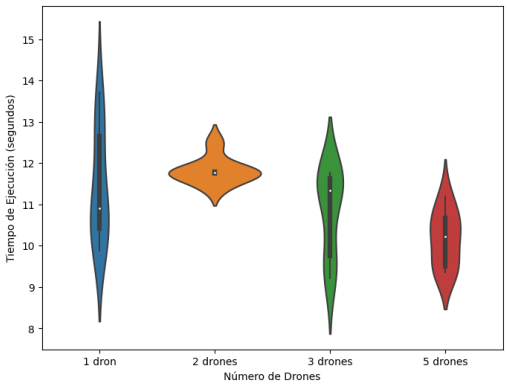
En general, estos gráficos de violín son útiles para entender visualmente la distribución y la variabilidad de los tiempos de ejecución en función del número de drones y del tipo de algoritmo utilizado, ayudando a identificar qué configuraciones podrían ofrecer los mejores equilibrios entre eficiencia y consistencia.



(a) *Violin plots* de tiempo de ejecución para entorno 5x5 con *Deep Q-Learning*



(b) *Violin plots* de tiempo de ejecución para entorno 5x5 con Algoritmo Genético



(c) *Violin plots* de tiempo de ejecución para entorno 5x5 con *Ant Colony Optimization*

Figura 4.4: *Violin plots* de tiempos de ejecuciones para entorno 5x5 con algoritmos *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*.

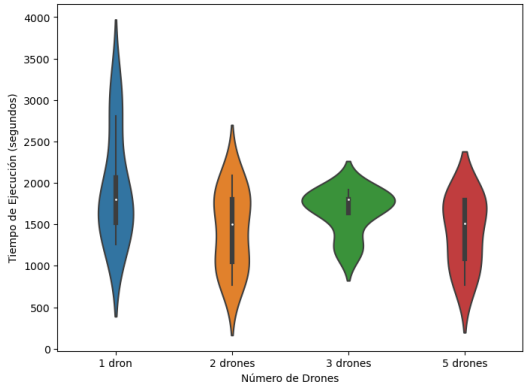
La figura 4.5, representa las gráficas de violín para los entornos 7x7, utilizando diferentes algoritmos revelan varias dinámicas interesantes en cuanto a los tiempos de ejecución se refiere.

Para el *Deep Q-Learning*, se observa una amplia distribución en los tiempos de ejecución para un dron que se estrecha significativamente con dos y tres drones, indicando una mayor consistencia en estos escenarios. Con cinco drones, sin embargo, la distribución se ensancha nuevamente, sugiriendo que agregar más drones puede no siempre resultar en una mejora de eficiencia debido a posibles complicaciones en la coordinación o la gestión de recursos.

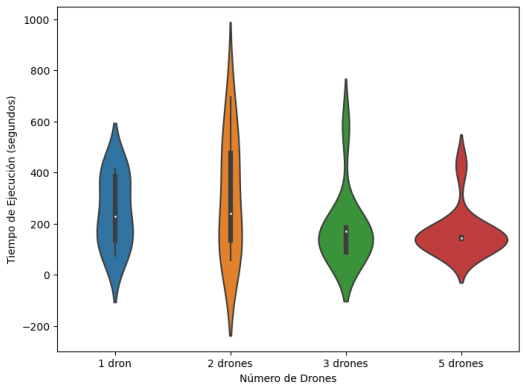
En el caso del Algoritmo Genético, los tiempos de ejecución son generalmente más bajos

y menos variados que los del *Deep Q-Learning*, especialmente notable con dos y tres drones, donde las distribuciones son estrechas y centradas, mostrando una optimización efectiva. Con cinco drones, el tiempo se reduce aún más, aunque la forma del violín se hace más irregular, lo que podría indicar algunos casos donde el rendimiento varía más notablemente.

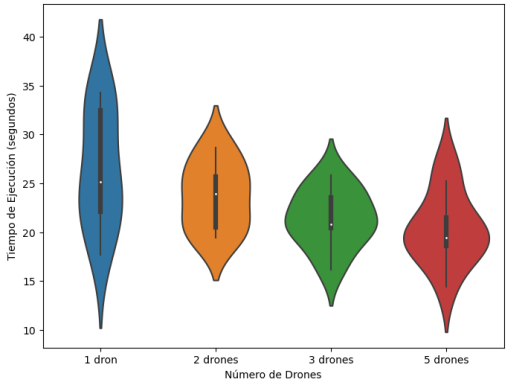
Para el *Ant Colony Optimization*, los tiempos de ejecución son consistentemente bajos y exhiben las distribuciones más estrechas de los tres algoritmos, destacando su eficacia en el entorno 7x7 con cualquier número de drones. La progresión desde uno a cinco drones muestra una disminución en la mediana de los tiempos de ejecución y una compactación de la distribución, lo que sugiere que este algoritmo es particularmente efectivo para manejar aumentos en la cantidad de drones sin sacrificar el rendimiento.



(a) *Violin plots* de tiempo de ejecución para entorno 7X7 con *Deep Q-Learning*



(b) *Violin plots* de tiempo de ejecución para entorno 7X7 con Algoritmo Genético



(c) *Violin plots* de tiempo de ejecución para entorno 7X7 con *Ant Colony Optimization*

Figura 4.5: *Violin plots* de tiempos de ejecuciones para entorno 7X7 con algoritmos *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*.

La gráfica 4.6 muestra el porcentaje de acciones válidas obtenidas utilizando diferentes algoritmos de planificación de rutas, como *Deep Q-Learning* (DQL), Algoritmo Genético (AG), y *Ant Colony Optimization* (ACO), con variaciones en el número de drones en un entorno de 5x5. Se observa que ACO muestra una notable mejora en el porcentaje de acciones válidas con el aumento del número de drones, alcanzando sus valores más altos y consistentes con tres y cinco drones, lo cual sugiere una eficacia considerable en entornos más complejos y dinámicos al aumentar el número de agentes. En contraste, el *Deep Q-Learning* y el Algoritmo Genético muestran un incremento menos pronunciado en la validez de las acciones conforme se incrementa el número de drones, con el *Deep Q-Learning* presentando una amplia variabilidad en sus resultados, especialmente con un solo dron donde las acciones válidas son notoriamente bajas. Esto podría indicar limitaciones en estos algoritmos para adaptarse eficazmente a la complejidad incrementada de manejar múltiples drones en términos de evitar acciones no productivas o inválidas dentro del proceso de simulación.

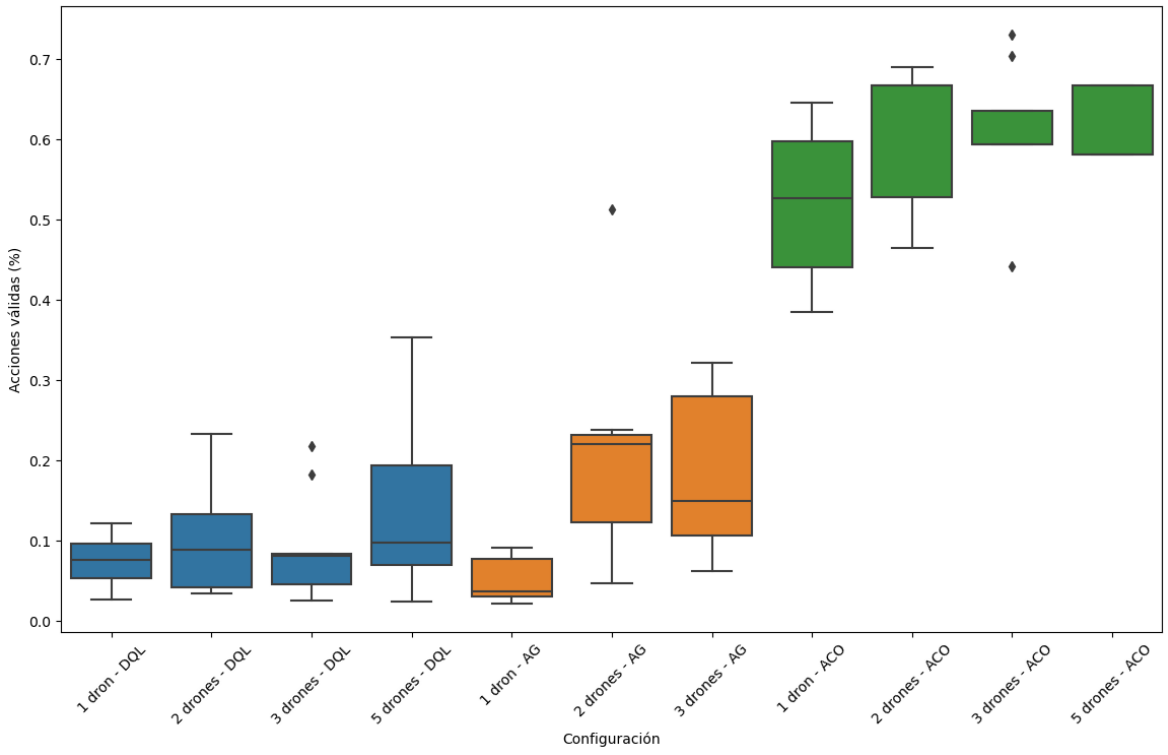


Figura 4.6: Diagrama de cajas de acciones válidas para entorno 5x5 con algoritmos *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*.

La gráfica 4.7 muestra el porcentaje de acciones válidas al utilizar algoritmos de planificación de rutas como *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*, en un entorno más grande y complejo de 7x7, con distintas cantidades de drones. En este escenario, ACO demuestra una eficiencia destacada, logrando porcentajes de acciones válidas consistentemente más altos, especialmente con dos y tres drones, lo que sugiere una mejor adaptación a entornos de mayor escala. Los resultados del algoritmo genético (AG) son menos consistentes, aunque mejoran con más drones, pero no alcanzan la eficiencia de ACO. Por otro lado, el *Deep Q-Learning* presenta resultados relativamente bajos y menos variabilidad en el porcentaje de acciones válidas, lo que podría indicar una capacidad limitada para adaptarse eficazmente a la complejidad del entorno 7x7. Estos hallazgos destacan la importancia de elegir el algoritmo adecuado según la dimensionalidad y las características específicas del entorno operativo para optimizar el rendimiento en la planificación de rutas de enjambres de drones.

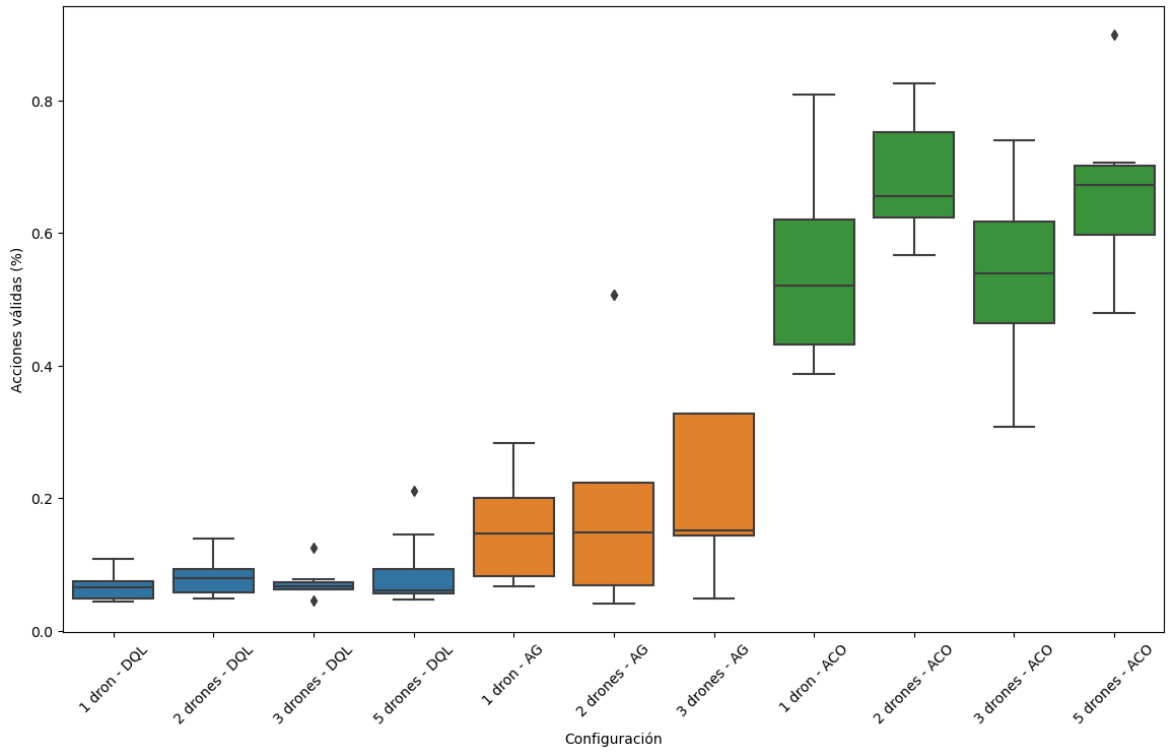


Figura 4.7: Diagrama de cajas de acciones válidas para entorno 7x7 con algoritmos *Deep Q-Learning*, Algoritmo Genético y *Ant Colony Optimization*.

Seguidamente, vamos a discutir los resultados que están presentes en la tabla 4.4, es decir, los resultados obtenidos al realizar 100 episodios con nuestro algoritmo de *Ant Colony Optimization*.

La gráfica 4.8 presentada muestra claramente cómo el número de acciones totales por episodio en un entorno 5x5 se desarrolla a lo largo de los episodios para configuraciones de dos y tres drones. Se observa una convergencia notable en la eficiencia del algoritmo, especialmente con tres drones, donde la línea muestra una disminución constante y estabilización en las acciones totales necesarias. Esto indica que la incorporación de un tercer dron ayuda significativamente a optimizar las tareas, logrando una convergencia hacia soluciones más eficientes a medida que avanza el número de episodios.

Durante los primeros 25 episodios, el algoritmo parece priorizar la exploración, permitiendo una búsqueda amplia en el espacio de soluciones para evitar una convergencia prematura hacia resultados subóptimos. Esto es crucial en las fases iniciales para asegurar que el algoritmo tenga la oportunidad de descubrir y evaluar una variedad de posibles soluciones. Posteriormente, en los 75 episodios restantes, el enfoque cambia hacia una mayor explotación, basándose más en las feromonas acumuladas para refinar y optimizar las soluciones encontradas. Esta estrategia de balance entre exploración inicial y explotación posterior fomenta no solo encontrar soluciones aceptables, sino también afinarlas hacia el óptimo, como se refleja en la estabilización y reducción en el número de acciones totales especialmente con tres drones.

Este enfoque estratégico muestra cómo la adaptación y ajuste de tácticas conforme progresa la simulación puede influir significativamente en la eficiencia y efectividad del algoritmo, destacando la importancia de una planificación cuidadosa en la implementación de algoritmos de optimización autónoma.

Observando la línea azul, que representa dos drones, se puede notar que la variabilidad en el número de acciones es considerable a lo largo de los episodios, con picos que sugieren episodios de mayor dificultad o ineficiencia en la coordinación. Sin embargo, la tendencia general muestra una disminución en el número de acciones a medida que avanza el número de episodios, lo que podría indicar una mejora en la eficacia del algoritmo a lo largo del tiempo.

En contraste, la línea roja, que representa tres drones, muestra una tendencia mucho más estable y baja en el número de acciones por episodio, con muy poca variabilidad después de los primeros 20 episodios. Este comportamiento sugiere que la incorporación de un tercer dron podría estar facilitando una mayor eficiencia y consistencia en la ejecución de las tareas, permitiendo que el sistema maneje el entorno de manera más efectiva con menos acciones totales.

La diferencia notable en las fluctuaciones y la estabilidad entre las dos configuraciones resalta cómo el número de drones puede impactar significativamente la eficiencia operativa en tareas de navegación y ejecución autónoma en entornos controlados. Este tipo de información es crucial para optimizar algoritmos y configuraciones en aplicaciones reales donde la precisión y la eficiencia son críticas.

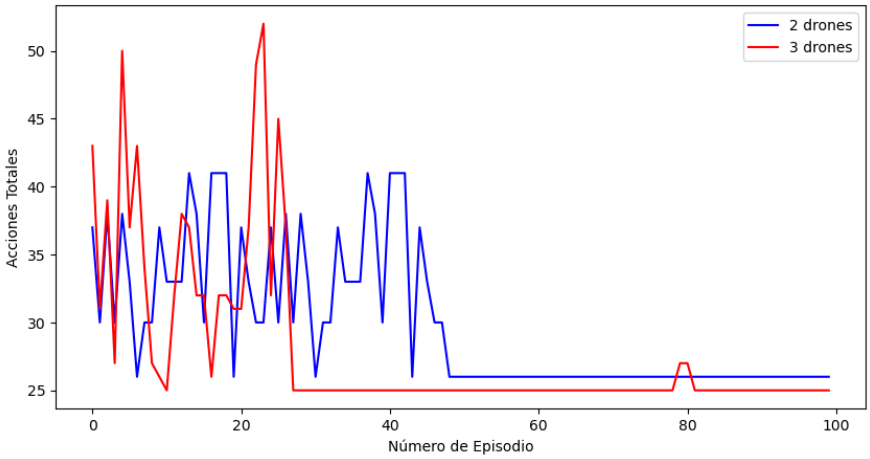


Figura 4.8: Gráfico de líneas de acciones totales para entorno 5x5 con algoritmo *Ant Colony Optimization* con 100 iteraciones.

La gráfica 4.9 muestra la comparación de acciones por episodio en un entorno de 7x7 utilizando dos y tres drones, ofreciendo una visión clara de cómo varían las acciones necesarias en cada configuración a lo largo de 100 episodios. La línea azul, que representa dos drones, exhibe una variabilidad notable en el número de acciones a lo largo de los episodios, con picos significativos que indican episodios de mayor dificultad o menor eficiencia en la coordinación entre los drones. Aunque la tendencia general muestra una reducción en el número de acciones hacia el final, la línea sigue mostrando fluctuaciones que sugieren que el sistema aún está ajustando su estrategia para manejar el entorno de manera más efectiva.

En contraste, la línea roja, representando tres drones, muestra una reducción mucho más rápida y estable en el número de acciones por episodio, alcanzando un nivel bajo de acciones consistentemente a partir del episodio 20 aproximadamente. Esta estabilidad sugiere que la adición de un tercer dron contribuye significativamente a la eficiencia del sistema, permitiendo una gestión más efectiva y uniforme de las tareas requeridas en el entorno. La presencia de tres drones parece facilitar una mejor distribución de las tareas y una mayor cobertura del espacio, lo que resulta en una disminución más rápida y sostenida de las acciones necesarias.

Estas observaciones subrayan cómo el número de drones influye en la eficacia operativa, especialmente en entornos más complejos donde la coordinación y la cobertura son críticas para el éxito de las operaciones. La configuración con tres drones no solo muestra una mayor eficiencia inicial sino también una mayor consistencia en el desempeño, lo que es crucial para aplicaciones prácticas donde la previsibilidad y la fiabilidad son fundamentales.

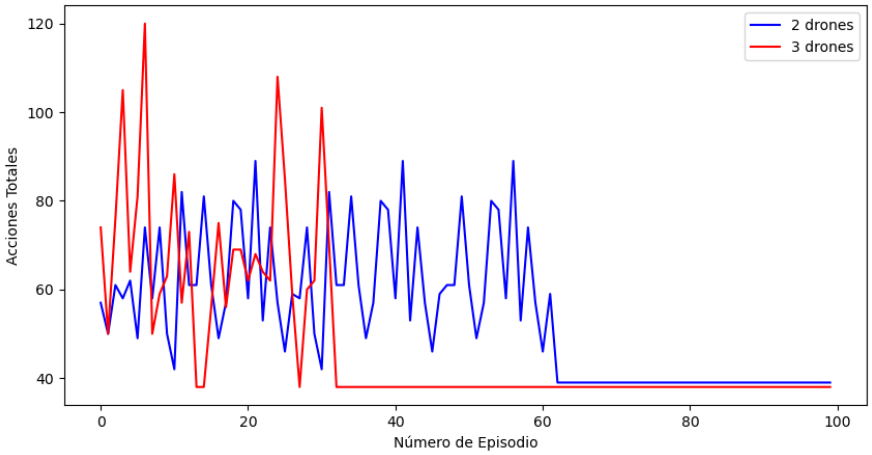


Figura 4.9: Gráfico de líneas de acciones totales para entorno 7x7 con algoritmo *Ant Colony Optimization* con 100 iteraciones.

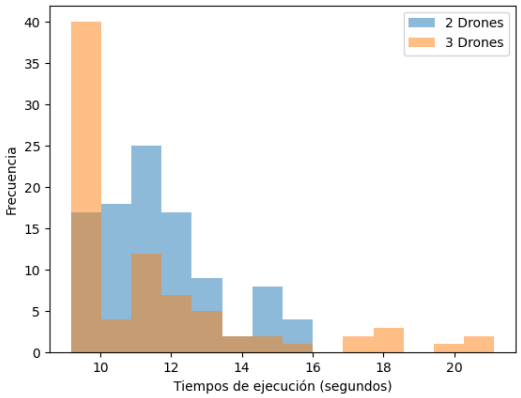
Para concluir, vamos a representar mediante histogramas en la figura 4.10, los tiempos de ejecución para los valores obtenidos en la tabla 4.4.

Las gráficas presentadas muestran los histogramas de tiempos de ejecución para entornos 5x5 y 7x7 utilizando el algoritmo *Ant Colony Optimization*, comparando la distribución de tiempos entre configuraciones de 2 y 3 drones.

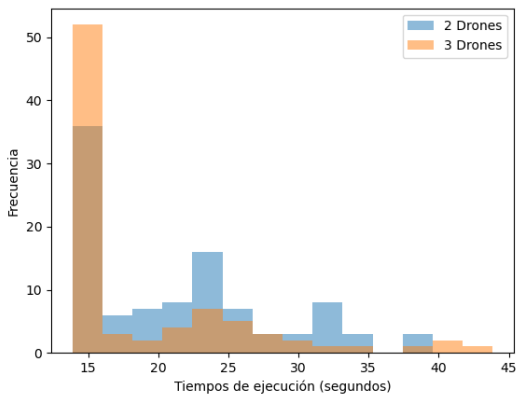
En el entorno 5x5, observamos que los tiempos de ejecución para dos drones están predominantemente agrupados alrededor de los 14 segundos, mientras que para tres drones, la distribución se expande notablemente hacia tiempos más cortos, con una concentración principal alrededor de los 12 segundos. Esto indica que agregar un dron adicional en este entorno puede mejorar significativamente la eficiencia, logrando completar las tareas más rápidamente.

Por otro lado, en el entorno 7x7, la diferencia es aún más marcada. Con dos drones, la mayoría de las ejecuciones se agrupan cerca de los 15 segundos, mientras que con tres drones, los tiempos se reducen drásticamente, con la mayoría de los tiempos de ejecución ubicados bajo los 25 segundos. Este cambio dramático sugiere que el algoritmo es capaz de optimizar mucho más eficientemente con tres drones en comparación con dos, en un entorno más grande.

Estos histogramas son útiles para visualizar cómo la variabilidad en los tiempos de ejecución puede ser afectada por el número de drones en uso. Además, proporcionan una visión clara de cómo la optimización de recursos y la coordinación entre drones puede afectar el rendimiento general del sistema en tareas específicas de optimización de rutas.



(a) Histograma de tiempos de ejecución para entorno 5x5 con algoritmo *Ant Colony Optimization* con 100 iteraciones



(b) Histograma de tiempos de ejecución para entorno 7X7 con algoritmo *Ant Colony Optimization* con 100 iteraciones

Figura 4.10: Histogramas de tiempos de ejecución con algoritmo *Ant Colony Optimization* con 100 iteraciones.

A continuación, se va a realizar la prueba de Tukey, la cual es una prueba estadística utilizada para determinar las diferencias entre las medias de las muestras y se comparan con una denominada "Diferencia honestamente significativa" (HSD), que se calcula mediante la siguiente ecuación.

$$HSD = Multiplicador \cdot \sqrt{\frac{MSe}{n}} \tag{4.1}$$

donde el multiplicador es el valor obtenido de la Tabla Tukey, MSe es el error cuadrático medio y n es el tamaño de la muestra en los grupos (Carmen Chacón de Isla, 2020).

En nuestro problema se está utilizando para comparar los resultados obtenidos con los diferentes algoritmos y determinar si existen diferencias significativas entre los distintos algoritmos respecto a las diferentes métricas que hemos obtenido.

Primeramente, vamos a comparar si existen diferencias significativas entre los algoritmos con respecto al porcentaje de acciones válidas. Los resultados se pueden visualizar en la tabla 4.5.

Para cada comparación de grupo, se presenta la diferencia media (*meandiff*), el valor p-ajustado (*p-adj*), el intervalo de confianza inferior (*lower*) y superior (*upper*), y si se rechaza la hipótesis nula (*reject*).

Podemos observar que hay una diferencia significativa entre *Ant Colony Optimization* (ACO) y el algoritmo genético (AG), debido a que el valor p-ajustado es 0.0 ($p < 0.05$), por lo que se rechaza la hipótesis nula, lo que indica una diferencia estadísticamente significativa.

group1	group2	meandiff	p-adj	lower	upper	reject
ACO	AG	-0.4329	0.0	-0.5304	-0.3354	True
ACO	DQL	-0.513	0.0	-0.6105	-0.4156	True
AG	DQL	-0.0802	0.1222	-0.1776	0.0173	False

Tabla 4.5: Resultados de la prueba de Tukey respecto a los porcentajes de las acciones válidas.

Seguidamente, se pueden observar diferencias significativas entre ACO y DQL debido a que ACO tiene un valor significativamente más bajo que DQL, con una diferencia media de -0.513 y un valor p-ajustado de 0.0 ($p < 0.05$) y, por lo tanto, se rechaza la hipótesis nula, lo que indica una diferencia estadísticamente significativa entre estos dos grupos.

Sin embargo, para los algoritmos AG y DQL no se encuentra una diferencia significativa, ya que, hay una diferencia media de -0.0802 y un valor p-ajustado de 0.1222 ($p > 0.05$), por lo que, la hipótesis nula no se rechaza, lo que sugiere que estos dos grupos pueden ser estadísticamente similares en términos de sus valores medios.

A continuación, se va a realizar la prueba de Tukey también para las métricas del número de acciones totales. En la tabla 4.6 se pueden ver representados los resultados de la prueba de Tukey.

Se observa que hay diferencias significativas entre los grupos *Ant Colony Optimization* y Algoritmo genético y, entre los grupos *Ant Colony Optimization* y *Deep Q-Learning*, ya que se rechaza la hipótesis nula en ambos casos. Sin embargo, no hay una diferencia significativa entre los grupos Algoritmo genético y *Deep Q-Learning*.

Por lo tanto, teniendo en cuenta que se tiene un nivel de confianza de 0.05, podemos concluir que:

- La comparación entre los grupos ACO y AG sigue siendo significativa, ya que el valor p ajustado (0.0131) es menor que el nivel de significancia de 0.05.
- La comparación entre los grupos ACO y DQL también sigue siendo significativa, ya que el valor p ajustado (menor que 0.0001) es mucho menor que el nivel de significancia de 0.05.
- La comparación entre los grupos AG y DQL sigue siendo no significativa, ya que el valor p ajustado (0.1671) es mayor que el nivel de significancia de 0.05.

Finalmente, se va a realizar la prueba de Tukey para los valores obtenidos para los tiempos de ejecución entre los distintos algoritmos.

group1	group2	meandiff	p-adj	lower	upper	reject
ACO	AG	171.902	0.0131	32.9168	310.8872	True
ACO	DQL	276.665	0.0001	137.6798	415.6502	True
AG	DQL	104.763	0.1671	-34.2222	243.7482	False

Tabla 4.6: Resultados de la prueba de Tukey respecto a las acciones totales.

En la tabla 4.7 se puede observar que se sigue teniendo algún caso similar en donde, entre ACO y DQL hay diferencias significativas, ya que el valor p-ajustado es 0.0 ($p < 0.05$), sin embargo, hay ciertos cambios, ya que, entre ACO y AG ya no hay diferencias significativas debido a que el p-ajustado es 0.6024 ($p > 0.05$), mientras que entre AG y DQL sí que las hay, en donde tenemos un p-ajustado igual a 0.0001 ($p < 0.05$).

group1	group2	meandiff	p-adj	lower	upper	reject
ACO	AG	2.6615	0.6024	-4.1498	9.4728	False
ACO	DQL	16.8565	0.0	10.0452	23.6678	True
AG	DQL	14.195	0.0001	7.3837	21.0063	True

Tabla 4.7: Resultados de la prueba de Tukey respecto al tiempo total de ejecución.

4.2. Comparativa con otros autores

En la comparación de nuestros resultados con los informados en el estudio de Puente-Castro et al. (2023a), específicamente para las configuraciones de 2 y 3 drones en espacios dimensionales de 5x5 y 7x7, observamos que nuestros resultados muestran un menor número de acciones totales. Esta mejora en la eficiencia sugiere la posibilidad de una mayor proporción de acciones válidas en nuestros experimentos.

Es crucial considerar que los tiempos de ejecución de los algoritmos pueden variar significativamente dependiendo de la capacidad del hardware utilizado. En nuestro caso, la implementación de *Ant Colony Optimization* resultó en tiempos de ejecución considerablemente más largos en comparación con los reportados en el artículo de Puente-Castro et al. (2022b), donde se indica que los experimentos se completaron en aproximadamente un minuto para configuraciones similares. Este incremento en el tiempo de ejecución hasta aproximadamente seis minutos en nuestro entorno se debe a la ausencia de hardware especializado, ya que Puente-Castro et al. (2022b) realizó sus pruebas en un supercomputador con múltiples GPUs, lo que facilita un procesamiento más rápido y eficiente.

Es importante destacar que la comparativa entre ambos estudios se limita a cuatro configuraciones específicas y que las diferencias observadas podrían no generalizarse a otros escenarios o configuraciones de drones. Por lo tanto, cualquier conclusión sobre la superioridad de un enfoque sobre el otro debe hacerse con cautela, teniendo en cuenta las limitaciones del hardware y la especificidad de los casos de uso examinados.

El artículo de Batista et al. (2024) se enfoca en la optimización de tareas logísticas en entornos de biblioteca mediante el uso de múltiples agentes autónomos. La investigación aplica técnicas avanzadas de aprendizaje automático, específicamente el Aprendizaje por Refuerzo con Q-Learning, para desarrollar estrategias de planificación de rutas eficientes y efectivas para robots encargados de transportar libros dentro de un edificio de dos pisos.

Al igual que en nuestro Trabajo de Fin de Máster, el artículo de Batista et al. (2024) muestra que el uso de un mayor número de drones conduce a mejores resultados en términos de eficiencia operativa. Este fenómeno se observa tanto en nuestra investigación como en el artículo mencionado, donde la escalabilidad y el rendimiento mejoran significativamente con el incremento en el número de agentes. Esto demuestra que tanto en entornos de 5x5, 7x7 como en 9x9, la capacidad de los algoritmos para manejar múltiples drones de manera efectiva resulta en una reducción notable de los pasos necesarios para completar las tareas asignadas, optimizando así el tiempo y la energía consumidos durante las operaciones.

En el artículo Meng et al. (2024) presenta un algoritmo híbrido de optimización por enjambre de partículas combinado para la planificación de rutas de múltiples UAVs en entornos complejos. Los resultados experimentales demuestran que el algoritmo propuesto mejora significativamente la calidad de las rutas, la velocidad de convergencia y el tiempo de ejecución en comparación con otros algoritmos, mostrando una excelente capacidad de procesamiento y escalabilidad.

Mientras que en nuestro trabajo se consideró escenarios con 2 y 3 drones, el estudio de Meng et al. (2024) aborda la planificación de rutas para hasta 500 drones. Los resultados indican que conforme aumenta el número de drones, el algoritmo no solo mantiene su eficiencia, sino que también mejora los resultados en términos de costos de ruta y estabilidad.

Por lo tanto, tal y como se ha podido observar anteriormente en los artículos citados y en los resultados obtenidos en nuestro trabajo, al aumentar el número de drones obtenemos en la mayoría de casos, mejores resultados.

Para concluir, es de gran importancia destacar que el uso del algoritmo de *Ant Colony Optimization* se muestra eficiente y relativamente sencillo de implementar, ofreciendo soluciones robustas con requisitos computacionales reducidos. *Ant Colony Optimization* es particularmente efectivo en entornos estáticos y bien definidos de tamaño moderado. En contraste, cuando se enfrenta a escenarios de planificación de rutas en ambientes dinámicos, donde las condiciones cambian constantemente y la dimensionalidad es elevada, el *Deep Reinforcement Learning* se convierte en una alternativa más apropiada. Gracias a su capacidad para aprender y adaptarse en tiempo real, DRL es adecuado para manejar complejidades y variabilidades intensas, aunque esto requiera de una inversión significativa en términos de recursos computacionales. Esta capacidad lo hace ideal para aplicaciones donde la adaptabilidad es crucial para el éxito de la operación.

Además, en el análisis de algoritmos para la planificación de rutas en entornos con múltiples drones, el *Deep Q-Learning* muestra una mejora pronunciada en su rendimiento a medida que aumenta el número de drones. Esta tendencia sugiere que *Deep Q-Learning* se beneficia significativamente de la capacidad adicional y la diversidad de estrategias que ofrece un mayor número de agentes, permitiendo una exploración y explotación más eficaz del espacio de estados. Por otro lado, el algoritmo de *Ant Colony Optimization* también exhibe mejoras al incrementar los drones, aunque estas son menos pronunciadas en comparación con *Deep Q-Learning*. *Ant Colony Optimization* tiende a mostrar una convergencia más rápida hacia soluciones competentes, pero con menos potencial para mejorar dramáticamente con el aumento de agentes. Esto se debe a que *Ant Colony Optimization* ya optimiza la colaboración y la formación de caminos eficientes desde una etapa temprana, lo que puede limitar su ganancia incremental de rendimiento al añadir más drones.

En resumen, mientras DQL se adapta y escala mejor con números más grandes de drones, *Ant Colony Optimization* ofrece consistencia y eficacia, aunque con incrementos más modestos en la mejora del rendimiento con cada dron adicional.

5. Simulación en entorno real

En este capítulo se va a presentar cómo se ha conseguido digitalizar una carta real en cuadrícula con obstáculos y veremos cómo se comportan varios drones en dichos entornos.

Para la realización de estas simulaciones se va a utilizar:

- **PX4 Autopilot**. Es un software de control de vuelo de código abierto para drones y otros vehículos no tripulados, que ofrece un conjunto flexible de herramientas para que los desarrolladores de drones compartan tecnologías con el fin de crear soluciones a medida para aplicaciones de drones (PX4 Autopilot, 2024).
- **Unreal Engine**. Es un completo conjunto de herramientas de creación para el desarrollo de juegos, la visualización arquitectónica y automovilística, la creación de contenidos lineales para cine y televisión, la emisión y producción de eventos en directo, la formación y simulación y otras aplicaciones en tiempo real (Unreal Engine, 2024). En nuestro problema utilizaremos la versión 4.27.
- **AirSim**. Es una nueva plataforma que se ejecuta en Microsoft Azure para construir, entrenar y probar de forma segura aeronaves autónomas a través de simulación de alta fidelidad (Siegel, 2022).
- **QGroundControl**. Es una herramienta que proporciona un control de vuelo y una planificación de misión completos para cualquier dron habilitado para MAVLink. Su principal objetivo es la facilidad de uso para usuarios profesionales y desarrolladores (QGroudControl, 2024).

5.1. Configuración del entorno

Para crear un entorno virtual realista, se utilizó Unreal Engine 4.27. Este motor gráfico permite diseñar escenarios detallados donde los drones pueden operar. La configuración inicial incluye la creación de un pequeño terreno predeterminado y, la creación del aspecto de los drones correspondientes. Esta herramienta se quiso implementar para poder visualizar cómo se comportarían los drones, es decir, cómo girarían en ciertos momentos, cómo de rápido se moverían y muchas otras funciones.

AirSim se integrará con Unreal Engine para simular la física y la dinámica de los drones. Se instaló el plug-in de AirSim en Unreal Engine y se realizó la configuración de los parámetros de los drones. En dicha configuración destaca el correcto uso de IPs y puertos para asegurar la comunicación entre AirSim, PX4 y QGroundControl.

5.2. Control y Navegación

PX4 Autopilot se utilizó para controlar los drones durante la simulación. Se configuró el firmware PX4 en el simulador para gestionar las rutas de vuelo.

QGroundControl se usó para monitorizar y gestionar la misión. Esta herramienta nos permitió visualizar en tiempo real la posición y el estado de los drones. Además, se configuró QGroundControl para recibir datos de telemetría de PX4 y facilitar la supervisión de la misión.

5.3. Diseño del experimento

A lo largo de esta sección se va a tratar el diseño del experimento desde la definición del escenario de prueba y la configuración de los parámetros hasta la ejecución de la simulación y la recogida de datos.

Primeramente, se va a definir el entorno sobre el que se ha probado la simulación. En este caso se ha decidido digitalizar una carta real en cuadrícula con obstáculos para ver cómo se desarrollaría en entorno real simulado. En la figura 5.1 se puede observar el mapa original sobre el que se realizará la simulación y sobre donde nuestros drones operarán y, en la figura 5.2 se puede ver una representación digital de las calles y los edificios para especificar el área sobre el que operaremos.

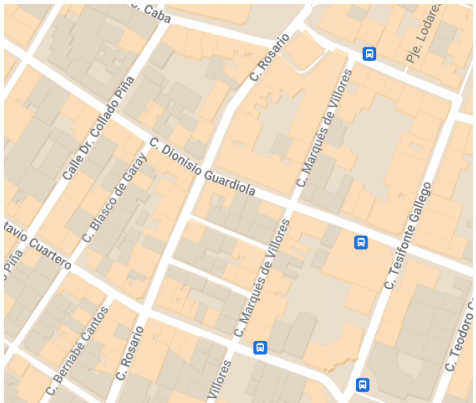


Figura 5.1: Mapa real original.

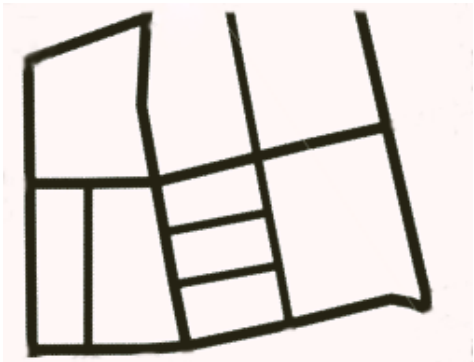


Figura 5.2: Extracción de calles y edificios.

Por lo tanto, se decidió digitalizar el mapa real en una cuadrícula, es decir, en forma de malla o matriz para poder obtener rutas de la misma manera con la que hemos trabajado a lo largo de este Trabajo Fin de Máster. En la imagen 5.3 se puede observar la imagen digitalizada en donde se puede ver un mapa con obstáculos. Dichos obstáculos son los edificios que se han visualizado anteriormente representados como unos conjuntos de celdas negras y las calles se han representado como el conjunto de celdas blancas.

Asimismo, hay que destacar que se definieron 5 drones, los cuales empezarán en distintas posiciones siguiendo una distribución similar a la que se sigue en los modelos creados en anteriores capítulos.

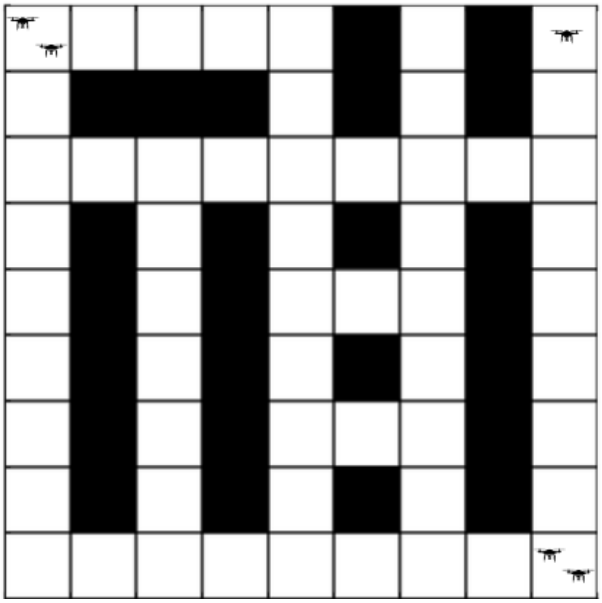


Figura 5.3: Mapa digitalizado en cuadrícula.

5.4. Ejecución de la simulación

Esta sección está dedicada a mostrar cómo se ha ejecutado la simulación y qué se puede visualizar mientras dicha ejecución se está produciendo.

En este caso tendremos dos aplicaciones, ejecutándose de manera simultánea, las cuales son QGroundControl, utilizada para visualizar en tiempo real la posición y el estado de los drones y, Unreal Engine, mediante la cual podremos ver cómo se desplazarían los drones en un escenario más realista.

En la figura 5.4 podemos ver una captura tomada en un instante de tiempo de la simulación realizada, en la cual se puede observar dónde están los drones, qué dirección están siguiendo y cuáles son las rutas que se van a seguir para poder llevar a cabo el problema de recorrer todo el espacio dimensional.

Además, en la parte inferior de la pantalla de QGroundControl, se encuentran varios indicadores de estado cruciales para el seguimiento de la misión del vehículo. Estos incluyen la altitud actual del vehículo, indicada como +50.0 metros. También se muestra el tiempo de vuelo transcurrido, que en este caso es de 00:01:53. Además, se indican distancias relevantes, como 117.1 metros, que representa la distancia recorrida, y 393.9 metros, que es distancia máxima alejada del punto de partida. Por último, hay una medida de velocidad, 5.1 metros por segundo, que se refiere a la velocidad horizontal del vehículo.

Estos datos son los recogidos para el vehículo 1, es decir, para el primer dron. Sin embargo, podremos alternar la vista entre drones para ver la información que se está recopilando sobre cada uno de ellos.

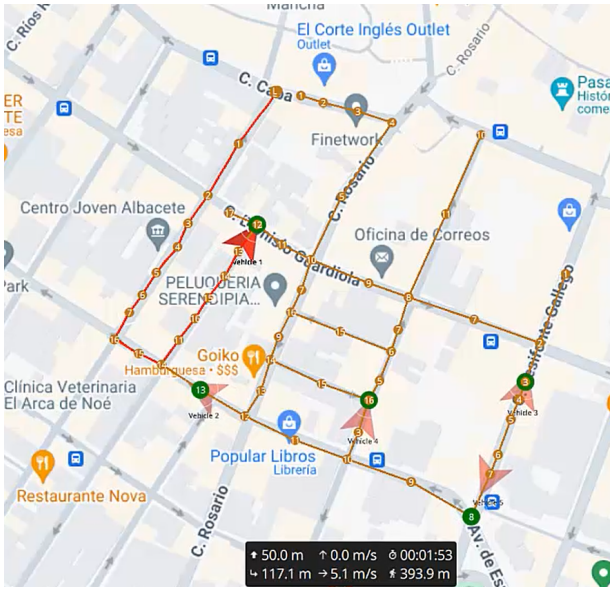


Figura 5.4: Simulación en QGroundControl.

A continuación, en la imagen 5.5 se pueden visualizar dos capturas que corresponden a la ejecución de la simulación en Unreal Engine, en donde se muestran cómo comparten un entorno los drones, pero que no colisiona, aunque puedan pasar cerca, debido a que están a diferente altitud unos de otros. Este motor gráfico nos ofrece una visión más realista de la simulación y de cómo se comportarían los drones correspondientes en este tipo de problemas.

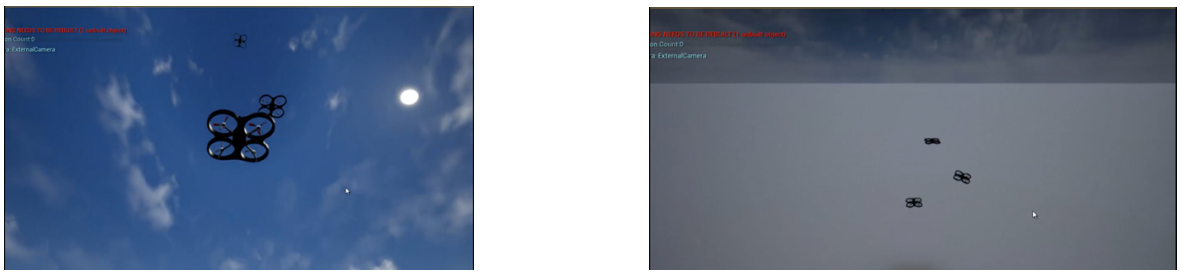


Figura 5.5: Simulación en Unreal Engine.

6. Conclusiones del proyecto

A lo largo de este capítulo se van a detallar las conclusiones finales de este Trabajo Fin de Máster. Aunque ya se hayan tratado en anteriores capítulos algunas conclusiones y futuras mejoras, estas eran específicamente relacionadas con la sección en las que se encontraban, sin embargo, en este capítulo se tratarán aquellas relacionadas con todo el proyecto.

6.1. Conclusiones

En este Trabajo Fin de Máster, se ha llevado a cabo un exhaustivo análisis comparativo entre técnicas de Reinforcement Learning y algoritmos de Evolutive Computing aplicados al control de enjambres de drones. A lo largo del estudio, se implementaron y evaluaron varios algoritmos, incluyendo *Deep Q-Learning*, *Ant Colony Optimization* y Algoritmos Genéticos.

Los resultados obtenidos destacan que los algoritmos de *Ant Colony Optimization* demostraron ser particularmente efectivos en la optimización de trayectorias de drones en diferentes entornos dimensionales. *Ant Colony Optimization* mostró una alta estabilidad y eficiencia en espacios más pequeños. Por otro lado, *Deep Q-Learning*, tuvo también un rendimiento robusto y adaptativo, pero con una mayor necesidad de recursos computacionales y tiempo de entrenamiento.

Además, se observó que la variabilidad en el rendimiento de los algoritmos es influenciada significativamente por el tamaño del espacio y el número de drones, lo que subraya la importancia de seleccionar el algoritmo adecuado según el contexto específico del problema.

El uso de enjambres de drones en lugar de un solo dron ofrece ventajas significativas en términos de eficiencia, cobertura y resiliencia de las misiones. Los enjambres permiten la división de tareas complejas en subtareas más manejables, lo que resulta en una mayor cobertura del área y una reducción en el tiempo de ejecución. Además, los enjambres pueden coordinarse para realizar operaciones simultáneas y complementar sus capacidades, lo que mejora la precisión y la eficacia de las misiones.

Conforme se incrementa el número de drones, generalmente se observan mejoras en los resultados debido a la capacidad de distribuir y optimizar mejor las rutas y las tareas asignadas. Sin embargo, el rendimiento también dependerá de factores como el algoritmo utilizado

para la planificación y coordinación, las dinámicas del entorno, y la capacidad de comunicación entre los drones.

6.2. Futuras mejoras

Para seguir mejorando y optimizando el rendimiento de los algoritmos de aprendizaje por refuerzo y algoritmos evolutivos en enjambres de drones, hay varias áreas clave que pueden ser exploradas. Aquí se presentan algunas sugerencias de futuras mejoras o implementaciones que podrían tratarse:

- **Entornos de simulación más grandes**: Evaluar los algoritmos en entornos de simulación más grandes y complejos para analizar su escalabilidad y robustez.
- **Entornos dinámicos**: Introducir dinámicas adicionales en los entornos de simulación, como obstáculos móviles o cambios en el terreno, para evaluar cómo los algoritmos se adaptan a situaciones impredecibles.
- **Capacidad de sobrevolar obstáculos**: Implementar la capacidad de los drones para sobrevolar obstáculos pequeños mediante un algoritmo de detección y evasión de obstáculos que considere movimientos laterales y verticales, aumentando así la eficiencia y seguridad en entornos urbanos.
- **Hibridación de algoritmos**. Desarrollar enfoques híbridos que combinen las fortalezas de *Reinforcement Learning* y *Evolutive Computing* para aprovechar las ventajas de ambos mundos, mejorando así la eficiencia y efectividad en la planificación de rutas y control de enjambres de drones.
- **Desarrollo de algoritmos adaptativos**. Investigar y desarrollar algoritmos adaptativos que puedan ajustar sus estrategias de planificación de rutas en tiempo real basándose en cambios dinámicos del entorno y en las condiciones de la misión.
- **Nuevas Estrategias de Evolución**. Probar nuevas estrategias de evolución, como la evolución diferencial (DE) o la optimización por enjambre de partículas (PSO) para comparar su rendimiento con los enfoques actuales.
- **Simulación en entornos más realistas**. Ampliar las pruebas a entornos de simulación más realistas y complejos, incluyendo variaciones ambientales dinámicas y obstáculos imprevistos, para validar la robustez y adaptabilidad de los algoritmos en condiciones más cercanas a la realidad.
- **Implementación en hardware real**. Trasladar las implementaciones de simulación a pruebas en hardware real, evaluando el desempeño de los algoritmos en drones físicos y midiendo su efectividad en escenarios prácticos de uso.

A. Anexo

A.1. Recursos software

Durante la realización de este proyecto, se han utilizado distintos recursos software, los cuales se detallarán a continuación ordenados alfabéticamente.

- **Jupyter notebooks.** Aplicación web de código abierto que permite crear y compartir documentos interactivos que contienen código, visualizaciones y texto. Utilizado para la realización de ciertos gráficos del TFM.
- **TensorFlow.** Biblioteca de código abierto utilizada para el aprendizaje automático y la inteligencia artificial. Esta biblioteca es empleada para la realización de los modelos con redes neuronales.
- **Keras.** Biblioteca de *Deep Learning* de alto nivel escrita en Python, la cual es utilizada para la realización de los modelos estáticos con redes neuronales.
- **Matplotlib.** Biblioteca de visualización de datos en Python, la cual es utilizada extraer las gráficas en el análisis exploratorio de datos y las gráficas que muestran la importancia de las características en los modelos realizados.
- **NumPy.** Biblioteca de Python utilizada principalmente para el cálculo numérico, la cual es empleada en la creación de la base de datos y en la realización de los modelos.
- **Overleaf.** Plataforma en línea que permite la edición y colaboración en tiempo real de documentos LaTeX, la cual es utilizada para la realización de este documento.
- **Pandas.** Biblioteca de software de código abierto para el lenguaje de programación Python. Esta biblioteca fue utilizada para el tratamiento y manipulación de datos.
- **Python.** Lenguaje de programación de alto nivel, interpretado y de propósito general, el cual es utilizado durante la realización de la base de datos, el análisis exploratorio de datos, creación y evaluación de los modelos y creación de la aplicación web.
- **Seaborn.** Biblioteca de visualización de datos en Python utilizada en el análisis exploratorio de datos.

- **Time.** Es una biblioteca en Python que proporciona funciones para trabajar con el tiempo, la cual es empleada para el cálculo del tiempo de ejecución de los modelos.
- **Random.** Es una librería en Python que proporciona funciones para generar números pseudoaleatorios y realizar operaciones relacionadas con la aleatoriedad.
- **Visual Studio Code.** Software creado por Microsoft que se utiliza para abrir, ver y editar archivos de código fuente, el cual utilizaremos en nuestros archivos Python.
- **Windows 11.** Sistema operativo desarrollado por Microsoft donde se ha ejecutado todo el proyecto.
- **PX4 Autopilot**. Es un software de control de vuelo de código abierto para drones.
- **Unreal Engine**. Es un completo conjunto de herramientas de creación para el desarrollo de juegos.
- **AirSim**. Plataforma que se ejecuta en Microsoft Azure para construir, entrenar y probar de forma segura aeronaves autónomas a través de simulación de alta fidelidad.
- **QGroundControl**. Es una herramienta que proporciona un control de vuelo y una planificación de misión completos para cualquier dron habilitado para MAVLink.

A.2. Recursos hardware

A lo largo de este trabajo de fin de grado se han utilizado una serie de recursos hardware, los cuales son los citados a continuación.

- **CPU**. 13th Gen Intel(R) Core(TM) i9-13900K
- **Tarjeta gráfica**. NVIDIA GeForce RTX 4090.
- **RAM**. 64,0 GB.

Referencia bibliográfica

Abdelkader, M., Güler, S., Jaleel, H., and Shamma, J. S. (2021). Aerial swarms: Recent applications and challenges. *Current robotics reports*, 2:309–320.

Aguilar-Rivera, R., Valenzuela-Rendón, M., and Rodríguez-Ortiz, J. (2015). Genetic algorithms and darwinian approaches in financial applications: A survey. *Expert Systems with Applications*, 42(21):7684–7697.

Batista, H., Carvalho, K., Oliveira, I. R., and Brandão, A. (2024). Smart library: A multi-agent path planning logistics operation using reinforcement learning.

Biesiadecki, J., Leger, C., and Maimone, M. (2007). Tradeoffs between directed and autonomous driving on the mars exploration rovers. *I. J. Robotic Res.*, 26:91–104.

Bortoff, S. A. (2000). Path planning for uavs. In *Proceedings of the 2000 american control conference. ACC (IEEE Cat. No. 00CH36334)*, volume 1, pages 364–368. IEEE.

Brand, M., Masuda, M., Wehner, N., and Yu, X.-H. (2010). Ant colony optimization algorithm for robot path planning. In *2010 International Conference On Computer Design and Applications*, volume 3, pages V3–436–V3–440.

Brownlee, J. (2011). *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee.

Carmen Chacón de Isla, J. L. I. C. (2020). Análisis de varianza prueba de tukey. https://www.colegiojuanxxiii.info/wp-content/uploads/2020/05/1_ANOVA-INVESTIGACION-Y-TECNOLOGIA-QUINTO.pdf. Accedido 27-05-2024.

Castillo, D. P. (2024). Tema 3: Aprendizaje por refuerzo con aprendizaje profundo. Máster Universitario en Ciencia de Datos. Universidad Camilo José Cela.

Chen, H., Wang, X.-m., and Li, Y. (2009). A survey of autonomous control for uav. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 2, pages 267–271. IEEE.

Chen, W., Liu, J., Guo, H., and Kato, N. (2020). Toward robust and intelligent drone swarm: Challenges and future directions. *IEEE Network*, 34(4):278–283.

Deng, L., Chen, H., Zhang, X., and Liu, H. (2023). Three-dimensional path planning of uav based on improved particle swarm optimization. *Mathematics*, 11(9).

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

Gad, A. G. (2022). Particle swarm optimization algorithm and its applications: a systematic review. *Archives of computational methods in engineering*, 29(5):2531–2561.

Ghaheri, A., Shoar, S., Naderan, M., and Hoseini, S. S. (2015). The applications of genetic algorithms in medicine. *Oman medical journal*, 30(6):406.

Giesbrecht, J. (2004). Global path planning for unmanned ground vehicles. page 56.

Haddad, M. N., Santos, A. C., Duhamel, C., and Coco, A. A. (2023). Intelligent drone swarms to search for victims in post-disaster areas. *Sensors*, 23(23).

Hayat, S., Yanmaz, E., Bettstetter, C., and Brown, T. (2020). Multi-objective drone path planning for search and rescue with quality-of-service requirements. *Autonomous Robots*, 44.

Hintze, J. L. and Nelson, R. D. (1998). Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184.

Janga Reddy, M. and Kumar, D. N. (2012). Computational algorithms inspired by biological processes and evolution. *Current Science*, 103:370–380.

Jia, J. and Wang, W. (2020). Review of reinforcement learning research. In *2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 186–191.

Kuffner, J. and LaValle, S. (2000). Rrt-connect: An efficient approach to single-query path planning. volume 2, pages 995–1001.

Lamini, C., Benhlima, S., and Elbekri, A. (2018). Genetic algorithm based approach for autonomous mobile robot path planning. *Procedia Computer Science*, 127:180–189. PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON INTELLIGENT COMPUTING IN DATA SCIENCES, ICDS2017.

Lamont, G. B., Slear, J. N., and Melendez, K. (2007). Uav swarm mission planning and routing using multi-objective evolutionary algorithms. In *2007 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making*, pages 10–20. IEEE.

McLain, T. (1999). Coordinated control of unmanned air vehicles.

Meng, Q., Chen, K., and Qu, Q. (2024). Ppswarm: Multi-uav path planning based on hybrid pso in complex scenarios. *Drones*, 8(5):192.

Millonas, M. M. (1993). Swarms, phase transitions, and collective intelligence. *arXiv preprint adap-org/9306002*.

Mitchell, M. (1995). Genetic algorithms: An overview. In *Complex.*, volume 1, pages 31–39. Citeseer.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Ni, H., Deng, X., Gong, B., and Wang, P. (2018). Design of regional logistics system based on unmanned aerial vehicle. In *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 1045–1051.

Pederi, Y. and Cheporniuk, H. (2015). Unmanned aerial vehicles and new technological methods of monitoring and crop protection in precision agriculture. In *2015 IEEE International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, pages 298–301.

Pervaiz, S., Ul-Qayyum, Z., Bangyal, W. H., Gao, L., Ahmad, J., et al. (2021). A systematic literature review on particle swarm optimization techniques for medical diseases detection. *Computational and Mathematical Methods in Medicine*, 2021.

Pham, H. X., La, H. M., Feil-Seifer, D., and Nefian, A. (2018). Cooperative and distributed reinforcement learning of drones for field coverage.

Phan, K. and Manjanna, S. (2012). Q-learning for markov decision processes.

Pinto, M. F., Melo, A. G., Marcato, A. L. M., and Urdiales, C. (2017). Case-based reasoning approach applied to surveillance system using an autonomous unmanned aerial vehicle. In *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, pages 1324–1329.

Puente-Castro, A., Rivero, D., Pazos, A., and Fernandez-Blanco, E. (2022a). A review of artificial intelligence applied to path planning in uav swarms. *Neural Computing and Applications*, 34.

Puente-Castro, A., Rivero, D., Pazos, A., and Fernandez-Blanco, E. (2022b). Uav swarm path planning with reinforcement learning for field prospecting. *Applied Intelligence*, 52.

Puente-Castro, A., Rivero, D., Pedrosa, E., Pereira, A., Lau, N., and Fernandez-Blanco, E. (2023a). Q-learning based system for path planning with unmanned aerial vehicles swarms in obstacle environments. *Expert Systems with Applications*, 235:121240.

Puente-Castro, A., Rivero, D., Pedrosa, E., Pereira, A., Lau, N., and Fernandez-Blanco, E. (2023b). Uav_swarms_rl_fixed_obstacles_maps. https://github.com/TheMVS/UAV_SWARMS_RL_FIXED_OBSTACLES_MAPS. Accedido 18-05-2024.

PX4 Autopilot (2024). About. https://px4.io/. Accedido 04-06-2024.

QGroudControl (2024). Qgroudcontrol. http://qgroundcontrol.com/. Accedido 04-06-2024.

Qi, C. (2011). Application of improved discrete particle swarm optimization in logistics distribution routing problem. *Procedia Engineering*, 15:3673–3677.

Qin, Y.-Q., Sun, D.-B., Li, N., and Cen, Y.-G. (2004). Path planning for mobile robot using the particle swarm optimization with mutation operator. In *Proceedings of 2004 international conference on machine learning and cybernetics (IEEE Cat. No. 04EX826)*, volume 4, pages 2473–2478. IEEE.

Reddy, M. J. and Kumar, D. N. (2020). Evolutionary algorithms, swarm intelligence methods, and their applications in water resources engineering: a state-of-the-art review.

Siegel, J. (2022). Microsoft lanza project airsim, una plataforma integral para acelerar el vuelo autónomo. https://news.microsoft.com/es-xl/features/microsoft-lanza-project-airsim-una-plataforma-integral-para-acelerar-el-vuelo-auto Accedido 04-06-2024.

Singh, A., Kumar, S., Singh, A., and Walia, S. (2020). *Parallel 3-Parent Genetic Algorithm with Application to Routing in Wireless Mesh Networks*, pages 1–28.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Barto, A. G., et al. (1999). Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134.

Tezza, D. and Andujar, M. (2019). The state-of-the-art of human–drone interaction: A survey. *IEEE Access*, 7:167438–167454.

The European Commission (2019). Commission implementing regulation (eu) 2019/947 of 24 may 2019 on the rules and procedures for the operation of unmanned aircraft. https://eur-lex.europa.eu/eli/reg_impl/2019/947/oj. Accedido 16-05-2024.

Tran, T. H. (2007). *Modelling and control of unmanned ground vehicles*. PhD thesis.

Tu, J. and Yang, S. (2003). Genetic algorithm based path planning for a mobile robot. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 1, pages 1221–1226 vol.1.

Unreal Engine (2024). Frequently asked questions. https://www.unrealengine.com/en-US/faq. Accedido 04-06-2024.

Vásárhelyi, G., Virágh, C., Somorjai, G., Nepusz, T., Eiben, A. E., and Vicsek, T. (2018). Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 3(20):eaat3536.

Vervoort, J. (2009). Modeling and control of an unmanned underwater vehicle. *Master Traineesh. Rep*, pages 5–15.

Vu, T. and Tran, L. (2020). Flapai bird: Training an agent to play flappy bird using reinforcement learning techniques. *CoRR*, abs/2003.09579.

Wang, D., Tan, D., and Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408.

Watkins, C. and Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, 8:279–292.

Wong, K.-C. (2015). Evolutionary algorithms: Concepts, designs, and applications in bioinformatics. *Nature-Inspired Computing: Concepts, Methodologies, Tools, and Applications*.

Yan, R.-j., Pang, S., Sun, H.-b., and Pang, Y.-j. (2010). Development and missions of unmanned surface vehicle. *Journal of Marine Science and Application*, 9:451–457.

Yoshida, E. and Kanehiro, F. (2011). Reactive robot motion using path replanning and deformation. pages 5456–5462.