TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Estudio y Análisis de Metaheurísticas modernas para el problema de Selección de Características

Autor

Miguel García López

Directores

Daniel Molina Cabrera

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Enero de 2024

Título del proyecto

Subtítulo del proyecto.

Autor
Nombre Apellido1 Apellido2 (alumno)

Directores
Nombre Apellido1 Apellido2 (tutor1)
Nombre Apellido1 Apellido2 (tutor2)

Estudio y Análisis de Metaheurísticas modernas para el problema de Selección de Características

MIGUEL GARCÍA LÓPEZ

Palabras clave: Metaheurística, selección, de características, binario, optimización, población, fitness, aprendizaje automático

Resumen

En el ámbito del Machine Learning, algunos algoritmos, como KNN o SVM, muestran excelentes resultados, pero muchas veces estos requieren de un procesamiento previo para identificar las características más relevantes. Ya sea por escoger un problema amplio y complejo, por falta de conocimiento o por falta de contexto a la hora de recolectar los datos, es posible que se llegue a aglomerar información de más, dando lugar a conjuntos de datos inmensos. Incluso aquellos algoritmos que internamente realizan esta identificación de características pueden beneficiarse de un procesamiento adicional. Este preprocesamiento, conocido como selección de características (features selection), se considera un problema complejo de optimización combinatoria.

Las metaheurísticas son algoritmos diseñados para resolver problemas de optimización complejos cuando los recursos son limitados. Aunque inicialmente se desarrollaron para abordar principalmente problemas combinatorios, en la actualidad se están proponiendo y aplicando cada vez más en problemas que implican variables continuas o reales. En respuesta a la creciente demanda en el campo de la selección de características, se han adaptado versiones especializadas de estas metaheurísticas para abordar este tipo de problemas combinatorios. Sin embargo, a pesar del número creciente de propuestas en este campo, las comparaciones objetivas entre ellas son limitadas. Aunque existen revisiones bibliográficas, muchas de ellas carecen de comparaciones adecuadas debido a la importancia y actualidad del problema de selección de características. Por lo tanto, hay una necesidad de estudios que proporcionen una evaluación comparativa más rigurosa y exhaustiva de las diferentes propuestas en este ámbito.

En este trabajo de carácter científico, se llevará a cabo una revisión bibliográfica de diversas metaheurísticas recientes para abordar el problema de selección de características. Se estudiarán e implementarán aquellas consideradas más prometedoras, con el objetivo de construir un repertorio amplio y variado de propuestas. Posteriormente, se realizará un estudio comparativo exhaustivo utilizando diversos algoritmos de machine learning y conjuntos de datos

representativos. Finalmente, se llevará a cabo un análisis crítico utilizando diversas métricas y valoraciones, como la tasa de acierto y el tiempo de ejecución, entre otras.

Study and Analysis of Modern Metaheuristics for the Feature Selection Problem

MIGUEL GARCÍA LÓPEZ

Keywords: Metaheuristic, feature selection, binary, optimization, population, fitness, machine learning

Abstract

In the field of Machine Learning, some algorithms like KNN or SVM often yield excellent results, but they often require preprocessing to identify the most relevant features. Whether due to tackling a broad and complex problem, lack of domain knowledge, or collecting data without proper context, it's possible to gather excessive information, resulting in vast datasets. Even algorithms internally performing feature identification can benefit from additional processing. This preprocessing, known as feature selection, is considered a complex combinatorial optimization problem.

Metaheuristics are algorithms designed to solve complex optimization problems when resources are limited. Initially developed mainly for combinatorial problems, they are increasingly proposed and applied to problems involving continuous or real variables. In response to the growing demand in feature selection, specialized versions of these metaheuristics have been adapted to tackle combinatorial problems. However, despite the increasing number of proposals in this field, objective comparisons between them are limited. Although there are literature reviews, many lack adequate comparisons due to the importance and timeliness of the feature selection problem. Therefore, there is a need for studies providing a more rigorous and exhaustive comparative evaluation of different proposals in this area.

In this scientific work, a literature review of various recent metaheuristics for feature selection will be conducted. The most promising ones will be studied and implemented to build a broad and varied repertoire of proposals. Subsequently, a comprehensive comparative study will be conducted using various machine learning algorithms and representative datasets. Finally, a critical analysis will be carried out using various metrics and assessments, such as accuracy rate and execution time, among others.

Yo, **Miguel García López**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77159865E, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Miguel García López

Granada a 29 de Enero de 2024.

D. **Daniel Molina Cabrera**, Profesor del Departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Estudio y Análisis de Metaheurísticas modernas para el problema de Selección de Características***, ha sido realizado bajo su supervisión por **Miguel García López**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 29 de Enero de 2024.

El director:

Daniel Molina Cabrera

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Definición del problema

El problema de la selección de características se define como el proceso de seleccionar un subconjunto de características relevantes [0]. Una característica es una propiedad individual medible de un fenómeno concreto. Este problema es considerado un problema **NP duro** [0]. La reducción de dimensionalidad, y con ello de características, suele ser necesario a la hora de crear un modelo predictivo por medio del aprendizaje automático ya que muchas de las características dentro de un conjunto de datos pueden no llegar a ser relevantes para solucionar aquellos problemas que se intentan solucionar, ya sea por que no aporta información, porque puede ser agrupada junto a otras tantas en una sola propiedad o incluso porque hay ruido en los datos, lo cual es inevitable [0].

Gracias a la reducción de características es posible mejorar tanto la capacidad de generalización como la precisión del modelo predictivo gracias a la reducción de *ruido*.

Siendo f la función objetivo a predecir, H^n el conjunto de hipótesis o conjunto de modelos de dimensión n posibles, $h^*(x)$ el mejor modelo aprendido y x una variable de entrada. El ruido conocido como ruido estocástico es aquel que atiende a una variación aleatoria que puede surgir de diversos factores, como mediciones imprecisas de señales o la falta de precisión en sensores. Por otro lado, el ruido determinista está directamente relacionado con la complejidad de un modelo. Su presencia aumenta la probabilidad de sobre-ajuste. El ruido determinista puede explicarse como la parte de la función f que el conjunto de hipótesis H^n no puede capturar, es decir, $f(x) - h^*(x)$. Este tipo de ruido se considera así porque la función (modelo) no es lo suficientemente compleja como para comprender esa parte. Este ruido depende de H^n y permanece constante para un valor dado de x [0].

La reducción de características ayuda a manejar ambos tipos de ruido [0] al simplificar el modelo, lo que puede reducir el impacto del ruido estocástico y disminuir la complejidad del modelo, lo que a su vez puede ayudar a mitigar el ruido determinista al mejorar la capacidad del modelo para capturar las características relevantes y descartar las irrelevantes. Esto puede conducir a una mejor capacidad de generalización y a una reducción del sobre-ajuste.

Además de la simplificación del modelo, que conduce a una reducción del ruido, la selección de características es un preprocesamiento necesario por varias razones:

1. Interpretabilidad: La presencia de características irrelevantes puede complicar innecesariamente la interpretación y el rendimiento de los modelos de aprendizaje automático [0]. La selección de un subconjunto relevante de características puede simplificar el modelo resultante, haciéndolo más comprensible y fácilmente interpretable.

2. Mejora de la eficiencia computacional: La reducción de la dimensionalidad puede conducir a un ahorro significativo en términos de tiempo y recursos computacionales necesarios para el entrenamiento y la evaluación de modelos. Al eliminar características irrelevantes, se reduce la complejidad del problema y se acelera el proceso de aprendizaje.

3. Evita la maldición de la dimensionalidad [0]: Cuando la dimensionalidad se incrementa en un problema, el volumen del espacio también lo hace, y esto ocurre tan rápido que hace que los datos disponibles se vuelvan dispersos. De forma que para obtener un resultado seguro/-fiable, la cantidad de datos necesitados debe verse incrementada de manera exponencial con la dimensionalidad [0]. A menor dimensionalidad (características en el conjunto de datos) menos datos harán falta para obtener un buen modelo.

En este trabajo, se lleva a cabo una investigación y análisis comparativo entre varios métodos de la familia **wrapper** o métodos de envoltura. Existen multitud de estrategias [0] que intentan dar solución a este problema. Los métodos de búsqueda más famosos son los de filtrado (**filter**), los cuáles seleccionan las características más discriminativas según la naturaleza de los datos [0]. Por lo general, estos métodos realizan la selección de características antes de las tareas de clasificación y agrupamiento. Ejemplos de algoritmos de filtrado son *relieF* [0] o F-statistic [0].

Los métodos **wrapper**, en cambio, utilizan el algoritmo de aprendizaje usado post-procesamiento para evaluar las características y seleccionar así las más útiles [0].

Los algoritmos clasificatorios de aprendizaje utilizados en este trabajo son

SVM [0] y *kNN* [0], siendo las máquinas de vectores de soporte un método robusto y eficiente y los vecinos más cercanos un método simple, interpretable y muy eficaz. Se analizará el resultado entre ambos clasificadores entre otros muchos análisis comparativos.

1.1.1. Motivación

El reciente interés del problema de la selección de características en el ámbito de las metaheurísticas en los últimos años es más que evidente. Puede comprobarse como en los últimos años hay una tendencia en la publicación de artículos presentando nuevos métodos metaheurísticos, mejores con respecto a los clásicos o incluso comparativas y análisis entre distintos algoritmos.

Esta crecimiento viene acompañado, sin embargo, de comparaciones que distan de ser objetivas por varios motivos. Entre varios artículos se comparan algoritmos del mismo tipo con soluciones y resultados muy variables entre sí a pesar de mismas configuraciones a la hora de experimentar, artículos sin código referenciado, de forma que sea más fácil interpretar los resultados o duplicarlos, y algoritmos novedosos presentados por su autor o autores que superaban al resto en alguna métrica concreta sin llegar a la rigurosidad adecuada.

Por ello, la motivación principal de este trabajo es la de proveer información no sesgada y todo lo objetiva posible por medio de un análisis comparativo entre los algoritmos optimizatorios metaheurísticos más populares y más citados junto con los algoritmos más robustos y clásicos en el campo de la optimización pseudo estocástica.

1.1.2. Objetivos

Objetivo General:

Realizar una comparación exhaustiva y objetiva de diversas metaheurísticas utilizadas en la selección de características, con el propósito de proporcionar una visión integral y evaluativa sobre su eficacia y aplicabilidad en diferentes contextos de análisis de datos.

Objetivos Específicos:

1. Evaluar el desempeño de las metaheurísticas más relevantes en el ámbito de la selección de características, analizando métricas clave como precisión, estabilidad de las soluciones y eficiencia computacional.

Se emplearán conjuntos de datos de referencia y metodologías de validación cruzada para garantizar la robustez de los resultados.

2. Investigar la transferibilidad de las técnicas diseñadas para dominios continuos y binarios en el contexto de la selección de características. Se analizará si las metaheurísticas efectivas en un dominio son igualmente eficaces cuando se aplican a otro, identificando posibles ventajas y limitaciones de cada enfoque.

3. Identificar las fortalezas y debilidades de cada metaheurística según el tipo de representación de las características. Se realizará un análisis detallado del comportamiento de las técnicas en problemas de selección de características con diferentes tipos de datos, destacando su rendimiento relativo y sus áreas de aplicación más adecuadas.

4. Proporcionar recomendaciones prácticas basadas en los resultados obtenidos, con el objetivo de orientar a practicantes y académicos en la selección y aplicación de metaheurísticas en problemas reales de selección de características.

5. Evaluar los resultados de las metaheurísticas en problemas de selección de característica usando distintos como algoritmos de aprendizaje los métodos *kNN* y *SVM*. Se realizará una comparativa a nivel de eficiencia en tiempo, estabilidad y calidad de los resultados.

1.1.3. Planificación

Un trabajo de fin de grado consta de 12 créditos ECTS, donde se estima que cada crédito debe valer unas 25 horas de trabajo aproximadamente. Teniendo en cuenta estos datos, se calcula que la duración del TFG no debería ser superior a 300 horas. Ha de tenerse en cuenta también que el alumno trabaja 25 horas semanales y debe superar algunas asignaturas además de su proyecto final para terminar la carrera. Por lo tanto, el proyecto se planifica con una duración extendida en el tiempo, pero con una carga de trabajo semanal menos intensiva.

Se planifica una duración de 5 meses aproximadamente. Se utilizará un diagrama de Gantt [0] para describir la planificación del proyecto, de manera que se realizarán tareas en un orden cronológico. Sin embargo, se reconoce que algunas tareas probablemente requerirán iteraciones posteriores, ya que es probable que se mejore y perfeccione el proyecto a lo largo de su ciclo de vida.

Las fases del ciclo de vida son:

- **Investigación inicial** Esto incluye investigar sobre conceptos básicos ya aprendidos, en forma de repaso sobre conceptos generales de

aprendizaje automático, tipos de metaheurísticas, tipos de codificación, optimización de funciones, test estadísticos y conceptos básicos, código Python y librerías asociadas, instalación de estas a partir de un entorno virtual, configuración del entorno de trabajo e investigación sobre el problema de selección de características.

- **Diseño del software**: Planificación de la estructura general del código, uso de patrones de diseño que puedan ser de utilidad de cara a al mantenimiento del software a lo largo del desarrollo, concepto de modularización inicial del código (estructura del proyecto), uso de entornos virtuales.

- **Investigación metaheurísticas**: Realización de un estudio más exhaustivo acerca de las metaheurísticas a implementar y sus diferentes versiones binarias. Esto incluye un listado de 12 metaheurísticas, siendo estas:

  - Binary Firefly Algorithm
  - Binary Whale Optimization Algorithm
  - Binary Bat Swarm Optimizer
  - Binary Grey Wolf Optimizer
  - Binary Dragonfly Algorithm
  - Binary Grasshoper Algorithm
  - Binary Cuckoo Search
  - Binary Differential Algorithm
  - Ant Colony Optimization
  - Binary Artificial Bee Colony Optimization
  - Binary Particle Swarm Optimization
  - Genetic Algorithm (binary & real)

  De cada una de ellas se investigará su inspiración, funcionamiento, implementación y versiones binarias, normalmente asociadas al problema de selección de características.

- **Implementación del software**: Una vez claros los requisitos programáticos quedan establecidos, se implementará el software base. Esto incluye código en Python para la generación de gráficas, manejo de datasets en formato *arff*, codificación de los algoritmos metaheurísticos en versión binaria, implementación de función objetivo (*fitness*) y parametrización del programa para distintas pruebas.

- **Pruebas y refactorizado**: En esta etapa se llevarán a cabo pruebas exhaustivas para verificar la robustez y eficacia de los diferentes algoritmos implementados. Además, se considerará la refactorización del código si es necesario, con el fin de mejorar su estructura, claridad y mantenibilidad.
- **Análisis de resultados**: En esta fase se recopilarán datos de la ejecución de los algoritmos en sus diferentes versiones, así como entre ellos, utilizando los conjuntos de datos seleccionados para el proyecto. Esta recopilación de métricas permitirá una evaluación del rendimiento y la eficacia de cada algoritmo en comparación con los demás, así como su comportamiento en diferentes conjuntos de datos.
- **Documentación**: En esta etapa final se generará una documentación del proyecto que incluirá de forma general la descripción del problema, los objetivos, planificación, implementación, resultados y pruebas.

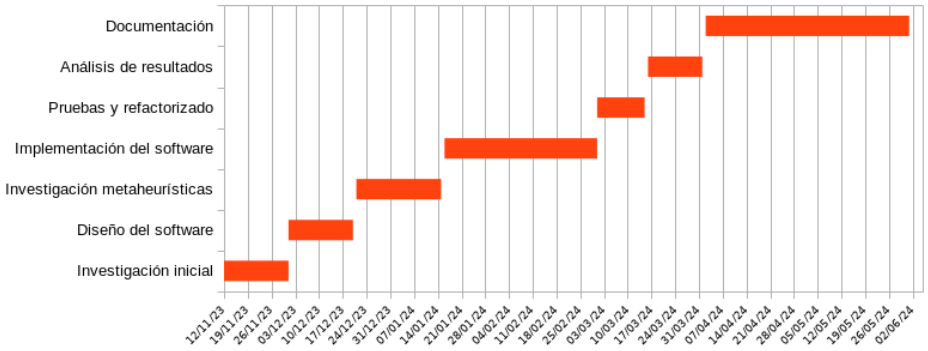

Figura 1.1: Diagrama de Gantt inicial

La planificación inicial tiene en cuenta un curso ideal del ciclo de vida del proyecto, siendo las etapas más extensas la de creación del software e implementación de la documentación. Son etapas excluyentes, no pueden ocurrir a la vez según este tipo de planificación. Al terminar una etapa se pasa inmediatamente a la siguiente.

La planificación final del proyecto se ha modificado significativamente debido a una serie de contratiempos y obstáculos surgidos durante su desarrollo, así como la influencia de numerosos eventos externos que han afectado a su cronograma. En particular, se han experimentado retrasos y bloqueos que han incidido en la duración prevista del proyecto. Por ejemplo, las etapas de implementación del software y la investigación de las metaheurísticas se han entrelazado debido a que la implementación efectiva del algoritmo se facilitaba una vez que se había estudiado a fondo la metaheurística correspondiente.

Esto ha llevado a una re-evaluación de la estrategia de planificación original, reconociendo que no habría sido eficiente estudiar todas las metaheurísticas simultáneamente y luego proceder con su implementación.

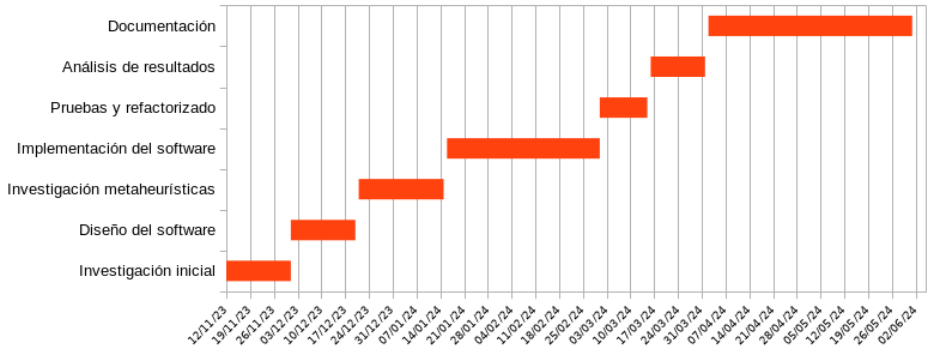

Figura 1.2: Diagrama de Gantt final

El coste estimado del proyecto se divide en varios sub-costes:

- **Sueldo**: Considerando un precio de proyecto que deba cubrir gastos y compensar, se estima un salario por investigador de 52.000€ al año, es decir, 25€/hora. Al durar el proyecto aproximadamente 8 meses, el salario final total es de 34.000€.

- **Ordenador portátil**: El estudiante consta con un portátil HP HP Pavilion Laptop 14-ec0xxx como herramienta principal de trabajo, con un precio de 1000€. Ha sido usado durante unos 8 meses aproximadamente. Si se tiene en cuenta que la vida media de un portátil es de aproximadamente $4 - 5$ años [0] entonces el costo del ordenador sería de 133.3€.

- **Servidor** Para la experimentación del proyecto se ha hecho uso de un servidor para ejecutar las distintas pruebas de los algoritmos. Se estima el precio por mes y total dada una estimación de los servicios de *Google Cloud - Compute Engine*.

Name	Quantity	Region	Service ID
Compute optimized Core running in London	150.0	europe-west2	6F81-5844-456A
Compute optimized Ram running in London	600.0	europe-west2	6F81-5844-456A
Storage PD Capacity in London	0.2	europe-west2	6F81-5844-456A

Tabla 1.1: Costos servidor Google Cloud - Part 1

SKU	Total Price (USD)
271A-7F2A-C5C5	6.56775
0BD6-E233-D705	3.5202
BF1A-6647-009D	0.0096
Total Price:	10.09755

Tabla 1.2: Costos servidor Google Cloud - Part 2

En la primera tabla, se detallan tres servicios distintos implementados en la región de Londres. Cada servicio se identifica por su nombre, cantidad (indicando la cantidad de recursos utilizados), región de implementación y un identificador único del servicio. Lo importante es que se ha calculado un servicio con 30 vCPUs para paralelizar los trabajos.

La segunda tabla complementa la primera proporcionando detalles adicionales sobre los servicios en términos de SKU (Stock Keeping Unit, una identificación única para un producto) y el precio total en dólares estadounidenses asociado con cada uno de ellos. El total, hecho el cambio de divisa a euros, es de 9.48€ al mes, es decir, 75.84€ en total (calculado para 8 meses). Dado este desglose, se calcula el coste final del proyecto:

Item	Costo (€)
Salario	34.000
Ordenador portátil	133.3
Servidor CPU - GC Compute Engine	75.84
Total	**34.209,14**

Tabla 1.3: Costo estimado del proyecto

Capítulo 2

Fundamentos Teóricos

En este capítulo se describirán aquellos conceptos teóricos fundamentales para comprender el trabajo realizado en este proyecto.

2.1. Optimización

La optimización es un campo de estudio que trata, mediante el uso de las adecuadas herramientas matemáticas, de maximizar o minimizar una función objetivo. Esto significa, obtener la mejor solución posible para un problema dado dentro de un conjunto de alternativas y normalmente sujeto a una serie de restricciones que hacen de una solución satisfacible. Es un área interdisciplinar que aborda desde campos tales como la Economía, Ingeniería, Biología y muchas otras tantas disciplinas.

La optimización es una disciplina arraigada en la naturaleza humana. Este impulso innato hacia la optimización ha llevado al desarrollo de diversas metodologías y técnicas a lo largo de la historia, desde los rudimentarios métodos de prueba y error hasta los sofisticados algoritmos de optimización computacional utilizados en la actualidad.

2.1.1. Definición general de un problema de optimización

Para poder convertir un problema abstracto en un problema de optimización concreto, con el que se pueda trabajar, es necesario establecer ciertos elementos fundamentales que lo definan de manera precisa y clara. En general, un problema de optimización se expresa de la siguiente forma [0]:

Función objetivo a minimizar	$f(x)$	(2.1a)
Sujeto a		
s restricciones de desigualdad	$g_i(x) \leq 0, \quad j = 1, 2, ..., s$	(2.1b)
w restricciones de igualdad	$h_j(x) = 0, \quad j = 1, 2, ..., w$	(2.1c)
Donde el número de variables es dado por	$x_i, \quad i = 1, 2, ..., n$	

La definición general de un problema de optimización proporciona una estructura sólida para abordar el problema abstracto. Establece la función objetivo que se busca minimizar o maximizar, junto con las restricciones que deben cumplirse. Estas restricciones pueden ser tanto desigualdades como igualdades, y todas juntas definen el **espacio de búsqueda** del problema.

2.1.2. Función objetivo y función fitness

Ambos términos, aunque a menudo se utilizan como sinónimos, desempeñan roles distintos en el ámbito de la optimización. La función *objetivo*, como su nombre indica, establece el objetivo a alcanzar en la resolución del problema. Esta función cuantifica el rendimiento de las soluciones encontradas en relación con el objetivo específico del problema, que puede ser maximizar ganancias, minimizar distancia, entre otros objetivos. Por otro lado, la función de *fitness* evalúa la idoneidad de una solución dentro de una población de soluciones. Es decir, determina la calidad relativa de la solución respecto a otras alternativas.

Debe destacarse que la métrica de la función de *fitness* suele ser estrictamente positiva, ya que representa la calidad de una solución, generalmente en un rango de valores entre 0 y 1. Por otro lado, la función objetivo puede ser positiva o negativa, dependiendo de si se busca maximizar o minimizar el objetivo. Además, la función de aptitud también puede llegar a ser una aproximación de la función objetivo, pero no necesariamente coinciden exactamente.

Resumiendo, podría decirse que la función *fitness* es un tipo particular de función objetivo que se utiliza como métrica de rendimiento [0].

2.1.3. Puntos globales o locales

Se conocen como punto de óptimo global (mínimo o máximo) la solución (o vector hablando en términos matemáticos) cuyo valor para la función

objetivo es el más grande en todo el espacio de soluciones posible, es decir, el espacio de búsqueda. Los puntos locales en cambio son varios, no solo uno como es el global. Son soluciones máximas o mínimas dentro de una región de soluciones.

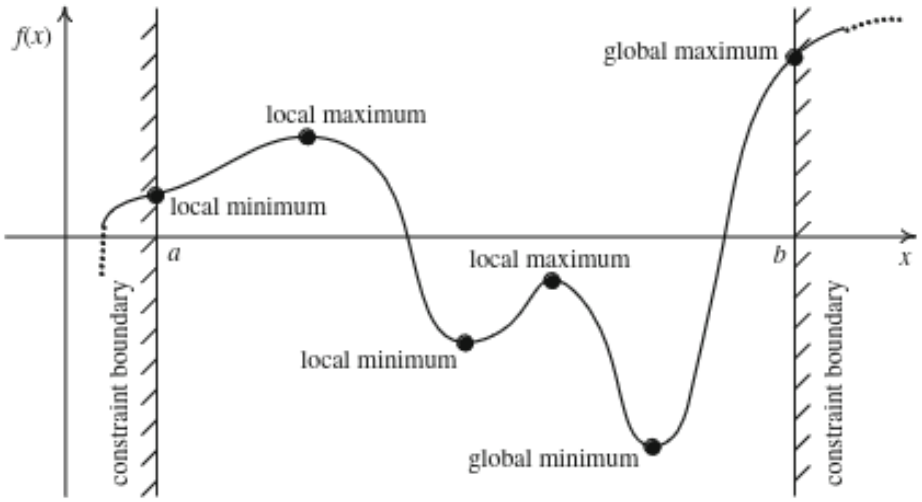

Figura 2.1: En esta figura extraída de [0] puede observarse de manera intuitiva la diferencia entre punto global y puntos locales. El máximo global es el valor más alto para $f(x)$ en el espacio, mientras que un máximo global solo representa el valor más alto dentro de un "vecindario"

Sea $f(x)$ una función a maximizar y x^* una solución óptima. Un objetivo $G(x)$ está en su máximo global sí y solo si [0]:

$$f(x^*) \geq f(x) \quad \forall x \tag{2.2}$$

En cambio el objetivo está en un máximo local en el punto x^* si:

$$\begin{aligned} f(x^*) \geq f(x) \quad & \forall x \\ & \text{dentro de un vecindario de } x^* \text{ [0]} \end{aligned} \tag{2.3}$$

2.2. Teoría de la computación

La teoría de la computación es un campo que estudia los fundamentos matemáticos y los límites de la computación. En esencia, se ocupa de preguntas fundamentales sobre qué se puede y qué no se puede computar, cómo se pueden resolver problemas de manera eficiente y qué tan difíciles son ciertos problemas en términos de recursos computacionales.

Suele usarse la máquina de Turing, como modelo abstracto de computadora, para las demostraciones realizadas, creando un marco sencillo en el que entender cómo se ejecutan los algoritmos y cómo se resuelven estos.

2.2.1. Máquina de Turing

Una máquina de Turing, cuyo nombre proviene de su creador Alan Turing, es un modelo de computación matemático que es capaz de ejecutar algoritmos mediante el control de una serie de símbolos en la tira de una cinta y de acuerdo a una serie de reglas establecidas [0].

Es capaz de general un modelo abstracto mediante el cual representar todo de algoritmos que resuelvan problemas computables.

2.2.2. Grupos de complejidad P y NP

Se puede definir un lenguaje como una serie de cadenas dentro de un alfabeto, siendo este último un conjunto finito de símbolos [0].

Dada esta definición se dice que un lenguaje que es *reconocible*, es decir, que una máquina de Turing puede determinar si una cadena pertenece al lenguaje o no, en un tiempo "polinómico determinista"son conocidos como problema de la clase P. Que un problema sea resoluble en tiempo polinómico determinista en una máquina de Turing, significa que para cada estado solo existe una posible acción y que su tiempo de resolución puede expresarse como una función polinómica [0].

En contraposición, un problema de la clase NP no es resoluble en tiempo polinómico determinista, sino en tiempo polinómico no determinista. Esto es, que puede resolverse expresando el tiempo como una función polinómica, pero con la diferencia de que el no determinismo implica que para pasar de un estado a otro es posible considerar múltiples acciones.

La diferencia entre los problemas de las clases P y NP es fundamental en la teoría de la complejidad computacional. Mientras que los problemas de la clase P son eficientemente resueltos, los problemas de la clase NP pueden ser verificados en tiempo polinómico, pero su resolución en tiempo polinómico no está demostrada y tiende a ser exponencial.

2.2.3. NP-Hard

Un problema de decisión H se considera NP-Hard o duro si, para cada problema L en NP, hay una reducción de muchos a uno de tiempo polinómico de L a H [0]. Esto significa que cualquier problema en NP puede ser

transformado en H en tiempo polinómico, lo que sugiere que H es al menos tan difícil como cualquier problema en NP.

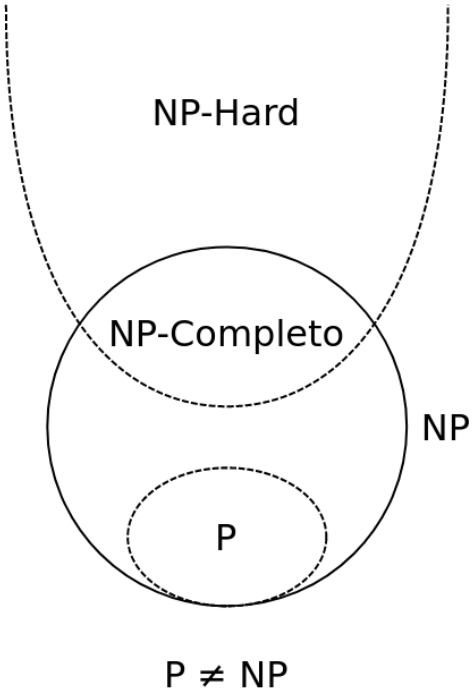

Figura 2.2: En esta figura se puede ver la relación de pertenencia entre clases de complejidad. Puede observarse que NP-Hard no es un subconjunto de NP. Una solución para un problema NP-Hard podría transformarse mediante reducción y solucionar un problema NP, pero no siempre es así al revés.

2.3. Selección de características

La selección de características es un ejemplo de problema NP-Hard y uno de los problemas más importantes en el mundo de la inteligencia artificial, más concretamente en el **machine learning** o aprendizaje automático.

2.3.1. Necesidad y motivo

El aprendizaje automático se basa en los datos como fuente de aprendizaje. Es indispensable tener un buen conjunto de datos para poder obtener un

modelo robusto y preciso. Por norma general o como se diría en inglés *"as a rule of thumb"*, a más grande el conjunto de datos, mayor calidad del modelo. De hecho hay una correlación inmediata con esta sentencia.

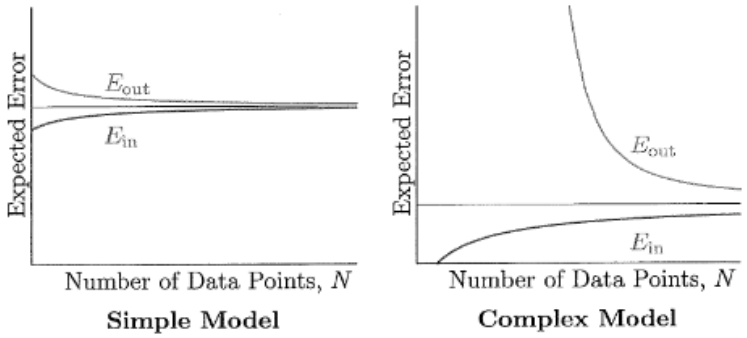

Figura 2.3: Esta figura extraída de [0] visualiza la relación directa entre calidad del modelo (menos error esperado) y número de datos que se le proporciona al algoritmo de aprendizaje.

Es inmediato pensar que a mayor cantidad de datos, mejor calidad del modelo, y así es. Sin embargo, también existe una "normal general" que relaciona la complejidad del modelo y su capacidad de generalización. Es cierto que un modelo muy complejo (muchos parámetros) es capaz de ajustar mejor funciones más complejas, pero dentro de un conjunto con una serie de modelos suficientemente complejos, suele ser mejor idea elegir el más simple. Hay una serie clara de ventajas para ello:

1. Menor sensibilidad al sobre-ajuste.
2. Mayor interpretabilidad del modelo y sus resultados.
3. Mayor eficiencia y por tanto menor tiempo de ejecución y menor peso en memoria.

De hecho, como puede observarse en la figura 2.3, el modelo más complejo tiene un error esperado mucho mayor que el simple. El E_{out} o error fuera de la muestra (error de generalización) es mucho más abrupto, es decir, generaliza peor.

Por supuesto, con suficientes datos, con un N (número de puntos) suficientemente grande, la tendencia del error es a la baja [0]. Ocurre, sin embargo, que la recolección, limpieza y transformación de datos es una tarea compleja,

por ello es mejor ceñirse, de nuevo, a el modelo más pequeño que obtenga una solución con suficiente calidad.

2.3.2. Concepto

El funcionamiento y motivo es explicado ya en la definición del problema en 1.1. Por evitar redundancias se procede a explicar el concepto de manera abreviada y puntualizando en los apartados más importantes.

La selección de características es una conocida y necesaria técnica de preprocesamiento de datos para la construcción de un modelo de aprendizaje. Su función es escoger un subconjunto óptimo de características dentro del conjunto inicial, de forma que la complejidad del modelo se reduzca.

Este problema es del tipo NP-Hard, como ya se ha mencionado previamente en 1.1 y explicado en 2.2.2. La selección de características ayuda a una serie de factores como son la interpretabilidad, mejora de la eficiencia temporal y espacial del modelo y sobre todo con la **maldición de la dimensionalidad** [0].

2.3.3. Maldición de la dimensionalidad

El término fue acuñado por Bellman en 1961 [0]. Este fenómeno ocurre cuando la dimensionalidad de los datos es muy grande. En un espacio de características de alta dimensión, es común que los datos estén muy dispersos, lo que significa que hay muy pocos datos en comparación con la cantidad posible de características. Esto dificulta que los modelos representen correctamente todo el espacio de características [0].

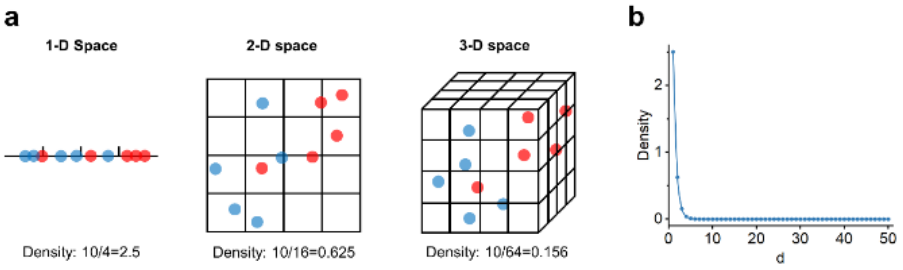

Figura 2.4: Esta figura extraída de [0] muestra la tendencia en la densidad a medida que se incrementa la dimensionalidad del espacio. En (a) se muestra la densidad con diez puntos de datos en un espacio $1D$, en (b) se muestra la tendencia al incrementar las dimensiones.

Además, en espacios de alta dimensión, la medición de distancias puede volverse inválida. Esto se debe a que las distancias entre puntos de datos diferentes tienden a converger a un mismo valor a medida que aumenta la dimensionalidad. Esto significa que las medidas de distancia ya no son tan útiles para medir la similitud entre datos [0]. En el trabajo de Beyer y compañeros [0] se observó como esto ocurría y por tanto un escaneo lineal (recorrer los puntos uno a uno) resultaba en altas dimensiones más práctico que otras técnicas complejas.

2.4. Metaheurísticas

Dentro de la optimización hay muchos tipos de métodos, dentro de los pseudoaleatorios pueden encontrarse las metaheurísticas. Estas son algoritmos basados en una abstracción de mayor nivel de la **heurísticas**. Mientras que las heurísticas se apoyan en el conocimiento específico del campo en el que se encuentra el problema, y están restringidas a su dominio, las metaheurísticas son aplicables a todo tipo de problemas, independientemente de su área de optimización [0]. Es cierto que hay algoritmos que son más convenientes para ciertos problemas que otros, pero su aplicación es generalizada. Normalmente, las metaheurísticas son diseñadas a siguiendo una inspiración en la naturaleza, ya sea en fenómenos físicos o en el comportamiento animal. Ejemplo de estos son algoritmos como el *Búsqueda Cuckoo*, *Enfriamiento Simulado* o incluso *Algoritmos Genéticos*.

Las metaheurísticas son especialmente útiles en problemas cuya resolución no es factible debido a altos costos computacionales, ya sea porque es posible analíticamente pero computacionalmente costoso, o porque el problema no es abordable mediante algoritmos convencionales. Son capaces de encontrar óptimos locales lo suficientemente aceptables, soluciones no óptimas, pero si muy buenas [0].

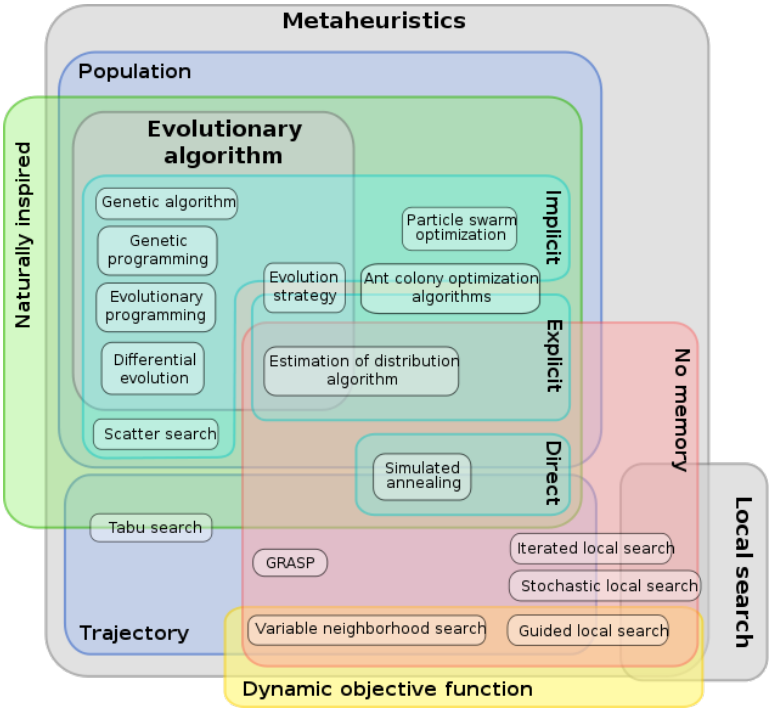

Figura 2.5: En esta figura de los autores Johann "nojhan" Dréo y Caner Candan, se clasifican las distintas metaheurísticas por inspiración.

2.4.1. Exploración vs explotación

Las metaheurísticas, y algoritmos basados en técnicas pseudoaleatorias, deben tener un balance entre sus factores explorativos y explotativos. De no ser así, los algoritmos tendrían tendencias a la convergencia temprana, dejando de lado mucho espacio por explorar y por ende soluciones posiblemente mejores (estancamiento en óptimos locales) o serían muy lentos en la convergencia hacia una solución.

La **exploración** se refiere a la habilidad del algoritmo de buscar nuevas y diversas regiones en el espacio de búsqueda/solución. Es una característica de la búsqueda global, que también puede ser llamada *diversificación*. En cambio, la explotación es la habilidad de la búsqueda de explotar las mejores soluciones encontradas hasta el momento y mejorarlas localmente, dentro de un "vecindario" [0].

No existe un equilibrio *de facto* entre exploración y explotación en las metaheurísticas. Aún no se ha alcanzado una respuesta definitiva a esta cuestión. Desde una perspectiva de sistemas, una metaheurística puede entenderse como un sistema dinámico compuesto por numerosos "individuos"

que interactúan entre sí. Estos individuos representan las distintas soluciones o posiciones en el espacio de búsqueda que la metaheurística explora [0]. Las interacciones entre estos individuos son las que conforman comportamientos exploratorios o explotativos y dependen del problema a solucionar y del propio algoritmo su equilibrio.

2.5. Teorema No Free Lunch

El Teorema de "No Free Lunch" (NFL) establece que, en promedio, ningún algoritmo de búsqueda puede superar a otros algoritmos en la búsqueda de todas las funciones objetivo posibles. En otras palabras, no existe un algoritmo universalmente óptimo que pueda dominar en todos los problemas de búsqueda. Esto implica que, si un algoritmo es efectivo para un conjunto particular de problemas, es probable que no lo sea para otros [0].

Relacionar este teorema con las metaheurísticas implica reconocer que no hay una única metaheurística que sea la mejor para todos los problemas de optimización. Cada problema puede tener características únicas que lo hacen más o menos adecuado para ciertas metaheurísticas. Por lo tanto, en lugar de buscar una solución universal, las metaheurísticas se centran en explorar y explotar diferentes áreas del espacio de búsqueda para encontrar soluciones aceptables o incluso óptimas para problemas específicos.

Pese a ello, las suposiciones de este tipo de teoremas, como conjuntos de datos extraídos de una distribución uniforme sobre todos los conjuntos de datos posibles, están completamente desalineadas con el mundo real, donde los datos suelen ser altamente estructurados y no están uniformemente muestreados [0].

De esta forma y faltando evidencia concluyente al respecto, es interesante mencionar el **NFL**, pero sin llegar a negar la posibilidad de nuevas y más precisas interpretaciones.

2.6. Aprendizaje automático

El aprendizaje automático es una sub-rama de estudio de la inteligencia artificial o **IA**, la cual aglomera una serie de métodos que pueden, de manera automática, detectar patrones en conjuntos masivos de datos para predecir datos futuros [0]. El aprendizaje automático o *machine learning* en inglés, es usado en una amplia variedad de campos por su utilidad trasversal. Algunos de ellos son la agricultura, marketing, videojuegos, meteorología, física, etc.

Los algoritmos de aprendizaje automático son capaces de encontrar patrones en los datos, como ya se ha mencionado. Por ello, es necesario nutrir a estos

algoritmos con datos de calidad. Un modelo de aprendizaje automático será tan bueno como los datos que se le puedan proveer, no más. De esta forma la recoleción de datos y su procesamiento se convierten en una prioridad a la hora de crear modelos.

Hay muchas formas de estructurar los datos y muchos tipos de algoritmos acorde a estos "inputs". En este documento se tratará con información en formato tabular, es decir, información en tablas con filas y columnas, donde cada fila representa un registro de información y cada columna una característica asociada. Esta última es la que determinará la complejidad del modelo.

2.6.1. Aprendizaje supervisado

Este subtipo de aprendizaje automático se caracteriza por tener un conjunto de datos sobre los que se entrena y una salida esperada (etiquetas) para cada punto de los datos [0]. Es bastante costoso porque etiquetar cada dato es una tarea laboriosa y poco escalable.

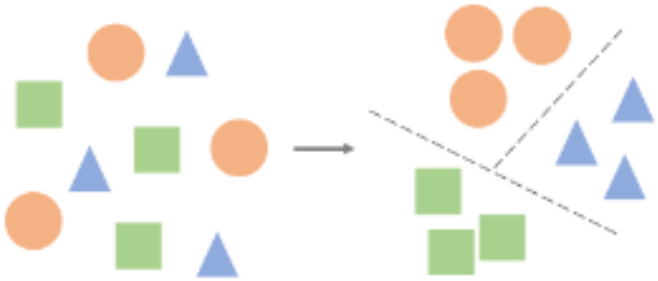

Figura 2.6: Figura extraída de [0] que muestra un ejemplo de aprendizaje supervisado.

2.6.2. Aprendizaje no supervisado

Cuando los datos solo vienen en forma de entrada y no se tiene ninguna salida (no hay etiquetas) entonces se trata de un aprendizaje no supervisado. Los algoritmos de este tipo se basan en la diferenciación de los datos mediante los patrones subyacentes que puedan encontrar. Son comunes en el aprendizaje no supervisado los algoritmos de *clustering* [0].

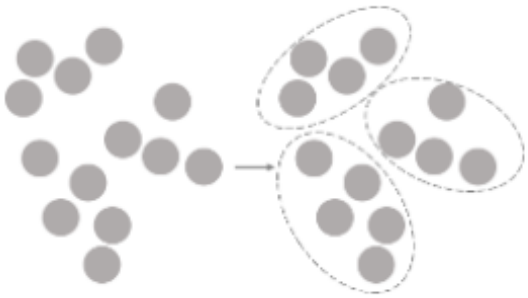

Figura 2.7: Figura extraída de [0] que muestra un ejemplo de aprendizaje no supervisado. Se trata de dividir en clústers sin información de etiquetas.

2.6.3. SVM

Las máquinas de vectores de soportes o *SVM* fueron introducidas por primera vez en [0]. En este documento será uno de los algoritmos de clasificación usados en la función *fitness* para cuantificar la calidad de los pesos aprendidos por los algoritmos optimizadores.

Las máquinas de vectores de soporte son algoritmos cuya principal característica se basa en la creación de un hiperplano o conjunto de hiperplanos en un espacio n-dimensional [0]. Este hiperplano o hiperplanos es elegido de manera que maximice el margen entre los puntos de datos de todas las clases a predecir. El margen es la distancia entre el hiperplano y los puntos de datos más cercanos de cada clase, estos puntos se denominan vectores de soporte. De esta manera se consigue minimizar el error de generalización [0].

La idea más intuitiva es, que al haber más espacio entre los puntos de distintas clases, hay más posibilidad de que los puntos no vistos, los puntos a predecir, caigan en zonas correctas.

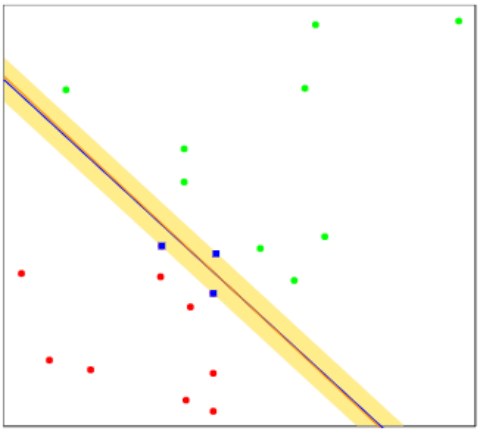

Figura 2.8: Figura extraída de [0] en la que se muestra de forma gráfica en un espacio 2-dimensional el margen óptimo de separación de dos clases.

2.6.4. K-nearest neighbors

El algoritmo de k vecinos más cercanos fue presentado en [0] por Evelyn Fix y Joseph Hodges, y más tarde ampliado en [0] por Thomas Cover.

Puede ser utilizado para regresión o clasificación, pero su uso suele darse más en este último problema. Este algoritmo ofrece una premisa sencilla, la clase de un punto p dependerá de cuál sea la clase mayoritaria dentro del subconjunto de k vecinos más cercanos [0].

Valores mayores de k incrementan la varianza, pero decrementan el sesgo (equilibrio sesgo-varianza [0]). De forma inversa, a menor k mayor sesgo y menor varianza.

Otra forma de verlo es que si $k = 1$ (el valor mínimo), el modelo entrenado será muy complejo, pues cada punto puede variar de clase por mínimo que sea el cambia en su entorno. Es un modelo que tiende a sobre-ajustar. Con $k = n$ el modelo es lo más simple posible, al tener en cuenta absolutamente todos los puntos, la clasificación será siempre la de la clase más repetida.

Capítulo 3

Estado del arte

La selección de características es un problema cuya popularidad ha ido en aumento con el paso de los años. No es algo casual, pues la cantidad de datos recogidos para tareas de aprendizaje automático y áreas derivadas, como el aprendizaje profundo, ha ido en aumento de manera casi exponencial.

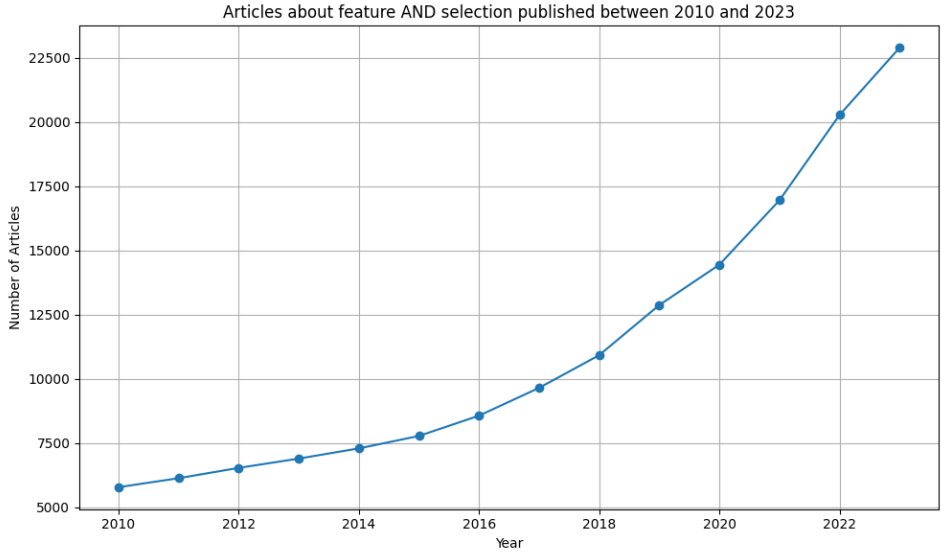

Figura 3.1: En esta figura se muestra el número de artículos publicados relacionados con la selección de características. Se ha usado el buscador Scopus para estos resultados.

También se puede observar una tendencia prácticamente igual en la popularidad sobre los años del problema de selección de características, pero abordado con técnicas metaheurísticas.

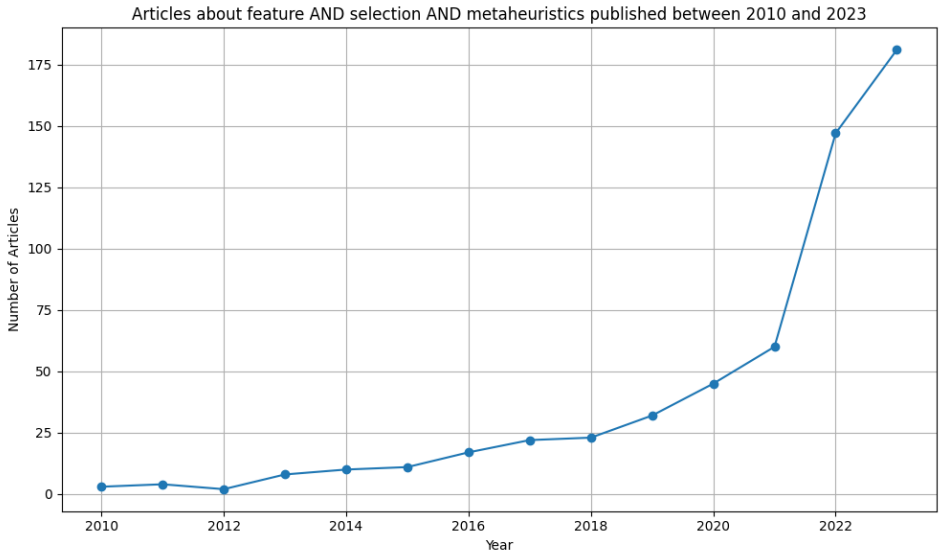

Figura 3.2: De igual forma que en la figura 3.1 las mateheuristicas han ido muy ligadas a la resolución de este problema. Se ha usado el buscador Scopus para estos resultados.

Y es que las metaheurísticas son algoritmos muy convenientes en la resolución de este tipo de problemas. Algoritmos analíticos no son válidos cuando el conjunto de datos tiene un número de características muy elevado debido a una cantidad de combinaciones simplemente inabordable en términos computacionales. Debido a ello, se suelen utilizar algoritmos más rápidos cuya solución no es óptima en términos globales, pero si suficientemente buena.

Muchos algoritmos metaheurísticos han sido propuestos a lo largo de los años. Se han escogido los más destacables en cuanto a resultados y más mencionados entre todos ellos, dando una selección de algoritmos metaheurísticos modernos de, en principio, muy alta calidad. No solo son seleccionados algoritmos modernos, sino que también se han escogido una serie de algoritmos más "clásicos", pero cuya aplicación es más extendida y con resultados que, de forma empírica, han demostrado ser más que buenos a lo largo de décadas de uso.

Algoritmos
GWO
FA
GOA
WOA
DA
CS
BA

Tabla 3.1: Algoritmos modernos

Algoritmos
PSO
GA
ACO
DE
ABCO

Tabla 3.2: Algoritmos clásicos

Para aclarar ciertas cuestiones referentes a las soluciones en este tipo de algoritmos estocásticos de tipo evolutivo, si no se especifica un tipo de representación se da por hecho que las soluciones se representan como vectores en un espacio n-dimensional. Un conjunto de individuos/vectores dan como resultado una población. La población es una matriz de $N \times M$, donde N es el número de filas o número de individuos en la población y M es el número de características del problema (M dimensiones). Algunos algoritmos admiten codificación binario y otros continua, ese tipo de cuestiones si serán especificadas.

3.1. Algoritmos genéticos

3.1.1. Introducción

Los algoritmos genéticos están inspirados en la selección natural y se utilizan tanto en problemas de optimización con restricciones como sin ellas. Es una metaheurística que modifica una población de soluciones individuales de manera repetida, seleccionando soluciones "padre" que dan lugar a la siguiente generación de soluciones en la siguiente iteración del algoritmo. En su forma más básica, un algoritmo genético opera sobre una población de soluciones potenciales a un problema dado. Cada solución potencial, a menudo llamada individuo o cromosoma, está representada como una cadena de símbolos, que puede ser binaria, numérica o simbólica [0].

3.1.2. Funcionamiento y Operadores

Las soluciones o **fenotipos** están compuestas por una serie de propiedades, características, cromosomas o **genotipos**, que pueden ser mutados y alterados. La representación más común es la binaria, pero otras codificaciones son posibles.

El proceso típico incluye [0]:

- Inicialización aleatoria de la población.
- Evaluación del *fitness* de cada individuo.
- Selección de padres basada en el *fitness*.
- Creación de una nueva generación mediante cruces y mutaciones.
- Elitismo: selección de los mejores individuos [0].

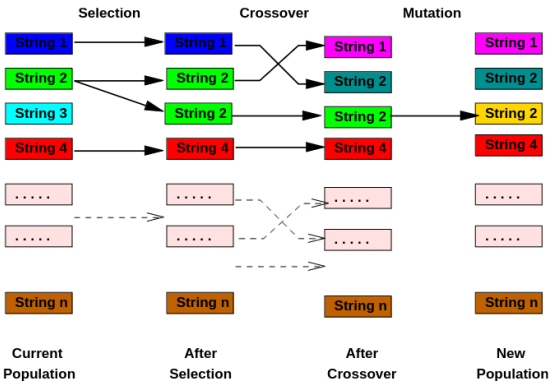

Figura 3.3: Figura obtenida de [0]

El algoritmo termina cuando se alcanza cierta tolerancia de error o número máximo de iteraciones.

Selección

La selección se puede llevar a cabo con varios métodos, uno de los más utilizados en selección por torneo.

En la selección por torneo, los individuos compiten entre sí en grupos pequeños. Cada grupo (torneo) tiene varios competidores (individuos). El competidor con la mejor aptitud (**fitness**) dentro del grupo tiene más probabilidad de ganar. Los ganadores son seleccionados para "reproducirse". Este proceso se repite hasta que se completa el *pool* de apareamiento con los ganadores de los torneos [0]. La probabilidad de cada individuo en base a su **fitness** se calcula de la siguiente manera:

$$P(\text{parent}_i) = \frac{\text{fitness}(\text{parent}i)}{\sum_{j=1}^{N} \text{fitness}(\text{parent}_j)} \tag{3.1}$$

Cruce

Existen múltiples tipos de crossover o cruces. Se describen los dos usados en este proyecto (uno para la versión binaria del algoritmo y otra para la real):

- **One-point crossover**: Se elige al azar un punto en los cromosomas de ambos progenitores y se designa como "punto de cruce". Los bits a la derecha de ese punto se intercambian entre los dos cromosomas parentales. El resultado son dos descendientes, cada uno con información genética de ambos progenitores [0].

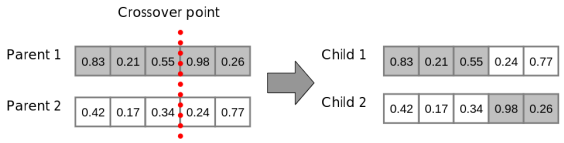

Figura 3.4: One-point crossover [0]

- **Blend crossover**: Dado dos números reales para cada uno de los genes de los padres al hijo se le asignará un número aleatorio entre ese rango de gen para cada gen que conforme al vector cromosómico [0].

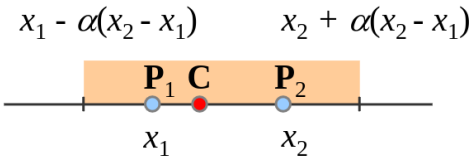

Figura 3.5: Blend crossover [0]

Mutación

Su propósito es mantener la diversidad genética de los cromosomas de una población de un algoritmo evolutivo. El operador más básico y clásico consiste en cambiar un bit arbitrario de un genotipo o solución de un algoritmo genético binario a su estado inverso dada una probabilidad de mutación [0].

Elitismo

El elitismo garantiza que las soluciones de alta calidad sobrevivan en las generaciones futuras, ayudando en la explotación del espacio de búsqueda. La cantidad de individuos élite seleccionados debe ser elegida con precisión [0].

3.2. Algoritmos de colonias de hormigas

3.2.1. Introducción

El algoritmo de colonia de hormigas o **ACO** es una técnica probabilística que trata de resolver problemas computacionales que pueden ser reducidos a la búsqueda de caminos óptimos a través de grafos. Se basa en el comportamiento de las hormigas reales y su comunicación vía feromonas. Las hormigas vagan por el mundo en búsqueda de comida de forma aleatoria. Cuando estas encuentran comida, dejan trazas de feromonas en su camino, de esta forma, si otras hormigas encuentran ese rastro, se hace más probable que recorran ese camino y dejen de vagar de forma aleatoria. Sin embargo, las trazas de feromonas se evaporan con el tiempo, perdiendo su fuerza de atracción sobre otras hormigas. De esta forma, si el camino es muy largo, la hormiga tardará mucho en recorrerlo y la traza de feromonas tendrá más tiempo a evaporarse. De manera análoga, a más corto es el camino, más rápido se recorre y más densidad de feromonas acumula [0].

3.2.2. Funcionamiento y Operadores

Para poder adaptar el algoritmo ACO a un problema es necesario reducir ese problema a la búsqueda del camino más corto dentro de un grafo ponderado. De esta forma, la **representación** en este tipo de algoritmo debe ser la de un grafo. Cada característica del problema original se representa como un nodo n. Los caminos que conectan a los nodos (e) representan la elección del subconjunto de características. El camino deberá ser lo más corto posible maximizando a su vez el **accuracy** [0].

Regla de transición probabilística

$$P_{ij}^{k}(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l}\tau_{il}^{\alpha}\eta_{il}^{\beta}} & \text{Si } l \text{ y } k \text{ son nodos admisibles} \\ 0 & \text{De lo contrario} \end{cases} \tag{3.2}$$

Donde:

- $P_{ij}^{k}(t)$ denota la probabilidad de transición de un nodo de i a j en la k-hormiga (agente) en el instante de tiempo t.
- τ_{ij} es la cantidad de traza de feromona en la arista (i, j) en el momento t. η_{ij} es la heurística de deseabilidad o visibilidad de la arista.
- β y α son dos parámetros que controlan la importancia relativa del valor de la feromona vs la información de la heurística.

Evaporación de feromona

Después de que todas las hormigas hayan terminado su camino, la evaporación de feromonas comienza. El contenido de feromonas del camino (i, j) en el instante $t + 1$ es:

$$\tau_{ij}(\text{new}) = (1 - p)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) + \Delta\tau_{ij}^{g}(t) \tag{3.3}$$

Donde:

- $\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{F^k} & \text{Si la hormiga k pasa por la arista (i,j) en } T^k \\ 0 & \text{De lo contrario} \end{cases}$
- $p \in (0, 1]$ es el ritmo de evaporación.
- m es el número de hormigas.
- $\Delta\tau_{ij}^{k}$ y $\Delta\tau_{ij}^{g}$ son respectivamente, la cantidad de feromonas colocadas en la arista (i, j) por la hormiga k y la cantidad de feromonas depositadas por la mejor hormiga g en el instante t sobre la arista (i, j).
- Q es una constante y F^k es el valor de coste de la solución encontrada por la hormiga k en el tour T^k, es decir, F^k es el *fitness* de la hormiga k.

Se utiliza el sistema *max-min*, solo la mejor hormiga puede depositar feromona.

En el algoritmo original, no había heurística de deseabilidad o visibilidad (η), solo información en términos de feromona [0].

3.3. Optimización por enjambre de partículas

3.3.1. Introducción

El algoritmo conocido como optimización por enjambre de partículas o en inglés *particle swarm optimization* fue concebido en 1995 por James Kennedy y Russel Eberhart [0]. En este algoritmo, un conjunto de soluciones candidatas, llamadas partículas, se mueven en un espacio de búsqueda multidimensional. Cada partícula ajusta su posición de acuerdo con su experiencia personal y la experiencia del grupo.

3.3.2. Funcionamiento y Operadores

Las soluciones son descritas como individuos dentro de un enjambre (conjunto total de posibles soluciones) que tienen una posición y una velocidad en todo momento. La posición de estos individuos representa la solución en sí.

Velocidad

$$v_i(t+1) = v_i(t) + c_1 \cdot \text{rand}() \cdot (pbest_i - x_i(t)) + c_2 \cdot \text{rand}() \cdot (gbest - x_i(t)) \quad (3.4)$$

Donde:

- $v_i(t+1)$ es la velocidad de la partícula i en la siguiente iteración.
- $v_i(t)$ es la velocidad de la partícula i en la iteración actual.
- c_1 y c_2 son factores de aceleración que controlan la influencia de las mejores posiciones.
- rand() es un número aleatorio en el rango [0, 1].
- $pbest_i$ es la mejor posición histórica de la partícula i.
- $x_i(t)$ es la posición actual de la partícula i en la iteración t.
- $gbest$ es la mejor posición global del grupo.

Posición

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (3.5)$$

Estas ecuaciones se utilizan para guiar el movimiento de las partículas a través del espacio de búsqueda hacia las mejores soluciones, adaptándose tanto a la mejor solución encontrada por la propia partícula ($pbest$) como a la mejor solución encontrada por cualquier partícula en el enjambre ($gbest$) [0]. Con el tiempo, las partículas tienden a converger hacia las mejores soluciones conocidas.

3.4. Evolución diferencial

La evolución diferencial surgió como un algoritmo de optimización para problemas no diferenciables y no lineales, con pocas variables de control,

robusto, paralelizable y muy eficaz. El algoritmo de evolución diferencial o *differential evolution* en inglés fue diseñado para resolver problemas continuos, a diferencia del propósito original de su "hermano gemelo" el algoritmo genético, cuya finalidad inicial era resolver problemas binarios. Este algoritmo fue presentado en 1996 por Storn y Price en [0].

En la evolución diferencial, la diferencia de vectores es un concepto clave que se utiliza en el operador de mutación. Esta se calcula restando dos vectores de la población. Gracias a esta operación puede calcularse la distancia entre vectores y esto conlleva una serie de ventajas [0]:

- Las posiciones de los individuos proporcionan información ya que si los individuos están bien distribuidos (y si la población es lo suficientemente grande), la población inicial será una buena representación de todo el espacio de búsqueda.
- Las distancias entre estos individuos serán inversamente proporcionales al tamaño de la población.
- A medida que avanza la búsqueda y los individuos comienzan a converger hacia un óptimo local, las distancias entre los individuos comenzarán a disminuir.

3.4.1. Funcionamiento y Operadores

La representación suele ser la misma que en otros algoritmos metaheurísticos. DE fue diseñado para optimizar un problema continuo, por lo que la población debería ser un conjunto de números reales.

Mutación

La operación de mutación produce un vector, conocido como vector de prueba o en inglés *trial vector*. Para ello se hace uso de un vector objetivo y una diferencia de vectores ponderada. La mutación elige a un padre x_i, generando un vector de prueba u_i siguiendo lo siguientes pasos:

1. Se selecciona un vector objetivo x_{i_1} tal que $i \neq i_1$.
2. Se selecciona aleatoriamente dos individuos x_{i_2} y x_{i_3} de la población tal que $i \neq i_1 \neq i_2 \neq i_3$ y i_2, i_3 sean seleccionados con una probabilidad uniforme. Es necesario que todos los individuos tengan la misma probabilidad de selección.

Finalmente, se calcula el vector de prueba:

$$u_i = x_{i_1} + \beta\dot{(}x_{i_2} - x_{i3}) \tag{3.6}$$

Donde β es un factor de escalado que controla la amplificación de la variación diferencial (mutación más diversa/grande) [0].

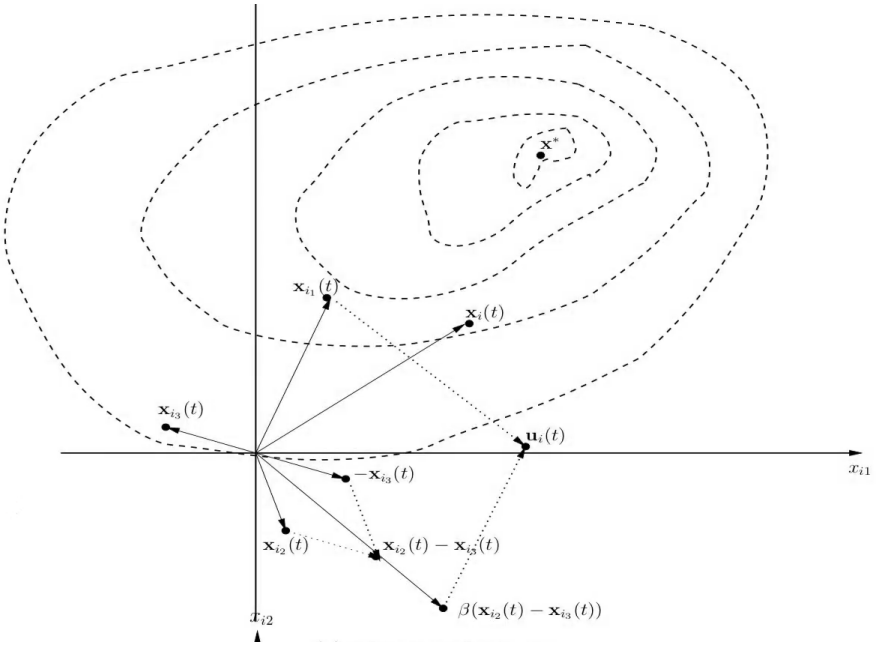

Figura 3.6: Esta figura ha sido seleccionada de [0]. Muestra de manera gráfica en un espacio bidimensional como funciona el operador de mutación y cómo se crea el "trial vector".

Una ventaja que tiene el uso de diferencias de vectores en la mutación, es que si todos lo individuos tienen la misma probabilidad de selección, la media de la distribución será cero. Con ello se consigue mantener la deriva genética lo más pequeña posible. La deriva genética es un proceso estocástico en el que la frecuencia de alelos (forma alternativa de un gen) en una población cambia aleatoriamente con el tiempo debido a evento aleatorios. Esto puede llevar a cambios en la composición genética de la población a lo largo de las generaciones, especialmente en poblaciones pequeñas. Por tanto, una media distinta de cero daría pie a sesgos en la deriva genética [0].

Cruce

El cruce implementa una recombinación del vector de prueba u_i y el vector padre x_i para producir un vector hijo x'_i.

$$x'_{ij} = \begin{cases} u_{ij} & \text{Si } j \in \mathcal{J} \\ x_{ij} & \text{De lo contrario} \end{cases} \tag{3.7}$$

Donde $\mathcal{J}$ es un conjunto de puntos de perturbación. Estos índices pueden ser calculados de muchas formas, por ejemplo, de manera aleatoria.

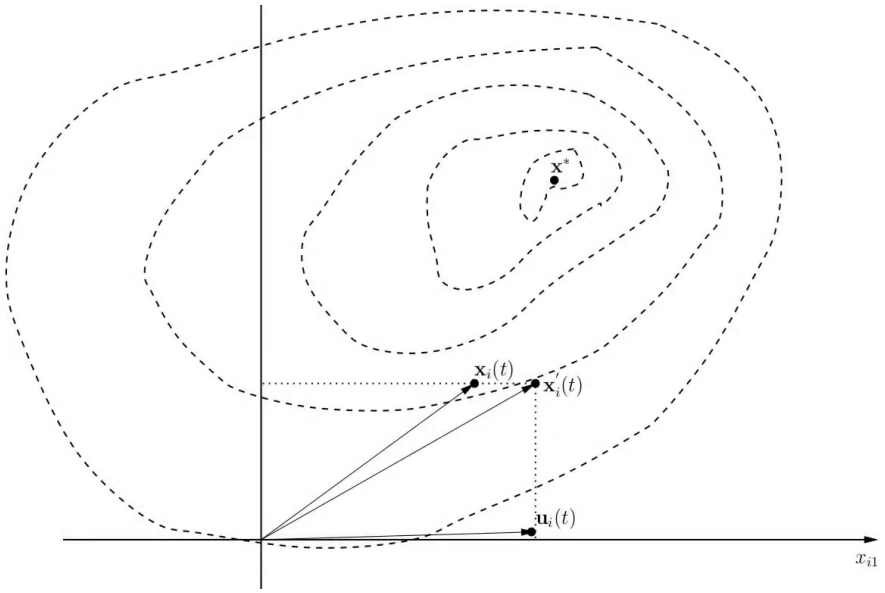

Figura 3.7: Esta figura ha sido seleccionada de [0]. Muestra de manera gráfica en un espacio bidimensional como funciona el operador de cruce al recombinar los genes/características de los vectores padre y "trial".

Selección

Si el vector hijo, al evaluarlo, tiene un valor **fitness** mejor que el vector padre, entonces se sustituye el vector x_i por el vector x_i'. De otra forma, el vector padre continuará en la población para la siguiente generación [0].

3.5. Optimización por colonia de abejas artificial

3.5.1. Introducción

El algoritmo del enjambre de abejas artificial es un algoritmo estocástico evolutivo, al igual que los ya mencionados anteriormente, que se basa en el comportamientos de los enjambres de abejas para la optimización de problemas. El modelo usado en este algoritmo depende de tres factores, las fuentes de alimento, las abejas buscadores empleadas y las abejas buscadoras desempleadas (hay dos tipos) [0].

3.5.2. Funcionamiento y Operadores

Los operadores del algoritmo son los siguientes:

1. **Abejas Empleadas (Employed Bees)**: Estas abejas están asociadas con fuentes de alimento específicas y son responsables de explotar estas fuentes. Llevan información sobre la fuente de alimento, incluyendo la distancia, la dirección y la rentabilidad, y comparten esta información dentro de la colmena, influyendo en las decisiones de otras abejas.

2. **Abejas Observadoras (Onlooker Bees)**: Estas abejas esperan dentro de la colmena y toman decisiones sobre a qué fuentes de alimento dirigirse basándose en la información proporcionada por las abejas empleadas. Seleccionan las fuentes de alimento según la probabilidad que está directamente relacionada con la rentabilidad de las fuentes.

3. **Abejas Exploradoras (Scout Bees)**: Son abejas que buscan nuevas fuentes de alimento sin información previa. Su rol es crucial para la exploración y el descubrimiento de nuevas fuentes de alimento. Cuando una fuente se agota o no mejora después de cierto número de intentos (determinado por un parámetro de control llamado "límite"), una abeja empleada puede convertirse en exploradora.

Estos operadores trabajan en un ciclo repetitivo donde las abejas empleadas primero van a las fuentes de alimento y actualizan la información sobre su rentabilidad. Luego, las abejas observadoras seleccionan fuentes de alimento basadas en esta información actualizada y las abejas exploradoras buscan nuevas fuentes [0]. Este proceso se repite hasta cumplir con los criterios de terminación del algoritmo.

3.6. Algoritmo de optimización del saltamontes

3.6.1. Introducción

Los **grasshoppers** (saltamontes o langosta cuando están en enjambre) son insectos cuyo ciclo de vida consiste en tres etapas: *huevo, ninfa, adulto*. Un saltamontes puede ser encontrado en el enjambre en su estado ninfa y adulto. La diferencia entre ninfa y adulto es la velocidad de movimiento con pasos pequeños en la ninfa y lo contrario en su vida adulta (Las soluciones adultas se encargan de la exploración y las ninfas de la explotación) [0].

3.6.2. Funcionamiento y Operadores

Operador de posición (primera propuesta)

La posición de un individuo se define en la siguiente ecuación:

$$x_i = r_1 s_1 + r_2 g_i + r_3 a_i \tag{3.8}$$

Donde x_i es la posición del agente número i, s la interacción social, g la fuerza gravitacional, a la advección del viento y r_1, r_2, r_3 son números aleatorios entre $[0, 1]$. Esta función de actualización de posición no puede ser usada para la optimización debido a que se alcanza la zona de comfort de manera prematura y no se converge en un punto específico [0].

Posición (propuesta final)

$$X_i^d = c_1 \left(\sum_{j=1, j\neq i}^{N} c_2 \frac{ub_d - lb_d}{2} s(|x_j^d - x_i^d|) \frac{x_j - x_i}{d_{ij}} \right) + \hat{T}_d \tag{3.9}$$

Donde ud_b es la cota superior en la dimensión D, lb_d es la cota inferior en la dimensión D, $\hat{T}_d$ es el valor de la dimensión D en el objetivo (la mejor solución encontrada hasta la fecha) y c es el coeficiente decreciente para hacer cada vez más pequeña la zona de comfort, zona de atracción y zona de repulsión.

No se considera la gravedad (G) y la dirección del viento siempre es a favor del objetivo $\hat{T}_d$.

Una diferencia con otros algoritmos de enjambres como PSO es que **GAO** actualiza la posición de cada saltamontes a partir de la posición actual, el mejor punto encontrado hasta el momento y la posición de todos los otros agentes, mientras que **PSO** lo hace a partir de la posición actual del agente y la mejor posición encontrada [0].

Esto significa que PSO no tienen en cuenta al resto de agentes, mientras que GAO sí.

Los parámetros c_1, c_2 tienen dos propósitos:

- $c_1 \rightarrow$ Reduce el movimiento del enjambre entero alrededor del objetivo, es decir, maneja la relación entre exploración y explotación para el conjunto global de soluciones.
- c_2 *rightarrow* Reduce la zona de comfort y las regiones de repulsión y atracción. s es la función que decide si un agente debe sentirse atraído

o repelido al objetivo, mientras que $\frac{ub_d - lb_d}{2}$ decrece de manera lineal el espacio de búsqueda a medida que c_2 decrece.

Ha de tenerse en cuenta también que en el proceso de búsqueda, la exploración debe ir antes que la explotación. En los saltamontes sin embargo, la fase explotativa que es la **ninfa** es la que se da primero. Por ello el parámetro c debe decrecer a medida que las iteraciones sobre el algoritmo van pasando. Este mecanismo hace que se promueva la explotación a medida que las iteraciones se incrementan, ya que se va reduciendo cada vez más el movimiento general del enjambre y se reduce la zona de comfort. c se actualiza por tanto así:

$$c = c_{\max} - l \cdot \frac{c_{\max} - c_{\min}}{L} \tag{3.10}$$

Donde l es la iteración actual y L el número máximo de iteraciones.

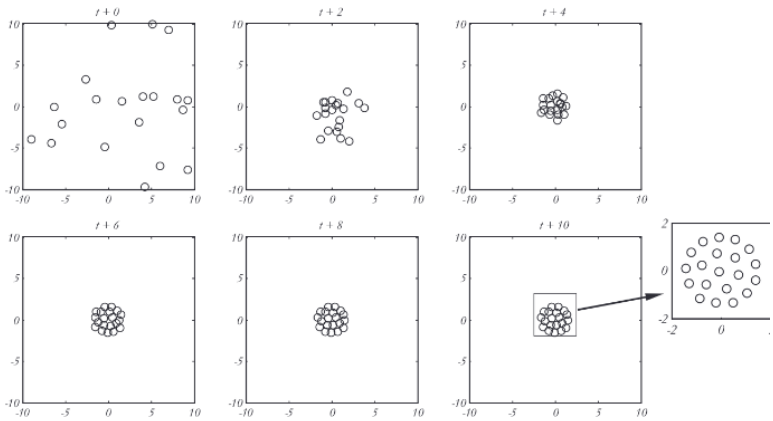

Figura 3.8: Esta figura ha sido seleccionada de [0]. Muestra la convergencia de los "saltamontes" en tan solo unas pocas iteraciones usando la fórmula 3.9.

3.7. Algoritmo de la libélula

3.7.1. Introducción

Los **dragonflies** (libélulas en español) son pequeños insectos que se pueden ver en enjambre solo cuando cazan o migran, dando lugar a dos tipos de enjambre:

- Las diferencias dadas entre ambos enjambres son notorias. En caza, se organizan en enjambres pequeños que vuelan en muchas direcciones en busca de comida. Este tipo de enjambre es llamado **estático**. Cuando

se organizan en enjambres para migrar, estos son numerosos y vuelan en una sola dirección. Estos enjambres en contraposición a los anteriores son llamados **dinámicos** [0].

- Es obvio el paralelismo que se da entre las principales fases de la búsqueda optimizatoria (**exploración y explotación**). La fase dinámica es un paralelismo con la exploración, ya que se buscan pequeños objetivos en grupos por el espacio. La fase estática (explotación) se da cuando todos los agentes empiezan a migrar a una dirección concreta.

3.7.2. Funcionamiento y Operadores

Separación

$$S_i = -\sum_{j=1}^{N} X - X_j \tag{3.11}$$

Se refiere a la elusión estática de colisión entre individuos con su vecindario. Donde X es la posición del agente actual y X_j la posición de su vecino número j. Mientras que N es el número total de agentes.

Alineamiento

$$A_i = \frac{\sum_{j=1}^{N} V_i}{N} \tag{3.12}$$

Se refiere a la coordinación de velocidad entre individuos de un vecindario. Donde V_j es la velocidad del individuo número j en el vecindario.

Cohesión

$$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X \tag{3.13}$$

Se refiere a la fuerza de atracción de los individuos hacia el centro de masa del vecindario. Se da por hecho un radio alrededor de cada libélula agrupando su vecindario, pues este es muy importante para el comportamiento de esta. En un enjambre dinámico, las libélulas tienden a alinear su vuelo al tiempo que mantienen una separación y cohesión adecuadas. En un enjambre estático, sin embargo, las alineaciones son muy bajas mientras que la cohesión es alta para atacar a las presas. Por lo tanto, se asigna a las libélulas pesos de alta alineación y baja cohesión cuando exploran el espacio de búsqueda y

de baja alineación y alta cohesión cuando explotan el espacio de búsqueda. Para la transición entre exploración y explotación, los radios de vecindad se incrementan proporcionalmente al número de iteraciones [0].

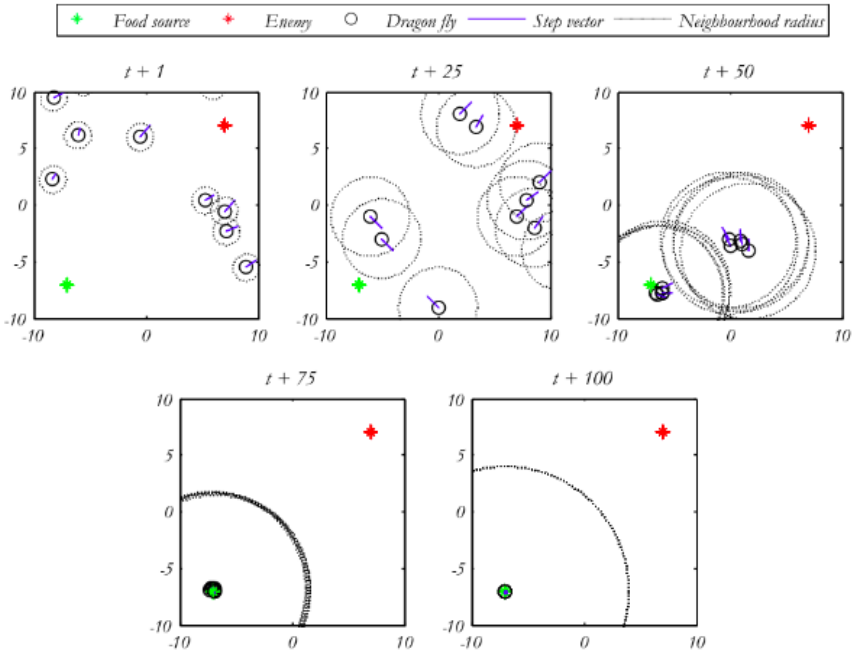

Figura 3.9: Vecindario de libélulas [0].

Atracción comida

$$F_i = X^+ - X \tag{3.14}$$

La fuerza de atracción hacia la comida (la mejor posición encontrada hasta el momento). Donde X^+ es la mejor posición encontrada hasta el momento.

Repulsión depredador

$$E_i = X^- + X \tag{3.15}$$

La fuerza de repulsión hacia un depredador (la peor posición encontrada hasta el momento). Donde X^- es la peor posición encontrada hasta el momento.

Dirección

$$\Delta X_{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\Delta X_t \tag{3.16}$$

El delta de X indica el vector dirección del movimiento de las libélulas. Cada elemento en minúscula es el factor del operador al que multiplica, de forma que es posible acentuar el efecto de cada uno de estos con múltiples combinaciones.

Posición

$$X_{t+1} = X_t + \Delta X_{t+1} \tag{3.17}$$

El operador de actualización de posición.

3.8. Algoritmo de optimización de ballenas

3.8.1. Introducción

Simula el comportamiento de las ballenas jorobadas en la caza, utilizando tanto el azar como el mejor agente de búsqueda para perseguir a la presa. Además, emplea una espiral para simular el mecanismo de ataque de la red de burbujas de las ballenas jorobadas [0].

3.8.2. Funcionamiento y Operadores

Rodeado de presa

$$\vec{D} = |\vec{C} \cdot \vec{X}^*(t) - \vec{X}(t)| \tag{3.18}$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D} \tag{3.19}$$

Donde t indica la iteración actual, $\vec{A}$ y $\vec{C}$ son vectores de coeficientes, X^* es el vector de posición de la mejor solución encontrada hasta el momento y $\vec{X}$ es el vector posición.

Los vectores $\vec{A}$ y $\vec{C}$ se calculan como sigue:

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \tag{3.20}$$

$$\vec{C} = 2 \cdot \vec{r} \tag{3.21}$$

Donde $\vec{a}$ es decrementado linealmente desde 2 hasta 0 y $\vec{r}$ es un vector aleatorio con valores en $[0, 1]$.

Ataque de red de burbujas (explotación)

- **Mecanismo de encogimiento del cerco** Este mecanismo es conseguido mediante el decremento del valor de $\vec{a}$. El vector $\vec{A}$ es un vector de valores aleatorios en el intervalo $[-a, a]$ donde a es decrementado desde 2 hasta 0.

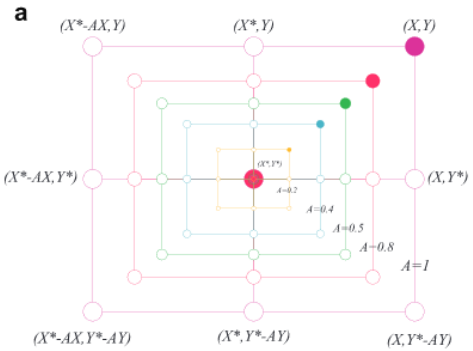

Figura 3.10: Ataque de red de burbujas usando el mecanismo de encogimiento del cerco [0].

- **Mecanismo de actualización de posición en espiral** Este mecanismo calcula la distancia entre la ballena (x, y) y la presa (x^*, y^*). Una ecuación con forma de espiral es creada entonces entre la ballena y la presa para simular el comportamiento de estas al cazar.

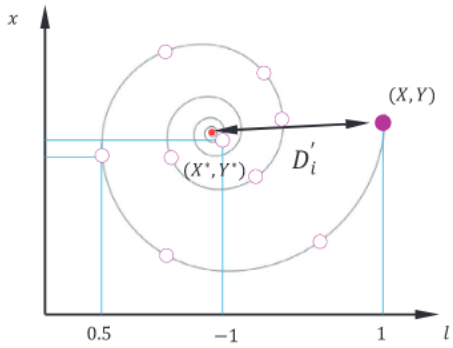

Figura 3.11: Ataque de red de burbujas usando el mecanismo de actualización de posición en espiral [0].

$$\vec{X}(t+1) = \vec{D}' \cdot e^{bl} \cdot cos(2\pi l) + \vec{X}^*(t) \tag{3.22}$$

Donde $\vec{D}' = |\vec{X}^*(t) - \vec{X}(t)|$ e indica la distancia entre presa y ballena. b es una constante para definir la forma de la espiral, l es un número

aleatorio entre $[-1, 1]$.

Se utilizan ambos mecanismos, dependiendo de si cierto número p aleatorio es mayor o menor a 0.5.

Búsqueda de la presa (exploración)

El mismo método utilizando el vector $\vec{A}$ puede ser utilizado para la exploración. Se puede usar $\vec{A}$ con valores más grandes a 1 o menores a -1 para forzar al agente a moverse lejos de otro agente.

$$\vec{D} = |\vec{C} \cdot \vec{X}_{rand} - \vec{X}| \tag{3.23}$$

$$\vec{X}(t+1) = \vec{X}_{rand} - \vec{A} \cdot \vec{D} \tag{3.24}$$

Cuando $|A| \geq 1 \rightarrow$ Exploración, cuando $|A| < 1 \rightarrow$ Explotación.

3.9. Algoritmo del murciélago

3.9.1. Introducción

El algoritmo *metaheurístico* de los murciélagos se basa en la capacidad de eco-localización de estos para detectar a su presa y cazarla finalmente. La eco-localización de los murciélagos no solo les permite detectar a su presa, sino discriminar entre distintos tipos de insectos. Más concretamente, este algoritmo se basa en los micro murciélagos, pues estos usan más frecuentemente la eco-localización en comparación a otras especies de murciélagos. Estos murciélagos emiten un pulso de sonido, el cual, rebota en los objetos y entidades cercanas, permitiendo al murciélago componer una "visión" general de sus alrededores. La frecuencia del pulso emitido podría estar relacionada con distintas estrategias de caza [0].

Acústica de la eco-localización

Aunque cada pulso dura solo unos pocos milisegundos (entre 8 y 10 ms), mantiene una frecuencia constante, generalmente entre 25 kHz y 150 kHz. La longitud de onda (λ) de estos impulsos ultrasónicos, con una frecuencia (f) y velocidad del sonido (v) en el aire a 340 m/s, está dada por

$$\lambda = \frac{v}{f} \tag{3.25}$$

con longitudes entre 2 mm y 14 mm para frecuencias de 25 kHz a 150 kHz, respectivamente. Esta longitud de onda coincide con el tamaño de

las presas. Los murciélagos pueden detectar obstáculos tan pequeños como cabellos humanos, utilizando la eco-localización para construir un modelo 3D del entorno, y discriminando distancias, orientaciones y tipos de presas. Aunque los murciélagos tienen buena vista y olfato, utilizan principalmente la eco-localización para la detección de presas y la navegación eficiente [0].

3.9.2. Funcionamiento y Operadores

Para simplificar:

- Los murciélagos usan la ecolocación para percibir la distancia y distinguir alimento de barreras.
- Vuelan aleatoriamente con velocidad v_i y frecuencia fija f_{min}, ajustando longitud de onda λ y sonido A_0.
- Se asume que la intensidad del sonido varía desde A_0 hasta un mínimo A_{min}.