TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Algoritmos socioinspirados

Implementación, estudio y comparativa

Autor
Juan José Sierra González

Director
Daniel Molina Cabrera

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, septiembre de 2018

Algoritmos socioinspirados

Implementación, estudio y comparativa

Autor
Juan José Sierra González

Directores
Daniel Molina Cabrera

Algoritmos socioinspirados: implementación, estudio y comparativa

Juan José Sierra González

Palabras clave: algoritmo, metaheurística, sociedad, socioinspirado, comparativa, Python...

Resumen

Poner aquí el resumen.

Socioinspired algorithms: implementation, study and comparative analysis

Juan José Sierra González

Keywords: algorithm, metaheuristic, society, socioinspired, comparative analysis, Python...

Abstract

Write here the abstract in English.

Yo, **Juan José Sierra González**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76589592Y, 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: Juan José Sierra González

Granada a 7 de septiembre de 2018.

D. **Daniel Molina Cabrera**, Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Algoritmos socioinspirados: implementación, estudio y comparativa***, ha sido realizado bajo su supervisión por **Juan José Sierra González**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 7 de septiembre de 2018.

El director:

Daniel Molina Cabrera

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La informática ha evolucionado mucho desde sus comienzos, y resolver complejas fórmulas matemáticas o computar cálculos de grandes dimensiones ya no es un problema. Existen multitud de algoritmos que se encargan de realizar estas tareas, y que cualquier usuario puede utilizar sin tener un conocimiento experto. Los nuevos retos de la informática están plagados de problemas que no tienen una solución clara y precisa, entre ellos los **problemas de optimización**, que se estudiarán en este trabajo. Así como un problema de cálculo está bien limitado y un programador puede idear un algoritmo que, en base a distintas fórmulas y operaciones, obtenga siempre el resultado deseado, no se puede decir lo mismo de los problemas de optimización, en los que influyen multitud de factores y que no tienen una solución absoluta.

Esta situación se pone de manifiesto cuando queremos obtener los valores que hacen mínima una función, y su dominio es tan grande que probar con el rudimentario método de fuerza bruta no es una opción. Para esto se diseñaron las **metaheurísticas**, algoritmos que buscan la mejor solución a una función y que no necesitan ser planteados de forma diferente dependiendo de cada una. A lo largo de su (aún breve) historia han existido ya multitud de algoritmos diferentes que aportan su solución a este problema, pero este trabajo se centrará en las nuevas metaheurísticas **socioinspiradas** y analizará lo que aportan al panorama actual y sobre qué se fundamentan.

1.1. Motivación

Las metaheurísticas, y en general la computación evolutiva, están a la orden del día en el ámbito de la investigación. Cada año se publican un gran número de nuevas propuestas y técnicas y muchas de ellas ven la luz en numerosos congresos en todo el mundo. En algunos de ellos incluso se realizan

competiciones entre las nuevas propuestas, como pasa con el Congress of Evolutionary Computation (CEC).

Con estas competiciones se busca incentivar a los investigadores a mejorar las soluciones actuales constantemente. En la actualidad, el auge de la ciencia de datos y del manejo de flujo de datos en tiempo real, y con ello de lo que conocemos como Big Data, premia que estos algoritmos estén muy optimizados y sean capaces de dar buenos resultados ya no sólo en términos de efectividad, sino de rapidez.

Las metaheurísticas que más éxito han tenido en este campo han sido habitualmente las **bioinspiradas**, es decir, aquellas que basan su funcionamiento en la naturaleza, y generalmente en animales. Tras el éxito cosechado por dichos algoritmos, nacen nuevas versiones que, tratando de emular a los anteriormente citados, realizan sus operaciones siguiendo un modelo basado en comportamientos de la sociedad. Estos algoritmos son denominados **socioinspirados**. Aportan un diseño interesante, innovador y más sencillo de comprender y de aplicar que los algoritmos evolutivos tradicionales, y son el centro del estudio de este trabajo.

1.2. Objetivos

El principal objetivo de este Trabajo de Fin de Grado es analizar cómo funcionan algunas de las versiones más interesantes de algoritmos socioinspirados, mediante un estudio comparativo con veinticinco funciones de un benchmark.

Se estudiarán un total de **seis algoritmos socioinspirados**, a saber:

- **Soccer League Competition (SLC)** [4] [5]
- **Imperialist Competitive Algorithm (ICA)** [3]
- **Parliamentary Optimization Algorithm (POA)** [6]
- **Social Emotional Optimization Algorithm (SEA)** [7]
- **Anarchic Society Optimization Algorithm (ASO)** [8] [9]
- **Ideology Algorithm (IA)** [10]

El objetivo global y final fruto del estudio de este trabajo es analizar cómo de efectivas son estas nuevas técnicas y si tienen un hueco entre la élite de la computación evolutiva, así como lo encontraron sus análogos bioinspirados. Para ello, en primer lugar se ha de **realizar un estudio en la**

literatura del algoritmo, a fin de comprender al completo cómo se comporta el mismo en cada momento del proceso de optimización. El conocimiento obtenido de cada uno será documentado en su correspondiente capítulo de la memoria.

Una vez comprendido todo el proceso, tiene lugar la fase de **implementación**, basada en los distintos *papers* que se han encontrado para cada algoritmo. Para ello se han utilizado los conocimientos adquiridos en la fase anterior, siendo lo más fiel posible al diseño planteado. Más información acerca de la implementación se detallará en el capítulo homónimo.

Con todos los algoritmos implementados, el siguiente paso, y más importante de cara al estudio a realizar, es la **experimentación**. Durante esta fase se enfrentará a cada uno de estos algoritmos a un benchmark de variadas funciones que evalúen apropiadamente cómo se comporta en una buena muestra de ejemplos.

Finalmente, el propio estudio en sí pasa por analizar los resultados obtenidos para cada algoritmo y función y compararlos de dos formas diferentes. En primer lugar, se incluirá dentro del capítulo de experimentación de cada uno una comparativa con un ya reconocido **algoritmo de referencia**, a fin de asegurar si dicho socioinspirado puede optar a una solución al menos tan buena como las estándares. En segundo lugar, y para finalizar el estudio, se compararán los resultados de **todos los algoritmos socioinspirados entre sí**, se analizarán pros y contras de las mejores soluciones y se dará una perspectiva global de la posición de estos algoritmos en el mundo de las metaheurísticas.

Capítulo 2

Planificación

En este capítulo se aborda el plan de trabajo a seguir para la realización de este estudio. En primer lugar se estimarán los requisitos a satisfacer por el mismo, a fin de lograr los objetivos planteados, y a continuación se planteará su planificación, como los presupuestos necesarios, la carga de trabajo por fase de dicho estudio o los tiempos esperados de realización para cada una de ellas.

2.1. Requisitos de investigación

Al tratarse de un trabajo de investigación, un análisis de requisitos habitual no puede ser aplicado correctamente a esta situación. En su lugar, sin embargo, se proponen una serie de objetivos a cumplir para la conclusión del estudio. En el caso de este en particular, los distintos requisitos que se pueden distinguir son:

1. **Realizar una investigación primeriza acerca de los algoritmos socioinspirados**: revisar la literatura buscando información sobre lo que representan estos algoritmos, valorar las motivaciones que impulsen a realizar el estudio y obtener propuestas de dichos algoritmos.

2. **Seleccionar las propuestas más interesantes**: de entre todos los algoritmos socioinspirados que se hayan podido encontrar en la fase anterior, seleccionar aquellos que resulten más interesantes o que aporten un enfoque diferente al panorama actual. También se busca que pertenezcan a campos distintos dentro de esta rama de algoritmos, a fin de aportar un punto de vista sobre las principales vertientes que existen.

3. **Analizar a fondo las propuestas seleccionadas**: el principal objetivo aquí es ser capaz de entender qué metodología sigue cada algoritmo, en qué carga teórica basa sus técnicas y qué es capaz de conseguir con lo que propone. Se intentará asemejar cada propuesta con otros algoritmos evolutivos que sean más reconocibles por cualquier investigador iniciado en el campo.

4. **Implementar aquellos algoritmos de los que no se posea código fuente**: ya que el análisis de este trabajo es experimental, es necesario contar con el código de los algoritmos para poder realizar adecuadamente las distintas pruebas. Una implementación propia facilita a su vez que se pueda adaptar el código de dicho algoritmo para seguir unas pautas de formato de soluciones comunes a todo el estudio.

5. **Estimar los parámetros de los algoritmos**: si se desconocen los parámetros ideales con los que ejecutar un algoritmo, se someterá a una pequeña experimentación con varias combinaciones de parámetros a fin de seleccionar los que mejores resultados aporten.

6. **Realizar la experimentación**: utilizar los algoritmos implementados y un benchmark de referencia para obtener resultados. Los algoritmos se lanzarán con las combinaciones de parámetros extraídas del apartado anterior, en función de las conclusiones obtenidas.

7. **Construcción de tablas y gráficas experimentales**: en base a los resultados obtenidos, se pueden disponer los datos en tablas representativas, así como en gráficas de convergencia con las que comprobar el comportamiento de cada algoritmo a lo largo de las ejecuciones.

8. **Estudio analítico de los datos obtenidos**: comparar los resultados de cada algoritmo con los de un algoritmo evolutivo de referencia, así como entre ellos, a fin de descubrir cómo se comportan con una serie variada de funciones complejas y qué algoritmos destacan más sobre los otros.

9. **Extraer conclusiones del estudio realizado**: dar una visión analítica de la eficacia de los algoritmos socioinspirados, basándose en la comparativa con el algoritmo de referencia, y justificar qué propuestas son más prometedoras.

10. **Trabajos futuros a realizar**: valorar en qué aspectos se puede innovar en este campo, aportar propuestas de mejora para las técnicas socioinspiradas y realizar un ajuste minucioso de parámetros para los algoritmos con mayor potencial.

Con los requisitos planteados, la planificación del trabajo debe abarcar cada uno de esos pasos y estimar un tiempo a priori con el que se pueda

solventar cada requisito. Además, debe abarcar todo el material e infraestructura necesarios y, junto al tiempo estimado, dar una idea del presupuesto que requiere realizar este estudio.

2.2. Planificación del trabajo

Este apartado está subdividido en la estimación de costes y la estimación de tiempo. Es necesario en un primer momento valorar el coste de la infraestructura y de todo aquel material que no esté disponible de forma libre o gratuita, y saber de qué presupuesto inicial debe partir el estudio. Además, la estimación de tiempo debe ser competente y ser capaz de abarcar los mejores y peores casos prácticos, para que no sea necesario aplazar la entrega a última hora debido a una mala planificación, y deberá influir a su vez en el coste, pues es un recurso más con el que se cuenta.

2.2.1. Estimación de coste de materiales e infraestructura

En primer lugar se tendrá en cuenta la máquina con la que se ha desarrollado el grueso de este estudio. Se trata de un ordenador portátil de la marca Acer, del año 2012, y que cuenta con un procesador i5-2450M a 2.5GHz, así como con 8GB de memoria RAM DDR3, un HDD de 1TB y un SSD de 120GB, que ha sido añadido posterior a la compra del equipo. Con este equipo se han realizado aquellas tareas de documentación, revisión de la literatura y redacción de la memoria, así como las labores de desarrollo e implementación de los algoritmos.

Para obtener la información y documentación requerida para el trabajo ha sido necesaria una red de internet de banda ancha, disponible tanto en el lugar de desarrollo del trabajo como en las instalaciones de la Universidad de Granada cuando ha sido necesario. Gracias a los convenios que la Universidad establece con algunas bases de datos documentales como Scopus [1] ha sido posible acceder a multitud de *papers* y publicaciones sobre el tema a investigar. Además ha sido de mucha utilidad la página ResearchGate [11] para acceder a otros papers no encontrados en Scopus.

El desarrollo del trabajo ha sido realizado completamente sobre el sistema operativo Linux, con una distribución Ubuntu 17.10, por lo que es totalmente libre y carente de costes. Las principales herramientas de software utilizadas también han sido de código abierto por lo que no han conllevado coste adicional, utilizando como IDE principal Visual Studio Code [12] y escribiendo la memoria en LaTeX utilizando el software TeXstudio [13].

Para la ejecución de los experimentos, debido al procesador antiguo del ordenador principal, así como de los grandes cómputos que son necesarios

para la experimentación, el departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada (DECSAI) facilitó el acceso a su clúster Hércules [14], ubicado en el Centro de Investigación en Tecnologías de la Información y de las Comunicaciones de la Universidad de Granada (CITIC-UGR). Este clúster, de acuerdo a la fuente anteriormente citada, «posee 46 nodos, cada uno de ellos equipado con un procesador Intel Core i7 930 a 2.8 GHz, 24 GB de RAM y HDD SATA2 de 1TB. Los nodos están interconectados mediante dos redes internas de tipo Gigabit Ethernet. El sistema incluye una cabina RAID con capacidad para 72 TB. Se emplean los S.O. Fedora 16 y CentOS 6.2». Se trata por tanto de una máquina mucho más potente que la utilizada en el estudio, y servirá para obtener de forma más eficaz los resultados de los experimentos.

De todo este material expuesto en los párrafos anteriores, el trabajo sólo ha requerido comprar el ordenador portátil personal y la red de banda ancha disponible en el lugar de desarrollo del mismo, dado que se trata de recursos personales. El resto de la infraestructura y materiales o bien han sido productos de software libre o han sido aportados por la Universidad de Granada.

2.2.2. Distribución de tiempo entre tareas

Organizar la carga de trabajo en las distintas fases del proceso es una labor que debe realizarse antes del comienzo del mismo. Gracias a ello se pueden plantear fechas de entrega, acotar los tiempos dedicados a cada tarea y, por consiguiente, ayudar al rendimiento general del problema.

En primer lugar, la labor de revisión de la literatura y documentación abarcará todo el proceso, ya que es habitual que a lo largo del desarrollo sea necesario acudir a nuevos documentos que contengan la información deseada. Sin embargo, de la primera tarea se dedicará una semana a seleccionar las propuestas más interesantes a priori, una vez conocidos los principales algoritmos. En dicha labor, cabe destacar la ayuda de este «bestiario» de algoritmos evolutivos recopilado por el usuario *fcampelo* en la plataforma GitHub [15], que ha servido para descubrir algunos de los algoritmos escogidos para el estudio.

En cuanto a la comprensión de cada algoritmo y la implementación del mismo, resulta una tarea compleja y que tiene una alta carga de trabajo. A pesar de que estos algoritmos no están necesariamente plagados de complejas fórmulas matemáticas, sí que es necesario escudriñar a fondo toda la información recabada de su definición en la literatura, a fin de poder realizar una implementación adecuada del mismo. Una estimación factible es que comprender e implementar cada algoritmo puede llevar entre dos semanas y un mes, dependiendo de la complejidad del mismo.

Antes de plantear los experimentos definitivos se ha propuesto realizar una estimación de parámetros, debida a la poca información que algunos de los *papers* esclarecen acerca de los mismos. Esto llevará un par de semanas para elegir el benchmark, preparar el entorno de trabajo y ejecutar los algoritmos. Lógicamente, cuantas más combinaciones de parámetros se quieran probar, más extenso será este tiempo.

La experimentación final requerirá de unos días más para seleccionar dichos parámetros adecuadamente, preparar una serie de scripts que lanzar en paralelo y recopilar los datos de forma conjunta. Una vez conseguido esto, la construcción de tablas y gráficas será también casi directa, aunque trabajosa, y puede llevar otra semana.

Finalmente, extraer conclusiones de los resultados puede ser más sencillo si se conocen los algoritmos y se han trabajado correctamente, de forma que sea fácil interpretarlos y descubrir patrones en los mismos. Al ser una tarea tan diversa y poco acotada, una extracción de conclusiones puede llevar desde una semana hasta más de un mes.

En resumen, la ruta de trabajo trazada por estas tareas puede estimarse en que llevará entre unos 6 y 7 meses, aunque naturalmente ese espacio de tiempo puede variar dependiendo de lo que se invierta cada día. Es de importancia recordar que el coste del tiempo invertido también debe ser igualmente valorado tal y como se hace con el coste de los materiales. Partiendo de la base de que este estudio es, en esencia, un trabajo de investigación en un campo puntero tecnológicamente, podría estimarse el coste total del trabajo como una media entre el sueldo a jornada completa de un ingeniero informático y el de un investigador de un departamento de universidad durante el transcurso del mismo.

Siendo un valor estimado de 18 euros/hora para una profesión como esta, a 6 meses trabajando (20 días al mes) a 8h diarias, se estipularía el presupuesto del proyecto en 17280 euros.

Capítulo 3

Revisión de la literatura: estado del arte

En este capítulo se realizará una revisión de los estudios y propuestas más innovadoras relacionadas con los algoritmos socioinspirados. Se tendrá en cuenta hasta dónde han llegado aquellos algoritmos en los que se basaron, los bioinspirados, qué impacto tienen en el panorama actual y de dónde surgieron los algoritmos socioinspirados. Además, se realizará un análisis general de aquellos algoritmos socioinspirados que resulten interesantes y se verá cómo de efectivos han resultado en problemas reales hasta la fecha.

La algorítmica experimentó un antes y un después con la llegada de los algoritmos bioinspirados. El planteamiento tan visual a la par que efectivo que poseen dichos algoritmos incitó que poco a poco cada vez más investigadores se dedicasen a este campo. En particular, ha tenido especial éxito en la computación evolutiva, donde las propuestas más conocidas como los algoritmos de colonias de hormigas o de colmenas de abejas han resultado no sólo innovadoras sino también efectivas.

Los algoritmos **Ant Colony Optimization (ACO)**, los basados en los comportamientos de las colonias de hormigas, han sido los pioneros en este campo. Marco Dorigo fue el padre de los ACO, publicando en su tesis doctoral [16] en 1992 lo que él definió como «Ant systems». A partir de esta primera aproximación, Dorigo continuó su investigación en dicha técnica, y años más tarde, en 1999, publicó su primera propuesta de algoritmo, explicando, en términos del autor, lo que él propuso como «Ant algorithms» [17].

Además de la vertiente computacional, algunas de las primeras propuestas bioinspiradas también trataban de aplicar estas técnicas al hardware, en particular a mejorar el diseño de los sistemas. Sánchez et al. (1997) plantearon la investigación en este campo. En el 1st International Conference

on Evolvable Systems, acontecido en Tsukuba, Japón, en 1996, presentaron en su paper *Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware* [18] las bases de la investigación en técnicas bioinspiradas, en este caso para dotar a sistemas hardware de procesos evolutivos. Más adelante lo publicaron en la revista IEEE Xplore [19].

Estos ejemplos citados no son, sin embargo, nada más que la pequeña primera muesca en la historia de las técnicas bioinspiradas. Consultando una base de datos bibliográfica se puede observar un creciente aumento en cuanto a las publicaciones relacionadas con algoritmos bioinspirados. En la base de datos de Scopus estos datos pueden ser plasmados en una gráfica como la que se muestra a continuación.

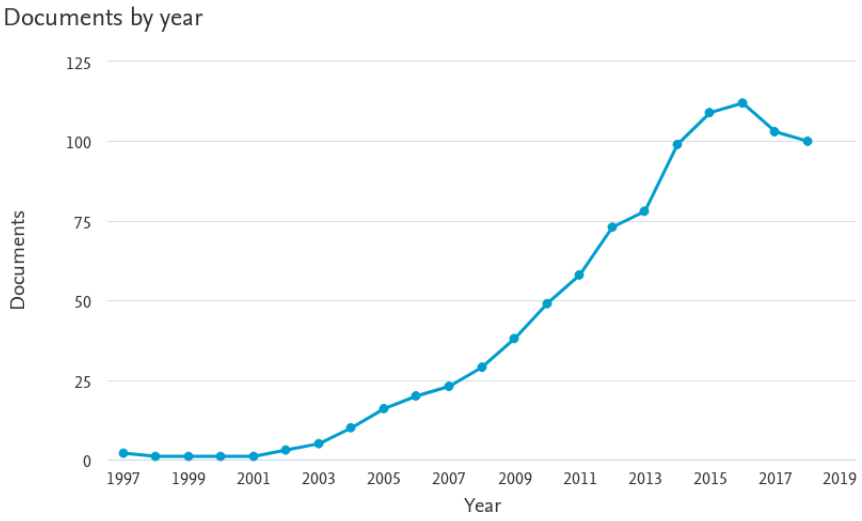

Figura 3.1: Gráfico de publicaciones sobre algoritmos bioinspirados registradas en Scopus [1].

Dicha gráfica refleja perfectamente el auge de los algoritmos bioinspirados; de menos de diez publicaciones por año hasta el año 2004 aproximadamente, hasta los más de 100 que llevan registrándose desde 2015. El éxito de estos algoritmos se fundamenta en dos pilares: su carga conceptual es sencilla de abstraer para cualquier usuario, y generalmente aportan buenos resultados a los problemas a los que se han enfrentado.

La principal motivación para construir algoritmos bioinspirados es simple, a la par que consistente: si un ser vivo ha evolucionado durante siglos y siglos hasta llegar a desarrollar una técnica con la que sobrevivir en la naturaleza, dicha técnica debe ser suficientemente óptima como para resultar digna de un estudio. Es por ello que se han propuesto multitud de técnicas que simulan el comportamiento de numerosos seres vivos en su búsqueda de un objetivo.

Actualmente existen algoritmos bioinspirados en cualquier rama del reino animal. Algoritmos de manadas de lobos o grupos de tiburones que acorralan a su presa, basados en bancos de peces que se mueven en grupo hacia una zona óptima, o incluso que simulan el movimiento de una polilla cuando se dirige hacia un foco de luz. Por supuesto, en los años de la explosión de los algoritmos bioinspirados (acorde con la gráfica de la figura **??**, se podría considerar a partir de 2005), con tanta diversidad en las propuestas no tardaron en surgir técnicas cuyo concepto se basaba en un animal muy particular: el ser humano.

Conocemos como **algoritmos socioinspirados** a aquellos que inspiran su funcionamiento en comportamientos basados en la especie humana, y que pueden aplicarse a resolver problemas de cualquier índole. Similar a sus contrapartes bioinspiradas, las técnicas socioinspiradas tienen en el centro de su modelo conceptual a un miembro que forma parte de la naturaleza, en este caso el ser humano, y se aprovechan de sus relaciones, formas de actuar y estilo de vida para intentar dar una solución óptima a un problema.

En el apartado anterior se defendía la investigación de técnicas bioinspiradas por la constante evolución en busca de supervivencia de las distintas especies animales, y esto sucede de forma análoga con las técnicas socioinspiradas. En el caso de estas últimas, sin embargo, la mejora de soluciones no se basa en la evolución genética de la especie a lo largo de los siglos y en la selección natural, sino en el desarrollo de técnicas que permiten a nuestra sociedad alcanzar metas colectivas o individuales. Dichas técnicas pueden resultar cotidianas, pero deteniéndose a analizar cada una de ellas se puede observar que dicho comportamiento se puede trasladar a un algoritmo que explore en un espacio de búsqueda.

Dada la multitud de variedades bioinspiradas que se han propuesto y estudiado desde el surgimiento de dichos algoritmos, era de esperar que se empezasen a valorar técnicas humanas para optimización de funciones. En el día a día de un ser humano suceden cuantiosas situaciones que, sin ser puramente consciente, le obligan a buscar una solución óptima a un problema menor, tales como buscar la ruta más óptima para desplazarse al lugar de trabajo, estimar la máxima cantidad de días que se puede pasar sin realizar la compra o en qué horas debe conectar un electrodoméstico para aprovechar al máximo el consumo energético. Son problemas en los que encontrar soluciones a corto plazo puede resultar trivial, pero las técnicas abordadas en este estudio abarcan temas más complejos, o a mayor escala.

Cabe destacar que son unas técnicas muy recientes, y que a día de hoy ni siquiera existe un estudio formal que analice algunas de las propuestas con experimentos y análisis de rendimiento, como se busca hacer en este Trabajo de Fin de Grado. De hecho, en la actualidad estos algoritmos se encuentran aún en una fase de divulgación y propagación, es decir, los investigadores

priorizan buscar nuevas propuestas de algoritmos antes que analizar las ya existentes, bien por falta de ellas o por considerarlas poco prometedoras.

Kumar, Kulkarni y Satapathy [2] han presentado la última propuesta socioinspirada actual, que data de abril de este mismo año, titulada *Socio evolution & learning optimization algorithm: A socio-inspired optimization methodology*. En este artículo, antes de explicar su algoritmo, los autores dan una pincelada sobre la situación de los algoritmos socioinspirados. En la gráfica ubicada a continuación se puede observar una clasificación de estos algoritmos, según los autores.

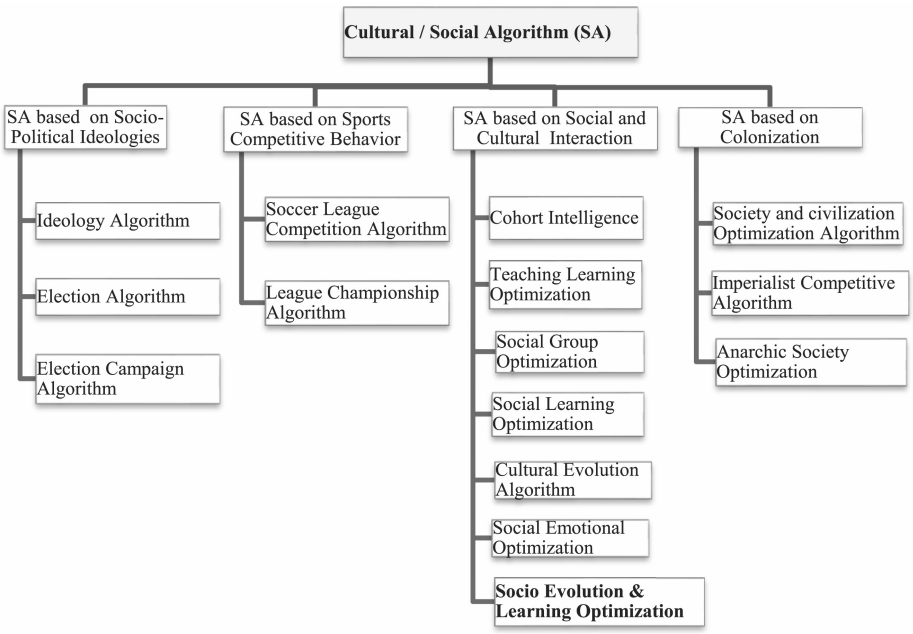

Figura 3.2: Clasificación estimada de los algoritmos socioinspirados [2].

En dicha gráfica se encuentran cinco de los seis algoritmos que son fruto de estudio de este trabajo: del grupo de socioinspirados basados en ideologías socio-políticas, el algoritmo *Ideology Algorithm* (IA) [10]; del grupo de basados en competiciones deportivas, el algoritmo *Soccer League Competition* (SLC) [4] [5]; del grupo de basados en interacción social y cultural, el algoritmo *Social Emotional Optimization Algorithm* (SEA) [7]; y del grupo de basados en colonización, los algoritmos *Imperialist Competitive Algorithm* (ICA) [3] y *Anarchic Society Optimization* (ASO) [8] [9].

El algoritmo restante, que no se muestra en esta gráfica, es *Parliamentary Optimization Algorithm* (POA) [6], y de haberlo incluido los autores en esta clasificación seguro que estaría en el primer grupo, los basados en ideologías socio-políticas. Su ausencia en la gráfica probablemente se deba al desconocimiento de su existencia por parte de los autores, ya que no es

una clasificación muy extensa en la que haya habido necesidad de eliminar información.

Precisamente llama la atención la falta de propuestas en dicha gráfica. Los algoritmos socioinspirados están pasando por la primera etapa de la que se ha hablado en la sección anterior de algoritmos bioinspirados. Las propuestas son innovadoras, llaman la atención, pero pocos investigadores abandonan su campo de trabajo para volcarse con ellas. Poco a poco, si los resultados acompañan, irán desarrollándose nuevas técnicas que, retroalimentándose unas a otras, conseguirán dar consistencia a esta rama de la computación evolutiva.

En la siguiente gráfica puede observarse el crecimiento de estos algoritmos en la historia reciente de la computación evolutiva.

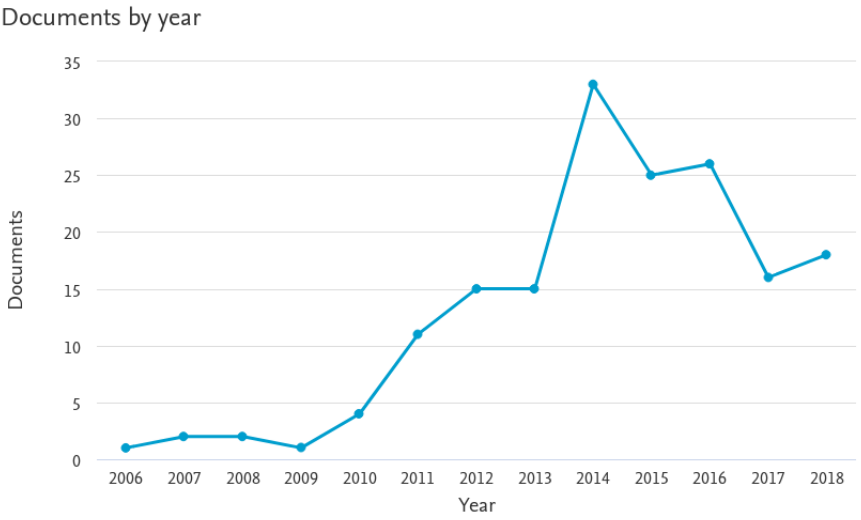

Figura 3.3: Gráfico de publicaciones sobre algoritmos socioinspirados registradas en Scopus [1].

Como puede observarse, su historia es notablemente más corta de lo que se apreciaba en la gráfica de propuestas bioinspiradas, comenzando aquí la gráfica en 2006 sin apenas propuestas hasta ya entrada la década actual.

En palabras de Kumar et al. [2], «estos métodos han ganado popularidad por sus pautas sencillas para buscar soluciones óptimas en problemas computacionales reales y complejos». Como se había anticipado en la introducción, esta es la principal motivación para trabajar con algoritmos socioinspirados. Cuando se trata de aproximar a usuarios primerizos a la computación evolutiva, el primer escalón puede ser difícil de sobrepasar. Estos algoritmos aportan una visión más cotidiana, una forma de hacer comprender a casi cualquier persona cómo se puede explorar un espacio de soluciones sin

tener que comprender a fondo el contexto de un algoritmo común de estas características.

Hablar de «población» en un algoritmo evolutivo no debe suponer ningún problema de comprensión para un investigador dedicado a este campo. Pero resulta mucho más atractivo si a esta población se le da un nombre propio, si se la asocia con un conjunto de individuos que una persona es capaz de reconocer y ubicar. Por ejemplo, con un parlamento político, o con los jugadores de una liga deportiva. Son sólo algunos de los casos que se plantean en los algoritmos estudiados en este trabajo.

Ya que una de las máximas de estos algoritmos es mantener la sencillez y hacer que las técnicas utilizadas sean lo más intuitivas posibles, la metodología que siguen para llegar a una solución óptima también debe ser asociable a conductas reconocibles. Al igual que las poblaciones están compuestas de soluciones habitualmente representadas como «seres humanos», las técnicas evolutivas se corresponden a aquellas acciones realizadas por los mismos para llegar hasta su objetivo. Continuando el símil anterior, realizar debates políticos o jugar partidos de una temporada serían los ejemplos equivalentes.

Dentro de esta metodología siempre hay alguna figura o grupo que sobresale entre la población, y que constituye (o en caso de ser un grupo, contiene) al mejor «individuo». Este se equivaldrá a la mejor solución encontrada para el problema, y es fácilmente reconocible si uno se abstrae a la capa socioinspirada del mismo. El mejor individuo de un parlamento político sería la figura del presidente, o el de una liga deportiva sería el mejor jugador admirado por todo el mundo.

Como se ha podido comprobar, el análisis funcional de los algoritmos socioinspirados es, a diferencia de lo que ocurre con otros algoritmos evolutivos de propósito general, bastante sencillo e intuitivo de transmitir. Esto es claramente debido a la naturalidad con la que un usuario puede asociar dichas pautas a acciones observables en su día a día. No obstante, la eficacia real de estos algoritmos, fuera de las funciones probadas en sus respectivos *papers* o artículos, está aún por probar, ya que de momento no son partícipes de las grandes competiciones acontecidas en congresos, entre las que destacan las del Congress of Evolutionary Computation (CEC).

Sin embargo, sí que se han propuesto en distintas publicaciones algunas aplicaciones de estos algoritmos a resolución de problemas reales muy particulares. En el caso de aquellos algoritmos que se han estudiado en este trabajo, existe una aproximación del algoritmo *Soccer League Competition* para optimizar el diseño de redes de distribución de agua [5], u otra del algoritmo *Anarchic Society Optimization* para manejar el controlador PID (*proportional–integral–derivative*) de un Regulador de Voltaje Automático (*Automatic Voltage Regulator* o AVR, en inglés) [20].

Con el paso de los años es de esperar que estos algoritmos sean perfeccionados, sigan evolucionando y se utilicen cada vez en más problemas reales, al igual que sucedió en su momento con los algoritmos bioinspirados.

Capítulo 4

Análisis de los algoritmos

En este apartado se realizará un análisis detallado sobre cómo funciona cada uno de los algoritmos seleccionados para el estudio desde un punto de vista teórico. Se explicarán las técnicas que utiliza cada uno para llegar a la solución óptima y cómo se asemejan dichas técnicas al comportamiento humano, para de ese modo poder considerarse un algoritmo socioinspirado.

Los algoritmos se analizarán en orden cronológico, siendo el más antiguo de todos Imperialist Competitive Algorithm (2007) [3] y el más reciente Ideology Algorithm (2016) [10].

4.1. Imperialist Competitive Algorithm (ICA)

El primer algoritmo a analizar, **Imperialist Competitive Algorithm** o, según sus siglas, **ICA**, fue presentado por primera vez en 2007 por Atashpaz-Gargari y Lucas [3]. Se trata de un algoritmo que podría ser agrupado en la subcategoría de socioinspirados basados en conquistas y colonialismo, como lo han hecho Kumar et al. [2].

Este algoritmo parte de la idea de un mundo completamente colonialista, plagado de países que luchan por alzarse vencedores y conquistadores. En esta guerra constante, los países con más poder se volverían los líderes, y podrían contar con numerosas colonias a su servicio. Por otra parte, aquellos países que no pueden vencer a los demás se verían relegados a ocupar un puesto de colonia para siempre.

Durante la guerra también se puede producir una revolución o alzamiento de una de las colonias contra su propio imperio. Si esta colonia resulta ser más poderosa que el actual líder, dicha posición pasa a ser suya. También puede darse el caso totalmente opuesto; si una colonia resulta realmente débil, un imperio colindante puede absorberla y hacerse más fuerte gracias

a ella. Y por supuesto, un imperio puede ser colonizado al completo por otro, haciéndose así cargo de sus colonias restantes y del propio país imperial.

En palabras de los propios autores del algoritmo, «la competición imperialista con suerte convergerá a un estado en el que sólo haya un imperio y todas sus colonias se hallen en la misma posición y tengan el mismo coste que el país imperialista».

Como se puede observar de esta hoja de ruta del algoritmo, la relación con la sociedad es evidente, aunque recuerde más a épocas pasadas que a la actualidad. Este es el funcionamiento del algoritmo a grandes rasgos, pero a continuación se detallarán las características más llamativas del mismo. Previamente se muestra un pseudocódigo del algoritmo, para ayudar a comprender cada paso del mismo antes de la explicación detallada.

Algorithm 1 Imperialist Competitive Algorithm

```
1: imperios ← inicializarImperios()
2: while not criterioParada do
3:     for each colonia do
4:         colonia ← colonia + (imperialista − colonia + rand())
5:         if peso(colonia) < peso(imperialista) then
6:             imperialista ← colonia
7:     imperios ← competicionImperial()
8:     if any imperio is empty then
9:         imperios ← eliminarImperio(imperio)
10: return min(peso(imperialistas))
```

Se puede comenzar este análisis en base a los distintos componentes del algoritmo, relacionándolos con la descripción anterior y comprobando por qué se trata de técnicas socioinspiradas.

En primer lugar conviene hablar acerca de la población. Si bien habitualmente en estos algoritmos socioinspirados la población suele estar constituida de individuos, siendo esta una agrupación de los mismos, en el caso del algoritmo ICA no es así. En este algoritmo se realiza un símil con una sociedad global donde cada una de las soluciones que compone la población es como un país, o en este caso como una colonia. Por tanto, la figura del ser humano como persona individual no aparece en esta propuesta.

Una vez que se han instanciado las primeras soluciones o colonias que ocuparán la población inicial, hay que seleccionar aquellas que sean más poderosas para considerarlas «imperialistas», es decir, serán los líderes de cada uno de los imperios en los que se divida el problema. Por supuesto, tanto el número de colonias como el número de imperios iniciales son definidos en los parámetros del algoritmo y pueden variar si así se desea.

A lo largo de todo este algoritmo se entenderá que un país es más poderoso que otro si **su evaluación para la función objetivo es menor**, dado que se trata de problemas de optimización buscando el menor valor posible. La asignación de los imperios se hace con respecto a las colonias iniciales con un menor valor para dicha función.

Con los imperios constituidos y las colonias distribuidas se comienza a desarrollar el bucle principal del algoritmo, que finalizará cuando se cumpla el criterio de parada. La implementación de este algoritmo originalmente paraba tras un número de «décadas» (iteraciones del bucle principal), pero a fin de integrarlo con el resto de algoritmos del estudio se añadió la posibilidad de definir un criterio de parada por número de evaluaciones de la función objetivo.

En cada iteración de dicho bucle cada colonia realiza un desplazamiento hacia su imperialista. El desplazamiento tiene un componente de aleatoriedad que permite que no converjan todas las soluciones en el primer imperialista nada más comenzar las iteraciones. A continuación se puede observar una representación gráfica de dicho movimiento.

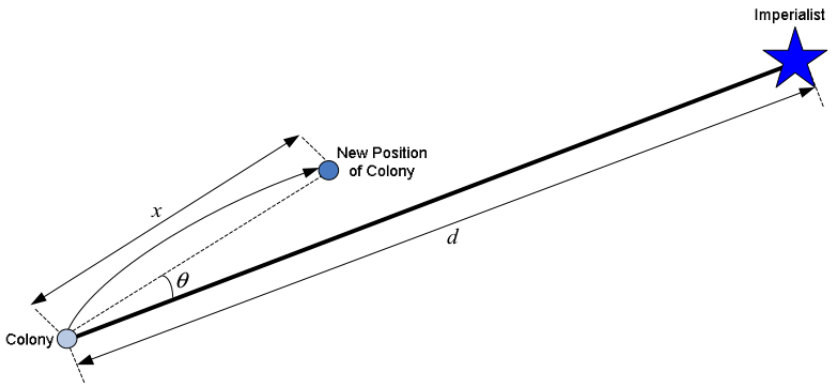

Figura 4.1: Desplazamiento de una colonia hacia su imperialista, con un componente de aleatoriedad [3].

El movimiento producido x queda definido por la siguiente fórmula

$$x \sim U(0, \beta \times d) \tag{4.1}$$

en la cual d es la distancia entre la colonia y el imperialista y β representa el llamado «coeficiente de asimilación». Es decir, se trata de un desplazamiento hacia el imperialista donde la distancia recorrida está definida por un valor uniforme aleatorio entre 0 y la máxima distancia.

Además, el ángulo θ que aparece en la figura, que desvía a la colonia de su dirección, viene calculado de la siguiente forma

$$\theta \sim U(-\gamma, \gamma) \tag{4.2}$$

donde γ es el parámetro que ajusta la desviación de la dirección, y que es llamado por los autores «coeficiente del ángulo de asimilación».

Los cambios producidos mediante estos operadores mencionados ayudan a diversificar las posiciones de las colonias y a que no se agrupen automáticamente en torno a la mejor solución del imperio. Esto permite al algoritmo explorar el espacio de búsqueda adecuadamente y no caer en posibles óptimos locales con facilidad en las primeras iteraciones, lo que limitaría mucho el proceso evolutivo. Si la nueva posición hacia la que se mueve una colonia resulta ser mejor que la del actual imperialista de dicho imperio, se intercambian los puestos y el resto de colonias intentará moverse hacia la nueva mejor solución.

La competición imperialista que tiene lugar en cada iteración del bucle principal aporta aún más variedad al proceso, llevando a la peor de las colonias del peor imperio hasta el imperio que más probabilidad tiene de adquirirla, que suele ser el más poderoso. Este poder de los imperios se calcula en función del peso del imperialista y del peso medio de sus colonias. Esta decisión también aporta más variedad a la par que realismo en el proceso imperialista, ya que si un el país líder del imperio es muy poderoso pero tiene muchas colonias débiles, generalmente dicho imperio será menos potente que uno que tenga un imperialista algo peor y menos colonias aunque más fuertes.

Una vez que un imperio se quede sin colonias, este pasa a ser absorbido por el imperio más fuerte, y la cantidad de imperialistas se reduce. El proceso se repite en cada iteración del bucle hasta que quede solamente un imperio que contenga todas las colonias; en tal caso, a partir de esa iteración en el bucle principal solamente se moverán las colonias hacia nuevas posiciones, tratando de explorar las cercanías del país imperialista.

Para concluir el análisis del ICA, se puede afirmar con certeza que basa muchísimo su comportamiento en los algoritmos de **Particle Swarm Optimization (PSO)** [21]. Este algoritmo está basado en nubes de partículas, donde cada partícula es una solución. Dicha partícula buscará en cada iteración dirigirse hacia la mejor de su población, y realiza un desplazamiento en función de un vector de velocidad. La velocidad es calculada en cada iteración en función del componente cognitivo (relacionado consigo mismo) y social (relacionado con la mejor) de cada una de las partículas. Se trata de un algoritmo muy conocido y que ha servido de base a muchas propuestas, como al ICA.

En este caso, cada imperio funcionaría como un algoritmo PSO en sí, siendo las partículas las colonias y la mejor posición la del país imperialista.

La principal variación consiste en que el algoritmo simula varios PSO compitiendo entre sí, aunque mezclando sus partículas mediante revoluciones o intercambiándolas en el proceso. La propuesta es interesante y una de las primeras en aportar un enfoque colonialista al planteamiento del algoritmo, por lo que resultó especialmente interesante para el estudio.

El punto crítico al algoritmo viene en torno a la exploración del espacio de búsqueda. Para ello es necesario distinguir bien en este caso cuál es la fase de exploración y cuál la de explotación. En este algoritmo se produce una exploración del espacio de búsqueda cuando hay varios imperios, y además las colonias del imperio se encuentran suficientemente separadas del imperialista como para que su avance hacia él permita valorar un considerable número de posiciones en los alrededores, pero no en las inmediaciones. Esta búsqueda en las posiciones más cercanas a la mejor solución es la que se considera la explotación, cuyo principal objetivo es comprobar si en las cercanías hay alguna posibilidad de mejora para apuntillar la solución. Esta fase suele darse en las iteraciones más avanzadas del algoritmo, cuando ya se tiene más o menos claro dónde puede estar el óptimo.

Si bien se ha comentado lo interesante de la propuesta para que no se caiga de inmediato en óptimos locales y la ventaja que aporta que cada colonia tenga posibilidad de revolucionarse y dar lugar a una nueva exploración, el problema es que una o dos colonias colocadas de forma aleatoria no suponen una exploración decente. Tener pronto todas las colonias agrupadas en un mismo imperio supone contar con una fase de exploración corta, lo que hace bastante factible la posibilidad de que el algoritmo se quede en un óptimo local y no vuelva a moverse de ahí ya que el resto de las iteraciones implican casi esencialmente una fase de explotación.

4.2. Parliamentary Optimization Algorithm (POA)

El segundo algoritmo en la lista cronológica es **Parliamentary Optimization Algorithm**, o **POA** acorde a sus siglas, que fue presentado en 2008 [6] en la revista *International Journal of Innovative Computing, Information & Control*, también conocida como *IJICIC*. Este algoritmo está englobado en el apartado de socioinspirados basados en ideologías socio-políticas, obviamente en la vertiente de políticas.

Esta propuesta simula ubicarse dentro de un marco político, en un parlamento. En el mismo tendrán cabida una serie de partidos políticos que tratarán de hacerse con el control del gobierno. Dentro de cada uno tendrá lugar una competición por ascender hasta los puestos de candidato del partido, donde los distintos miembros pueden evolucionar hasta parecerse a sus candidatos, y si alguno de ellos aporta soluciones diferentes y mejores,

incluso sucederles en el cargo.

Constantemente se producen también una serie de debates políticos que pueden llevar a los partidos más minoritarios a desaparecer del parlamento, o a algunos de los más votados a confluir y unirse en un pacto para asegurarse estar presentes en el gobierno. Entre todos los candidatos aportados por cada partido se escoge como presidente del parlamento a aquel que sea considerado el mejor político. En caso de que haya un único partido o confluencia, se escoge igualmente al mejor representante de dicha agrupación.

Este algoritmo tiene un funcionamiento sencillo pero a la vez es muy intuitivo de asociar con la sociedad, ya que se ubica en un entorno muy habitual y cotidiano. La política está presente en prácticamente cada acto de la vida diaria y la estructura de un parlamento es por todos conocida. En todo caso, los debates políticos que allí acontecen son accesibles para cualquier ciudadano y es común entender cómo funcionan y cómo se desenvuelven los partidos políticos para ganar escaños y hacerse con las elecciones.

A continuación se incluye un pseudocódigo del algoritmo para comprender de forma general cuál es el funcionamiento en cuanto a pasos a seguir y procedimientos.

Algorithm 2 Parliamentary Optimization Algorithm

1: $parlamento \leftarrow inicializarPartidos()$
2: $candidatos_{partido_i} \leftarrow mejoresMiembros(partido_i)$
3: **while not** *criterioParada* **do**
4: **for each** *partido* **do**
5: $miembro_i \leftarrow (candidatos_{partido} - miembro_i) \times bias$
6: **if** $peso(miembro_i) < peso(candidato_{j,partido})$ **then**
7: $candidato_{j,partido} \leftarrow miembro_i$
8: $poder_{partido} \leftarrow computarPoder(partido)$
9: **confluir los 2 mejores partidos con probabilidad** p_m
10: **eliminar el peor partido con probabilidad** p_d
11: **return** *min(peso(candidatos))*

Se empezará a analizar este algoritmo relacionando los conceptos técnicos presentados en este pseudocódigo con la descripción en lenguaje natural realizada previamente.

En el caso de esta propuesta, la población inicial está compuesta de soluciones que se interpretan como individuos humanos. A lo largo de este algoritmo se asociará cada una de las soluciones con un político particular, y su evaluación en la función objetivo del problema determinará lo válido que es dicho político dentro del parlamento.

Con los primeros políticos generados, este algoritmo los agrupa por par-

tidos, tantos como se haya indicado en los parámetros del mismo. Además, el algoritmo recibe como parámetro el número de candidatos que tiene cada partido, así que debe seleccionar de entre los mejores políticos tantos como candidatos haya, para asignarlos a los distintos grupos parlamentarios. Estos se repartirán equitativamente el resto de miembros del parlamento, para partir desde una situación de igualdad.

En este punto ya se ha generado la situación inicial del problema, y se procede a entrar en el bucle principal. En cada iteración primero se produce una competición interna dentro del propio partido, y después se da pie a una cooperación entre partidos que puede llegar a buen puerto o no.

La competición interna consiste en mover todos los políticos hacia sus candidatos, con un sesgo determinado que evite una congregación de todo el partido en un único punto. Para ello se sigue la siguiente fórmula

$$p' = p_0 + \eta(\frac{\sum_{i=1}^{\theta}(p_i - p_0) \times f(p_i)}{\sum_{i=1}^{\theta} f(p_i)}) \tag{4.3}$$

donde:

- p' es la nueva posición
- p_0 es la posición actual del político
- p_i representa a cada candidato del partido
- η es el sesgo que se aplica al movimiento
- $f(p)$ es la evaluación de la función objetivo para el político p
- θ es el número de candidatos por partido

Lo importante en este algoritmo es que, según viene especificado en la propia descripción del mismo, el desplazamiento únicamente tiene lugar **si la nueva posición es mejor que la anterior.** Si en el proceso de desplazamiento algún político resulta estar mejor ubicado que alguno de los actuales candidatos, estos intercambian sus puestos y el anterior miembro regular del partido pasa a ser un nuevo candidato. De esta forma el resto de los políticos en futuras iteraciones se desplazarán hacia esta nueva solución.

Cuando termina la competición interna del partido, se produce la fase de cooperación. En este momento existen dos parámetros p_m y p_d que simbolizan la probabilidad de que los dos mejores partidos se unan (*merge*) y de que el peor partido sea expulsado del parlamento (*delete*), respectivamente. En caso de que se dé la primera condición probabilística, los dos mejores partidos dejan a un lado sus diferencias y confluyen en uno solo, escogiendo

como candidatos a los mejores miembros entre ambos partidos sin importar su partido de origen. Si por otro lado, se produce el segundo caso, el partido es eliminado del parlamento y todos los miembros desaparecen del mismo, por lo que el parlamento ve reducido su tamaño, haciendo hincapié en las mejores soluciones para el resto de sus iteraciones.

Este algoritmo basa su funcionamiento en el **Particle Swarm Optimization** o **PSO**, comportándose de manera muy similar pero dando un carácter socio-político al entorno del mismo. En este caso cada partido funciona como un algoritmo PSO, siendo los políticos las partículas y los candidatos aquellas situadas en mejor posición a las que el resto se acercan. En cada iteración la posibilidad de hacer confluir dos partidos puede permitir aumentar la exploración, entendiéndose que un político puede optar a acercarse a un candidato que antes no conocía, y por el camino explorar una zona desconocida.

La principal novedad que aporta este algoritmo con respecto al PSO es la de tomar como referencia de desplazamiento un número mayor de partículas, en particular el que se desee y se especifique en la llamada al algoritmo. Esto permite que se puedan generar más de una «buena posición» hacia la que dirigirse, explorando también los caminos intermedios.

La crítica a este algoritmo se basa en la forma de realizar el desplazamiento, que limita la exploración. Al no permitir que las partículas pasen a ocupar una peor posición se puede caer fácilmente en un óptimo local si se estabilizan en las primeras iteraciones. Además, el sesgo es un parámetro fijado según han documentado los propios autores [6] y eso resta aleatoriedad a la búsqueda y puede frenarla totalmente en un determinado punto en el que siempre se realice el mismo desplazamiento y ninguna partícula mejore su posición.

4.3. Social Emotional Optimization Algorithm (SEA)

El siguiente algoritmo de esta lista en orden cronológico es el **Social Emotional Optimization Algorithm**, o **SEA** de acuerdo a las siglas elegidas por los autores [7] para representarlo. Esta propuesta se presentó en 2010 en el libro *Swarm, evolutionary, and memetic computing. First international conference on swarm, evolutionary, and memetic computing, SEMCCO* [22]. Este algoritmo podría ser englobado en la subcategoría de socioinspirados basados en interacción social y cultural según Kumar et al. [2].

Esta técnica socioinspirada está fundada en las relaciones sociales de las personas y cómo reaccionan en función de sus aciertos y errores o de la forma de actuar de los demás. Para ello hay que tener en cuenta las acciones pasadas y cómo influyeron en el resto de la sociedad, y a su vez comprobar

cómo influye en un individuo el comportamiento de los demás.

En cuanto a esto, es de esperar que si un individuo ve que el resto de la sociedad encuentra posiciones mejores a la suya, su índice de emoción disminuya, al igual que aumenta por completo si se coloca como el individuo con mejor posición en toda la población. Al principio todos los individuos comienzan con la moral al mismo nivel, pero esta variará dinámicamente en función de los resultados que se vayan obteniendo.

Cuando llega el momento de que cada individuo realice su siguiente movimiento, tiene tres posibles formas de hacerlo:

- Si su moral está muy alta porque es consciente de que tiene una buena posición, su movimiento simplemente le lleva a alejarse de las peores posiciones a la par que se fija en su mejor posición histórica.
- Si su moral es media porque aún está algo conforme consigo mismo a pesar de que sabe que puede mejorar, se alejará de las peores posiciones a la vez que busca un movimiento hacia la mejor posición que él mismo ha tenido y la mejor posición registrada por la población hasta ese momento.
- Si su moral es muy baja porque es conocedor de lo mala que es su posición, buscará inspirarse en la mejor solución de la población hasta el momento.

El paso del tiempo da lugar a que las distintas soluciones evolucionen y sepan el lugar al que pertenecen dentro de la sociedad. En el siguiente pseudocódigo se puede ver con más claridad cómo funciona el algoritmo.

Se puede empezar el análisis del algoritmo entendiendo cómo se representan técnicamente la población y los distintos componentes observados en el pseudocódigo, y por tanto cómo se pueden relacionar los mismos con una técnica socioinspirada.

En primer lugar, la población representa una sociedad de seres humanos, donde cada uno de los individuos de la misma es una solución al problema, generada aleatoriamente para la población inicial. En esta sociedad será donde los individuos crezcan intentando ser mejores y situarse por encima del resto. El índice de emoción de cada individuo será un valor real entre 0 y 1, siendo 1 para todos los individuos al inicio del algoritmo. El índice aumentará o decrecerá en función de los éxitos o los fracasos del individuo con respecto a sí mismo y al resto de la población.

En la primera iteración del algoritmo, como todos los índices están a 1, se produce un movimiento especial que no tiene lugar en el resto del proceso. Este movimiento simplemente aleja al individuo de las peores posiciones

Algorithm 3 Social Emotional Optimization Algorithm

```
1: sociedad ← inicializarPoblacion()
2: emocion_i ← 1.0
3: sociedad_i ← comportamiento1()
4: while not criterioParada do
5:     if emocion_i > limite1 then
6:         sociedad_i ← comportamiento4()
7:     else if limite2 > emocion_i ≥ limite1 then
8:         sociedad_i ← comportamiento3()
9:     else
10:        sociedad_i ← comportamiento2()
11:    if peso(sociedad_i) < peso(minimoHistorico) then
12:        emocion_i ← 1.0
13:    else
14:        emocion_i ← emocion_i − Δ
15:    actualizar mejores posiciones históricas
16: return peso(minimoHistorico)
```

que hay actualmente en la población, como mecanismo de defensa para no acabar en una situación como esa. Las soluciones a las que les haya tocado caer en las peores posiciones verán sus índices de emoción reducidos hasta que logren desplazarse a una zona mejor siguiendo el camino de un individuo con éxito.

Este algoritmo guarda una posición histórica para cada individuo, la mejor que ha conseguido a lo largo de las iteraciones. Esta posición se usará tanto para las fórmulas de desplazamiento como para guardar la mejor solución hasta el momento, ya que este algoritmo no es elitista y permite que las mejores soluciones se pierdan de la población actual. Guardándolas se asegura sin embargo poder obtener la mejor solución encontrada a lo largo de las iteraciones cuando haya que devolver el valor final del proceso.

Cada vez que se realiza un movimiento, el índice de emoción del individuo cambia. Si el movimiento le coloca en la mejor posición histórica, su emoción aumenta hasta el máximo valor, mientras que si no lo consigue dicho índice se verá decrementado en un parámetro Δ definido previamente con el algoritmo. En función de la emoción del individuo, en cada iteración optará por seguir un comportamiento u otro a la hora de desplazarse. Un comportamiento u otro queda determinado por unos límites que dividen el dominio del índice, [0,1], en tres secciones. Para ello son necesarios dos límites, que se definen como parámetros del algoritmo.

El SEA puede ser interpretado como un **Particle Swarm Optimization (PSO)**, al que se le han realizado algunas modificaciones, como la

memoria de posiciones históricas y la variación del comportamiento en función del índice de emoción. La idea de tener un comportamiento distinto adecuado a la situación en el tiempo de cada solución es muy interesante y da lugar a múltiples ideas que pueden mejorar esta técnica. Como algoritmo socioinspirado, este intenta darle un significado con sentido a la toma de decisiones de los individuos, para que cada rango de valores del índice de emoción corresponda a un comportamiento u otro.

Al final, sin embargo, el proceso sigue siendo el mismo que en un PSO en el que las partículas (individuos) buscan acercarse a la que saben que es la mejor, aunque en este caso esta puede ser la mejor histórica y no estar presente en la población del momento. La crítica a este algoritmo recae en el manejo de los índices de emoción, ya que es muy fácil que decaigan si se ha llegado a un óptimo local ya que el algoritmo en ningún momento volverá a explorar lejos de ese punto. Si todas las soluciones posteriores resultan son peores que aquella que ha ocupado el puesto de mejor histórica los índices de todos los individuos decaerán hasta el 0, y por tanto se limitarán a buscar en los alrededores de dicho óptimo local.

4.4. Anarchic Society Optimization (ASO)

El algoritmo **Anarchic Society Optimization**, por sus siglas **ASO**, fue presentado por primera vez en 2011 por Ahmadi Javid [8] en el Congress of Evolutionary Computation (CEC). A partir de ese momento se ha utilizado este algoritmo para resolver algunos problemas reales, como este [20].

Esta técnica ha sido agrupada por Kumar et al. [2] en el subapartado de socioinspiradas basadas en colonización. Sin embargo, de acuerdo al análisis que se ha realizado para este estudio se ha determinado que un mejor grupo para él sería el de ideologías socio-políticas, y en la siguiente explicación se exponen los argumentos.

ASO es un algoritmo basado en una sociedad anarquista, en la que la figura del gobierno no es necesaria y donde los propios individuos se organizan a sí mismos. Por separado o mediante grupos, podrán determinar cuál es el la mejor decisión posible en cada caso en base a sus propias experiencias y a las del resto de la sociedad.

Para ello, cada individuo tiene que tomar una decisión cuando realice un movimiento: tiene que valorar si le conviene más basarse en su propia situación, en las experiencias del resto de la sociedad actual o en lo que ocurrió en el pasado. Si realiza esta comparación de forma equilibrada y aprovechando toda la información de la que dispone (se supone que en esta sociedad el individuo está al tanto de las decisiones que toman el resto) podrá llegar hasta una posición más favorable, o incluso óptima.

Además, los individuos de una sociedad anárquica pueden experimentar cambios radicales de parecer y dirigirse hacia posiciones que no tienen por qué ser necesariamente mejores. En palabras del propio autor original, los individuos «también se comportan aventurada e irracionalmente, moviéndose hacia posiciones inferiores que ya han visitado».

Cuando han valorado todas las posibilidades de movimiento según las directrices indicadas anteriormente, los individuos deciden qué estrategia seguir para tomar su nueva posición. Pueden quedarse con la estrictamente mejor de las soluciones planteadas o bien intentar escoger lo mejor de cada una de ellas.

A lo largo del tiempo se espera que al menos un buen porcentaje de la sociedad haya aprendido las buenas conductas y descubra la zona del espacio donde mejor ubicados pueden estar, aunque haya otros miembros que no lo tengan tan claro y deambulen por el resto del espacio explorando diferentes rutas. Al final resulta ambicioso esperar que toda la sociedad converja en un mismo punto, y es normal que se produzcan escisiones así, como bien representa el algoritmo.

A continuación se muestra un pseudocódigo que ayuda a expresar este proceso en términos más cercanos a la implementación.

Algorithm 4 Anarchic Society Optimization

1: $sociedad \leftarrow inicializarPoblacion()$
2: **while not** *criterioParada* **do**
3: $\quad Mp_i^{current} \leftarrow movimientoBasadoEnPropiaPosicion()$
4: $\quad Mp_i^{society} \leftarrow movimientoBasadoEnSociedad()$
5: $\quad Mp_i^{past} \leftarrow movimientoBasadoEnHistoria()$
6: $\quad sociedad_i \leftarrow seleccionMovimiento(Mp_i^{current}, Mp_i^{society}, Mp_i^{past})$
7: $\quad$ **actualizar mejores posiciones históricas**
8: **return** *peso(minimoHistorico)*

Este algoritmo no es muy complejo en cuanto a instrucciones se refiere, sin embargo a continuación se analizará cómo se ha implementado la propuesta y qué decisiones se han tomado.

En primer lugar es conveniente notar que todo el peso de este algoritmo recae sobre las decisiones de movimiento dado que es realmente lo que tiene carga de trabajo en cada iteración. Este algoritmo solamente debe mantener la mejor posición que cada individuo ha conseguido hasta esa iteración y utilizar tres fórmulas matemáticas que determinen la nueva posición acorde a cada movimiento. La cuestión es averiguar si dichas fórmulas son suficiente para hallar una solución óptima en el espacio de búsqueda.

El movimiento basado en su situación actual viene determinado por un

parámetro definido como el autor como «inestabilidad» (*fickleness*) o tendencia al cambio. Un valor bajo de este atributo indicará al individuo que debería seguir explorando el espacio por sus cercanías, mientras que un valor alto repercutirá en que actúe de forma individualista y cambie radicalmente de posición.

En el movimiento basado en el resto de la sociedad influye, por supuesto, la mejor solución que el individuo puede encontrar en la iteración actual. Él conoce la posición del resto de sus compañeros, y sabe en quién debe fijarse, pero en lugar de automáticamente copiarlo calcula un «índice de irregularidad externa». De nuevo existe un umbral relacionado con este índice que determina si el individuo debe proceder a desplazarse hacia la mejor posición de la iteración actual o por el contrario se rebela y actúa por su cuenta generando una posición alejada de la misma.

Por último, el movimiento basado en el registro histórico es muy similar al que se acaba de explicar. En este caso la posición en la que el individuo se fija no es la mejor actual sino la mejor histórica, y el índice calculado se denomina «índice de irregularidad interna». Existe también un umbral análogo, solamente que en este caso el desplazamiento se realiza lógicamente hacia la mejor posición que ha habido en todas las iteraciones.

Con todos los posibles movimientos evaluados, el individuo debe tomar una decisión. En este punto entra el operador de selección para elegir con qué movimiento se queda, y la definición del algoritmo en el *paper* original aporta dos posibles soluciones: que se elija según un modelo **elitista**, quedándose con la mejor posición de las tres generadas, o que se elija según un modelo **secuencial**, y en cada iteración cambie de uno a otro siguiendo la misma serie. Cada uno tiene sus ventajas e inconvenientes, que además son principalmente excluyentes del otro método. En el caso elitista se consumen muchas evaluaciones de la función objetivo, el triple que en otro algoritmo evolutivo común, pero sin embargo se produce una exploración del espacio más variada. En el caso secuencial tan sólo se producen una tercera parte de las evaluaciones, pero estas no dan la seguridad de ser la mejor en cada caso. En lo que a la implementación de este estudio corresponde, se ha optado por desarrollar la opción **elitista**.

Este algoritmo también basa su funcionamiento en un **Particle Swarm Optimization (PSO)**, ya que hasta el propio autor dedica un apartado a la comparación del ASO con el PSO [8]. En dicho apartado hace una similitud entre los vectores de velocidad del PSO y los movimientos basados en cada uno de los comportamientos anteriormente descritos. En general, asemeja la forma de desplazarse de cada uno de los algoritmos buscando en cada caso del ASO un objetivo distinto dependiendo del movimiento escogido.

4.5. Soccer League Competition (SLC)

El siguiente algoritmo en orden cronológico es el **Soccer League Competition**, **SLC** según sus siglas. Fue presentado en 2014 por Moosavian y Kasaee Roodsari [4], en una propuesta para resolver ecuaciones no lineales. Más adelante el propio Moosavian ha presentado otras publicaciones aplicando este algoritmo a problemas de la mochila (*knapsack problems*) [23] y a un problema más real, como la distribución de agua en una red [5]. El algoritmo puede ubicarse en el subapartado de socioinspirados basados en comportamiento competitivo, según Kumar et al. [2]

En este algoritmo se ubica el problema en un marco deportivo, en particular en una liga de fútbol. La población está constituida por equipos, cada uno de ellos con sus diferentes jugadores. A su vez, se distingue entre jugadores titulares y suplentes, y unos lógicamente tienen más importancia que los otros. En este panorama el objetivo de cada jugador es llegar a ser el mejor de la liga, o en su defecto al menos ser lo suficientemente bueno como para que los mejores equipos de la competición quieran contar con él en su plantilla.

Los equipos compiten entre sí a lo largo de temporadas, y los jugadores mejoran en función de su resultado en los partidos, y de la influencia que hayan tenido en el juego. Un jugador titular mejorará más que uno suplente, y si algún titular baja el nivel puede ser relegado al banquillo. A su vez, si un jugador suplente no da la talla para el equipo, este puede despedirle y fichar un nuevo recambio para el próximo partido. El movimiento dicta que los titulares buscan parecerse al «*Super Star Player*» para optar a jugar en mejores equipos, mientras que los suplentes intentan llegar al nivel medio de los titulares de su equipo, a fin de encontrar un puesto en la rotación. Con respecto al equipo perdedor, un porcentaje de sus jugadores intentará entrenar alguna nueva técnica con la que vencer a ese equipo en el próximo partido.

Cuando acaba la temporada los mejores equipos se hacen con los mejores jugadores, dejando los peores para los equipos más humildes. Así, la siguiente temporada los jugadores más prometedores pueden seguir mejorando al estar rodeados de un entorno a su altura, y los humildes pueden destacar dentro de su propio equipo, pudiendo llegar a llamar la atención de uno más potente. El mejor jugador, o «*Super Star Player*» según el autor, será el que el algoritmo devuelva como resultado de todas las temporadas.

A continuación se muestra un pseudocódigo del algoritmo que explique de manera más técnica los detalles que lo rodean.

Con este pseudocódigo comprendido, se procede a relacionar el comportamiento explicado anteriormente con las instrucciones que se muestran, así

Algorithm 5 Soccer League Competition

1: $liga \leftarrow inicializarLiga()$
2: **while not** *criterioParada* **do**
3: **for each** $par(equipo_i, equipo_j)$ **do**
4: $vencedor, perdedor \leftarrow partido(equipo_i, equipo_j)$
5: $titulares_{vencedor} \leftarrow operadorImitacion()$
6: $suplentes_{vencedor} \leftarrow operadorProvocacion()$
7: **reordenar jugadores en función de su fitness**
8: **if** *condicionDescenso* **then**
9: $equipo_{peor} \leftarrow nuevoEquipo()$
10: **return** *peso(superStarPlayer)*

como a asociar cada uno de los componentes a su contraparte socioinspirada.

En primer lugar la población en este caso está compuesta de jugadores. Ellos son los que representan las soluciones al problema, y que irán evolucionando conforme el proceso algorítmico tenga lugar. Esta población se divide en equipos, cada uno de los cuales cuenta con sus propios jugadores, y dicha plantilla con jugadores titulares y jugadores suplentes. Los titulares serán las mejores soluciones del equipo y los suplentes aquellas no tan buenas.

Las soluciones deben estar ordenadas en cada «temporada» (que corresponde a una iteración del bucle principal), por lo que el primer equipo de la liga siempre empezará la temporada con los mejores jugadores, mientras el último será el que se quede con los peores. Sin embargo, a mitad de cada temporada puede darse el caso de que un jugador de un equipo más humilde llegue a dar una solución suficientemente buena como para ocupar las mejores posiciones de la liga. En dicho caso, este jugador será fichado por el mejor equipo cuando termine la temporada.

El proceso de la temporada consiste en que cada equipo juegue un partido contra el resto de equipos, y en función del resultado sus jugadores mejorarán de una u otra manera. Este partido se simula calculando el coste de cada equipo participante, estimando la probabilidad de cada uno para salir vencedor, y a continuación se genera un valor aleatorio que determine qué equipo es el que gana.

Los jugadores titulares del equipo ganador realizan un movimiento hacia el mejor jugador de la liga, el «*Super Star Player*». Si dicho movimiento no supone una mejora, realizan un movimiento hacia el «*Star Player*» de su propio equipo, es decir, su mejor jugador. En caso de que tampoco se encuentre una mejora, intenta hacer un último movimiento aleatorio hacia una posición totalmente distinta. Si el resultado sigue sin mejorar, el jugador no se mueve a consecuencia de ese partido.

Los jugadores suplentes del equipo vencedor deben intentar parecerse a los jugadores titulares, con esperanza de ocupar su puesto. Para ello se introduce el operador de gravedad, que aplicado a los jugadores titulares del equipo representa la posición media que ocupan en el espacio de búsqueda. Ese será el punto al que el jugador suplente intente desplazarse. En caso de que no dé un buen resultado, realiza un movimiento similar a la inversa, alejándose de ese centro de gravedad. Si este movimiento tampoco supone una mejora a la posición inicial el jugador es despedido del equipo y un nuevo suplente generado aleatoriamente ocupa su lugar.

Con respecto al equipo perdedor, el *paper* original no especifica nada, y sin embargo el código MATLAB descargado [24] incluye una ligera mutación que aporta variedad al proceso de búsqueda. Esta modificación ha sido plasmada en la implementación de este estudio, y se basa en seleccionar tres miembros de la plantilla titular y mutar un porcentaje de sus componentes. Solamente si la mejora resulta mejor que el original, la mutación queda registrada. A continuación se seleccionan dos suplentes de dicho equipo perdedor y se genera un cruce genético entre ellos, obteniendo dos nuevas soluciones. El equipo reordena su plantilla y se queda con los mejores jugadores de todos los generados, liberando a los peores.

Finalmente, al final de la temporada los jugadores vuelven a ser ordenados, pero esta vez a nivel global, lo que simula el fichaje de los mejores jugadores por el mejor equipo. Los suplentes del primer equipo son mejores que los titulares del segundo equipo, y así sucesivamente, planteando una liga donde claramente existe un equipo con suficiente poder adquisitivo para hacerse con todos los mejores jugadores. Si se ha especificado el componente de descenso, cuando se haya cumplido la condición necesaria el peor equipo queda eliminado de la competición y es rellenado con nuevos jugadores generados aleatoriamente, a fin de aportar mayor exploración a lo largo de todo el proceso.

4.6. Ideology Algorithm (IA)

Por último, el algoritmo socioinspirado más reciente del estudio es el **Ideology Algorithm**, también referido como **IA** por sus siglas. Se trata de un algoritmo presentado por Ting Huan et al. [10] en 2016, por lo que se puede considerar muy actual en el momento de redacción de este trabajo. En su agrupación de algoritmos socioinspirados, Kumar et al. [2] lo clasifican en el grupo de los basados en ideologías socio-políticas.

Esta propuesta parte de una idea similar a la del algoritmo **POA** [6], presentado previamente en este mismo capítulo, ya que también tiene lugar en un contexto político en el que distintos partidos tratan de hacerse con

el control del gobierno. Estos están representados por un líder, el que se equiparía al presidente del partido, aunque la figura del vicepresidente, o segundo mejor miembro del partido, también juega un papel fundamental.

En este algoritmo, el líder tiene que intentar mantenerse al frente de su partido a la par que intenta ascender hasta colocarse como presidente del gobierno. Para esto pasa por tres fases, una de introspección, otra de competición local y una tercera de competición global. Durante la fase de introspección, el líder del partido busca en torno a su actual posición para intentar mejorar de cara a la siguiente fase. Durante la competición local busca en torno al segundo mejor político de su partido, al vicepresidente, para intentar mantener su puesto y adelantarse a un posible movimiento futuro de este. Por último, en la competición global el líder calcula un movimiento similar a los anteriores pero con respecto al actual presidente del gobierno. Una vez que ha valorado todos los movimientos, selecciona uno utilizando el método de la ruleta («*Roulette Wheel Selection*») presentado por Kumar y Jyotishree en 2012 [25].

Cuando el líder de cada partido ha realizado su movimiento, y previamente a que el resto de los miembros tomen su propio movimiento en el debate político, el peor miembro de cada partido tiene que valorar si sigue sintiéndose afín a dicha organización o si prefiere abandonarla. En caso de que sus diferencias con el segundo peor miembro sean suficientemente grandes, el miembro abandona el partido y se afilia a otro seleccionado al azar.

En este momento sí se produce el movimiento del resto de miembros. Estos buscan mejorar su posición en los alrededores, a la par que intentan parecerse al líder de su partido. Tras actualizar su posición, si alguno de los miembros se ha convertido en el mejor político del partido pasa a ocupar el puesto de líder. En resumen, se reorganiza el partido en función de las nuevas aptitudes de los miembros.

Finalmente, el político que ocupe el puesto de presidente del gobierno cuando termine la ejecución del algoritmo será considerado la solución óptima encontrada por el algoritmo. En el siguiente pseudocódigo se puede este funcionamiento.

Con el pseudocódigo claro se puede proceder a analizar el funcionamiento del algoritmo adecuadamente, utilizando terminología más directa.

En primer lugar se puede relacionar la población inicial con el parlamento, o el gobierno, depende de la interpretación que se dé a este contexto. Sea como fuere, cada individuo del mismo, reflejado como político en el análisis representado anteriormente, es un vector solución al problema. Estas soluciones serán las que vayan evolucionando a lo largo de las iteraciones del algoritmo buscando encontrar el óptimo global de la función objetivo.

Algorithm 6 Ideology Algorithm

```
1: partidos ← inicializarPoblacion()
2: while not criterioParada do
3:     ordenar políticos en función de su fitness
4:     introspeccion_i ← introspeccion(lider_i)
5:     compLocal_i ← competicionLocal(lider_i)
6:     compGlobal_i ← competicionGlobal(lider_i)
7:     lider_i ← seleccionMovimiento(introspeccion_i, compLocal_i, compGlobal_i)
8:     if (penultimo_i − peor_i) > T then
9:         peor_i ← elegirNuevoPartido()
10:    for each politico do
11:        politico ← nuevoDesplazamiento()
12: return min(peso(lider_i))
```

Ordenando a los políticos en función de su ajuste o fitness se puede obtener en cada iteración quién es el miembro de cada partido que actúa como líder. Este será el que experimente las tres fases previas al movimiento del resto de miembros. En la fase de introspección, se realizará una pequeña mutación en las características de su vector solución, a fin de explorar el espacio de búsqueda contiguo a su actual posición. En la competición local y en la global, en la que el líder del partido se intenta aproximar a las posiciones contiguas a la segunda mejor solución de su partido y a la mejor posición global existente en dicha iteración, respectivamente, se sucede un proceso similar pero cambiando el objetivo de desplazamiento.

Estos movimientos constituyen **una búsqueda local** que tiene lugar siempre sobre la mejor solución de cada partido, lo que puede ayudar a mejorar los resultados dedicando más evaluaciones a aquellas soluciones con potencial de ser óptimas.

Y si estos movimientos potencian la fase de explotación, la fase de exploración a su vez se ve reforzada por la posibilidad que tienen los políticos de abandonar un partido. El símil con no ser afín a su partido viene determinado por la distancia existente entre el inmediatamente peor representante del partido y él mismo. Si esta distancia supera el límite de deserción establecido en el algoritmo, el político se unirá a otro partido al azar. Esto permite que se puedan explorar nuevas situaciones ya que acercarse a uno u otro líder político puede ser sustancialmente distinto dependiendo del miembro que realice el desplazamiento.

Cuando todos los demás miembros del parlamento se han movido, en este caso hacia el líder de su partido y explorando a la par su posición contigua, se da por finalizada una iteración del bucle principal y se reordenan las posiciones de cada partido. Además, si alguno de los nuevos líderes resulta

ser mejor que el actual presidente, porque posee un valor de fitness mejor, este pasa a ocupar dicho cargo.

El algoritmo IA tiene sus raíces también en un Particle Swarm Optimization, como los propios autores indican en su artículo [10]. Cada partido funciona como un pequeño modelo de PSO, en el que el líder político del partido simboliza la partícula mejor situada hacia la que el resto se desplazan. Al existir un variado número de pequeños PSOs, la exploración es mayor, pero aún destaca más por el hecho de que estos grupos son capaces de interactuar entre ellos en el proceso de exploración. Así, el líder de un partido puede verse influenciado por la posición de otro, o bien un miembro recién afiliado al partido puede ayudar a dirigirlo a una zona inexplorada del espacio.

Capítulo 5

Implementación

En este capítulo se mostrarán los detalles más destacados de la implementación de cada algoritmo estudiado, comentando sus particularidades principalmente en cuanto a parámetros, explicando cómo utilizarlo y resumiendo lo que ha supuesto esta fase en cuanto al desarrollo completo del proyecto. Además se realizará una crítica acerca de aquellas situaciones que, por malas prácticas ajenas a este estudio, han supuesto taras en el desarrollo del mismo.

5.1. Detalles acerca de la implementación

En primer lugar, antes de comenzar a valorar las características de implementación de los algoritmos, conviene dar una visión general acerca de cómo se ha llevado a cabo este estudio y con qué herramientas.

Los algoritmos se han implementado utilizando el lenguaje de programación **Python** en su versión 3.6 [26], elección personal del alumno, dada su facilidad para trabajar con vectores y matrices utilizando librerías tan útiles como **Numpy** [27]. Poder operar de forma sencilla con múltiples matrices que habitualmente constituirán la población de soluciones de cada problema resultó un detalle determinante para tomar la decisión.

Encontrar el código fuente de los algoritmos mencionados resulta una ardua tarea, ya que la mayoría de los autores no liberan su código en ninguna plataforma. En algunas ocasiones es posible encontrar versiones en el lenguaje de programación **MATLAB**, uno de los lenguajes más utilizados en el ámbito científico y de investigación por su capacidad de cómputo y de operar con fórmulas matemáticas de gran dimensión. Sin embargo, MATLAB es un software privativo y de pago, y el alumno ha querido liberar su proyecto desde el primer momento, por lo que se rechazó la idea de implementarlos en este lenguaje. Sin embargo, sí que se ha utilizado como referencia para algún

algoritmo una versión de MATLAB sobre la que basar la implementación en Python.

En relación a lo comentado en el párrafo anterior, el proyecto ha estado libre y alojado en la plataforma **GitHub** [28] desde que se comenzó la implementación, para que cualquier usuario interesado pudiese no sólo acceder a él sino también comprobar cómo está hecho, y en un futuro incluso realizar los cambios pertinentes que considerase, cumpliendo así el paradigma del software libre.

Con respecto a detalles de diseño de los algoritmos, se han implementado como **funciones que pueden ser importadas y llamadas** desde cualquier fichero Python que tenga acceso a ellas. Cada algoritmo recibe una serie de parámetros distinta, que depende de la interpretación que se ha hecho con respecto al *paper* o de los que se encuentren en uno de los códigos previamente implementados en otro lenguaje. Algunos de los parámetros son comunes dado a la evidente importancia de los mismos en cualquier algoritmo evolutivo, como la función de evaluación, el tamaño de la población, el dominio de las variables, la dimensión del problema o el número máximo de evaluaciones. El resto de parámetros propios de cada algoritmo que haya sido interesante modificar para la experimentación se especificarán en el siguiente apartado.

Como extra que no incluían las propuestas, y como hay funciones que así lo requieren, se ha implementado para cada algoritmo la posibilidad de incluir un dominio de variables para la población inicial, para aquellos casos en los que el óptimo se encuentre fuera de esos límites que se fijan al inicio del problema.

Por último, a fin de obtener valores de convergencia para las ejecuciones de los algoritmos, se devuelve junto a la mejor solución un vector con los valores mínimos que se obtuvieron cada 10 % de evaluaciones.

5.1.1. Imperialist Competitive Algorithm (ICA)

La implementación del ICA en Python no pudo ser encontrada en internet, sin embargo sí que se encontró el código fuente en MATLAB, subido por el propio autor del *paper* [29]. Este código se ha usado como referencia para implementarlo de forma similar en Python, ya que algunos detalles varían con respecto a la literatura.

Para lanzar este algoritmo hay que importar la función `ICA` del fichero Python con el mismo nombre. Para esta función son muy importantes los siguientes parámetros, aparte de los mencionados en el apartado anterior:

- `ncountries` ⇒ indica el tamaño de la población inicial.

- `nimperialists` ⇒ indica el número de imperios en los que se dividirá el problema en la primera iteración.
- `evaluationCriteria` ⇒ valor booleano que especifica el criterio de parada a seguir por el algoritmo. Si se especifica a `True` será por evaluaciones de la función objetivo, o será por iteraciones del bucle principal si se indica `False`.
- `decades` ⇒ se tendrá en cuenta este valor como tope de iteraciones a realizar por el algoritmo si no se especifica criterio de parada por evaluaciones.

El resto de valores son parámetros técnicos especificados por el *paper* o extraídos del código MATLAB, por lo que se han dejado como aparecen por defecto.

5.1.2. Parliamentary Optimization Algorithm (POA)

Del algoritmo POA no ha sido posible encontrar código fuente, ni siquiera en MATLAB, por lo que la implementación es de autoría propia basada únicamente en lo que refleja el artículo, siendo lo más fiel posible a lo explicado en el mismo.

Para lanzar este algoritmo es necesario importar la función `POA` del fichero Python homónimo. Sus principales parámetros, aparte de los comunes, son los siguientes:

- `nparties, nmembers, ncandidates` ⇒ estos parámetros determinan la población inicial y su estructura, en lugar de ser definida mediante un parámetro único. `nmembers` representa el total de miembros de un partido, incluyendo miembros regulares y candidatos.

El resto de los parámetros son valores numéricos especificados en el paper que sirven para ajustar fórmulas, y que generalmente no van a cambiar salvo que se busque un ajuste de parámetros específico.

5.1.3. Social Emotional Optimization Algorithm (SEA)

El algoritmo SEA no ha sido posible encontrarlo implementado, ni en Python ni en ningún otro lenguaje, así que su implementación ha sido derivada únicamente del contenido del que se disponía en el *paper*.

Para lanzar este algoritmo se debe importar la función `SEA` del fichero Python del mismo nombre, y llamarla con los parámetros que se prefieran.

Aparte de los comunes indicados al comienzo de este capítulo, SEA cuenta con los siguientes, que han resultado importantes:

- `nindividuals` ⇒ define el tamaño de la población inicial.
- `k1, k2, k3` ⇒ son parámetros que afectan a los movimientos de los individuos, y que no tienen un significado semántico claro.
- `emotion_decrease` ⇒ el valor de emoción que pierde un individuo cada vez que no logra encontrar la mejor solución.
- `lower_threshold, upper_threshold` ⇒ definen los umbrales de separación de unos movimientos y otros.

5.1.4. Anarchic Society Optimization (ASO)

Los autores no han revelado el código de este algoritmo, y tampoco se ha encontrado ninguna implementación en ningún lenguaje en internet. Por tanto, este algoritmo se ha implementado únicamente en función de lo interpretado de los *papers*.

Para ejecutar el ASO se debe importar la función `ASO` del fichero del mismo nombre, y llamarla con los argumentos indicados al comienzo de este capítulo, además de estos particulares:

- `nindividuals` ⇒ el número de individuos de la población inicial.
- `fickleness_rate` ⇒ el índice de inestabilidad de los individuos, que influye en su toma de decisiones. Debe estar entre 0 y 1.
- `external_rate, internal_rate` ⇒ dos índices que influyen en la toma de decisiones, el primero para basarse en la mejor solución actual y el segundo para basarse en la mejor solución histórica.
- `external_threshold, internal_threshold` ⇒ dos umbrales que definen cómo de probable es que el individuo siga el camino correspondiente al parámetro anterior o que realice un movimiento totalmente aleatorio.

5.1.5. Soccer League Competition (SLC)

El autor del SLC subió el código a MathWorks [24] por lo que se pudo utilizar este como referencia para la implementación en Python. Tras leer detenidamente las publicaciones sobre este algoritmo, se ha seguido principalmente la estructura de este código apoyándose en los detalles explicados en los *papers*.

Para utilizarlo es necesario importar la función `SLC` del fichero Python homónimo, y llamarla con algunos de los siguientes parámetros principales:

- `nteams` ⇒ el número de equipos que competirán en la liga.
- `nmain, nsubs` ⇒ simbolizan, respectivamente, el número de jugadores titulares y suplentes que tendrá cada equipo.

El resto de los parámetros son algunos valores numéricos que ajustan los movimientos de los jugadores. En particular, se hablará de dos de ellos en el siguiente apartado.

5.1.6. Ideology Algorithm (IA)

El algoritmo IA ha sido el único del que se ha encontrado código en Python, en particular en este repositorio en GitHub, implementado por Néstor Rodríguez [30]. Por lo tanto, para este estudio no se ha tenido que implementar el algoritmo desde cero sino que, mediante un *fork* del repositorio, se han hecho los retoques que han sido necesarios para que funcione de manera similar al resto de algoritmos, y pueda devolver los mismos valores en el mismo formato, esencialmente.

Por la forma de implementación del algoritmo hay que tratarlo un poco distinto. Con la nueva implementación en el *fork*, accesible desde el GitHub del proyecto (ver README) [28], se puede importar la clase `IA` del fichero del mismo nombre, pero para poder realizar la ejecución es necesario primero crear el objeto con sus parámetros y luego llamar a su método `ideology_algorithm`.

Cabe destacar que el objeto no se reinicia una vez que se termina la ejecución del algoritmo y por tanto hay que utilizar un nuevo objeto cada vez si se quiere repetir cada ejecución una serie de veces, como ha sido el caso en este estudio.

Los parámetros más a destacar de este algoritmo son:

- `n_parties` ⇒ define el número de partidos del problema.
- `politicians` ⇒ define el tamaño de la población inicial. El número de políticos entre el número de partidos debe ser exacto para que el algoritmo no lance un error.
- `desertion_threshold` ⇒ define el umbral que determina cuándo un miembro del partido se siente descontento dentro del mismo.

5.2. Dificultades encontradas: ambigüedades y falta de información

En este apartado se realizará una crítica a aquellas malas prácticas relacionadas con la publicación de *papers* que se han cruzado en el camino de este estudio, y que han limitado su desarrollo por momentos, o lo han complicado en exceso.

5.2.1. Imperialist Competitive Algorithm (ICA)

En el código fuente que el autor subió a la página **MathWorks** [29] hay algunos detalles que cambian con respecto al *paper*. Ante la ambigüedad existente entre teoría e implementación, para este estudio se ha tomado la decisión de implementar en Python lo obtenido en el código. En este caso se trata de un componente de revolución que puede afectar a algunas colonias en cada iteración, y que permite que cambien radicalmente de posición. Gracias a este componente aleatorio se podría llegar a explorar una zona del espacio de búsqueda a la que no se haya llegado anteriormente y que aporte una mejor solución.

No obstante, implementar esta funcionalidad no ha supuesto un gran gasto de tiempo, ya que aparte de tratarse de una operación sencilla existía código de referencia en el que basarse.

5.2.2. Parliamentary Optimization Algorithm (POA)

Sobre el *paper* de este algoritmo no existe queja alguna. Los parámetros con los que se han realizado los experimentos están bien definidos, las fórmulas son claras y el planteamiento del problema está bien enfocado. Implementar este algoritmo no supuso ningún impedimento adicional a la carga de trabajo ya expuesta en su correspondiente capítulo. Lo único que puede recriminarse a los autores es no haber compartido el código, una práctica por desgracia habitual en el mundillo.

5.2.3. Social Emotional Optimization Algorithm (SEA)

Sobre el SEA sí conviene hablar de la falta de información, uno de los mayores problemas a los que se debe enfrentar quien quiera implementar un algoritmo en base a la información de una publicación. Muchos algoritmos requieren de parámetros que modifican los desplazamientos, o que determinan la decisión a tomar por una solución en un momento determinado. Sin estos parámetros es muy complicado tanto reproducir los experimentos

que se han hecho en la publicación como obtener buenas soluciones trabajando con dicho algoritmo sobre otras funciones. Además, en algunos casos ni siquiera se indica un rango de valores en el que se puedan inferir dichos parámetros.

En el caso del SEA se tratan de los parámetros *k1, k2* y *k3*. Al no definirse un valor para los mismos, ni siquiera un dominio estimado, estos han tenido que ser determinados mediante la experimentación. En una muestra básica de 4 funciones se comprobaron distintas combinaciones de estos parámetros para utilizar la mejor en los experimentos con el benchmark. Esta tarea, aparte de no asegurar un resultado óptimo, podía haberse evitado con una definición correcta y completa del problema en el *paper*.

5.2.4. Anarchic Society Optimization (ASO)

El caso del algoritmo ASO ha resultado especialmente complicado, ya que no solamente omite valores para parámetros vitales en las fórmulas, sino que ni siquiera especifica el operador de movimiento. Es decir, las publicaciones leídas sobre este algoritmo admiten la existencia de los parámetros, pero el único dato que aportan sobre ellos es su rango, o simplemente que no deben ser negativos.

Para solventar este problema en este caso se han tomado valores medios para los que se especifica un rango, y para el resto se ha realizado una prueba similar a la que se ha hecho para el algoritmo SEA. Por desgracia no se puede asegurar que la estimación sea óptima pero eso requeriría de un trabajo mucho más extenso.

5.2.5. Soccer League Competition (SLC)

El algoritmo SLC no especifica algunos parámetros en el paper pero al menos el autor subió el código para poder extraerlos de ahí. Lo cual provocó otro problema derivado, la ambigüedad entre código y teoría, como ha pasado en el otro ejemplo del que existe código fuente, el ICA.

Para el caso del SLC hay tanto cosas especificadas en la teoría y no en el código como al contrario. En el código aparece una modificación para los jugadores que pierden partidos, y además una función bastante extensa, mientras que en la teoría no se hace referencia al movimiento de los jugadores que salen derrotados. Con respecto a lo que aparece en la teoría pero no en la práctica, se trata del descenso de equipos cuando acaba la temporada. Basándose en lo que dicta el *paper* del algoritmo se podría pensar que en el código aparecería así, pero no. Aquí entran en juego dos nuevos parámetros, `mutation_probability` y `mutation_rate`, que determinan cuántas variables se mutan y en qué proporción, respectivamente.

La táctica a seguir ha sido implementar lo que aparece en la implementación del autor, si es que esta existe, ya que se entiende que es el código final sobre el que ha hecho las ejecuciones de su benchmark de funciones.

Capítulo 6

Diseño experimental

En este capítulo se analizará cómo se ha realizado el estudio de los algoritmos. Se comentará el benchmark de funciones que se ha utilizado, cómo se han diseñado los experimentos para poder ser ejecutados individualmente permitiendo paralelismos y cómo se han recopilado los datos.

Los algoritmos socioinspirados no han competido hasta ahora de forma profesional en las competiciones de los **Congress of Evolutionary Computation (CEC)**, por lo que ponerlos a evaluar funciones muy recientes podría ser precipitado. En su lugar, para este primer estudio de los algoritmos se ha preferido utilizar un benchmark que, aparte de ser un poco más sencillo que los más actuales, es más intuitivo de interpretar. Se trata del benchmark utilizado en el **CEC2005** [31], compuesto por:

- 5 funciones unimodales
- 20 funciones multimodales, de las cuales:
  - 7 básicas
  - 2 expandidas
  - 11 híbridas compuestas

Existe un estudio detallado [32] de las funciones de este benchmark que ha sido de mucha utilidad a la hora de tratarlas, de conocer sus propiedades e incluso de obtener una representación gráfica que ayude a visualizar los puntos óptimos. Este estudio servirá de referencia a la hora de analizar los resultados reflejados por el los algoritmos en cada una de las funciones del benchmark.

Para realizar los experimentos se han construido 10 scripts por cada algoritmo, 5 para problemas de dimensión 10 y 5 para los de dimensión 30, ejecutándose en cada uno de ellos un total de 5 funciones. El objetivo de

dividir el trabajo en varios scripts es aprovecharse del paralelismo disponible al contar con el cluster Hércules [14], ya que como además las evaluaciones de una y otra función no dependen entre ellas no es necesario recopilar los datos con un modelo map-reduce.

Cada función se ejecuta un total de 10 veces, recopilando el valor medio obtenido al final de las evaluaciones para cada una de esas ejecuciones. A fin de hacer cada experimento divisible del resto a la vez que reproducible, se fija una semilla para valores aleatorios previamente a cada función. Si la semilla se fija correctamente antes de que se procese el bucle que ejecuta el algoritmo 10 veces se obtendrá siempre el mismo conjunto de resultados. Es muy importante no confundir esto con inicializar la semilla dentro del bucle, lo que provocaría tener el mismo resultado en cada una de las 10 veces, y por tanto lo haría inútil.

Como suele ser habitual en los problemas de este estilo, el número máximo de evaluaciones por ejecución viene determinado por la dimensión del problema, siendo $10000 \times dim$ la fórmula en cuestión. Para cada algoritmo se extraen además los datos de convergencia para poder mostrarlos en una gráfica. Estos datos aportarán conocimiento sobre cómo de necesarias son todas las evaluaciones, o qué algoritmos encuentran el óptimo más rápido.

Para todos los algoritmos se ha elegido una población de tamaño 30 tanto para dimensión 10 como para dimensión 30. Esta limitación en el tamaño para la versión de dimensión 30 permitirá apreciar en los resultados qué algoritmos se adaptan mejor con un margen de exploración más reducido.

A continuación se incluyen en tablas los parámetros que se han escogido para la experimentación para cada algoritmo.

Parámetro	**Valor**	**Parámetro**	**Valor**
Núm. Imperialistas	8	ζ	0.1
Ratio revolución	0.3	**Ratio frenada**	0.99
Coef. Asimilación	2	**Umbral unificación**	3
Coef. Ángulo Asimilación	$\frac{\pi}{4}$		

Tabla 6.1: Parámetros de ejecución del ICA.

Parámetro	Valor	Parámetro	Valor
Núm. Partidos	6	**Probabilidad de unión**	0.01
Núm. Miembros	5	**Probabilidad de eliminación**	0.001
Núm. Candidatos	2	**Miembros a unir**	2
Ratio Peso Miembros	0.01	**Miembros a eliminar**	1
Ratio Peso Candidatos	1	**Sesgo**	0.3

Tabla 6.2: Parámetros de ejecución del POA.

Parámetro	Valor	Parámetro	Valor
$k1$	0.1	Δ	0.05
$k2$	0.4	**Umbral inferior**	0.3
$k3$	0.4	**Umbral superior**	0.6

Tabla 6.3: Parámetros de ejecución del SEA.

Parámetro	Valor	Parámetro	Valor
Ratio Inestabilidad	0.5	**Ratio Evolución**	0.5
Ratio Externo	3	**Ratio Interno**	1
Umbral Externo	0.5	**Umbral Interno**	0.5

Tabla 6.4: Parámetros de ejecución del ASO.

Parámetro	Valor	Parámetro	Valor
Núm. Equipos	5	**Probabilidad de mutación**	0.1
Núm. Titulares	3	**Ratio de mutación**	0.2
Núm. Suplentes	3		

Tabla 6.5: Parámetros de ejecución del SLC.

Parámetro	Valor	Parámetro	Valor
Núm. Partidos	5	**Umbral de deserción**	10
r	0.5		

Tabla 6.6: Parámetros de ejecución del IA.

Capítulo 7

Resultados del estudio

En este capítulo se recopilan las tablas y gráficas más representativas con los resultados obtenidos en el estudio. Se mostrará una tabla global para cada dimensión, que incluya el error medio obtenido con respecto al óptimo por función y algoritmo, además de una tabla de rankings que ayude a determinar qué algoritmos se comportan mejor en general. Además de las tablas, se incluirán algunas gráficas de convergencia para poder observar de manera más visual qué algoritmos llegan antes a los óptimos y cuáles necesitan de más evaluaciones, así como qué resultados son mejores y en qué proporción con respecto al resto. Por último, se dará una visión analítica acerca de los resultados, dando pie a distintas valoraciones bien fundamentadas que se pueden extraer de las tablas para obtener las conclusiones del estudio. Las tablas y gráficas han sido generadas gracias a la página *TACOLab* [33].

Para valorar mejor cómo funcionan los algoritmos del estudio, se ha escogido un algoritmo de referencia en el que muchos de ellos están inspirados, y se ha añadido a las tablas de resultados. Se trata de un Particle Swarm Optimization, un algoritmo que ya se describió anteriormente en este documento. Comparar los resultados que arroje este algoritmo con el de aquellos socioinspirados que se comportan de manera similar servirá para determinar si dicha modificación al PSO es significativa o por el contrario no aporta gran cosa. El código para ejecutar el PSO ha sido obtenido de este repositorio de GitHub [34].

En primer lugar se van a analizar los resultados para la dimensión 10. Para ello conviene fijarse en la tabla 7.1 que contiene el error medio con respecto al óptimo obtenido por cada algoritmo a lo largo de 10 ejecuciones sobre una misma función en dicha dimensión. Con un simple vistazo a la tabla se puede observar que en general en las funciones unimodales (desde F1 hasta F5) los algoritmos dan resultados muy malos, en especial PSO, el algoritmo de referencia. Y es que para estas funciones la mayoría de los algoritmos mejoran su resultado, salvo el POA que obtiene errores aún

	ASO	IA	ICA	POA	PSO	SEA	SLC
F01	5.071e+00	1.981e+03	6.705e+03	1.460e+04	2.828e+03	1.348e+01	**0.000e+00**
F02	1.980e+02	3.436e+03	9.925e+03	1.297e+04	1.719e+04	1.292e+02	**1.160e-08**
F03	1.137e+06	9.482e+06	5.012e+07	1.251e+08	9.043e+07	5.509e+06	**2.971e+04**
F04	6.140e+01	4.165e+03	1.169e+04	1.365e+04	7.709e+04	9.347e+02	**3.592e-05**
F05	8.036e+02	7.204e+03	8.684e+03	1.901e+04	1.752e+04	6.230e+02	**4.590e-03**
F06	1.388e+04	6.219e+07	9.836e+08	3.383e+09	3.263e+08	2.764e+04	**8.802e+01**
F07	5.061e+00	1.890e+03	1.259e+03	3.131e+03	2.253e+03	1.920e+03	**5.673e-01**
F08	2.035e+01	2.041e+01	2.053e+01	2.093e+01	2.022e+01	2.028e+01	**2.019e+01**
F09	**3.529e+00**	6.505e+01	7.433e+01	8.867e+01	1.201e+02	3.292e+01	3.881e+00
F10	9.487e+00	9.127e+01	9.884e+01	1.162e+02	2.073e+02	2.739e+01	**4.584e+00**
F11	7.778e+00	9.458e+00	1.028e+01	1.345e+01	7.542e+00	8.258e+00	**3.801e+00**
F12	1.230e+03	2.248e+04	5.419e+04	1.107e+05	5.515e+04	2.686e+03	**4.385e+02**
F13	**5.364e-01**	6.426e+00	1.014e+01	9.040e+00	6.991e+00	2.393e+00	5.710e-01
F14	3.754e+00	3.927e+00	4.111e+00	4.501e+00	4.853e+00	3.606e+00	**2.460e+00**
F15	**2.538e+02**	5.274e+02	6.972e+02	9.631e+02	7.499e+02	4.281e+02	2.721e+02
F16	1.216e+02	2.799e+02	3.968e+02	6.118e+02	5.502e+02	1.679e+02	**1.108e+02**
F17	5.554e+01	1.346e+02	8.380e+01	1.128e+02	2.099e+02	9.551e+01	**3.215e+01**
F18	**7.774e+02**	1.091e+03	1.129e+03	1.383e+03	1.245e+03	1.037e+03	8.838e+02
F19	**8.096e+02**	1.084e+03	1.161e+03	1.383e+03	1.194e+03	8.563e+02	8.373e+02
F20	**8.158e+02**	1.082e+03	1.161e+03	1.383e+03	1.267e+03	9.720e+02	8.648e+02
F21	7.045e+02	1.252e+03	1.340e+03	1.424e+03	1.384e+03	8.994e+02	**6.702e+02**
F22	7.250e+02	9.516e+02	1.076e+03	1.350e+03	1.248e+03	8.940e+02	**6.624e+02**
F23	9.590e+02	1.256e+03	1.317e+03	1.474e+03	1.391e+03	1.071e+03	**9.556e+02**
F24	**2.365e+02**	1.134e+03	1.164e+03	1.446e+03	1.459e+03	8.663e+02	2.800e+02
F25	**5.109e+02**	1.346e+03	1.101e+03	2.118e+03	1.214e+03	5.243e+02	5.188e+02
Best	8	0	0	0	0	0	**17**

Tabla 7.1: Resultados medios del benchmark CEC2005 para dimensión 10.

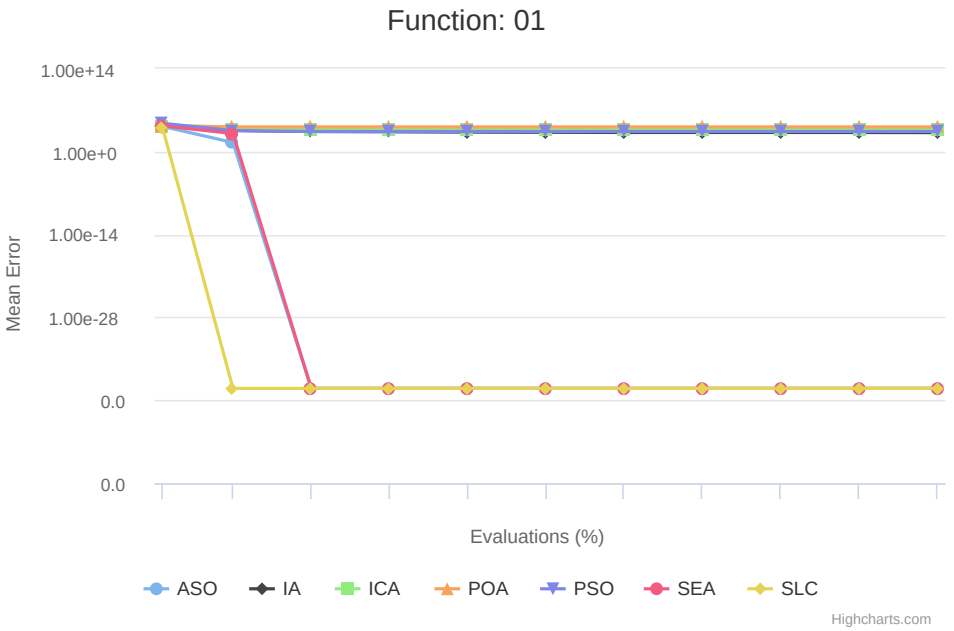

Figura 7.1: Gráfica de convergencia de la función 1 para dimensión 10.

	ASO	**IA**	**ICA**	**POA**	**PSO**	**SEA**	**SLC**
F01	2	4	6	7	5	3	**1**
F02	3	4	5	6	7	2	**1**
F03	2	4	5	7	6	3	**1**
F04	2	4	5	6	7	3	**1**
F05	3	4	5	7	6	2	**1**
F06	2	4	6	7	5	3	**1**
F07	2	4	3	7	6	5	**1**
F08	4	5	6	7	2	3	**1**
F09	**1**	4	5	6	7	3	2
F10	2	4	5	6	7	3	**1**
F11	3	5	6	7	2	4	**1**
F12	2	4	5	7	6	3	**1**
F13	**1**	4	7	6	5	3	2
F14	3	4	5	6	7	2	**1**
F15	**1**	4	5	7	6	3	2
F16	2	4	5	7	6	3	**1**
F17	2	6	3	5	7	4	**1**
F18	**1**	4	5	7	6	3	2
F19	**1**	4	5	7	6	3	2
F20	**1**	4	5	7	6	3	2
F21	2	4	5	7	6	3	**1**
F22	2	4	5	7	6	3	**1**
F23	2	4	5	7	6	3	**1**
F24	**1**	4	5	6	7	3	2
F25	**1**	6	4	7	5	3	2
Mean	1.920	4.240	5.040	6.640	5.800	3.040	**1.320**

Tabla 7.2: Ranking de resultados del benchmark CEC2005 para dimensión 10.

	ASO	IA	ICA	POA	PSO	SEA	SLC
F01	1.288e+02	3.626e+04	6.173e+04	6.584e+04	8.855e+04	1.866e+03	**0.000e+00**
F02	2.489e+03	4.569e+04	7.734e+04	8.917e+04	1.313e+05	1.220e+04	**1.179e-02**
F03	1.602e+07	3.321e+08	1.194e+09	1.466e+09	1.633e+09	2.487e+07	**5.643e+05**
F04	1.092e+04	5.443e+04	1.054e+05	8.943e+04	5.277e+05	3.946e+04	**4.680e+00**
F05	4.663e+03	2.588e+04	3.480e+04	4.441e+04	4.695e+04	1.343e+04	**4.157e+03**
F06	1.317e+07	9.779e+09	3.827e+10	2.916e+10	6.250e+10	1.173e+08	**7.574e+02**
F07	1.646e+02	8.951e+03	9.200e+03	1.252e+04	1.069e+04	5.696e+03	**3.926e-01**
F08	2.097e+01	2.096e+01	2.112e+01	2.134e+01	**2.064e+01**	2.082e+01	2.090e+01
F09	**2.940e+01**	3.278e+02	4.076e+02	4.047e+02	4.455e+02	2.046e+02	9.910e+01
F10	2.528e+02	4.707e+02	6.905e+02	6.595e+02	9.259e+02	2.872e+02	**9.004e+01**
F11	3.956e+01	4.008e+01	4.330e+01	4.749e+01	3.427e+01	3.429e+01	**2.410e+01**
F12	2.710e+04	1.026e+06	1.512e+06	1.579e+06	4.473e+05	1.481e+05	**9.489e+03**
F13	3.567e+00	8.211e+01	1.858e+02	4.855e+01	1.075e+02	1.657e+01	**3.425e+00**
F14	1.352e+01	1.359e+01	1.381e+01	1.437e+01	1.488e+01	1.337e+01	**1.207e+01**
F15	**2.950e+02**	8.825e+02	1.049e+03	1.200e+03	9.587e+02	6.849e+02	5.160e+02
F16	**8.460e+01**	6.003e+02	9.486e+02	1.191e+03	8.238e+02	4.667e+02	2.964e+02
F17	5.693e+01	1.875e+02	8.462e+01	1.054e+02	2.195e+02	1.872e+02	**1.831e+01**
F18	9.354e+02	1.184e+03	1.284e+03	1.268e+03	1.402e+03	9.586e+02	**9.107e+02**
F19	**9.170e+02**	1.178e+03	1.283e+03	1.261e+03	1.367e+03	9.666e+02	9.192e+02
F20	9.170e+02	1.158e+03	1.283e+03	1.261e+03	1.365e+03	9.758e+02	**9.094e+02**
F21	**6.006e+02**	1.325e+03	1.412e+03	1.386e+03	1.419e+03	1.066e+03	7.375e+02
F22	9.227e+02	1.306e+03	1.558e+03	1.740e+03	1.599e+03	1.193e+03	**9.108e+02**
F23	9.780e+02	1.323e+03	1.383e+03	1.385e+03	1.422e+03	1.211e+03	**9.097e+02**
F24	**2.021e+02**	1.367e+03	1.468e+03	1.464e+03	1.602e+03	1.220e+03	9.086e+02
F25	**3.574e+02**	1.410e+03	1.746e+03	1.978e+03	1.563e+03	9.508e+02	3.995e+02
Best	7	0	0	0	1	0	**17**

Tabla 7.3: Resultados medios del benchmark CEC2005 para dimensión 30.

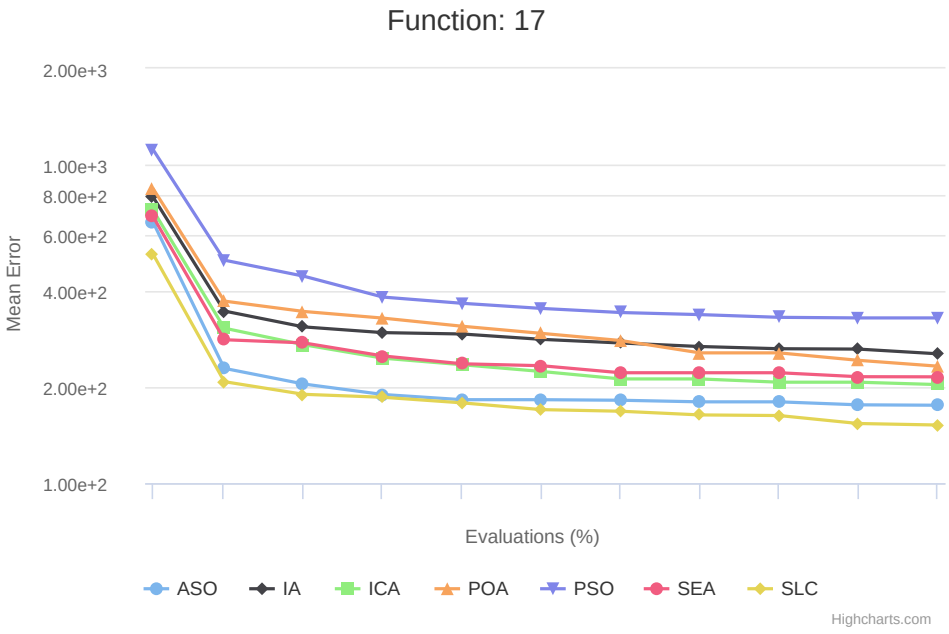

Figura 7.2: Gráfica de convergencia de la función 17 para dimensión 10.

	ASO	**IA**	**ICA**	**POA**	**PSO**	**SEA**	**SLC**
F01	2	4	5	6	7	3	**1**
F02	2	4	5	6	7	3	**1**
F03	2	4	5	6	7	3	**1**
F04	2	4	6	5	7	3	**1**
F05	2	4	5	6	7	3	**1**
F06	2	4	6	5	7	3	**1**
F07	2	4	5	7	6	3	**1**
F08	5	4	6	7	**1**	2	3
F09	**1**	4	6	5	7	3	2
F10	2	4	6	5	7	3	**1**
F11	4	5	6	7	2	3	**1**
F12	2	5	6	7	4	3	**1**
F13	2	5	7	4	6	3	**1**
F14	3	4	5	6	7	2	**1**
F15	**1**	4	6	7	5	3	2
F16	**1**	4	6	7	5	3	2
F17	2	6	3	4	7	5	**1**
F18	2	4	6	5	7	3	**1**
F19	**1**	4	6	5	7	3	2
F20	2	4	6	5	7	3	**1**
F21	**1**	4	6	5	7	3	2
F22	2	4	5	7	6	3	**1**
F23	2	4	5	6	7	3	**1**
F24	**1**	4	6	5	7	3	2
F25	**1**	4	6	7	5	3	2
Mean	1.960	4.200	5.600	5.800	6.080	3.000	**1.360**

Tabla 7.4: Ranking de resultados del benchmark CEC2005 para dimensión 30.

	D-10	**D-30**
ASO	8	7
IA	0	0
ICA	0	0
POA	0	0
PSO	0	1
SEA	0	0
SLC	**17**	**17**

Tabla 7.5: Mejores soluciones de cada algoritmo para cada dimensión.

	D-10	D-30
ASO	1.92	1.96
IA	4.24	4.20
ICA	5.04	5.60
POA	6.64	5.80
PSO	5.80	6.08
SEA	3.04	3.00
SLC	**1.32**	**1.36**

Tabla 7.6: Ranking global de cada algoritmo para cada dimensión.

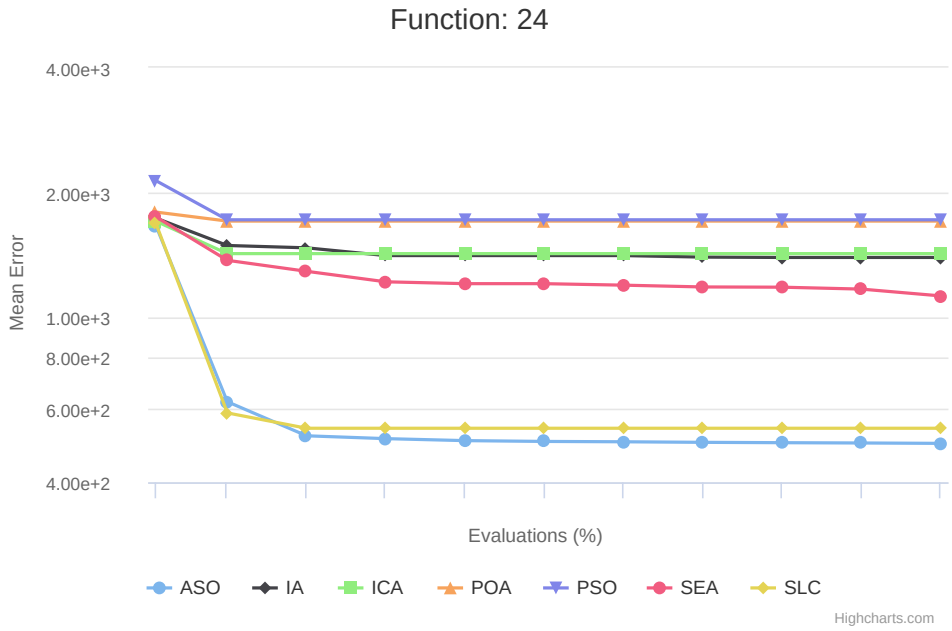

Figura 7.3: Gráfica de convergencia de la función 24 para dimensión 10.

mayores.

Con respecto al POA, fijándose en la tabla 7.2 se puede observar que únicamente en 8 de las 25 funciones no acaba en última posición. Por tanto, hay un total de 17 funciones en las que no consigue vencer ni al algoritmo de referencia en el que se basa, un resultado que resulta muy poco favorecedor. Se puede afirmar que POA es el peor algoritmo para cada grupo de funciones en dimensión 10.

Yendo al extremo contrario, a los que mejor se comportan en estas funciones unimodales, se encuentran ASO y SLC. Esto se puede observar tanto en la tabla 7.1 en la que se aprecia que sus errores son varios órdenes de magnitud menores que los del resto, y en las gráficas de convergencia como la de la figura 7.1, donde aparte de verse la diferencia en cuanto a ajuste de la solución destaca la rápida convergencia de estos algoritmos.

SLC es el mejor en unimodales, llegando o acariciando el óptimo en dos de ellas, y siendo en todo momento sus errores varios órdenes de magnitud inferiores a los del resto, a excepción de ASO en algunos casos. SLC también destaca con algunas de las mejores soluciones para las multimodales básicas, y aunque para las híbridas también funciona bien y se mantiene en un más que correcto segundo lugar, ASO es el algoritmo que mejores resultados da en ese campo.

Se puede decir que ambos algoritmos van casi de la mano en multimodales, ya que aunque el ASO explota sus ventajas cuanta más compleja sea la función, también es cierto que SLC obtiene muy buenos resultados para funciones con ruido, como son en este caso F4 y F17. Es curioso el caso de este algoritmo ya que maneja tan bien el ruido que para la misma función encuentra una mejor solución con ruido (F17) que sin él (F16).

Si se estudia la complejidad y el relieve del espacio de búsqueda de las funciones, se puede afirmar que SLC funciona especialmente bien en aquellas funciones con relieve suave (F1, F2) mientras que ASO lo hace para las que tienen un relieve más tortuoso y con más óptimos locales (F9, F18, F19). También SLC funciona bien con las que están rotadas, como F7, F8, F10 o F11. En este tipo de funciones los algoritmos basados en PSO sufren, ya que PSO es menos robusto frente a rotaciones que un Differential Evolution, por ejemplo.

Con respecto al resto de algoritmos que se encuentran en una posición intermedia, al menos se puede valorar que mejoran al algoritmo de referencia PSO en casi todas las funciones. En el caso del ICA, las diferencias no son muy significativas pero sin embargo sí que obtiene un error menor que PSO de manera generalizada, exceptuando F8 y F11 donde PSO destaca, siendo sólo superado por SLC. Si se valora en las funciones híbridas, sí que se puede encontrar una tendencia clara de mejora, y es que es sistemáticamente mejor

para aquellas funciones entre F15 y F25. Este dato se puede apreciar de manera menos confusa en la tabla 7.2, comprobando que la columna de ICA es siempre menos que la de PSO a partir de dichas funciones.

SEA es el tercer algoritmo que mejores resultados encuentra de manera general, de acuerdo a la tabla de ranking 7.2. Sin embargo, como se puede observar en la gráfica de convergencia de la función 24 (figura 7.3) la distancia entre las soluciones de los algoritmos ASO y SLC y la del SEA es muy grande. A pesar de ser el tercer mejor algoritmo y destacar sobre los otros 4, está a bastante de acercarse a SLC y ASO. Esto puede observarse para casi todas las funciones si se compara en la tabla 7.1 la columna de SEA con las de estos algoritmos. Con respecto a IA ocurre una situación parecida, resultando mejor que ICA, POA y PSO pero encontrándose lejos de SEA, que a su vez lo está de los algoritmos líderes del estudio.

Si se estudian los distintos grupos de algoritmos en función a su relación con PSO, se pueden sacar conclusiones interesantes. ICA y POA son los que más se asemejan a este algoritmo, en los que su mayor innovación consiste realmente en manejar varias estructuras similares a un PSO dentro del propio problema, y hacer que interactúen entre ellas. Sin embargo su interacción es bastante pobre, y la exploración se puede ver que no es nada del otro mundo con respecto al algoritmo original, como puede observarse en la gráfica de la figura 7.1. IA es similar a POA pero con una interacción más compleja entre sus soluciones, lo que se puede ver reflejado en las tablas que supone un margen de mejora amplio entre ambas técnicas. SEA y ASO implementan un componente histórico que ayuda a mejorar las soluciones, ya que además viene acompañado en ambos casos de una toma de decisiones basada en el estado temporal del problema. Dicha bifurcación en el camino del individuo parece estar mejor planteada en ASO, que consigue los mejores resultados de los algoritmos que parten de una idea basada en PSO, y es que SLC, el algoritmo que indiscutiblemente encuentra mejores resultados generales, es un algoritmo similar a Differential Evolution que utiliza mutaciones. Sus fundamentos básicos, algo distintos a los del resto de socioinspirados que se han estudiado, permiten que destaque sobre el resto y llegue a alcanzar esas soluciones tan positivas en la mayoría de los casos.

En dimensión 30 los resultados siguen una ruta similar, como se puede valorar comparando en la tabla 7.6 las columnas correspondientes a dimensión 10 y 30 para cada algoritmo. Se observa que los valores medios de ranking para cada algoritmo son muy parecidos, solamente cambiando de situación POA, que ahora sí mejora a PSO. En particular, se nota la mejora en todas las unimodales y en bastantes funciones complejas (ver tabla 7.3).

El resto de algoritmos sigue una evolución similar. IA, ICA y SEA mantienen un ranking similar, sin destacar en ninguna faceta especial pero siendo al menos constantes en sus ejecuciones si aumenta la dimensionalidad.

Lo interesante de esta dimensión se halla en la particular competición que mantienen SLC y ASO. Si en dimensión 10 se citaba la mejora de ASO en funciones multimodales a medida que avanzaba la complejidad, en esta nueva tabla con dimensionalidad 30 puede apreciarse una tendencia hacia lo mismo, que se procede a explicar.

Si se observan detenidamente los resultados de las funciones híbridas en la tabla 7.3 y se comparan los de ASO y los de SLC, se puede encontraar una pauta curiosa que refuerza la idea presentada en el párrafo anterior. En aquellas funciones en las que SLC es mejor, la distancia entre dicha solución y la que aporta ASO es muy pequeña; sin embargo, si es ASO el algoritmo que mejor resultado aporta, la solución del SLC se encuentra bastante más alejada. Es decir, la diferencia que justificaba que ASO cumplía mejor con las funciones más complejas se acentúa en mayor dimensión, y esto revela que ASO gestiona mejor la diversidad en su espacio de búsqueda.

Como conclusión final a este análisis, puede dar la sensación de que **SLC**, que es el que más destaca, es un algoritmo que podría formar parte de alguna competición en un CEC. Sin embargo, los resultados aún quedan muy lejos del óptimo para la mayoría de las funciones. Por hacer un símil, se compararán estos resultados del SLC con los aportados por un algoritmo memético que fue presentado en 2008 [35] y que comparaba con este mismo benchmark.

Como se observa en la tabla 7.7, donde se comparan para dimensión 30 todas las funciones multimodales del benchmark de 2005, un total de 20, solamente hay 2 en las que SLC resulte ser mejor. Cabe recordar que SLC es una técnica que vio la luz en 2014 y estos resultados comparativos fueron presentados en 2008, y ya están más que superados por nuevas propuestas. Así que, desde un punto de vista crítico, se debe afirmar que ni siquiera el mejor de los socioinspirados supone un digno rival para algoritmos de hace 10 años.

	MA-LSCh-CMA	SLC
F06	**1.191e+01**	7.574e+02
F07	**8.871e-04**	3.926e-01
F08	**2.027e+01**	2.090e+01
F09	**7.800e-09**	9.910e+01
F10	**1.839e+01**	9.004e+01
F11	**4.351e+00**	2.410e+01
F12	**7.690e+02**	9.489e+03
F13	**2.345e+00**	3.425e+00
F14	1.268e+01	**1.207e+01**
F15	**3.080e+02**	5.160e+02
F16	**1.363e+02**	2.964e+02
F17	1.346e+02	**1.831e+01**
F18	**8.157e+02**	9.107e+02
F19	**8.164e+02**	9.192e+02
F20	**8.158e+02**	9.094e+02
F21	**5.120e+02**	7.375e+02
F22	**5.258e+02**	9.108e+02
F23	**5.342e+02**	9.097e+02
F24	**2.000e+02**	9.086e+02
F25	**2.108e+02**	3.995e+02
Best	**18**	2

Tabla 7.7: Comparativa de funciones multimodales entre SLC y MA-LSCh-CMA.

Capítulo 8

Conclusiones y posibles extensiones

A lo largo de todo este estudio se han estudiado una serie de propuestas de algoritmos socioinspirados, a fin de arrojar algo de luz sobre un campo de la algorítmica del que aún no se tienen muchos resultados. Estos algoritmos tienen un interés muy obvio en la relación de algoritmos evolutivos con patrones cotidianos que son sencillos de entender y de aplicar. Pero lo que aún no se ha comprobado es si realmente este tipo de algoritmos está preparado para afrontar un benchmark de una competición y vencer a otras propuestas más conocidas.

El estudio realizado ha abarcado todas las fases de una experimentación completa, desde la revisión de la literatura hasta la experimentación y el análisis resultante, pasando por un previo análisis teórico que permitiese entender y valorar cada uno correctamente.

En el apartado del análisis experimental se ha llegado a la conclusión de que los algoritmos basados en una agrupación de PSOs, como ICA y POA, son menos robustos y tienden a dar resultados peores que los que van un paso más allá y modifican aún más el comportamiento estándar de un PSO. En este grupo puede entrar el algoritmo IA, que explora más posiciones en cada iteración y por tanto tiene más margen de maniobra para llegar a una mejor solución. En el último escalón de los socioinspirados basados en una estructura PSO se encontrarían los algoritmos SEA y ASO, dos técnicas que utilizan la memoria histórica en el problema para tomar decisiones de desplazamiento en base a ella. Gracias a esto, la mejor posición no tiene por qué estar necesariamente siempre presente en la población, lo que da lugar a que haya un movimiento más variado y se produzca una mayor exploración en el espacio de búsqueda que evite estancarse pronto en óptimos locales. Finalmente, el algoritmo socioinspirado que mejor resultado ha dado no es

una variación de un PSO sino que recuerda más a un algoritmo genético, o a un Differential Evolution, y se trata del SLC.

No obstante, el hecho de que el SLC haya arrasado dentro del estudio no implica que sea un algoritmo que refleje buenos resultados. A pesar de que todos los algoritmos, en mayor o menor medida, han terminado venciendo al PSO, el algoritmo que se ha utilizado como referencia, este no supone un reto real ya que no se trata de ninguna versión pulida. Al enfrentar los resultados del SLC frente a un algoritmo pulido de hace 10 años, los resultados han quedado bien expuestos: ni siquiera el mejor de los algoritmos socioinspirados es suficientemente bueno como para ser digno de participar en una competición. Al menos, es así por el momento, hasta que una nueva propuesta lo desmienta.

Y es que realmente la motivación que rodea a estos algoritmos no es tanto computacional sino didáctica. La complejidad de un algoritmo evolutivo queda reducida sustancialmente cuando se explica desde un contexto socioinspirado, y como se ha notado a lo largo del estudio, estos algoritmos ni siquiera son difíciles de entender aun no teniendo base conceptual sobre el tema. Valorando esto, y desde el punto de vista de la enseñanza, desde este estudio se anima a que sean utilizados para ayudar a comprender los algoritmos más complejos que puedan ser un rompecabezas de primeras.

Como posibles extensiones a este trabajo, en primer lugar cabría la posibilidad de comparar los algoritmos socioinspirados con nuevas variantes de PSO que hay surgido con el paso de los años, para hacer una estimación de en qué punto se encuentran. Más adelante, dependiendo de cómo resulten ser estas comparativas, se puede hacer un nuevo estudio para determinar qué características de los algoritmos que mejor resultado reflejen se pueden aplicar sobre el resto de algoritmos, y valorar su eficacia. Aunque, por supuesto, será mejor si todo tiene una base socioinspirada a la que aferrarse.

Bibliografía

[1] Elsevier Scopus Database. `https://www.scopus.com`.

[2] M. Kumar, A. Kulkarni, and S. Satapathy. Socio evolution & learning optimization algorithm: A socio-inspired optimization methodology. *Future Generation Computer Systems*, 81:252–272, April 2018.

[3] E. Atashpaz-Gargari and C. Lucas. Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition. In *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, volume 7, pages 4661–4667, October 2007.

[4] N. Moosavian and B. Kasaee Roodsari. Soccer League Competition Algorithm, a New Method for Solving Systems of Nonlinear Equations. *International Journal of Intelligence Science*, 4:7–16, January 2014.

[5] N. Moosavian and B. Kasaee Roodsari. Soccer League Competition Algorithm: A Novel Meta-heuristic Algorithm For Optimal Design of Water Distribution Networks. *Swarm and Evolutionary Computation*, 17, August 2014.

[6] A. Borji and M. Hamidi. A new approach to global optimization motivated by parliamentary political competitions. *International journal of innovative computing, information & control: IJICIC*, 5, January 2008.

[7] Y. Xu, Z. Cui, and J. Zeng. Social Emotional Optimization Algorithm for Nonlinear Constrained Optimization Problems, January 2010.

[8] A. Ahmadi Javid. Anarchic Society Optimization: A human-inspired method. In *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, pages 2586–2592, June 2011.

[9] A. Bozorgi, O. Bozorg-Haddad, and X. Chu. Anarchic Society Optimization (ASO) Algorithm, July 2018.

[10] T. Ting Huan, A. Kulkarni, J. Kanesan, J. Chuah, and A. Abraham. Ideology algorithm: a socio-inspired optimization methodology. *Neural Computing and Applications*, June 2016.

[11] ResearchGate Scientific Knowledge Database. `https://www.researchgate.net`.

[12] Visual Studio Code - Open Source, repositorio de GitHub. `https://github.com/Microsoft/vscode`.

[13] TeXstudio, página oficial. `https://www.texstudio.org`.

[14] CITIC-UGR Sala de Servidores y Clústeres de Computadores. `http://citic.ugr.es/pages/informacion_general/equipamiento/cluster`.

[15] Evolutionary Computation Bestiary, repositorio de GitHub. `https://github.com/fcampelo/EC-Bestiary`.

[16] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[17] M. Dorigo and G. Di Caro. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5:137–172, 1999.

[18] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, and A. Stauffer. Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware. In *1st International Conference on Evolvable Systems, ICES*, 1996.

[19] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, and A. Stauffer. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Xplore*, 1997.

[20] H. Shayeghi and J. Dadashpour. Anarchic Society Optimization Based PID Control of an Automatic Voltage Regulator (AVR) System. *Electrical and Electronic Engineering*, 2:199–207, August 2012.

[21] J. Kennedy and R. Eberhart. Particle Swarm Optimization. *IEEE Intl. Conf. on Neural Networks*, 4:1942–1948, 1995.

[22] B. Panigrahi, S. Das, P. Suganthan, and S. Dash. *Swarm, evolutionary, and memetic computing. First international conference on swarm, evolutionary, and memetic computing, SEMCCO 2010, Chennai, India, December 16–18, 2010. Proceedings*, volume 6466. January 2010.

[23] N. Moosavian. Soccer league competition algorithm for solving knapsack problems. *Swarm and Evolutionary Computation*, 20, October 2014.

[24] Soccer League Competition (SLC) Algorithm For Discrete Problems, File Exchange, MATLAB Central. `https:`

//www.mathworks.com/matlabcentral/fileexchange/56480-soccer-league-competition-slc-algorithm-for-discrete-problems.

[25] R. Kumar and Jyotishree. Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms. *International Journal of Machine Learning and Computing*, pages 365–370, January 2012.

[26] Documentación de Python 3.6. https://docs.python.org/3.6.

[27] Documentación de NumPy. http://www.numpy.org.

[28] Repositorio del proyecto en GitHub. https://github.com/JJSrra/Research-SocioinspiredAlgorithms.

[29] Imperialist Competitive Algorithm (ICA), File Exchange, MATLAB Central. https://www.mathworks.com/matlabcentral/fileexchange/22046-imperialist-competitive-algorithm-ica.

[30] Repositorio del Ideology Algorithm en GitHub. https://github.com/NestorRV/IdeologyAlgorithm.

[31] Special Session on Real-Parameter Optimization at CEC-05, Edinburgh. http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC-05/CEC05.htm.

[32] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. *Natural Computing*, 341-357, January 2005.

[33] TACO Website. https://tacolab.org/.

[34] Repositorio del Particle Swarm Optimization en GitHub. https://github.com/nathanrooy/Particle-Swarm-Optimization-with-Python.

[35] D. Molina, M. Lozano, C. García-Martínez, and F. Herrera. Memetic algorithm for intense local search methods using local search chains. *Hybrid Metaheuristics: 5th International Workshop*, 5296:58–71, October 2008.

[36] N. Moosavian and H. Moosavian. Testing Soccer League Competition Algorithm in Comparison with Ten Popular Meta-heuristic Algorithms for Sizing Optimization of Truss Structures. *IJE TRANSACTIONS A: Basics*, 30:926–936, July 2017.