UNIVERSIDAD
DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingeniería
Informática y de Telecomunicaciones

Doble Grado en Matemáticas e Ingeniería
Informática

TRABAJO DE FIN DE GRADO

Algoritmos Coevolutivos. Análisis teórico y práctico de su convergencia en arquitecturas distribuidas.

Presentado por:
Alejandro Alonso Membrilla

Tutor:
Julio Ortega Lopera
Departamento de Arquitectura y Tecnología de Computadores

Lidia Fernández Rodríguez
Departamento de Matemática Aplicada

Curso académico 2020-2021

Algoritmos Coevolutivos. Análisis teórico y práctico de su convergencia en arquitecturas distribuidas.

Alejandro Alonso Membrilla

Alejandro Alonso Membrilla *Algoritmos Coevolutivos. Análisis teórico y práctico de su convergencia en arquitecturas distribuidas.*.

Trabajo de fin de Grado. Curso académico 2020-2021.

Responsable de tutorización

Julio Ortega Lopera
Departamento de Arquitectura y Tecnología de Computadores

Lidia Fernández Rodríguez
Departamento de Matemática Aplicada

Doble Grado en Matemáticas e Ingeniería Informática

Facultad de Ciencias y Escuela Técnica Superior de Ingeniería Informática y de Telecomunicaciones

Universidad de Granada

Declaración de originalidad

D./Dña. Alejandro Alonso Membrilla

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2020-2021, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 6 de septiembre de 2021

Fdo: Alejandro Alonso Membrilla

Índice general

Breve Resumen

Los algoritmos evolutivos son un método de optimización de propósito general ampliamente estudiados y utilizados en problemas de ciencias e ingeniería. Lamentablemente, el hecho de que puedan adaptarse a muchos problemas de optimización distintos hace que su rendimiento varíe enormemente y que sea muy difícil predecir su eficacia. En este trabajo se expondrá un marco teórico conocido, basado en procesos estocásticos, para entender los conceptos básicos asociados a los algoritmos evolutivos. Nos centraremos en las dificultades que surgen desde el punto de vista matemático al aplicar metaheurísticas para resolver problemas de optimización a gran escala y en cómo afrontarlas mediante la descomposición de dichos problemas en subproblemas menores, concretamente desde el marco de la coevolución cooperativa.

Los algoritmos coevolutivos cooperativos (*CCEA*) son la familia de metaheurísticas que nace de la combinación de los *EA* clásicos con la descomposición antes mencionada. Se detallarán los elementos principales que caracterizan un *CCEA*, sobre los modelos matemáticos más importantes que existen a día de hoy para explicar el comportamiento de estos algoritmos y las dificultades que surgen al intentar coordinar correctamente la resolución de los distintos subproblemas cuando estos no son completamente independientes entre sí. También se describirán las arquitecturas distribuidas y cómo los *CCEA* se adaptan, por diseño, a este tipo de esquemas de computación.

El trabajo concluirá con un estudio experimental en el que se compararán los resultados de un *EA* clásico con los de un *CCEA* en un problema diseñado para ser fácilmente descomponible, para lo que se emplearán técnicas de agrupamiento mediante aprendizaje no supervisado. Se expondrán conclusiones nacidas de combinar los resultados analíticos y empíricos y se sugerirán nuevas líneas de investigación sobre algoritmos coevolutivos cooperativos en el futuro.

Palabras clave: Metaheurísticas, *CCEA*, Procesos Estocásticos, Teoremas *NFL*, Maldición de la Dimensionalidad, Arquitecturas Distribuidas.

Summary

Evolutionary algorithms are a general-purpose optimization method widely studied and used in science and engineering problems. Unfortunately, the fact that they can accommodate many different optimization tasks renders a wide variation in performance and a hard-to-predict effectiveness. In this work we will present a known theoretical framework based on stochastic processes in order to understand the basic concepts associated with **evolutionary algorithms**. We will focus on the difficulties that arise from the mathematical point of view when applying metaheuristics to solve large-scale optimization problems and how to deal with them by decomposing those problems into minor subproblems, specifically within the framework of **cooperative coevolution**.

In first chapter the concept of optimization problem wil be reviewed from a mathematical point of view, leading to a general explanation on **black-box optimization** and **metaheuristics**, both key ideas for the understanding of chapters 3 and 4. A black-box optimization problem is one that does not provide any information on the objective function beyond its evaluation. Techniques to solve this kind of problems usually rely on local search (evaluating solutions on the neighborhood of another given solution) or population based search (keeping track of and comparing multiple solutions each time), but dozens of different data structures and operators have been created to complement the search, such as curriculums, taboo lists, population reset... Metaheuristics are a problem-agnostic problem solver description that support the design of algorithms to look for solutions (usually in an iterative and stochastic way) to specific problems by providing a list of steps easily adaptable to different problem representations. Evolutionary algorithms are a type of metaheuristic.

The second chapter will focus on a deeper understanding of the behavior of evolutionary algorithms by explaining the mathematical framework presented by Xiaofeng Qi and F. Palmieri in 1994 based on stochastic processes and the hypothesis of infinite populations. The reasonableness of these assumptions will be justified by the Glivenko-Cantelli theorem and some other proven results. A solid understanding should be grasped on the importance of the population size, the process of selection, and the "exploration" induced by the mutation and crossover operators. Some basic concepts of drift analysis will be presented, as they are one of the most used and promising tools in the analytical study of metaheuristics today. For example, theorem 2.10, the Level-Based Theorem, is a relatively new result that allows to apply automatically drift analysis tools to evolutionary algorithms for a certain generic type of problems and it has been already used in the analysis of multiple EAs, including self-adaptive and noisy algorithms.

The third chapter explains different obstacles that can arise in black-box optimization, which includes evolutionary algorithms. Special emphasis will be placed on two mathematical concepts of vital importance for the understanding of the rest of this text: the **No Free Lunch theorems** and the **curse of dimensionality**. The conclusion that arises from the con-

cepts introduced in this chapter is: the choice of a correct algorithm must be based on *a priori* knowledge of the problem and its relationship with the method used to find its solution. An example of a recent mathematical study that "rows" in this direction by applying theoretical tools to relate concrete algorithms and problems based on the "ease" of the formers to solve the latters, *On the Easiest and Hardest Fitness Functions*[HCY15], will also be described.

The curse of dimensionality causes, among other things, the exponential growth of the computational cost to solve an optimization problem by increasing the number of variables in the solution space. Cooperative coevolutionary algorithms emerged in 1994 from Mitchell A. Potter and Kenneth A. De Jong to address this exponential growth by optimizing the objective function in each variable separately. Chapter 4 will approach the main elements that characterize a CCEA, the most important studies that exist today about these algorithms from a mathematical perspective and the difficulties involved in trying to coordinate properly the resolution of the different subproblems when they are not completely independent of each other. The different schemes explored in the literature on the decomposition and collaboration between subpopulations and the insights that the presented mathematical models provide in the understanding of CCEA will be discussed.

The "parallel" optimization performed in cooperative coevolutionary algorithms is carried out mainly by defining these two elements in a particular CCEA design: the way the domain is divided, or **problem decomposition**, and the methodology of coordination and communication between the subpopulations, or **collaboration scheme**. The problem decomposition allows to find and exploit the lack of dependency between different variables to optimize them distinctly. It could be considered as the most fundamental element of CCEAs, as there can be no coevolution without problem decomposition. There are plenty of methods to search for ways of splitting up the domain, as we cover in section 4.2.1, some based on direct interaction measurement, some based on more technical probabilistic and statistical modelling. When a problem is not separable into non-interdependent blocks of variables it starts to be required a method to communicate the different subpopulations to facilitate global convergence, as it will be justified from previous research, preferably in the most efficient possible way. The overwhelming amount of possibilities in the implementation of these two schemes not only opens the door to flexible distributed *hardware-software* co-designed implementations, but it also strengthens the hypothesis that an *a priori* knowledge of the problem is required for efficient CCEA design.

Chapter 5 is devoted to distributed architectures and to an empirical study of the influence of domain decomposition on the optimization convergence on easily separable problems.

Distributed computing consists of sharing out a parallelizable task between different nodes interconnected by a network. It is the single most scalable way of parallelization, and a fundamental element of modern high performance computing (HPC). Since CCEA adapts by design to the concept of distributed computing, basic concepts regarding these types of architectures will be presented and alternatives to cooperative coevolution whose implementation distributed schemes has already been researched in the literature will be compared.

The empirical study carried out consists of the optimization of a Boltzmann machine, an object of justifiable interest in the study of the optimization of functions following a predetermined structure, since in its definition we can generate problems with whatever de-

pendency structure that we want. Although the practical cases presented in our experiment are relatively simple (problems easily decomposable into blocks of variables with low interdependence), the exposed framework for their resolution is generalizable to more complex dependency structures and alternative techniques are suggested to adapt our methodology to other less manageable cases by applying more flexible data analysis techniques on the estimated dependencies.

The Boltzmann machines to be optimized comprise 800 and 8000 units, what leads to problems with the same number of dimensions, each of them with 8 blocks of highly dependent variables. Random noise is inserted into the weights of the network in order to make the division of the variables non-straightforward. The applied method for the decomposition is based on previous *differential grouping* methods for estimating dependencies together with *K-medoids*, a classic clustering algorithm based on distances. The metaheuristics compared are an EA with tournament selection and uniform crossover and a basic CCEA designed by adapting that same EA into evolving multiple disconnected subpopulations, each of them optimizing only one of the previously generated variable groups.

The obtained results show significant benefits in the long-term (after enough number of generations) when problem decomposition is applied to the confronted task. Besides, the difference in the evolution of the best solutions for both algorithms seems to follow the difference in order of growth predicted by the theory.

Overall, this project seeks to unify as much as possible the current theoretical knowledge on cooperative coevolutionary optimization that can give some useful insights when applying evolutionary techniques into real problems, while providing a relatively extensive list of references on the state-of-the-art of evolutionary and coevolutionary optimization. Some widely used analytical tools in mathematical research on evolutionary algorithms have also been briefly presented, since we believe that these can be helpful too in deepening the understanding of CCEAs and their usecases.

The final chapter ends with a section of conclusions and suggestions on future work. On the basis of the mathematical and empirical results presented in this work we conclude that having a reference on the type of dependency structures in each family of problems will be extremely helpful when deciding about the use of CCEAs in particular usecases. Therefore, we suggest that future research might be devoted to the search for patterns in the variable dependencies. Also, it is justified the interest in the theoretical study of evolutionary algorithms and we believe that combining the latest methodologies for the analysis of EAs with the cooperative coevolutionary framework is a promising yet unexplored path for future research.

Keywords: Metaheuristics, CCEA, Stochastic Processes, NFL Theorems, Curse of Dimensionality, Distributed Architectures.

Introducción

Los algoritmos evolutivos han demostrado ser uno de los métodos de optimización más útiles en multitud de problemas complejos. El modelo coevolutivo cooperativo constituye una aproximación que puede ser eficaz para afrontar este tipo de problemas cuando el espacio de soluciones consta de un gran número de dimensiones, distribuyendo dicho espacio en una arquitectura paralela donde cada nodo busca la mejor solución avanzando dentro de su propio subespacio.

El inmenso número de posibles elecciones en el diseño de este tipo de algoritmos y el hecho de que no sea posible encontrar una única metaheurística que sea capaz de resolver eficientemente un problema de optimización cualquiera nos hacen plantearnos si es posible recurrir a un análisis matemático para obtener información sobre qué algoritmos son más aptos para cada problema a resolver, y cómo ajustar el diseño de los algoritmos coevolutivos cooperativos (*CCEA*) a partir de conocimiento *a priori*.

En este sentido, las influencias principales de este proyecto son:

- Trabajos que estudian el comportamiento de los algoritmos evolutivos desde un punto de vista matemático, entre los que se destacan aquellos realizados por Xiaofeng Qi y F. Palmieri in 1994 [XP94a][XP94b].
- *Evolutionary Optimization: Pitfalls and Booby Traps* (2012) [WCT12], por Thomas Weise, Raymond Chiong y Ke Tang, que supone la recopilación y explicación más completa, a nuestro conocimiento, sobre las dificultades inherentes a los problemas de optimización de caja negra.
- Los teoremas *NFL* [WM97].
- El concepto de algoritmo coevolutivo cooperativo, definido por Mitchell A. Potter y Kenneth A. De Jong en 1994 [PDJ94a].
- Los avances aportados por R. P. Wiegand en 2003 [Wie03], R. Subbu y A. C. Sanderson en 2004 [SS04] y Liviu Panait en 2010 [Pan10] en la modelización matemática de los algoritmos coevolutivos cooperativos.
- La extensa discusión realizada en *A Survey on Cooperative Co-Evolutionary Algorithms* (2019) [MLZ+19], por X. Ma and X. Li and Q. Zhang and K. Tang and Z. Liang and W. Xie and Z. Zhu, sobre las principales tendencias y técnicas investigadas en coevolución cooperativa durante los últimos años.
- *Parallel Coarse Grain Computing of Boltzmann Machines* (1998) [ORDP98], por Julio Ortega, Ignacio Rojas, Antonio F. Diaz y Alberto Prieto, ha servido como ejemplo a la hora de experimentar con un *CCEA* para optimizar máquinas de Boltzmann. Supone la

principal inspiración para el estudio práctico realizado en el capítulo 5.

Los temas tratados en este proyecto se asientan sobre la optimización matemática y computacional, dentro de la cual las metaheurísticas y los algoritmos evolutivos suponen el eje central de nuestra exposición. Por otro lado, el enfoque de estos algoritmos desde el punto de vista de su adaptabilidad a ciertas arquitecturas *hardware* aportado en el capítulo 5 también conecta nuestro trabajo con el campo de la computación de altas prestaciones.

En lo que respecta a las técnicas matemáticas empleadas, los resultados y las demostraciones presentadas en el capítulo 2 y, en menor medida, en el 3 y el 4, requieren de conocimientos en teoría de la probabilidad, procesos estocásticos y teoría de la medida. Por otro lado, múltiples argumentos y razonamientos realizados a lo largo del texto se fundamentan en la complejidad asintótica de diversos algoritmos, de uso habitual en la informática y la algorítmica.

Parte de la metodología presentada en el capítulo 5 se basa en técnicas y conceptos del análisis de datos y el aprendizaje automático, especialmente dentro del área del aprendizaje no supervisado.

Objetivos

Los objetivos iniciales del alumno al afrontar este proyecto consistían en "exponer un modelo matemático para trabajar con los algoritmos coevolutivos implementados en computadores con paralelismo híbrido, tratar de encontrar condiciones relativas a la distribución de los subespacios de búsqueda y a la coordinación entre las distintas especies que favorezcan la convergencia de las soluciones hacia el óptimo global, y estudiar este tipo de modelos evolutivos aplicados a arquitecturas distribuidas, desde sus características hasta sus aplicaciones prácticas, pasando por aquellos aspectos que puedan afectar a su eficacia y rendimiento".

Estos objetivos se han cubierto parcialmente. Ciertamente, se han expuesto modelos matemáticos que han permitido profundizar en nuestro entendimiento sobre los algoritmos coevolutivos cooperativos, se han estudiado condiciones sobre la estructura de la función objetivo que facilitan o dificultan su convergencia y pueden afectar a su eficacia y rendimiento, además de cómo paliarlos y combatirlos. Se ha establecido una relación entre los *CCEAs* y las arquitecturas distribuidas y se han discutido ventajas e inconvenientes de combinar ambas herramientas, además de alternativas a este tipo de algoritmos previamente estudiadas adaptadas a este paradigma de paralelización.

Sin embargo, no puede decirse que se hayan cumplido todas las expectativas. El espacio dedicado a arquitecturas distribuidas ha sido menor del planteado originalmente, dado que el proyecto ha acabado enfocándose desde un punto de vista abstracto y cubriendo los elementos más fundamentales de los algoritmos evolutivos y coevolutivos. Los modelos matemáticos expuestos, aunque permiten alcanzar conclusiones útiles para comprender el comportamiento de los *CCEAs* de forma rudimentaria, son demasiado abstractos y simples como para permitir hacer predicciones sobre el uso de algoritmos particulares en arquitecturas específicas. Nuestro trabajo se ha limitado a recopilar y analizar resultados obtenidos por otros investigadores en el campo, sin lograr construir un modelo o encontrar resultados técnicos de interés originales. La causa puede encontrarse, en parte, en los pocos avances que hay hasta la fecha en cuanto a la modelización matemática de *CCEAs*. Los trabajos dedicados a estudiar este tipo de algoritmos desde un punto de vista matemático, hasta donde sabemos, se encuentran dispersos y tienden a centrarse en versiones altamente simplificadas como el $CC(1+1)EA$.

Por otro lado, sí consideramos que el presente proyecto ha acabado siendo una amplia e interesante introducción a los algoritmos coevolutivos cooperativos, así como una recopilación de las fuentes más importantes en este campo. También consideramos acertadas las sugerencias aportadas en el apartado de conclusiones, en tanto que creemos que podrían enriquecer la investigación sobre algoritmos coevolutivos cooperativos.

1 Optimización y algoritmos evolutivos

1.1. Optimización

En esta breve sección se explicará lo que es un problema de optimización, sus elementos (dominio, función objetivo y restricciones), mencionando algunos casos concretos.

También se hablará de la optimización de caja negra, qué inconvenientes presenta y en qué casos es necesaria.

En un sentido amplio, se habla de un **problema de optimización** como todo aquel en el que se busca aquella solución de entre un conjunto de soluciones posibles que sea la mejor en cierto sentido preestablecido.

Formalmente, es habitual ver un problema de optimización cualquiera representado de la siguiente forma:

Dado Ω un conjunto cualquiera, $d \in \mathbb{N}$ y $f\colon \Omega \to \mathbb{R}^d$ una función. Nuestro objetivo será encontrar los valores

$$\operatorname*{argmin}_{x\in\Omega} f(x) \qquad\qquad \min_{x\in\Omega} f(x) \tag{1.1}$$

Se llamará **dominio, espacio factible** o **espacio de soluciones** al conjunto Ω y **función objetivo** a f. Cabe destacar el hecho de que no se pierde generalidad al representar nuestro problema de optimización como uno de minimización, puesto que encontrar el mínimo de $-f$ equivale a hallar el máximo de f.

A menudo, un cierto dominio puede definirse imponiendo restricciones sobre otro más grande. En los problemas de optimización continuos, por ejemplo, la formulación matemática queda así:

$$\begin{aligned} \min_{x\in\mathbb{R}^m} \quad & f(x) \\ \text{sujeto a} \quad & g_i(x) \leq 0 \quad \forall i = 1, ..., I \\ & h_j(x) = 0 \quad \forall j = 1, ..., J \end{aligned} \tag{1.2}$$

donde g_i y h_j son $I + J$ funciones continuas que imponen restricciones sobre el dominio.

Cuando d sea mayor que 1, se dirá que el problema anterior es de **optimización multiobjetivo**. En estos casos, es común utilizar el concepto de **dominancia de Pareto**, es decir:

$f = (f_1, f_2, ..., f_d)$. Entonces:

$$x_1 \prec_f x_2 \iff f_i(x_1) \leq f_i(x_2) \quad \forall i \in \{1, ..., d\} \text{ y } \exists j \in \{1, ..., d\} \text{ tal que } f_j(x_1) < f_j(x_2) \quad (1.3)$$

y se lee de la forma "x_1 domina a x_2". El f subíndice de $\prec$ suele omitirse cuando no hay lugar a confusión.

Un elemento $x^* \in \Omega$ se dice óptimo en este sentido si no existe ningún otro $x \in \Omega$ que lo domine. La colección de todos los elementos no dominados se denomina **conjunto de Pareto**, y su imagen por f forma el **frente de Pareto**.

Si bien la optimización multiobjetivo es un campo muy amplio y estudiado a día de hoy [IYAN13][MLZ+19][HTZ+19][HYT21], este trabajo **se centrará principalmente en el caso mono-objetivo** ($d = 1$).

Existe un enorme número de técnicas de optimización diferentes. Métodos analíticos, como los derivados del Teorema de puntos críticos de Fermat para funciones reales diferenciables, de la ecuación de Euler-Lagrange [GdS77] o del lema de Lax-Milgram ([Bre10] corolario 5.8), ambos para funcionales bajo distintos requisitos. También existen métodos numéricos para optimización, como el descenso de gradiente o el método de Newton, que garantizan la convergencia a óptimos locales cuando se conoce la expresión del gradiente y (con el método de Newton) la Hessiana de f, además de los métodos de quasi-Newton como BFGS y L-BFGS, por ejemplo, que buscan el óptimo aproximando la Hessiana [NW06].

Los algoritmos que se tratarán en este trabajo pertenecen a la categoría de las **metaheurísticas.** Más que un algoritmo concreto, una metaheurística es una forma de diseñar algoritmos de optimización que puede adaptarse, con más o menos éxito, a casi cualquier problema particular.

Al carecer de información concreta del problema, las metaheurísticas se caracterizan por su naturaleza iterativa y estocástica. Esto también implica que operan con el sistema o función que buscan optimizar en forma de **caja negra**, esto es, la única información que obtienen de dicha función es el resultado de evaluarla en un punto específico del dominio. Esta puede ser la forma más manejable de aproximarse al algunos problemas de carácter combinatorio, como el problema del viajante de comercio (*TSP*), la selección de características, el problema de la mochila y otros muchos de tipo NP. También puede ser de interés si se quiere estudiar cómo afectan una serie de parámetros a un sistema complejo, discreto o continuo y, aunque no se disponga de modelos analíticos que lo representen, aún se puede simular el sistema y evaluar su comportamiento para distintos valores de dichas variables [SHC+17][KBLB20].

Dentro de las metaheurísticas existen muchas familias de algoritmos diferentes. El **enfriamiento simulado** parte de una solución inicial, generalmente aleatoria, buscando aquel **vecino** (solución válida cercana a la actual, incluyéndose a sí misma) que minimice la función objetivo. Este método incluye un parámetro adicional, la temperatura, que es inversamente proporcional a la probabilidad de elegir el mejor vecino como siguiente solución (otro vecino se escoge aleatoriamente en caso contrario), y que va reduciéndose con el tiempo. Esto permite que, al principio, se produzca una búsqueda local aleatoria, que ayuda a explorar el espacio de soluciones, que cambia progresivamente a una búsqueda *greedy* o de *descenso de colina*, convergente a un óptimo local. **GRASP** es otro método de búsqueda local basado en

la elaboración de una lista de posibles soluciones siguientes a partir de los entornos de las soluciones exploradas hasta el momento, y asignando una probabilidad de selección a cada elemento de la lista directamente proporcional a su valor objetivo. Otros son la búsqueda tabú, la búsqueda local iterativa... Cada uno aporta un elemento diferenciador con respecto a los demás, y tiene sus pros y contras.

Otra familia de metaheurísticas son los algoritmos de enjambre. Los más conocidos son los **enjambres de partículas** (*PSO*) y las **colonias de hormigas** (*ACO*). Los primeros parten de una población de soluciones, cada una de las cuales se desplaza en el espacio factible con una velocidad inicial en una dirección diferente. Cada iteración se evalúan todas las soluciones actuales, tras lo que el enjambre al completo se desplaza. Las mejores soluciones tienen cierta probabilidad de alterar la dirección de cambio de las peores soluciones [BVA07]. Los segundos se basan en un conjunto de búsquedas tabú simultáneas (búsquedas locales con una lista de soluciones recorridas, para evitarlas) que se denotarán como hormigas. Cada hormiga explora el espacio de soluciones y, en una estructura de datos compartida por todas las hormigas, cambia un parámetro asociado al camino seguido de forma proporcional a la bondad del valor objetivo de la solución alcanzada. Esta información se tiene en cuenta en iteraciones posteriores, de forma que la probabilidad de elegir un camino en particular depende del "rastro" dejado por el resto de hormigas [DMC96].

En este caso, los algoritmos que se estudiarán en más profundidad son los **evolutivos**, otros métodos de tipo poblacional, que se explicarán en el siguiente apartado.

1.2. Algoritmos evolutivos

En esta sección se dará una definición de algoritmo evolutivo (*EA*). Se hablará sobre las principales diferencias entre este tipo de métodos y otras metaheurísticas de búsqueda estocástica como el enfriamiento simulado. Se explicarán las partes fundamentales de un algoritmo evolutivo: espacio del problema, función objetivo, espacio de búsqueda o genotipo, y un mapeado entre el genotipo y los fenotipos, o elementos del espacio del problema.

Los **algoritmos evolutivos** son toda una clase de algoritmos de optimización basados en poblaciones regidos por el siguiente patrón:

Dada una población/conjunto de soluciones inicial, posiblemente generada de forma aleatoria, repetir hasta que se alcance cierta condición de parada:

1. **Evaluación:** se calcula la bondad de cada individuo de la población con respecto a la función objetivo de referencia.

2. **Selección:** se elige las soluciones que, o bien pasarán a la siguiente generación, o bien serán mezcladas para obtener las nuevas soluciones. Las formas más comunes de hacer esto son:

    a) Selección proporcional: se muestrean los individuos de la población actual con una probabilidad de selección proporcional a la bondad de la solución.

   b) Selección por k-torneo: $k \in \mathbb{N}$ individuos de la población actual se eligen de forma aleatoria. Seleccionamos el mejor de ellos.

3. **Cruce:** "se mezclan" las soluciones seleccionadas para explorar el espacio factible en búsqueda de otras soluciones mejores. Si se mezclan 2 individuos para dar lugar a un tercero, una estrategia habitual es la de descartar el peor de los 3. Otra opción es generar 2 descendientes y descartar los dos anteriores.

4. **Mutación:** se modifican los genes de la población resultante con una probabilidad determinada, generalmente muy pequeña. Esto permite introducir nuevas características en el conjunto de soluciones que pudieran haber quedado sin explorar en la población inicial.

En el próximo capítulo se expondrá un modelo matemático para los algoritmos evolutivos, basado en procesos estocásticos, que permitirá entender en mayor profundidad la importancia de cada uno de los operadores utilizados en el proceso anterior.

Nótese que es posible configurar y adaptar un algoritmo evolutivo de muchas formas distintas a partir de los parámetros establecidos en la descripción anterior. Se observa también el uso extensivo de términos provenientes de la Biología, campo que inspira la mayor parte de los conceptos que se implementan en los algoritmos evolutivos y en las metaheurísticas en general. Esta nomenclatura es la que se usa más habitualmente en la literatura sobre algoritmos evolutivos y, por tanto, la que se utilizará en esta memoria. Los más importantes, al margen de los explicados anteriormente, son:

- Los candidatos/individuos del espacio factible Ω, dominio de la función f a optimizar, se conocen en la literatura más clásica como *fenotipos*, aunque a día de hoy su uso no es tan habitual.

- Es posible modificar la estructura del espacio de soluciones, codificándolo mediante cierto **cambio de variable**. En el nuevo dominio es donde se aplicarían los operadores de cruce y de mutación antes mencionados, que se ven afectados por distancias y la geometría del espacio. Es habitual llamar a este dominio transformado un ***genotipo*** y notarlo como $\mathbb{G}$. Naturalmente, $\mathbb{G}$ puede ser igual a Ω.

- Las distintas variables de las soluciones representadas en el espacio $\mathbb{G}$ se denominan *genes*.

1.3. Principales algoritmos evolutivos clásicos

Existen numerosas versiones y variantes de los algoritmos evolutivos. La taxonomía compilada en [MPS+20] enumera 24 metaheurísticas distintas basadas en estrategias de evolución. Las dos más conocidas son los algoritmos genéticos y la evolución diferencial.

Los **algoritmos genéticos** se caracterizan por su uso del operador de cruce. Generalmente, en los algoritmos genéticos las soluciones candidatas se representan como cadenas de variables discretas (tradicionalmente, 0s y 1s [Hol75]). El cruce puede realizarse eligiendo dos

soluciones, que harán de padres, y generando una nueva solución tomando en torno a la mitad de los genes de cada progenitor. La proporción de variables seleccionadas de cada padre puede ajustarse, por ejemplo, para favorecer los genes de las mejores soluciones. También es habitual en estos algoritmos introducir una mutación aleatoria que puede cambiar el valor de un gen con una probabilidad relativamente baja.

La **evolución diferencial** [SP97] tiene más sentido en problemas continuos y se caracteriza por unificar el cruce y la mutación. Fijamos dos constantes, $CR \in [0,1]$, o *probabilidad de cruce*, y $F \in [0,2]$, o *peso diferencial*. Por cada individuo x de la población, una vez por generación o por muestreo aleatorio, generamos 3 posibles soluciones $a, b, c \in \Omega$, un índice $J \in \{1, ..., dim(\Omega)\}$ y un vector $r \in [0,1]^{dim(\Omega)}$, todos de forma aleatoria. Ahora generamos una nueva solución $\hat{x}$ con

$$\hat{x}_i = a_i + F(b_i + c_i) \quad \text{si} \quad r_i < CR \quad \text{ó} \quad i = J \qquad \text{y}$$

$$\hat{x}_i = x_i \qquad \text{en caso contrario.}$$

La evolución diferencial es una técnica profundamente estudiada teórica y empíricamente. En [Sto96] podemos ver una lista de sugerencias o "reglas de pulgar" para ajustar sus hiperparámetros.

2 Modelo teórico para algoritmos poblacionales

2.1. Glivenko-Cantelli. Aproximación con poblaciones infinitas

En la práctica, todos los algoritmos evolutivos, como es natural, limitan su número de evaluaciones de la función objetivo a una cantidad finita. Esto implica que el tamaño de las "poblaciones" de posibles soluciones que se hacen evolucionar a lo largo del proceso también está limitado. Sin embargo, el estudio teórico de algoritmos evolutivos con poblaciones finitas es muy complejo. Para hacer frente a esta dificultad, es habitual ceñirse al estudio del comportamiento asintótico del progreso de la distribución de probabilidad de una población de tamaño infinito a lo largo de varias generaciones. En esta sección se expondrá e interpretará el desarrollo matemático de este modelo realizado en [XP94a], [XP94b] y [QM04].

Se empezará justificando nuestro interés en el análisis asintótico de las distribuciones de soluciones. Se ja dicho que esto simplifica el estudio, pero ello sería irrelevante si no hubiese ninguna relación ni parecido entre los comportamientos de la población finita y la infinita. El teorema de Glivenko-Cantelli [DGL96] (pág. 193), que se enunciará a continuación, establece una relación entre el comportamiento de la distribución muestral (la de un conjunto de puntos) y su distribución de probabilidad subyacente.

Teorema 2.1: sean $Z_1, ..., Z_n$ variables reales independientes e idénticamente distribuidas, con una función de distribución $F(z) = P\{Z_1 \leq z\}$. Se denota la función de distribución muestral como

$$F_n(z) = \frac{1}{n} \sum_{i=1}^{n} I_{\{Z_i \leq z\}}$$

donde $I_{\{e\}}$ es una función indicadora que devuelve 1 si e es cierto y 0 si es falso. Entonces:

$$P\left[\sup_{z \in \mathbb{R}} \left| F(z) - F_n(z) \right| > \epsilon \right] \leq 8(n+1)e^{-n\epsilon^2/32}$$

y, en particular,

$$\lim_{n \to \infty} \sup_{z \in \mathbb{R}} \left| F(z) - F_n(z) \right| = 0 \qquad \text{c.s.}$$

La prueba de esta proposición es relativamente laboriosa y no será incluida aquí, aunque puede consultarse en la fuente citada con anterioridad. El teorema de Glivenko-Cantelli establece que existen condiciones bajo las cuales cabe esperar que la función de distribución empírica (eso sí, de variables reales) converja uniformemente a la función de distribución subyacente aumentando el tamaño de la muestra, por lo que da una motivación para el modelo con poblaciones infinitas.

En la sección siguiente se definirá matemáticamente el proceso de evolución y se probará un teorema análogo, adaptado al caso de los algoritmos evolutivos, que nos dará información sobre cómo puede esperarse que cambie su comportamiento en función del tamaño de la población.

2.1.1. Operadores de selección y mutación

En esta subsección se desarrollará el modelo con poblaciones infinitas considerando que el algoritmo evolutivo solo aplica operadores de selección proporcional y mutación, como se hace en [XP94a]. A nivel complementario, se añadirá el resultado obtenido en [QM04] que demuestra que, asumiendo una población infinita y aplicando un operador de selección mediante torneo, la distribución de la población tiende a degenerarse en el conjunto de soluciones óptimas.

El modelo utilizado considerará un problema de maximización continuo monobjetivo con $\Omega \subseteq \mathbb{R}^m$, para algún $m \in \mathbb{N}$. Como ya se ha explicado, maximizar una función es completamente análogo a minimizarla y hará más sencillas algunas de las consideraciones que se harán más adelante. Se asumirá que la función a maximizar, f, tiene una cantidad finita de máximos globales y de discontinuidades, y que es acotada y positiva.

De acuerdo con [XP94a], un algoritmo evolutivo genérico con selección proporcional y mutación puede modelarse como:

Algoritmo 1 (Algoritmo evolutivo con selección proporcional y mutación):

1. **Inicialización:** se fija un tamaño de población N y una función de densidad p_0 en Ω, de forma que para cada tiempo k se cuenta con N vectores aleatorios independientes e idénticamente distribuidos (i.i.d.), siguiendo una distribución de probabilidad con función de densidad p_k en Ω. Se define un criterio de parada y se inicia el algoritmo, igualando k a 0 y muestreando N vectores aleatorios $x_0^1, ..., x_0^N$ a partir de p_0.

2. **Selección:** dados $x_k^1, ..., x_k^N$, se toman $x_k'^1, ..., x_k'^N$ mediante selección proporcional. Esto es,
$$P\left[x_k'^i = x_k^j \middle| x_k^1, ..., x_k^N\right] = \frac{f(x_k^j)}{\sum_{l=1}^N f(x_k^l)}$$

3. **Mutación:** se altera cada vector solución x muestreando un nuevo candidato en su vecindad, mediante una función de densidad simétrica centrada en x y acotada, que se notará como $p_{w_k}(y|x)$. Una función gaussiana centrada en x, por ejemplo, sería una opción válida para aplicar esta mutación. El motivo de hacer que este operador de mutación dependa del tiempo k, es que esto permite ir reduciendo progresivamente el radio de la perturbación, ayudando a la convergencia en las últimas iteraciones.

4. **Terminación:** si se satisface algún criterio de parada, se devuelve la mejor solución. En caso contrario, $k = k + 1$ y se vuelve al paso 2.

Se ha representado la evolución de la población de soluciones como una sucesión de dis-

tribuciones de probabilidad dependientes del tiempo, esto es, como un proceso estocástico. Nótese que las funciones de densidad p_k no se definen de forma explícita, sino implícitamente a partir de la población en el tiempo k y de la aplicación de los operadores de selección proporcional y de mutación, el primero dependiente de la bondad de los candidatos y el segundo posiblemente cambiante con el tiempo.

El siguiente teorema, demostrado en [XP94a](Apéndice A), describe como se distribuirían en función del tiempo poblaciones infinitas sujetas al proceso de evolución descrito anteriormente.

Teorema 2.2: dado el algoritmo evolutivo con selección proporcional y mutación definidas en cada generación k por sendas funciones de densidad $p'_k, p_{w_k}(\cdot|\cdot)\colon \Omega^2 \to \mathbb{R}$. Si p_{w_k} es esencialmente acotada y se cumple que $p_{k+1} = p_{w_k} \cdot p'_k$ entonces se puede concluir que, cuando $N \to \infty$, la sucesión de funciones de densidad de la población en cada instante k puede representarse como:

$$p_{k+1}(x) = \frac{\int_\Omega p_k(y) f(y) p_{w_k}(x|y) dy}{\int_\Omega p_k(y) f(y) dy}$$

Demostración: se desarrollará la demostración expuesta en [XP94a], con ligeras modificaciones y aclaraciones. Sean $x_k^1, ..., x_k^N$ vectores aleatorios que toman valores en Ω las N soluciones candidatas en la generación k. Se denotará como X_k a la población completa en el tiempo k, esto es, $X_k = (x_k^1, ..., x_k^N)$, y p'_k será la función de densidad de la distribución de soluciones candidatas después de aplicar selección sobre los individuos de la generación k. A raíz del paso 2 de nuestro algoritmo, se puede escribir la probabilidad $p'_k(z|X_k)$ de observar un $z \in \Omega$, condicionada a la población actual X_k, de la forma siguiente:

$$p'_k(z|X_k) = \frac{\sum_{j=1}^N \delta_{x_k^j}(z) f(x_k^j)}{\sum_{l=1}^N f(x_k^l)} \quad \forall z \in \Omega$$

donde δ_x es la función *delta de Dirac* m-dimensional (recuérdese que $\Omega \subseteq \mathbb{R}^m$), la cual satisface que

$$\int_\Omega \delta_x(z) f(z)\, dz = f(x) \quad \forall x \in \Omega.$$

La función de densidad p_{k+1} corresponde al resultado de mutar las soluciones previamente seleccionadas. La hipótesis $p_{k+1} = p_{w_k} \cdot p'_k$ permite escribir

$$\begin{aligned} p_{k+1}(x|X_k) &= \int_\Omega p_{w_k}(x|z) p'_k(z|X_k)\, dz \\ &= \int_\Omega p_{w_k}(x|z) \frac{\sum_{j=1}^N \delta_{x_k^j}(z) f(x_k^j)}{\sum_{l=1}^N f(x_k^l)}\, dz \\ &= \frac{\sum_{j=1}^N f(x_k^j) \int_\Omega p_{w_k}(x|z) \delta_{x_k^j}(z)\, dz}{\sum_{l=1}^N f(x_k^l)} \\ &= \frac{\sum_{j=1}^N f(x_k^j) p_{w_k}(x|x_k^j)}{\sum_{l=1}^N f(x_k^l)} \end{aligned} \tag{2.1}$$

Por el Teorema de la Probabilidad Total y notando la función de densidad conjunta de X_k como p_{X_k}, se puede calcular p_{k+1} a partir de la expresión de su probabilidad condicionada a X_k como

$$\begin{aligned} p_{k+1}(x) &= \int_{\Omega^N} p_{k+1}(x|y^1, ..., y^N) \cdot p_{X_k}(y^1, ..., y^N)\, dy^1 ... dy^N \\ &= \sum_{j=1}^N \int_{\Omega^N} \frac{f(y^j) p_{w_k}(x|y^j)}{\sum_{l=1}^N f(y^l)} \cdot p_{X_k}(y^1, ..., y^N)\, dy^1 ... dy^N \end{aligned} \tag{2.2}$$

Se verá que todos los términos de la sumatoria anterior son iguales. Sean $i, j \in \{1, ..., N\}$:

$$\begin{aligned} &\int_{\Omega^N} \frac{f(y^j) p_{w_k}(x|y^j)}{\sum_{l=1}^N f(y^l)} \cdot p_{X_k}(y^1, ..., y^j, ..., y^i, ..., y^N)\, dy^1 ... dy^j ... dy^i ... dy^N \\ &= \int_{\Omega^N} \frac{f(y^j) p_{w_k}(x|y^j)}{\sum_{l=1}^N f(y^l)} \cdot p_{X_k}(y^1, ..., y^i, ..., y^j, ..., y^N)\, dy^1 ... dy^j ... dy^i ... dy^N \end{aligned}$$

Donde se ha usado que las coordenadas de $p_{X_k}(y^1, ..., y^j, ...y^i, ..., y^N)$ pueden permutarse sin afectar al valor de la integral, debido a que el etiquetado de los elementos resultantes de la selección es arbitrario y el operador de mutación afecta a cada elemento independientemente de su índice. Intercambiando ahora el orden de integración de y_j e y_i, se obtiene el resultado deseado:

$$\begin{aligned} &\int_{\Omega^N} \frac{f(y^j) p_{w_k}(x|y^j)}{\sum_{l=1}^N f(y^l)} \cdot p_{X_k}(y^1, ..., y^j, ..., y^i, ..., y^N)\, dy^1 ... dy^j ... dy^i ... dy^N \\ &= \int_{\Omega^N} \frac{f(y^j) p_{w_k}(x|y^j)}{\sum_{l=1}^N f(y^l)} \cdot p_{X_k}(y^1, ..., y^i, ..., y^j, ..., y^N)\, dy^1 ... dy^i ... dy^j ... dy^N \\ &= \int_{\Omega^N} \frac{f(y^i) p_{w_k}(x|y^i)}{\sum_{l=1}^N f(y^l)} \cdot p_{X_k}(y^1, ..., y^j, ..., y^i, ..., y^N)\, dy^1 ... dy^j ... dy^i ... dy^N \end{aligned}$$

Por tanto:

$$p_{k+1}(x) = N \int_{\Omega^N} \frac{f(y^j) p_{w_k}(x|y^j)}{\sum_{l=1}^{N} f(y^l)} \cdot p_{X_k}(y^1, ..., y^N)\, dy^1 ... dy^N$$
$$\text{para algún } j \in \{1, ..., N\}$$

Se definen dos nuevas variables aleatorias:

$$\eta_k^N = \frac{1}{N} \sum_{l=1}^{N} f(x_k^l) \qquad\qquad \xi_k(x) = f(x_k^j) p_{w_k}(x|x_k^j) \quad j \in \{1, ..., N\}$$

Se tiene que:

$$p_{k+1}(x) = \mathbb{E}_{X_k}\Big[\frac{\xi_k(x)}{\eta_k^N}\Big] \tag{2.3}$$

donde $\mathbb{E}_{X_k}[\cdot]$ es la esperanza matemática con respecto a la función de densidad p_{X_k}. Por las restricciones impuestas sobre f, es claro que $0 \leq \eta_k^N \leq \frac{1}{N} N \cdot \sup_{x\in\Omega} f(x) = \sup_{x\in\Omega} f(x) \quad \forall N \in \mathbb{N}$. Por tanto, cuando $N \to \infty$, η_k^N converge casi por doquier a alguna variable aleatoria η_k con media

$$\mathbb{E}_{X_k}[\eta_k] = \mathbb{E}_{X_k}[f(x_k^j)] = \int_{\Omega^N} f(y^j) p_k(y^j)\, dy^1 ... dy^N = \int_{\Omega} f(y^j) p_k(y^j)\, dy^j \quad \forall k, \quad j \in \{1, ..., N\}$$

η_k es, además, claramente independiente de η_k^N para cualquier N finito. En particular, es independiente de $\eta_k^N = g(x_k^j)$ y, por tanto, también de $\xi_k(x)$.

Se define

$$\Delta_k(N, x) = \left| p_{k+1}(x) - \frac{\int_\Omega p_k(y) f(y) p_{w_k}(x|y)\, dy}{\int_\Omega p_k(y) f(y)\, dy} \right|$$

Ver que $\Delta_k(N, x) \xrightarrow{N\to\infty} 0 \quad \forall k \in \mathbb{N}, \forall x \in \Omega$ concluirá nuestra demostración.

$$\Delta_k(N, x) = \left| \mathbb{E}_{X_k}\Big[\frac{\xi_k(x)}{\eta_k^N}\Big] - \frac{\mathbb{E}_{X_k}\Big[\xi_k(x)\Big]}{\mathbb{E}_{X_k}\Big[\eta_k\Big]} \right|.$$

Como η_k es positivo e independiente de $\xi_k(x)$ y η_k^N, se tiene:

$$\begin{aligned} \Delta_k(N, x) &= \frac{1}{\mathbb{E}_{X_k}[\eta_k]} \left| \mathbb{E}_{X_k}\Big[\frac{\xi_k(x)\eta_k}{\eta_k^N}\Big] - \xi_k(x) \right| \\ &= \frac{1}{\mathbb{E}_{X_k}[\eta_k]} \left| \mathbb{E}_{X_k}\Big[\frac{\xi_k(x)}{\eta_k^N}(\eta_k - \eta_k^N)\Big] \right| \\ &\leq \frac{1}{\mathbb{E}_{X_k}[\eta_k]} \mathbb{E}_{X_k}\Big[\frac{\xi_k(x)}{\eta_k^N}|\eta_k - \eta_k^N|\Big] \end{aligned} \tag{2.4}$$

Recuérdese que se ha asumido p_k acotada para todo $k \in \mathbb{N}$. Tomando f_{min}, M_k y f_{max} tal que $f_{min} \leq \eta_k, \eta_k^N$, $p_k \leq M_k$ y $\xi_k(x) \leq f_{max} M_k$, y sustituyendo en la última desigualdad:

$$\Delta_k(N, x) \leq \frac{M f_{max}}{f_{min}^2} \mathbb{E}_{X_k}\Big[|\eta_k^N - \eta_k|\Big]$$

Esto es suficiente para concluir el resultado buscado, puesto que se sabe que η_k^N tiende a η_k casi seguramente y la esperanza de su diferencia, por tanto, tenderá a 0. Sin embargo, llevando la prueba un poco más lejos se puede encontrar un resultado interesante.

Se tiene que:

$$\begin{aligned}
\mathbb{E}_{X_k}\Big[|\eta_k^N - \eta_k|\Big] &= \int_\Omega \chi_{\{|\eta_k^N - \eta_k| \leq \frac{1}{\sqrt{N}}\}} |\eta_k^N - \eta_k| \cdot p_k(\omega)\, d\omega + \int_\Omega \chi_{\{|\eta_k^N - \eta_k| > \frac{1}{\sqrt{N}}\}} |\eta_k^N - \eta_k| \cdot p_k(\omega)\, d\omega \\
&\leq \frac{1}{\sqrt{N}} \cdot P\Big\{|\eta_k^N - \eta_k| \leq \frac{1}{\sqrt{N}}\Big\} + ess\sup |\eta_k^N - \eta_k| \cdot P\Big\{|\eta_k^N - \eta_k| \geq \frac{1}{\sqrt{N}}\Big\} \\
&\leq \frac{1}{\sqrt{N}} + (f_{max} - f_{min}) \cdot P\Big\{|\eta_k^N - \eta_k| \geq \frac{1}{\sqrt{N}}\Big\}
\end{aligned} \tag{2.5}$$

Como f_{max} y f_{min} no dependen de k, puede concluirse que

$$\mathbb{E}_{X_k}\Big[|\eta_k^N - \eta_k|\Big] = O\Big(\frac{1}{\sqrt{N}}\Big)$$

□

A modo de aclaración, el hecho de imponer que $p_{k+1} = p_{w_k} \cdot p'_k$ como hipótesis del teorema 2.1.1 puede garantizarse mutando todas las soluciones en una misma generación de la misma forma, por ejemplo con ruido gaussiano.

Las últimas líneas de la demostración implican que la función de densidad de la distribución de las poblaciones finitas se acerca a la función de densidad teórica para poblaciones infinitas con una diferencia que decrece en orden $O(\frac{1}{\sqrt{N}})$, garantizando que para tamaños de población suficientemente grandes un análisis basado en poblaciones infinitas puede aproximar razonablemente bien el comportamiento de un algoritmo evolutivo con poblaciones finitas.

Ahora que se tienen herramientas para estudiar la evolución de un algoritmo evolutivo con poblaciones infinitas, se verá un teorema que muestra cómo afecta el proceso de selección a la búsqueda de mejores soluciones [QM04].

Se denotará como $x^* \in \Omega$ a la solución que maximiza f en Ω y $f_{max} = f(x^*)$. Se escribirá como $E(k)$ la esperanza matemática del valor objetivo de la población en el tiempo k, esto es

$$E(k) = \int_\Omega p_k(x) f(x)\, dx.$$

Se dirá que el algoritmo elegido convergerá globalmente si $E(k) \xrightarrow{k\to\infty} f_{max}$. Buscar que la población completa converja al óptimo global puede parecer innecesario, puesto que no se requiere la convergencia de toda la población para encontrar la solución óptima. En la práctica lo más común es ir guardando el candidato con mejor valor objetivo encontrado hasta el momento, por lo que basta con alcanzar el óptimo una vez. El análisis del número esperado de iteraciones hasta alcanzar la solución óptima, conocido como *"first hitting time analysis"*, ha sido estudiado por otros autores [CZH11][HY02], pero no será necesario para dar garantías de la convergencia de poblaciones infinitas bajo selección, como demuestra el teorema siguiente.

Teorema 2.3: [QM04] suponiendo que f tiene al menos un máximo global en un entorno de medida no nula donde f es continua. Si p_0 es estrictamente positiva y continua en todo Ω, entonces al aplicar sucesivamente el operador de selección sobre la distribución inicial el algoritmo converge:

1. Usando selección proporcional.

2. Usando selección por torneo binario.

Demostración:

1. Para la selección proporcional, se puede usar el esquema obtenido del teorema anterior para poblaciones infinitas, si se aplica un "operador de mutación" que no provoque ningún cambio. Esto puede lograrse tomando $p_{w_k}(x|y) = \delta_x(y)$, obteniendo así:

$$p_{k+1}(x) = \frac{\int_\Omega p_k(y) f(y) p_{w_k}(x|y)\, dy}{\int_\Omega p_k(y) f(y)\, dy} = \frac{\int_\Omega p_k(y) f(y) \delta_x(y)\, dy}{\int_\Omega p_k(y) f(y)\, dy} = \frac{p_k(x) f(x)}{E(k)}$$

De la definición de $E(k)$, se deduce que

$$E(k+1) = \int_\Omega p_{k+1}(x) f(x)\, dx = \frac{\int_\Omega p_k(x) f(x)^2\, dx}{E(k)}$$

Recuérdese que $E(k)$ representa el valor esperado de f sobre la distribución con función de densidad p_k, esto es, $E(k) = \mathbb{E}[f(x)]$. Del desarrollo de $E(k+1)$, también se deduce que $E(k+1) = \frac{\mathbb{E}[f(x)^2]}{E(k)}$. Dividiendo el valor esperado de la generación siguiente entre el de la actual, se obtiene:

$$\frac{E(k+1)}{E(k)} = \frac{\mathbb{E}[f(x)^2]}{\mathbb{E}[f(x)]^2} = \frac{\mathbb{E}[f(x)]^2 + Var(f(x))}{\mathbb{E}[f(x)]^2} \geq 1$$

donde se ha usado la fórmula $\mathbb{E}[X \cdot Y] = \mathbb{E}[X]\mathbb{E}[Y] + Cov(X, Y)$. Como el cociente anterior es mayor o igual que 1, es claro que

$$E(k+1) \geq E(k).$$

Como $E(k)$ no puede ser mayor que f_{max}, este debe converger a algún valor menor o igual

a f_{max}, que se llamará G. Asumiendo que $G < f_{max}$. Se cumple que:

$$p_k(x) = \frac{f(x)}{E(k-1)} \cdot \frac{f(x)}{E(k-2)} \cdots \frac{f(x)}{E(0)} p_0(x).$$

Sea $x^* \in \Omega$ un máximo global de f con un entorno de medida no nula en Ω, denotado como U. Dado que f es continua en U, debe existir otro entorno de x^* de medida no nula, V, tal que $V \subseteq U$ y $f(v) > G \ \forall v \in V$. Visto el desarrollo anterior de $p_k(x)$, esto implica que

$$\lim_{k\to\infty} p_k(x) = +\infty \qquad \forall x \in V$$

y, por tanto,

$$\lim_{k\to\infty} \int_V p_k(x) = +\infty$$

así que existe un cierto k_0 tal que

$$\int_\Omega p_k(x) > 1 \qquad \forall k > k_0$$

lo que contradice el hecho de que p_k es una función de densidad para todo $k \geq 0$. Se deduce, por tanto, que $G = f_{max}$, o lo que es lo mismo, el algoritmo converge globalmente.

2. De acuerdo con [Mü97] (sección 13), el proceso de selección mediante 2-torneo puede modelarse como

$$p_{k+1}(x) = 2p_k(x) \int_{\{y\in\Omega:\, f(y)\leq f(x)\}} p_k(y)\, dy$$

Para cualquier $\varepsilon > 0$, sea $N_\varepsilon = \{x \in \Omega \colon f(x) \geq f_{max} - \varepsilon\}$. Se define

$$\begin{aligned} r_k(\varepsilon) &= \int_{N_\varepsilon} p_k(x)\, dx \\ &\text{y} \\ q_k(\varepsilon) &= \int_{\Omega\setminus N_\varepsilon} p_k(x)\, dx \end{aligned} \tag{2.6}$$

Por la definición de p_{k+1} se cumple que

$$r_{k+1}(\varepsilon) = \int_{N_\varepsilon} \left[2p_k(x) \int_{\{y\in\Omega:\, f(y)\leq f(x)\}} p_k(y)\, dy \right] dx.$$

Para todo $x \in N_\varepsilon$, claramente:

$$\int_{\{y\in\Omega:\, f(y)\leq f(x)\}} p_k(y)\, dy = q_k(\varepsilon) + \int_{\{y\in\Omega:\, f_{max}-\varepsilon\leq f(y)\leq f(x)\}} p_k(y)\, dy.$$

Combinando las dos últimas igualdades y simplificando, se obtiene

$$r_{k+1}(\varepsilon) = 2q_k(\varepsilon)r_k(\varepsilon) + 2\int_{x\in N_\varepsilon} \int_{f_{max}-\varepsilon\leq f(y)\leq f(x)} p_k(x)p_k(y)\, dxdy \tag{2.7}$$

Por la simetría de x e y dentro de la integral, se tiene que

$$\int_{x\in N_\varepsilon}\int_{f_{max}-\varepsilon\leq f(y)\leq f(x)} p_k(x)p_k(y)\,dxdy = \int_{y\in N_\varepsilon}\int_{f_{max}-\varepsilon\leq f(x)\leq f(y)} p_k(x)p_k(y)\,dxdy$$

y, por tanto,

$$\begin{aligned} r_{k+1}(\varepsilon) &= 2q_k(\varepsilon)r_k(\varepsilon) + 2\int_{x\in N_\varepsilon}\int_{f_{max}-\varepsilon\leq f(y)\leq f(x)} p_k(x)p_k(y)\,dxdy \\ &= 2q_k(\varepsilon)r_k(\varepsilon) + \int_{x\in N_\varepsilon}\int_{f_{max}-\varepsilon\leq f(y)\leq f(x)} p_k(x)p_k(y)\,dxdy \\ &\qquad + \int_{y\in N_\varepsilon}\int_{f_{max}-\varepsilon\leq f(x)\leq f(y)} p_k(x)p_k(y)\,dxdy \\ &= 2q_k(\varepsilon)r_k(\varepsilon) + \int_{x\in N_\varepsilon}\int_{y\in N_\varepsilon} p_k(x)p_k(y)\,dxdy \\ &= 2q_k(\varepsilon)r_k(\varepsilon) + r_k(\varepsilon)^2 \end{aligned} \tag{2.8}$$

Usando que $q_k(\varepsilon) + r_k(\varepsilon) = 1$ se tiene

$$\left.\begin{array}{ll} r_k(\varepsilon)^2 & = 1 + q_k(\varepsilon)^2 - 2q_k(\varepsilon) \\ 2q_k(\varepsilon)r_k(\varepsilon) & = 2q_k(\varepsilon) - 2q_k(\varepsilon)^2 \end{array}\right\} \implies r_{k+1}(\varepsilon) = 1 - q_k(\varepsilon)^2 \implies q_{k+1}(\varepsilon) = q_k(\varepsilon)^2$$

lo que implica que

$$q_{k+1}(\varepsilon) = q_0(\varepsilon)^{2(k+1)}$$

Como N_ε tiene medida no nula por hipótesis, y p_0 es estrictamente positiva en todo Ω, se cumple que

$$\lim_{k\to\infty} q_k(\varepsilon) = 0 \qquad\qquad \lim_{k\to\infty} r_k(\varepsilon) = 1$$

Como esto se cumple para un ε arbitrariamente pequeño y f es continua en un entorno de x^*, esto implica que

$$\lim_{k\to\infty} E(k) = f_{max}$$

■

Ante el hecho de que un algoritmo evolutivo converge únicamente bajo presión selectiva, sería razonable preguntarse la utilidad de incluir mutación en el proceso de evolución o, incluso, cualquier otro tipo de operador. Partiendo de la hipótesis fundamental de contar con poblaciones infinitas, lo cual es imposible en la práctica. Técnicas distintas a la selección permiten *explorar* (término que aparece frecuentemente en la literatura sobre metaheurísticas en contraposición con el de *explotar*, que hace referencia a aquellas medidas dedicadas a conservar las características de las soluciones encontradas con mejor valor objetivo) el espacio de soluciones más allá de la población muestreada al inicializar el algoritmo. Si se contase solamente con un número finito de soluciones y se aplicase selección exclusivamente, el

algoritmo convergería con una probabilidad más alta hacia aquellas con mejor valor objetivo, pero no cabría la posibilidad de obtener una solución que no estuviese en dicho conjunto. Otra forma de ver el "absurdo práctico" de esta suposición es que el algoritmo obliga a evaluar cada solución observada. Si se contase con una distribución de soluciones que cubriese todo el dominio, sería tan sencillo como devolver directamente la mejor solución.

Se concluirá enunciando un teorema que demuestra la posibilidad de convergencia global añadiendo mutación no trivial al algoritmo evolutivo [XP94a] (teorema 3).

Teorema 2.4: sea $f\colon \Omega \to \mathbb{R}$ una función positiva, con al menos un máximo global. Suponiendo que cada óptimo global tiene un entorno simplemente conexo (función f bien definida y acotada en dicho entorno) en el que f es continua y la función de densidad inicial p_0 es estrictamente positiva en al menos uno de esos entornos. Entonces existe una sucesión de distribuciones de mutación radialmente simétricas, con densidades $\{p_{w_k}\}_{k\in\mathbb{N}}$, tales que la sucesión de distribuciones de probabilidad de la población definida por

$$p_{k+1}(x) = \frac{\int_\Omega p_k(y) f(y) p_{w_k}(x|y)\, dy}{\int_\Omega p_k(y) f(y)\, dy}$$

converge a un óptimo global.

Conocido el hecho de que el algoritmo converge globalmente aplicando únicamente selección, no es de extrañar que también lo haga aplicando mutaciones suficientemente pequeñas. Es por esto que, si bien su demostración tiene interés teórico, no se expondrá aquí. Puede consultarse en [XP94a], Apéndice C.

2.1.2. Operador de cruce

En esta subsección se explicarán los resultados obtenidos en [XP94b] que muestran los cambios en la distribución de la población al aplicar iterativamente el operador de cruce, dando así una intuición sobre su utilidad.

Este operador se añade a los algoritmos evolutivos con el objetivo de aumentar la diversidad de la población pero, al contrario que la mutación, no busca explorar nuevos valores de cada variable, sino nuevas combinaciones de aquellos valores que ya se encuentren en la población actual y que, se asume, den resultados razonablemente buenos.

Existen varios métodos de cruzar soluciones: media ponderada de los dos vectores, selección de subintervalos aleatorios... En este caso se considerará una de las formas más sencillas conocida como *cruce uniforme* de dos soluciones: consiste en elegir para cada coordenada el valor correspondiente a una de las soluciones con una probabilidad $P \in [0, 1]$, o a la otra solución con una probabilidad $1 - P$. El valor de P puede considerarse como un hiperparámetro del algoritmo. Fijarlo a valores distintos de 0.5 nos permite hacer que el cruce favorezca la herencia de genes del padre que tenga mayor valor objetivo.

Nótese que el cruce uniforme, tal como se ha descrito aquí, podría generar soluciones no factibles en caso de que el dominio Ω no fuese convexo. Es posible eliminar aquellos resultados que no caigan dentro del espacio factible, o bien introducir alguna regla adicional

en el cruce en caso de disponer, por ejemplo, de una parametrización de Ω.

Dentro de nuestro algoritmo evolutivo genérico, se insertaría el operador de cruce uniforme en cada iteración, después de la mutación.

Nuestro análisis no buscará modelar los resultados de aplicar el algoritmo evolutivo completo, que combina selección, mutación y cruce, sino únicamente entender los efectos del operador de cruce. El siguiente teorema [XP94b] (teorema 1) indica cómo pueden modelarse las distribuciones de probabilidad de los individuos de nuestra población aplicando repetida y exclusivamente el operador de cruce uniforme a lo largo de varias generaciones.

Teorema 2.5: Sea $x_k = (x_k^1, ..., x_k^m)$ un vector aleatorio que tome valores en $\mathbb{R}^m$, que representa una posible solución de nuestro problema de optimización. Al aplicar iterativamente cruce uniforme sobre los elementos de la población, la función de densidad de la distribución de soluciones en la generación $k+1$ puede representarse, en términos de la función de densidad correspondiente a la generación k, mediante la siguiente ecuación:

$$p_{k+1}(x^1, ..., x^m) = \sum_{l=0}^{d} [P^l(1-P)^{m-l} + P^{m-l}(1-P)^l] \cdot \sum_{i_1,...,i_l} p_k^{x^{i_1},...,x^{i_l}}(x^{i_1}, ..., x^{i_l}) p_k^{x^{i_{l+1}},...,x^{i_m}}(x^{i_{l+1}}, ..., x^{i_m})$$

donde $d = \frac{m}{2}$ si m es par y $d = \frac{m-1}{2}$ si es impar, y donde $p_k^{x^{i_1},...,x^{i_l}}$ y $p_k^{x^{i_{l+1}},...,x^{i_m}}$ son las funciones de densidad marginales de p_k con respecto a las variables con índices $i_1, ..., i_l$ y a todas las demás, respectivamente.

Demostración:

Sean dos pares de vectores aleatorios, $x_{k+1} = (x_{k+1}^1, ..., x_{k+1}^m)$ y $x'_{k+1} = (x'^1_{k+1}, ..., x'^m_{k+1})$, generados a partir de $x_k = (x_k^1, ..., x_k^m)$ y $x'_k = (x'^1_k, ..., x'^m_k)$ tomados independientemente de la misma distribución con función de densidad $p_k(x^1, ..., x^m)$. La función de densidad conjunta de tomar dos vectores particulares es, por tanto, $p_k(x_k) \cdot p_k(x'_k)$.

Ahora considérese la posibilidad de que un conjunto particular de exactamente l variables, con índices $\{i_1, ..., i_l\}$, sean las únicas en sufrir el cruce. La probabilidad de que dos padres generen el vector resultante de combinar dichos índices es la probabilidad de seleccionar dichos padres, modelada por p_k, multiplicada por la probabilidad de realizar ese cruce particular. En términos de la función de densidad, resulta en la siguiente fórmula:

$$P^l(1-P)^{m-l} p_k(x) p_k(x')$$

Se calcula la suma de la expresión anterior para todos los posibles $0 \leq l \leq m$, y para todas las posibles combinaciones de l índices distintos. Por el Teorema de la Probabilidad Total, al ser las distintas combinaciones de índices eventos excluyente cuya unión es el espacio de eventos total, se puede expresar la función de densidad conjunta de generar los vectores x y x' (independientes) de la siguiente forma:

$$p_{k+1}(x) p_{k+1}(x') = \sum_{l=0}^{m} P^l(1-P)^{m-l} \sum_{i_1,...,i_l} p_k(..., x^{i_1}, ..., x^{i_l}, ...) p_k(..., x'^{i_1}, ..., x'^{i_l}, ...).$$

Como se quiere calcular la probabilidad de realización de un único $x \in \Omega$, se quiere eliminar $p_k(x')$ de la fórmula anterior. Se integran ambos lados de la ecuación con respecto a x' y se separan p_k de x en producto de sus distribuciones marginales con respecto a los índices que se cruzan y los que no, respectivamente. Esto puede hacerse porque, al tratarse de cruce uniforme, se ha asumido que la probabilidad de intercambiar los valores en una coordenada es siempre P, luego es independiente de intercambiar los valores de cualquier otra variable. El resultado es el siguiente:

$$p_{k+1}(x) = \sum_{l=0}^{m} P^l (1-P)^{m-l} \sum_{i_1,\dots,i_l} p_k^{x^{i_1},\dots,x^{i_l}}(x^{i_1},\dots,x^{i_l}) p_k^{x^{i_{l+1}},\dots,x^{i_m}}(x^{i_{l+1}},\dots,x^{i_m})$$

La fórmula anterior puede modificarse ligeramente para alcanzar la forma dada en el enunciado del teorema. Nótese que

$$\begin{aligned} &\sum_{i_1,\dots,i_l} p_k^{x^{i_1},\dots,x^{i_l}}(x^{i_1},\dots,x^{i_l}) p_k^{x^{i_{l+1}},\dots,x^{i_m}}(x^{i_{l+1}},\dots,x^{i_m}) \\ &= \sum_{i_1,\dots,i_{m-1}} p_k^{x^{i_1},\dots,x^{i_{m-l}}}(x^{i_1},\dots,x^{i_{m-l}}) p_k^{x^{i_{m-l+1}},\dots,x^{i_m}}(x^{i_{m-l+1}},\dots,x^{i_m}) \end{aligned} \tag{2.9}$$

donde se han intercambiado los índices de una de las funciones de densidad marginales por los de la otra. No se introduce ningún error, al considerar todas las posibles combinaciones de índices tanto para los l índices a la izquierda como los $m-l$ índices a la derecha de la igualdad. Recorriendo por tanto la mitad de posibles valores de l y tomando $d = \frac{m}{2}$ si m es par, o $d = \frac{m-1}{2}$ si es impar, la función de densidad queda tal como se indicaba:

$$\begin{aligned} p_{k+1}(x^1,\dots,x^m) = \sum_{l=0}^{d} &[P^l(1-P)^{m-l} + P^{m-l}(1-P)^l] \cdot \\ &\cdot \sum_{i_1,\dots,i_l} p_k^{x^{i_1},\dots,x^{i_l}}(x^{i_1},\dots,x^{i_l}) p_k^{x^{i_{l+1}},\dots,x^{i_m}}(x^{i_{l+1}},\dots,x^{i_m}) \end{aligned} \tag{2.10}$$

■

El teorema anterior da una herramienta fundamental para estudiar el comportamiento de la población tras iteraciones de cruces sucesivos. En particular, permitirá demostrar el siguiente teorema, que es el principal resultado de esta sección [XP94b]:

Teorema 2.6: (Teorema de la epistasis)

Sea $p_0^{x^i}$ la función de densidad marginal de p_k con respecto a la i-ésima coordenada en el tiempo $k = 0$. La aplicación reiterada del operador de cruce resulta en la siguiente identidad:

$$\lim_{k\to\infty} p_k(x) = \prod_{i=1}^{m} p_0^{x_i}(x_i)$$

Esto se traduce en que el operador de cruce tiende a reducir la dependencia entre las distintas variables, lo cual resulta bastante razonable si se considera que este proceso intercambia coordenadas aleatorias entre dos individuos cualesquiera. El nombre de este resultado vie-

ne del campo de la genética, siendo la *epistasis* la denominación dada a la interacción y dependencia entre distintos genes. Se hablará más sobre esto en el capítulo 3.

Demostración:

Se realizará una prueba por inducción. Para el caso $m = 2$, la fórmula demostrada en el teorema 2.5 da que

$$p_{k+1}(x_1, x_2) = [P^2 + (1-P)^2]p_k(x_1, x_2) + 2P(1-P)p_k^{x_1}(x_1)p_k^{x_2}(x_2) \tag{2.11}$$

Aplicando ahora la definición de función de densidad marginal, con $\Omega = \Omega_1 \times \Omega_2$, se obtiene que

$$p_{k+1}^{x_1}(x_1) = \int_{\Omega_2} p_{k+1}(x_1, x_2)\, dx_2 = p_k^{x_1}(x_1)[P^2 + (1-P)^2 + 2P(1-P)] = p_k^{x_1}(x_1)$$

Y análogamente para la distribución marginal con respecto a su otra variable. Se tiene entonces que $p_k^{x_i} = p_0^{x_i} \quad \forall k \in \mathbb{N}, i \in \{1, 2\}$. Aplicando iterativamente 2.11, se obtiene que

$$p_k(x_1, x_2) = [P^2 + (1-P)^2]^k p_0(x_1, x_2) + (1 - [P^2 + (1-P)^2]^k)p_0^{x_1}(x_1)p_0^{x_2}(x_2)$$

Como $[P^2 + (1-P)^2] < 1 \quad \forall P \in (0, 1)$, para $m = 2$ obtenemos el resultado que se pide:

$$\lim_{k\to\infty} p_k(x_1, x_2) = p_0^{x_1}(x_1)p_0^{x_2}(x_2).$$

Ahora se supone que la hipótesis de inducción se cumple para un cierto número de dimensiones $r \in \mathbb{N}$. Esto es:

$$\lim_{k\to\infty} p_k(x_1, ..., x_r) = \prod_{i=1}^{r} p_0^{x_i}(x_i)$$

Por el teorema 2.5 (en particular, lo visto en su demostración), se cumple que

$$p_{k+1}(x^{i_1}, ..., x^{i_{r+1}}) = \sum_{l=0}^{r+1} [P^l(1-P)^{r+1-l}] \cdot \sum_{i_1,...,i_l} p_k^{x^{i_1},...,x^{i_l}}(x^{i_1}, ..., x^{i_l}) p_k^{x^{i_{l+1}},...,x^{i_{r+1}}}(x^{i_{l+1}}, ..., x^{i_{r+1}}). \tag{2.12}$$

Tomando límites a ambos lados de la igualdad y usando la hipótesis de inducción, se

obtiene que

$$\begin{aligned}\lim_{k\to\infty} p_{k+1}(x^{i_1},...,x^{i_{r+1}}) =& [(1-P)^{r+1}+P^{r+1}]\lim_{k\to\infty} p_k(x^{i_1},...,x^{i_{r+1}}) \\ &+\sum_{l=1}^{r}[P^l(1-P)^{r+1-l}]\prod_{i=1}^{r+1}p_0^{x_i}(x_i) \\ =& [(1-P)^{r+1}+P^{r+1}]\lim_{k\to\infty} p_k(x^{i_1},...,x^{i_{r+1}}) \\ &+\{1-[(1-P)^{r+1}+P^{r+1}]\}\prod_{i=1}^{r+1}p_0^{x_i}(x_i)\end{aligned} \tag{2.13}$$

Llevando el primer término de la derecha restando a la izquierda y teniendo en cuenta que $\lim_{k\to\infty} p_k(x^{i_1},...,x^{i_{r+1}}) = \lim_{k\to\infty} p_{k+1}(x^{i_1},...,x^{i_{r+1}})$, se tiene que

$$\lim_{k\to\infty} p_{k+1}(x^{i_1},...,x^{i_{r+1}}) = \prod_{i=1}^{r+1}p_0^{x_i}(x_i)$$

lo que completa la inducción. ■

Las curvas de nivel en la figura 2.1 representan la evolución de una población sobre la que se ha aplicado sucesivos cruces uniformes.

Claramente, tal como advierte el teorema 2.6, el operador de cruce actúa conservando las distribuciones marginales de la distribución original, eliminando la dependencia entre variables distintas.

Se concluye sintetizando la información desarrollada en las dos últimas subsecciones. Se han descrito los pasos de un algoritmo evolutivo genérico, en el que se aplican repetidamente un proceso de selección, mutación y cruce.

Asumiendo poblaciones infinitas, se ha visto que el proceso de selección por sí solo es suficiente como para hacer tender el algoritmo a la solución óptima. Dado que esta hipótesis no se sostiene en la práctica, se incluyen mutaciones aleatorias que añaden variedad a las soluciones generadas originalmente que serían las únicas exploradas en caso de aplicar únicamente selección.

El operador de cruce también ayuda a explorar el espacio de soluciones de una forma más fuerte, al mismo tiempo que controlada. Al crear nuevas soluciones mezclando las generadas previamente, se reduce la probabilidad de que el algoritmo asuma implícitamente dependencias espurias entre las variables debido a que la población actual no represente correctamente el espacio objetivo. De esta forma, se cruzan las soluciones más prometedoras generando otros candidatos potencialmente interesantes.

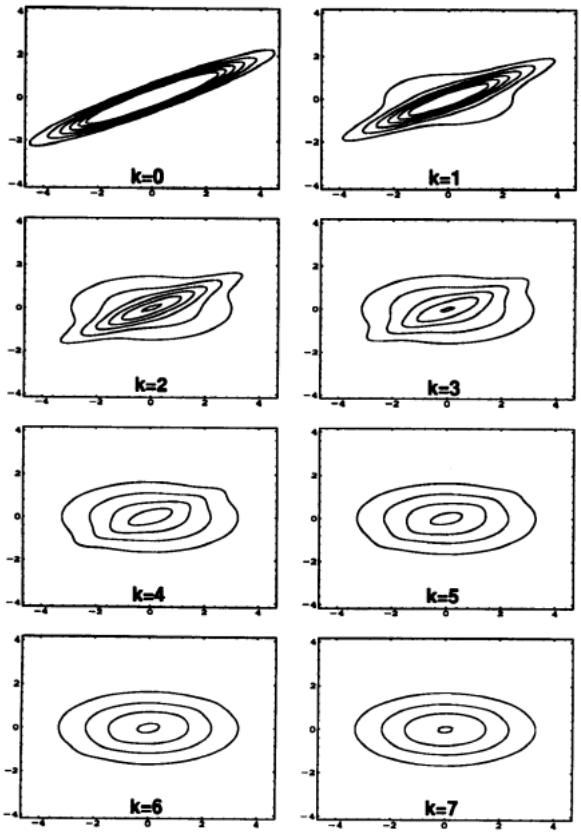


Figura 2.1: Tomada de [XP94b] (Xiaofeng Qi y F. Palmieri, *Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. part ii: Analysis of the diversification role of crossover*, figura 4). Evolución de una población originalmente muestreada de una distribución normal bivariante con vector de medias $(0,0)$ y matriz de covarianzas $\begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix} = \begin{pmatrix} 4 & 0.95 \cdot 2 \cdot 1 \\ 0.95 \cdot 2 \cdot 1 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 1.9 \\ 1.9 & 1 \end{pmatrix}$, sobre la que se ha aplicado sucesivos cruces uniformes con parámetro $P = 0.5$.

2.2. Análisis de la deriva

El análisis de la deriva o, como es más frecuentemente llamado en la literatura sobre *EAs*, *drift analysis*, es una herramienta usada para el análisis teórico de algoritmos evolutivos [HY04] [HY16] [LS17] [Len18]. En particular, la se usará en 3.4.

Si bien no parece haber a día de hoy resultados de importancia que apliquen el *drift analysis* al campo de los Algoritmos Coevolutivos Cooperativos (*CCEAs*), nuestro interés principal en este trabajo (capítulo 4), resulta natural pensar que una de las principales herramientas para el estudio teórico de los algoritmos evolutivos en los últimos años sea una buena candidata para impulsar nuestro conocimiento sobre los *CCEAs* en el futuro. Es por eso que

se dedicará esta sección a exponer algunos de los resultados más importantes encontrados sobre *drift analysis* hasta la fecha. Simplemente se enunciarán cada uno de ellos, remitiendo su demostración a sus respectivas fuentes.

El análisis de la deriva puede aplicarse a cualquier algoritmo de optimización iterativo. Para ello, se necesita una medida de la "distancia", en un sentido general, entre la solución actual en el tiempo k y la óptima. Esta medida, que se notará como X_k, es habitualmente denominada como *potencial*.

El potencial suele modelarse como una variable aleatoria con un espacio de estados $\mathcal{S} \subseteq \mathbb{R}_0^+$ finito. Es deseable que el algoritmo de optimización converja a la solución óptima y que, por tanto, pueda alcanzarse un potencial de 0. Por este motivo, se impone que 0 pertenezca a $\mathcal{S}$. Un par de ejemplos sencillos de potencial son la distancia de la solución actual a la óptima, asumiendo que esta tenga sentido en el espacio de soluciones de nuestro problema, o la distancia en el espacio objetivo, esto es, la diferencia entre el valor objetivo de la solución actual y la óptima (en un problema de minimización) o el opuesto de este valor (en uno de maximización). La elección de un potencial adecuado puede requerir, sin embargo, de un conocimiento previo de la estructura del problema y de soluciones deseables del mismo y es, por tanto, uno de los elementos más importantes y complicados del análisis de la deriva [Len18].

El objetivo del *drift analysis* es hallar condiciones sobre la evolución del potencial que permitan acotar el tiempo esperado de parada (nº de generaciones esperadas antes de alcanzar el óptimo) y/o la probabilidad de grandes desviaciones en el número de iteraciones. Esto se hace definiendo la siguiente función:

$$\Delta_k(s) = \mathbb{E}[X_k - X_{k+1} | X_k = s]$$

Esta función es conocida como *drift* y representa el cambio esperado de potencial al cambiar de generación, asumiendo un potencial actual de s. Para evitar problemas derivados del uso de la esperanza matemática sobre conjuntos de medida nula, se define $\Delta_k(s)$ solo sobre aquellos $s \in \mathcal{S}$ tal que $P[X_k = s] > 0$.

Se procede a enunciar el primero de los teoremas de esta sección.

Teorema 2.7: (Additive Drift Theorem [HY04], lemas 1 y 2) Sea $\{X_k\}_{k\in\mathbb{N}}$ una sucesión de variables aleatorias que toman valores en $S \subseteq \mathbb{R}_0^+$, con $0 \in \mathcal{S}$ y $|\mathcal{S}| < \infty$. Sea $T := \inf\{k \in \mathbb{N} | X_k = 0\}$, esto es, el tiempo de llegada al óptimo o *first hitting time*.

1. Si $\exists \delta > 0$ tal que, $\forall s \in \mathcal{S} \setminus \{0\}$ y $\forall k \geq 0$,

   $$\Delta_k(s) \geq \delta \quad \text{(condición de aditividad)}$$

   entonces

   $$\mathbb{E}[T] \leq \frac{\mathbb{E}[X_0]}{\delta}$$

2. Si $\exists \delta > 0$ tal que, $\forall s \in \mathcal{S} \setminus \{0\}$ y $\forall k \geq 0$,

$$\Delta_k(s) \leq \delta$$

entonces

$$\mathbb{E}[T] \geq \frac{\mathbb{E}[X_0]}{\delta}$$

El teorema anterior afirma un resultado bastante razonable. Si la velocidad de reducción de potencial es, en promedio, mayor a cierto número δ, es de esperar que el tiempo necesario para reducir el potencial a 0 sea menor o igual a la diferencia entre el potencial inicial y el final (se observa que $\mathbb{E}[X_0] - \mathbb{E}[X_T] = \mathbb{E}[X_0]$) partido por dicha "velocidad" promedio. El resultado inverso es igualmente razonable.

A continuación se comentará el siguiente teorema, que puede probarse a partir del anterior, y es a su vez complementario a este. Si bien antes se ha asumido posible acotar inferiormente por una constante el "progreso" del algoritmo en cada iteración, a menudo este se reduce progresivamente conforme este se acerca al valor óptimo. El teorema 2.8 permite acotar T bajo la circunstancia de conocer una función monótona creciente en $s \in \mathcal{S}$ (por tanto, menor cuanto más se encuentre uno del óptimo) que acote el *drift* inferiormente.

Teorema 2.8: (Variable Drift Theorem [Joh10], teorema 4.6)

Sea $\{X_k\}_{k \in \mathbb{N}}$ una sucesión de variables aleatorias que toman valores en $S \subseteq \mathbb{R}_0^+$, con $0 \in \mathcal{S}$ y $|\mathcal{S}| < \infty$. Sean $T := \inf\{k \in \mathbb{N} | X_k = 0\}$ y $s_{min} := \min(\mathcal{S} \setminus \{0\})$. Si existe una función creciente $h : \mathbb{R}^+ \to \mathbb{R}^+$ tal que, $\forall k \in \mathbb{N}$ y $\forall s \in \mathcal{S} \setminus \{0\}$ donde $\Delta_k(s)$ esté bien definida, se cumple que

$$\Delta_k(s) \geq h(s)$$

entonces

$$\mathbb{E}[T|X_0] \leq \frac{s_{min}}{h(s_{min})} + \int_{s_{min}}^{X_0} \frac{1}{h(\sigma)}\, d\sigma$$

Un caso particular muy importante del teorema anterior es cuando $h(s) = \delta s$, con $\delta > 0$ constante. El resultado siguiente, conocido como *Multiplicative Drift Theorem*, aborda este caso dando cotas más ajustadas que el caso general.

Teorema 2.9: (Multiplicative Drift Theorem [DJW12][Len18])

Sea $\{X_k\}_{k \in \mathbb{N}}$ una sucesión de variables aleatorias que toman valores en $S \subseteq \mathbb{R}_0^+$, con $0 \in \mathcal{S}$ y $|\mathcal{S}| < \infty$. Sean $T := \inf\{k \in \mathbb{N} | X_k = 0\}$ y $s_{min} := \min(\mathcal{S} \setminus \{0\})$. Si existe $\delta > 0$ tal que $\forall k \in \mathbb{N}$ y para todo $s \in \mathcal{S} \setminus \{0\}$ donde $\Delta_k(s)$ esté bien definida, se cumple que

$$\Delta_k(s) \geq \delta s \quad \text{(condición de multiplicatividad)}$$

entonces

$$\mathbb{E}[T] \leq \frac{1 + \mathbb{E}[\ln(X_0/s_{min})]}{\delta}.$$

Además, $\forall r \geq 0$,

$$P\Big[T > \Big\lceil \frac{r + \ln(X_0/s_{min})}{\delta} \Big\rceil\Big] \leq e^{-r}.$$

En [Len18] pueden encontrarse ejemplos de aplicación, entre otros, de los tres teoremas anteriores a múltiples problemas clásicos junto con una guía introductoria muy completa al *drift analysis*.

Estas herramientas son útiles para analizar todo tipo de algoritmos estocásticos de optimización iterativa. Jun He y Xin Yao aplicaron un análisis de este tipo para analizar algoritmos evolutivos [HY01][HY04][HCY15][HY16]. Aunque algunos de sus trabajos pueden considerarse precursores en el campo, a menudo asumen como potenciales las distancias en el dominio, o tienden a definir otros potenciales de forma artificiosa sin tratar de aportar una intuición sobre su elaboración que pueda ser de utilidad a otros investigadores en tareas similares. Los autores Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev y Per Kristian Lehre desarrollan un teorema llamado *Level-Based Theorem* [CDEL18] que permite aplicar automáticamente las herramientas del *drift analysis* a algoritmos evolutivos para un cierto tipo genérico de problemas. Se concluirá el capítulo enunciando y comentando dicho resultado.

Teorema 2.10: (*Level-Based Theorem* [CDEL18]) Sea $\lambda \in \mathbb{N}$. Dada una partición $(A_1, ..., A_n)$ de Ω, se define $P_k \in \Omega^\lambda$ como la población de nuestro algoritmo evolutivo en la generación k, y $T := \min\{k\lambda : |P_k \cap A_n| > 0\}$. Si existen $z_1, ..., z_{n-1}, \delta \in (0, 1]$, y $\gamma_0 \in (0, 1)$ tales que, para cualquier población $P_k \in \Omega^\lambda$,

1. para cada nivel $j \in \{1, ..., n-1\}$, si $|P_k \cap \Big(\cup_{i=j}^n A_i\Big)| \geq \gamma_0\lambda$, entonces para todo y siguiendo la misma distribución que las soluciones de la población P_k

$$P[y \in \cup_{i=j+1}^n A_i] \geq z_j$$

2. para cada nivel $j \in \{1, ..., n-2\}$, y cada $\gamma \in (0, \gamma_0]$,
si $|P_k \cap \Big(\cup_{i=j}^n A_i\Big)| \geq \gamma_0\lambda$ y
$|P_k \cap \Big(\cup_{i=j+1}^n A_i\Big)| \geq \gamma\lambda$, entonces

$$P[y \in \cup_{i=j+1}^n A_i] \geq (1+\delta)\gamma$$

3. y el tamaño de la población $\lambda \in \mathbb{N}$ satisface

$$\lambda \geq \Big(\frac{4}{\gamma_0\delta^2}\Big) \ln\Big(\frac{128n}{z_*\delta^2}\Big), \text{ donde } z_* := \min_{j \in \{1,...,n-1\}} \{z_j\}$$

entonces

$$\mathbb{E}[T] \leq \Big(\frac{8}{\delta^2}\Big) \sum_{j=1}^{n-1} \Big(\lambda \ln\Big(\frac{6\delta\lambda}{4 + z_j\delta\lambda}\Big) + \frac{1}{z_j}\Big).$$

El teorema anterior asume que es posible dividir el dominio de nuestro problema en n

secciones o niveles de forma que, cuanto mayor sea el índice de la sección, más cerca se encuentra la solución óptima, y tal que se cumplen las condiciones:

- Dada una cantidad de candidatos suficientemente alta (dependiente del tamaño de población λ) concentrados en el nivel j (o superior), la probabilidad de escoger para la siguiente generación un individuo en niveles superiores está por encima de cierto umbral.
- El tamaño de la población está también por encima de un umbral determinado, que es a su vez dependiente del conjunto de umbrales para la probabilidad de pertenencia a los distintos niveles.

Se ve, por tanto, que las cláusulas impuestas requieren encontrar un tamaño de población que permita un equilibrio entre las tres condiciones. Esto tiene mucho sentido, si se considera que el propio teorema define el *first hitting time* como la generación en la que se alcanza el óptimo multiplicada por el tamaño de la población, lo que suele corresponder más concretamente con el número de ejecuciones de la función objetivo. Tomar un λ muy grande no nos permitiría, por tanto, minimizar en este sentido el tiempo de parada.

Para conocer algunas de las aplicaciones que este teorema ha permitido, desde su primera publicación en 2014, se recomienda leer la sección 4.4 de [Len18].

3 Problemas en la optimización de caja negra

Este capítulo irá dirigido a hablar de los principales obstáculos encontrados al resolver problemas de optimización de caja negra, esto es, en los que la única información de la función a optimizar es su evaluación en puntos específicos del espacio de soluciones y que, por tanto, se basan en modificar las variables de decisión de una forma preestablecida que ayude a acercarse al óptimo lo más rápido posible.

La bibliografía referenciada y los resultados mostrados en este apartado ayudarán, en general, a entender aspectos fundamentales de la investigación en metaheurísticas de los últimos 25 años y, en particular, a comprender la motivación del diseño de los algoritmos coevolutivos.

3.1. Algunos problemas comunes en el diseño de EAs

Se empezará esta sección explicando algunos de los problemas principales que pueden encontrarse al tratar de buscar óptimos en funciones arbitrarias: convergencia prematura, no uniforme y dominada; causalidad débil (o *ruggedness*), neutralidad, *epistasis*, ruido y dimensionalidad del espacio de decisión, de la que se hablará en más profundidad en la sección siguiente. Nuestra principal referencia para este apartado será [WCT12].

Las dificultades anteriores se encuentran, individualmente o en conjunto, en todos los algoritmos de optimización de caja negra. Se destaca, por tanto, que las cuestiones tratadas en esta sección no se circunscriben únicamente al algoritmo evolutivo genérico tratado en el capítulo anterior. Debido a la gran variedad de métodos dentro de la clase de los algoritmos evolutivos, todas ellas se presentan en estos, de un modo u otro, y es habitual que se diseñen medidas y operadores específicos dentro de cada algoritmo para paliar sus efectos. Se verá que será necesario encontrar un equilibrio entre convergencia y exploración de nuevas soluciones, y se deberá tener en cuenta la forma de la función objetivo a la hora de elegir un algoritmo apropiado para optimizarla.

Claramente, muchos de los problemas derivados de la forma de la función están relacionados con la elección del genotipo G que se definen en el capítulo 1, al ser la forma en la que se representa el espacio de soluciones y del que depende su estructura.

También cabe destacar que aquellas cualidades que hacen a muchos problemas tradicionalmente "difíciles" no tienen que ser las mismas para los algoritmos evolutivos. Muchos de los problemas comentados en la sección 1.1 cuyas soluciones se aproximan a menudo utilizando *EAs*, pertenecen a la categoría *NP-hard*. En [OHY09] se demuestra que para algunas subclases del problema de la *cobertura de vértices mínima* (tipo *NP-hard*), el algoritmo $(1+1)EA$, la

versión más sencilla de algoritmo evolutivo, alcanza soluciones de forma eficiente e incluso en tiempo polinomial.

Se limitará el desarrollo a problemas de optimización mono-objetivo. Se empezará comentando aquellas dificultades que el proceso evolutivo pueda afrontar de cara a una **convergencia correcta** y en **tiempo razonable**.

En un primer lugar, se tiene el caso de la **convergencia prematura**. Esta se da cuando nuestro algoritmo se detiene antes de lo que sería conveniente tras alcanzar un óptimo local distinto al óptimo global y con un valor objetivo "mucho peor". La causa puede ser centrar la búsqueda de candidatos en torno a la vecindad más cercana a las soluciones actuales o, en el caso de los algoritmos basados en poblaciones, una tendencia demasiado exagerada a utilizar y conservar únicamente las mejores soluciones. En resumen, la convergencia prematura se debe a una *exploración* insuficiente.

En términos de la forma de la función objetivo, los principales elementos que pueden contribuir a una convergencia prematura o errónea son la **"rugosidad"** (*ruggedness*) y el **"engaño"** (*deceptiveness*).

La rugosidad consiste en una alta tasa de cambio de la función, lo que habitualmente desemboca en múltiples máximos y mínimos. Una búsqueda local convergerá rápidamente al óptimo más cercano. Factores como una alta presencia de no linealidades en la naturaleza del objeto medido (a veces conocida como *causalidad débil*) o el ruido en la evaluación de la función pueden originar rugosidad. La figura 3.1 muestra un ejemplo de función rugosa.

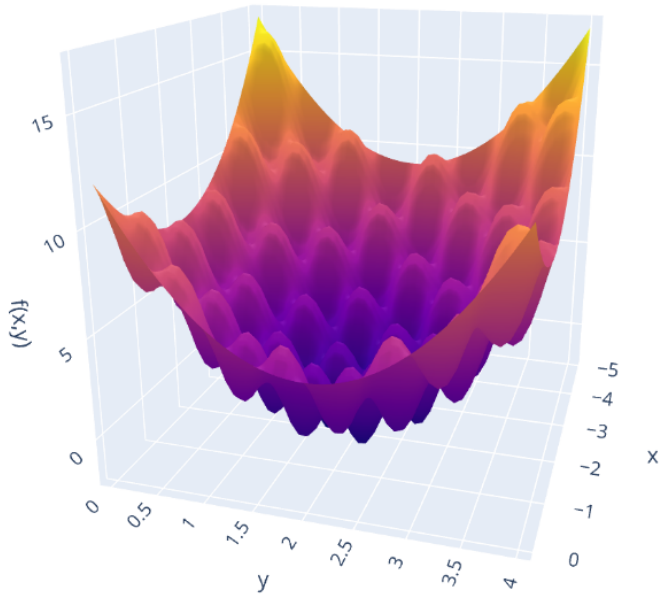


Figura 3.1: Muestra la gráfica de la función $f(x,y) = (x+2)^2 + 2(y-2)^2 + 2\sin(2\pi x)\sin(2\pi y)$ como ejemplo de función rugosa. De elaboración propia.

Para combatir el ruido, es posible utilizar toda una clase de algoritmos evolutivos denomi-

nados como "Algoritmos de Estimación de la Distribución" (*EDAs*), en los que la población actual se utiliza para generar una distribución de probabilidad sobre la que se muestrea la nueva generación de soluciones [QM04] (esto es, se intenta recrear la función de densidad p_k que se ha visto en el capítulo 2). También es posible aproximar la función objetivo por otra más sencilla y suave que procure no alterar la localización de las soluciones óptimas o, en presencia de ruido aleatorio en las mediciones, evaluar varias veces la función objetivo en cada candidato y tratarla como una distribución de probabilidad [WHB98]. Cualquiera de los métodos anteriores puede alternarse con otros algoritmos, como la evolución diferencial, que favorezcan la explotación sobre la auténtica función objetivo.

Una causalidad débil en la función objetivo puede deberse a una representación poco adecuada del espacio de soluciones. Pueden consultarse en [CWM12] casos prácticos de diseño de múltiples metaheurísticas y ejemplos de codificación de problemas reales y de sus respectivos dominios.

Por otro lado, una función a optimizar es **engañosa** cuando la explotación de regiones con un valor objetivo aparentemente bueno puede alejarnos del óptimo global. Con "aparentemente" es común referirse a que, en un entorno con una extensión razonable de una solución dada, la dirección de avance de las soluciones tiende a mejorar su valor objetivo.

Una función engañosa no tiene por qué ser necesariamente rugosa. Asúmase, por ejemplo, que se quiere minimizar el laplaciano de la función gaussiana, cuya expresión es

$$LoG(x,y) = -\frac{1}{\pi\sigma}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

La gráfica de esta función puede observarse en la figura 3.2. Desde una perspectiva global, se observa un mínimo claro en el centro de la gráfica. En un entorno de dicho mínimo la función tiende a crecer y, cuando nos alejamos lo suficiente, de nuevo a decrecer. Un método de búsqueda local que parta de un punto suficientemente alejado del óptimo tenderá a alejarse de este porque, localmente hablando, acercarse hará crecer la función objetivo.

En estos casos, encontrar las regiones cercanas a los valores óptimos es cuestión de azar. En los algoritmos evolutivos, una baja presión selectiva y reiniciar la población, guardando las mejores soluciones hasta el momento, son mecanismos que favorecen la exploración. Algunos algoritmos también incluyen métodos de control de la densidad y la diversidad de las soluciones (*niching*), lo que puede prevenir la convergencia prematura y ayudar a la exploración de soluciones alejadas de la población actual. Al igual que con las funciones rugosas, encontrar una representación más adecuada de las soluciones podría cambiar la forma de la función objetivo de una forma provechosa [SY00].

Otro mecanismo de convergencia subóptima es la **convergencia dominada**, esto es, cuando una variable o conjunto de variables tiene un mayor efecto sobre la función objetivo que el resto, de forma que las demás variables del dominio son ignoradas por el proceso de optimización. Identificar las coordenadas con un menor efecto relativo sobre la función objetivo, ya sea analítica o empíricamente, y aplicar operadores que favorezcan la exploración sobre esas variables, puede ayudar a paliar este efecto.

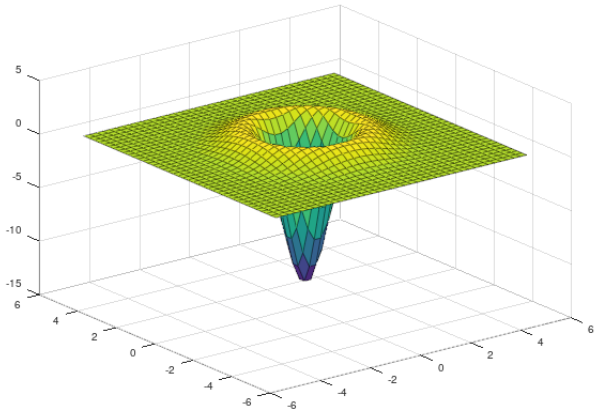


Figura 3.2: Gráfica del laplaciano de la función de densidad gaussiana bidimensional. De elaboración propia.

Un fenómeno similar al engaño es la **neutralidad**. Esta consiste en que existan regiones del dominio en las que la función objetivo no varíe. Una búsqueda local en dichas regiones carece de información útil para elegir entre diferentes trayectorias y acabará estancada o realizando un camino aleatorio. Algunos de los problemas más difíciles de resolver son de la forma *needle-in-a-haystack*, esto es, aquellos en los que la función es mayoritariamente neutral, salvo por una región relativamente pequeña con soluciones muy buenas.

Un concepto algo más abstracto y que conecta, y en cierto sentido provoca, los problemas vistos anteriormente es que exista una compleja interrelación entre las distintas variables o genes y cómo esta se manifieste en la función objetivo. De ahí que se haya destacado la representación de nuestro espacio de soluciones como algo a tener en cuenta para paliar varios de estos problemas. En la literatura sobre algoritmos evolutivos, y metaheurísticas en general, esta dependencia entre variables se conoce como **epistasis**. Ya en el capítulo 2 se menciona el término epistasis en un sentido parecido aunque se destaca que, entonces, la dependencia entre variables referida se presentaba en la distribución de la población. Si bien en un algoritmo poblacional es deseable que la distribución de la población se adapte a las dependencias entre las variables de la función objetivo, ambas definiciones solo se aproximan en un sentido intuitivo. En la figura 3.3 se ve cómo la epistasis se relaciona con el resto de problemas vistos anteriormente.

Aparte de los métodos comentados para las funciones rugosas y engañosas, la epistasis puede ser analizada en general mediante técnicas de **aprendizaje de conexiones** (o *linkage learning*). Consisten en cuantificar de alguna forma la interacción entre distintas variables y/o agruparlas en función de su nivel de dependencia. Algunos ejemplos de algoritmos que aplican este tipo de conexiones son el *Bayesian Optimization Algorithm* [PGCP99] (y sus variaciones), el *messy Genetic Algorithm* [GKD89], el uso de matrices de estructura de dependencias (*DSM*) [YGS+09] o la técnica de *Variable Interaction Learning (VIL)* [CWYT10]. En [pClYS+07] se encuentra una discusión cualitativa sobre diferentes formas de abordar el *linkage learning*. Este tema será de vital importancia en el uso de los *CCEAs* y se retomará en los capítulos 4 y 5.

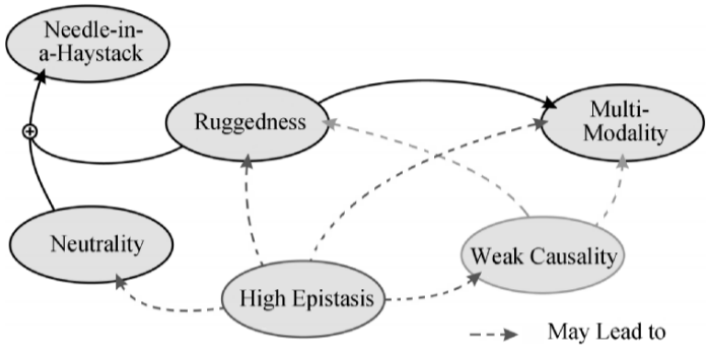


Figura 3.3: Extraída de [WCT12] (Thomas Weise, Raymond Chiong y Ke Tang, *Evolutionary optimization: Pitfalls and booby traps. Journal of Computer Science and Technology)*.

A continuación, se tratará la cuestión de la **dimensionalidad**, esto es, el número de variables de la función objetivo. Algunos autores prefieren usar el término escalabilidad, reservando la palabra dimensionalidad para el número de funciones objetivo que se optimizan simultáneamente. En el contexto de la optimización mono-objetivo, la dimensión del espacio de soluciones es el principal factor que determina el tamaño del problema de optimización y el tiempo medio necesario para alcanzar soluciones adecuadas.

Al ser de gran importancia para la motivación de los algoritmos coevolutivos cooperativos, se le dedicará íntegramente la siguiente sección.

3.2. Maldición de la dimensionalidad

En esta sección se hablará sobre la maldición de la dimensionalidad [CNByM01] [KM17][K¨], la cual describe una serie de dificultades que surgen al aplicar métodos computacionales para resolver problemas con un número de parámetros muy alto, o aplicar análisis estadístico de datos con muchas dimensiones. Estos problemas suelen derivarse del aumento exponencial con respecto al número de variables de la cantidad de puntos necesaria para tomar una muestra uniforme en todo el espacio, asumiendo una distancia entre nodos adyacentes fija.

Supóngase que se desea optimizar una función f definida en el intervalo $[0, 1]$. Para poder tener cierta confianza en que un método de optimización estocástica cualquiera devuelva una solución razonablemente buena, este deberá haber explorado, en mayor o menor profundidad dependiendo de la complejidad de la función, todo el dominio. Cabe entonces preguntarse cuánto puede tardar una búsqueda que explore adecuadamente dicho dominio, que se aproxima por un muestreo uniforme en el intervalo.

Asúmase que se elige un número k de puntos equiespaciados en el intervalo $[0, 1]$. Si nuestra función f tuviese dos parámetros a optimizar, tomando ambos valores en $[0, 1]$, en este caso se necesita comprobar cada valor del primer parámetro con cada valor del segundo. Considerando k valores para cada uno de ellos, se observa que una muestra uniforme análoga al caso unidimensional requiere de $k \cdot k = k^2$ evaluaciones de la función objetivo. No es difícil

ver que esta tendencia puede generalizarse como

$$\text{Tamaño de la muestra } = k^m$$

donde m es el número de variables, o la dimensión del espacio de soluciones.

El aumento exponencial del tamaño de la muestra no es el único inconveniente. Considérese un hipercubo de lado l, y una esfera concéntrica al mismo contenida dentro del hipercubo que tiene, por tanto, un radio de $\frac{l}{2}$. El cociente entre los volúmenes contenidos dentro de la esfera y dentro del cubo es [oST]:

$$\frac{V_S}{V_C} = \frac{\pi^{\frac{m}{2}} l^m}{\Gamma\left(\frac{1}{2}m+1\right)\cdot l^m} = \frac{\pi^{\frac{m}{2}}}{\Gamma\left(\frac{1}{2}m+1\right)} \xrightarrow{m\to\infty} 0$$

Cualitativamente, el cálculo anterior nos indica que, cuando el número de dimensiones m es grande, los vértices del hipercubo son malos representantes de los puntos cercanos a su centro. Esto quiere decir que al aumentar el número de variables la cantidad de puntos necesaria para muestrear uniformemente el dominio no solo crece exponencialmente, sino que para conservar la calidad de la representación hay que mantener la densidad de puntos por volumen a un ratio de $\frac{\Gamma\left(\frac{1}{2}m+1\right)}{\pi^{\frac{m}{2}}}$. En [K¨] (sección 4) es posible encontrar un desarrollo extendido sobre esta misma idea.

La discusión anterior es análoga, salvo isomorfismos, para cualquier dominio parametrizable a partir de un hipercubo en $\mathbb{R}^m$.

Existen métodos para combatir la maldición de la dimensionalidad, como seleccionar un subconjunto de las características a optimizar (aquellas con un mayor impacto en la función objetivo) o empezar la búsqueda aplicando un cambio de variable a un genotipo de menor dimensión que el dominio real, o aproximando la función f por otra más sencilla (por ejemplo, usando una red neuronal de pequeño tamaño [DWT12]). Para casos específicos también es posible encontrar formas de muestrar los puntos en el dominio de manera conveniente, como puede verse en [KS05] para integración numérica. En los capítulos 4 y 5 se aplicará un enfoque distinto para mitigar este problema, basado en explotar la separabilidad de la función y en la paralelización del cómputo.

La tendencia general siempre será, sin embargo, que nuestros algoritmos empeoren o se ralenticen notablemente al aumentar el número de variables, como se deducirá a raíz del resultado expuesto en el siguiente apartado.

3.3. No hay barra libre

En 1995 David H. Wolpert and William G. Macready publicaron un artículo llamado *No Free Lunch Theorems for Search*, y una revisión de este en 1997 [WM97], en los que demostraron los primeros teoremas de *No Free Lunch* para optimización, o "No Hay Barra Libre" como suele traducirse en español. Estos teoremas afirman que el resultado de dos algoritmos de

optimización de caja negra cualesquiera, bajo una serie de hipótesis comunes a la mayor parte de algoritmos de búsqueda implementables en una computadora y que abarcan a los algoritmos evolutivos, promediado sobre todas las posibles funciones objetivo, es el mismo.

Se concluirá usando este resultado para justificar los efectos adversos de la maldición de la dimensionalidad sobre los algoritmos evolutivos, y revalorando las afirmaciones finales realizadas en la primera sección de este capítulo sobre la necesidad de tener información *a priori* del "paisaje"de la función objetivo antes de decidir aplicar un algoritmo u otro para su optimización.

Se comenzará definiendo los términos necesarios para enunciar los teoremas de *NFL*. Sean $\mathcal{X}$ e $\mathcal{Y}$ dos conjuntos finitos, y $\mathcal{F}_{\mathcal{X}}^{\mathcal{Y}} = \mathcal{Y}^{\mathcal{X}}$ es el conjunto de funciones de la forma $f\colon \mathcal{X} \to \mathcal{Y}$. Sobre $\mathcal{Y}$ es necesaria la imposición adicional de que este conjunto esté totalmente ordenado, puesto que será necesario para comparar las diferentes soluciones. Habitualmente, $\mathcal{Y}$ es un subconjunto discreto de los números reales.

El hecho de definir $\mathcal{X}$ e $\mathcal{Y}$ como conjuntos finitos no supone, en el contexto de la optimización mediante computadores, un inconveniente a la hora de aplicar los teoremas *NFL* a problemas con dominios continuos. El motivo es que no es posible representar valores continuos (con precisión infinita) en un ordenador, ya sea por limitaciones de tiempo o de memoria disponible. Por la misma razón el dominio de búsqueda de soluciones suele estar acotado, por lo que $\mathcal{X}$ e $\mathcal{Y}$ serán necesariamente finitos.

Se notará al conjunto de valores distintos de $\mathcal{X}$ explorados hasta el momento como $d_m^x = (d_m^x(1), ..., d_m^x(m)) \in \mathcal{X}^m$, listados en el orden en que se han explorado por primera vez. Análogamente, se puede definir el historial de sus valores objetivo como un vector de la forma $d_m^y = (f(d_m^x(1)), ..., f(d_m^x(m))) = (d_m^y(1), ..., d_m^y(m)) \in \mathcal{Y}^m$. Se medirá la bondad de dicho historial mediante una función, que se notará como Φ. Para un problema de minimización, se puede definir $\Phi(d_m^y) = \min(d_m^y(1), ..., d_m^y(m))$.

Para esta discusión, a partir de las definiciones anteriores, Wolpert y Macready definen **algoritmo** como cualquier función $a : \cup_{m\in\mathbb{N}}(\mathcal{X}^m \times \mathcal{Y}^m) \to \mathcal{X}$ tal que $a(d_m^x, d_m^y)$ no es ningún elemento de d_m^x. En otras palabras, un algoritmo es cualquier función que, dado el historial de puntos visitados y de sus valores objetivo, indica cuál es el siguiente punto a visitar.

Se procede a enunciar el primer teorema de *No Free Lunch*.

Teorema 3.1: [WM97]

Para cualquier par de algoritmos a_1 y a_2,

$$\sum_{f\in\mathcal{F}_{\mathcal{X}}^{\mathcal{Y}}} P(d_m^y|f, m, a_1) = \sum_{f\in\mathcal{F}_{\mathcal{X}}^{\mathcal{Y}}} P(d_m^y|f, m, a_2)$$

donde $P(d_m^y|f, m, a)$ representa la probabilidad condicionada de contar con el historial de valores objetivo d_m^y después de m iteraciones de aplicar el algoritmo a para optimizar la

función f. Además, como corolario,

$$\sum_{f \in \mathcal{F}_{\mathcal{X}}^{\mathcal{Y}}} P(\Phi(d_m^y)|f, m, a_1) = \sum_{f \in \mathcal{F}_{\mathcal{X}}^{\mathcal{Y}}} P(\Phi(d_m^y)|f, m, a_2).$$

La interpretación del resultado anterior es bastante clara. Dado el mismo número de iteraciones, dos algoritmos cualesquiera obtendrán en promedio, considerando todas las posibles funciones con dominio $\mathcal{X}$ y codominio $\mathcal{Y}$, los mismos resultados. Esta misma idea puede reformularse de la siguiente forma: cuanto mejor sea un algoritmo para resolver un cierto conjunto de funciones, peores serán sus resultados en los posibles problemas restantes.

El caso anterior puede adaptarse a funciones que dependen también del tiempo. Esto es, en cada iteración la función f puede cambiar, de forma que se cuenta, en vez de con una única función f, con una sucesión de funciones $\{f_i\}_{i \in \mathbb{N}} \subset \mathcal{F}_{\mathcal{X}}^{\mathcal{Y}}$. La evolución en el tiempo de f_i puede modelarse como una función $T_m \colon \mathcal{F}_{\mathcal{X}}^{\mathcal{Y}} \times \{1, ..., m\} \to \mathcal{F}_{\mathcal{X}}^{\mathcal{Y}}$, de forma que para $i \leq m$ se puede calcular $f_{i+1} = T_m(f_i, i)$. Cabe destacar que, dado un $m \in \mathbb{N}$ fijo, el número de posibles funciones T_m de la forma dada es finito, y se denotará como $\mathcal{T}_m$.

En este caso, pueden considerarse razonablemente al menos dos formas de evaluar el resultado obtenido por el algoritmo: a partir de $d_m^y = (f_1(d_m^x(1)), ..., f_m(d_m^x(m)))$, o a partir de $D_m^y = (f_m(d_m^x(1)), ..., f_m(d_m^x(m)))$.

Si bien no se le dedicará más atención en lo que resta de proyecto, se enunciará el teorema de *NFL* para funciones dependientes del tiempo.

Teorema 3.2: [WM97]

Para cualquier par de algoritmos a_1 y a_2, dada una función inicial f_1 y un número de iteraciones fijo m:

$$\sum_{T \in \mathcal{T}_m} P(d_m^y|f_1, T, m, a_1) = \sum_{T \in \mathcal{T}_m} P(d_m^y|f_1, T, m, a_2)$$

y

$$\sum_{T \in \mathcal{T}_m} P(D_m^y|f_1, T, m, a_1) = \sum_{T \in \mathcal{T}_m} P(D_m^y|f_1, T, m, a_2)$$

En este caso se obtiene que el resultado promedio de nuestra búsqueda, considerando todas las posibles formas en las que puede evolucionar la función f a lo largo de m iteraciones, es independiente del algoritmo utilizado.

Gracias al teorema 3.1 puede argumentarse por qué, como se ha afirmado en la sección anterior, todo algoritmo de optimización tiende a necesitar un aumento del número de soluciones evaluadas exponencial con respecto al incremento de la dimensión del problema. **En promedio**, todo algoritmo obtendrá **los mismos resultados** que se alcanzarían muestreando el dominio uniformemente y eligiendo la mejor solución encontrada, que es exactamente el método discutido la sección 3.2 sobre la maldición de la dimensionalidad.

De esta forma, se llega a la principal lección que enseñan los teoremas *NFL* y que ha influenciado significativamente la investigación en metaheurísticas desde su publicación en 1995: no existen algoritmos de optimización mejores que otros. El principal interés al analizar una metaheurística es conocer para qué clase de problemas esta es especialmente útil, o bien identificar aquellos algoritmos que obtienen mejores resultados para un problema o clase de problemas particular.

3.4. Funciones más fáciles y más difíciles

A raíz de los teoremas de *No Free Lunch*, los estudios teóricos en optimización de caja negra tendieron a centrarse en analizar el comportamiento de un algoritmo o conjunto de algoritmos sobre una clase de problemas concreta.

En esta sección se expondrán los resultados principales de [HCY15], donde se da una definición de funciones objetivo *más fáciles* y *más difíciles* dentro de una clase de funciones, y se demuestra formalmente mediante *drift analysis* cuales son estas funciones para el $(1+1)EA$ elitista, un algoritmo evolutivo muy simplificado y ampliamente usado en el estudio teórico de los *EAs*.

Se intentará justificar la utilidad práctica de trabajos como el anterior y del uso de algoritmos simplificados como el $(1+1)EA$, a los que también se recurrirá en la sección 4.3 para intentar comprender algunas propiedades formales de los algoritmos coevolutivos.

Definición 3.3: dado un algoritmo evolutivo y una clase de funciones $\mathcal{F}$ donde todas las funciones alcanzan el óptimo en el mismo punto, se dice que una función $f \in \mathcal{F}$ es **la más fácil** para el *EA* establecido si, dado cualquier punto inicial, el número de generaciones necesarias para converger a la solución óptima es menor o igual que para cualquier otra función $g \in \mathcal{F}$ partiendo del mismo punto inicial. Análogamente, es **la más difícil** si el número de generaciones necesarias para converger al óptimo es mayor o igual que para cualquier otra función de $\mathcal{F}$ para todo punto inicial.

En esta sección se estudiará también un problema de búsqueda del máximo de una función $f\colon S \to \mathbb{R}$, donde S es un conjunto finito. Por tanto, f toma un número finito de valores, $f_0 > f_1 > ... > f_L$. En el caso de la maximización, se define el conjunto de valores donde se alcanza el óptimo como $S_{opt} = \{x \in S\colon f(x) = f_0\}$, y $S_{non} = S \setminus S_{opt}$. En general, se definen los **conjuntos de nivel** como

$$S_l := \{x \in S\colon f(x) = f_l\}, \quad l = 0, 1, ..., L$$

Ahora se procederá a explicar el algoritmo $(1+1)EA$ elitista. La notación usada corresponde a $(\mu+\lambda)EA$, que nota un algoritmo evolutivo en el que se usan μ soluciones para generar, mediante operadores de mutación y recombinación independientes del tiempo, otras λ soluciones que se añaden a la población. El adjetivo "elitista" indica que siempre se conserva el mejor individuo de la población para la siguiente generación. El algoritmo $(1+1)EA$ se describe entonces como:

Algoritmo 2: $(1+1)EA$ elitista

Input: $f \in \mathcal{F}$
Output: $x \in S$ donde f alcanza su máximo
se genera una solución inicial ϕ_0 aleatoria;
se inicia un contador de generaciones pasadas $t \leftarrow 0$;
while $f(\phi_t) < f_0$ **do**
  $\phi_{t.m} \leftarrow$ mutación de ϕ_t;
  if $f(\phi_{t.m}) > f(\phi_t)$ **then**
    $\phi_{t+1} \leftarrow \phi_{t.m}$;
  else
    $\phi_{t+1} \leftarrow \phi_t$;
  $t \leftarrow t+1$

En resumen, lo que hace el algoritmo anterior es generar una solución aleatoria y mutarla iterativamente hasta obtener otro candidato estrictamente mejor al anterior. La condición de parada es alcanzar el máximo de f, aunque en la práctica esta se sustituiría por alcanzar un umbral de la función objetivo, o un número máximo de iteraciones.

Se expondrán los elementos principales del *drift analysis* que se usan en [HCY15] para estudiar el caso anterior. Se llamará $G_f(x)$ al número esperado de ejecuciones de la función objetivo necesarias para alcanzar el óptimo, tomando x como valor inicial. $G_f(x) = \mathbb{E}[T|\phi_0 = x]$, siguiendo la notación utilizada en el capítulo 2. La sucesión de variables aleatorias $\{\phi_t\}_{t\in\mathbb{N}}$ corresponderá al estado de la población (de un único individuo) en la generación t.

Se notará la probabilidad condicionada de seleccionar una solución, dada la solución anterior, como

$$P_\phi(x,z) = P(\phi_{t+1} = z|\phi_t = x).$$

Como se ha asumido el operador de mutación independiente del tiempo, $P(x,z)$ también lo es.

Dada una función potencial d no especificada, cumpliendo que $d(x) \geq 0$ en cualquier $x \in S$ y $d(x^*) = 0$ en todo x^* óptimo de $f \in \mathcal{F}$ (recuérdese que todas las funciones en $\mathcal{F}$ tienen el mismo máximo), se puede definir la deriva o *drift* tal como lo se hizo en la sección 2.2.

$$\Delta_\phi(x) = \mathbb{E}[d(x) - d(y)] = \sum_{y\in S} P_\phi(x,y)(d(x) - d(y)).$$

Cuando no se produzca ambigüedad sobre la posible función a la que nos estemos refiriendo, se usará $G = G_f$ y $\Delta = \Delta_\phi$. Se enuncian, sin demostrar, los resultados principales en los que se basa el trabajo realizado en [HCY15].

Lema 3.4: [HCY15] Si $\Delta(x) \geq 1$ para cualquier $x \in S_{non}$, entonces $G(x) \leq d(x)$ para cualquier punto $x \in S$.

Lema 3.5: [HCY15] Si $\Delta(x) \leq 1$ para cualquier $x \in S_{non}$, entonces $G(x) \geq d(x)$ para

cualquier punto $x \in S$.

Lema 3.6: [HCY15] Tomando la función potencial $d(x) = G(x)$, se cumple que $\Delta(x) = 1$ para cualquier $x \in S_{non}$.

Lema 3.7: [HCY15] Para un algoritmo del tipo $(1+1)EA$ elitista, el tiempo de parada esperado con punto inicial x puede expresarse como:

$$G(x) = \begin{cases} 0 & x = S_0 \\ \frac{1+\sum_{k=0}^{l-1}\sum_{y\in S_k} P(x,y)G(y)}{\sum_{k=0}^{l-1}\sum_{y\in S_k} P(x,y)} & x \in S_l,\ l > 0 \end{cases}$$

Teorema 3.8: [HCY15]

Dado un algoritmo del tipo $(1+1)EA$ elitista, y una clase de funciones objetivo con los mismos valores óptimos, sea $G_f(x_0)$ el tiempo de parada esperado para optimizar la función f con valor inicial x_0.

Si para dos puntos cualesquiera $x, y \in S$ se cumple que

$$G_f(x) < G_f(y) \implies f(x) > f(y)$$

entonces f es la más fácil dentro de dicha clase de funciones.

No se mostrarán la prueba completa del teorema anterior, pero sí se esbozará el proceso seguido por sus autores. Sea $\{\phi_t\}_{t\in\mathbb{N}}$ la sucesión de soluciones seguidas por el algoritmo $(1+1)EA$ elitista para optimizar f. Dada otra función g de la misma clase de funciones, sea $\{\gamma_t\}_{t\in\mathbb{N}}$ la sucesión de soluciones para optimizar g. Tomando la función potencial $d(x) = G_f(x)$, aplicando el lema 3.6, se tiene que $\Delta_\phi(x) = 1$ para todo punto x en el que no se alcance el máximo. Aplicando la hipótesis del teorema, puede demostrarse que $\Delta_\gamma(x) \leq \Delta_\phi(x) = 1$ lo que implica, por el lema 3.5, que $G_g(x) \geq d(x) = G_f(x)$, independientemente de la función g escogida. Por propia definición, esto implica que f es la función más fácil de la clase.

Cabe destacar que el teorema anterior da una condición que depende únicamente de la función f. No era así en la definición original de "función más fácil", que dependía de comparar la relación entre G_f y G_g para cualquier g en la clase de funciones. Se observa, sin embargo, que el teorema 3.8 no es una caracterización de las funciones más fáciles. La nueva condición es suficiente pero no necesaria, puesto que en una clase de funciones con una única función esta será la más fácil independientemente de si esta se cumple o no.

Argumentamos que resulta muy razonable analizar las funciones más fáciles y más difíciles para un algoritmo dado aplicando este enfoque. Nuestra motivación para buscar este tipo de funciones viene dada por los teoremas *NFL*, y encontrar una condición suficiente como la dada por el teorema anterior garantiza que, independientemente de que otras funciones se añadan a la clase de f, esta siempre será una de las más fáciles.

La hipótesis del teorema 3.8, llamada de *decrecimiento monótono*, no establece más que el número esperado de iteraciones hasta la convergencia decrezca al crecer el valor objetivo de f. Se ve que la condición opuesta, denominada de *crecimiento monótono*, permite detectar, a

su vez, funciones "más difíciles".

Teorema 3.9: [HCY15]

Dado un algoritmo del tipo $(1+1)EA$ elitista, y una clase de funciones objetivo con los mismos valores óptimos, sea $G_f(x_0)$ el tiempo de parada esperado para optimizar la función f con valor inicial x_0.

Si para dos puntos cualesquiera $x, y \in S$ se cumple que

$$G_f(x) < G_f(y) \implies f(x) < f(y)$$

entonces f es la más difícil dentro de dicha clase de funciones.

La prueba es similar a la del teorema 3.8 y tampoco se mostrará. En su lugar, se hablará de dos ejemplos de uso de los teoremas anteriores que pueden encontrarse en [HCY15] y que ayudará a justificar el interés práctico en el *drift analysis* y en los resultados previos.

Considérese el problema de la mochila o-1. En general, el problema de la mochila es un ejemplo clásico del campo de la *investigación operativa* en el que se considera un conjunto de objetos, cada uno con un valor y con un peso o coste. Los valores y pesos totales se calculan sumando los correspondientes a los objetos "seleccionados". El objetivo final es encontrar la combinación de objetos disponibles que maximice el valor total sin superar un peso máximo.

Puede formularse matemáticamente el problema de la mochila o-1 como:

$$\begin{aligned} \text{maximizar} \quad & f(x_1, ..., x_n) = \sum_{i=1}^{n} v_i x_i \\ \text{sujeto a} \quad & \sum_{i=1}^{n} w_i x_i < C \\ & x_i \in \{0, 1\} \end{aligned} \tag{3.1}$$

donde $v_i > 0$ y $w_i > 0$ son, respectivamente, el valor y el peso del i-ésimo elemento de una lista de $n > 2$ objetos disponibles. La restricción $x_i \in \{0, 1\}$ indica que la elección de meter un elemento en la mochila es binaria, esto es, no se puede meter solamente "una parte" de dicho elemento. C corresponde a la capacidad máxima, en peso, de la mochila.

Ejemplo 1:

Este ejemplo se restringirá a los casos donde $v_1 > v_2 + ... + v_n$, $v_2 = v_3 = ... = v_n$, $w_1 = C$ y $w_2 + ... + w_n \leq C$. La solución óptima es, claramente, $x = (1, 0, ..., 0)$, esto es, cuando se elige el primer elemento (por ser más valioso que todos los demás juntos) y no se añade ninguno más, al no tener más espacio disponible.

Se aplica un algoritmo $(1+1)EA$ elitista con probabilidad de mutación sobre cada bit de la solución de $\frac{1}{n}$. Cada bit puede mutar independientemente del resto. Si una solución, generada aleatoriamente en la inicialización o por mutación, no satisface las restricciones del problema, esta es eliminada directamente.

Puede expresarse la probabilidad de mutación de una solución x en otra y como el producto de las probabilidad de invertir todos los bits en los que son diferentes y conservar aquellos en los que coinciden. Esto es:

$$P(x,y) = \left(1 - \frac{1}{n}\right)^{n-h(x,y)} \left(\frac{1}{n}\right)^{h(x,y)}$$

donde $h(x,y)$ es la *distancia Hamming* entre x e y, esto es, el número de bits en los que no coinciden. La probabilidad de transición de x a y, por tanto, será $P(x,y)$ si $f(y) > f(x)$ y 0 en otro caso. Además, asumiendo $n > 2$, se observa que $P(x,y)$ será mayor cuanto menor sea $h(x,y)$. Esto implica que $G(x)$ crecerá con la distancia Hamming de x a la solución óptima.

Se ha supuesto que $v_2 = ... = v_n$ por lo que x e y, ambas soluciones válidas no maximales, pertenecen al mismo conjunto de nivel si y solo si tienen el mismo *peso Hamming* (nº de bits encendidos), y su valor objetivo aumenta con dicho peso. Se concluye que $G(x)$ aumenta con el peso Hamming de x, igual que $f(x)$. Esto es, para todo $x, y \in S_{non}$:

$$G(x) < G(y) \iff \text{peso Hamming de } x < \text{ peso Hamming de } y \iff f(x) < f(y)$$

La equivalencia anterior implica la condición de decrecimiento monótono, así que por el teorema 3.9 se obtiene que f es de las funciones *más difíciles* para el algoritmo estudiado.

Ejemplo 2:

Ahora se considerará el caso en que $w_1 + ... + w_n \leq C$ y $v_1 = ... = v_n$, con solución óptima $x^* = (1, 1, ..., 1)$. Se aplica el mismo algoritmo de optimización que en el ejemplo anterior.

Se vuelve a tener la misma expresión de P. En este caso, G decrece al aumentar el peso Hamming (se acerca a la solución óptima) mientras que f aumenta, puesto que $v_1 = ... = v_n$. Se obtiene la equivalencia

$$G(x) < G(y) \iff \text{peso Hamming de } x > \text{ peso Hamming de } y \iff f(x) > f(y)$$

y, por el teorema 3.8, f es de las *más fáciles*.

En esta sección se ha expuesto algunas herramientas básicas para el estudio de los algoritmos de búsqueda y optimización. De nuevo, nuestra motivación es obtener información que pueda ser útil a la hora de elegir utilizar un algoritmo u otro para resolver un problema dado o, análogamente, conocer qué problemas resuelve mejor cada algoritmo.

El desarrollo elaborado en [HCY15] puede parecer demasiado sencillo y poco aplicable al campo de los *EA*, al centrarse principalmente en el algoritmo $(1+1)EA$, que no deja de ser una búsqueda local con un único individuo. Sin embargo, este tipo de trabajos sienta las bases para otros posteriores y proporciona herramientas para una mejor comprensión formal de los algoritmos de búsqueda. Además, muestra ejemplos de como los resultados teóricos del *drift analysis* pueden aplicarse a problemas reales.

4 Algoritmos Coevolutivos Cooperativos

En este capítulo se expondrán los elementos principales concernientes al diseño de un algoritmo coevolutivo cooperativo, pasando por los casos en los que puede ser adecuado su uso y las dificultades con las que nos podríamos encontrar.

4.1. Motivación para los algoritmos coevolutivos cooperativos

En esta sección se definirá lo que es un algoritmo coevolutivo y, en particular, un algoritmo coevolutivo cooperativo (*CCEA*) [PDJ94a][SS04]. Además, se darán razones para su interés.

En general, se dice que un algoritmo evolutivo es **coevolutivo** cuando el valor objetivo de cada candidato se mide de forma subjetiva, esto es, de forma dependiente al resto de candidatos. A menudo la población se subdivide en grupos, que pueden interactuar entre sí en un sentido **competitivo** o **cooperativo**.

Los *EA* coevolutivos competitivos encuentran sentido principalmente cuando el problema puede modelarse a partir de la interacción de múltiples agentes, como es habitual en las aplicaciones prácticas de la teoría de juegos.

Es posible tener varias poblaciones de soluciones evolucionando independientemente para resolver la misma tarea lo mejor posible, generalmente enfrentados en un contexto de suma 0 como, por ejemplo, entrenando dos redes neuronales para ganarse una a la otra al ajedrez. También es factible el caso en que se desea resolver un problema genérico, pero no se es capaz de representarlo al completo de forma abstracta en forma de una función objetivo. En esta situación, se puede generar un conjunto de instancias de dicho problema, cuyas soluciones sí se puede evaluar, y hacer evolucionar una población de soluciones a partir de cómo estas resuelven las instancias concretas. El conjunto de problemas particulares, a su vez, puede modificarse iterativamente con otro método evolutivo que valore favorablemente aquellas formas del problema que tiendan a "ser más difíciles", esto es, para los que haya un mayor número de soluciones que tiendan a obtener peores resultados. Muchos problemas en Aprendizaje Automático tienden a afrontarse de una manera similar, aunque la población de "problemas específicos" suele ser un conjunto de datos obtenido o generado previamente.

Ambos enfoques han sido estudiados teóricamente en profundidad desde los años 90 [AP93][JP96][RB97][PB98][PL02][Cas15], y han encontrado aplicaciones en múltiples problemas de optimización dentro y fuera de la teoría de juegos [NCFL13][WCH14]. Recientemente, el mismo concepto de optimización iterativa por competición entre dos agentes ha sido la base de importantes avances en el área del Deep Learning tanto en el enfoque simétrico (AlphaZero [SHS+17]) como en el asimétrico (*GANs* y síntesis de imágenes [GPAM+14]),

aunque este tipo de redes profundas no suelen entrenarse mediante algoritmos evolutivos.

Por otro lado, los **algoritmos coevolutivos cooperativos** (*CCEA*, por sus siglas en inglés) son aquellos en los que se hacen evolucionar varias poblaciones de soluciones en paralelo, de forma que cada población evoluciona cambiando un cierto subconjunto de las variables y dejando fijas todas las demás. Fueron descritos por primera vez en 1994 por Mitchell Potter y Kenneth De Jong [PDJ94b]. En 2004, R. Subbu y A.C. Sanderson expusieron un modelo matemático general para este tipo de algoritmos e hicieron un análisis simplificado sobre su convergencia [SS04].

La principal motivación para la invención de los *CCEA* es la reducción de la complejidad computacional de los *EA* tradicionales. Recuérdese que el orden de complejidad de la búsqueda de soluciones a un problema es exponencial con respecto al número de variables (sección 3.2). Se asumen que el espacio de soluciones se divide en l subespacios, cada uno con n dimensiones tal que $l \cdot n = m$. El coste computacional de muestrear uniformemente, con k posibles valores para cada variable, los l subespacios es el siguiente:

$$\text{Tamaño de la muestra } = l \cdot k^n$$

donde se aplica el mismo razonamiento realizado en la sección 3.2, tan solo que repetido l veces para problemas con un dominio de n dimensiones. Llevando al extremo el procedimiento anterior, se llega al algoritmo propuesto por Potter y De Jong en el que se optimiza una variable cada vez, pasando a una complejidad de orden $m \cdot k$. Sorprendentemente, se ha pasado de un algoritmo de orden exponencial a uno lineal.

Ahora cabría preguntarse cuál es el interés de resolver problemas con un elevado número de variables. Si, en la práctica, todos los problemas de interés contasen con un bajo número de grados de libertad, la motivación para paliar la maldición de la dimensionalidad sería únicamente teórica.

Muchos problemas en ciencias, ingeniería y economía pueden contar con un número de variables muy alto, del orden de los miles o decenas de miles [FKP19][CZY+18]. Muchos problemas en *Machine Learning* se reducen a uno de optimización con, al menos, tantos parámetros como datos o como atributos. En el caso de las redes neuronales, el número de parámetros a optimizar puede moverse entre los cientos y (a día de hoy) los miles de millones (véase el artículo de GPT3 [BMR+20], sección 2.1).

La llamada *Large Scale Optimization* es un campo ampliamente estudiado hoy en día y de un extendido interés práctico.

4.2. Elementos del diseño de un CCEA

Esta sección estará destinada a describir los elementos fundamentales que hay que tener en cuenta al adaptar un *EA* al marco de la coevolución cooperativa. Su desarrollo se basará principalmente en la exposición hecha en [MLZ+19].

4.2.1. Descomposición del problema

Dentro del marco de los *CCEAs*, se busca explotar la separabilidad del problema a resolver para reducir su complejidad. Por tanto, agrupar las variables de una forma que represente la estructura interna de la función objetivo es fundamental para sacarle partido a este método. Con esta finalidad se han diseñado muchos esquemas de descomposición que a menudo buscan aprender las interacciones entre distintas variables para una mejor agrupación.

Cabe destacar que una descomposición correcta del problema puede ayudar también a conocer qué partes del mismo tienen una mayor contribución sobre la función objetivo. De cara a la implementación de un *CCEA*, partiendo de una buena descomposición se puede analizar dicha contribución sobre cada población y reservar más o menos recursos para cada una de forma dinámica y proporcional. Trabajos como [OLMY14] exponen las posibles ganancias de explotar este conocimiento.

4.2.1.1. Separabilidad

Se empezará explicando el concepto de separabilidad.

Definición 4.1: una función $f : \Omega \to \mathbb{R}$ es **separable** o **totalmente separable** cuando cada una de sus variables puede optimizarse de forma independiente a las demás. Formalmente, escribiendo $\Omega = \prod_{i=1}^{m} \Omega_i$, esto equivale a

$$\begin{aligned} \underset{(x_1,\dots,x_m)\in\Omega}{\operatorname{argmin}} \; f(x_1,\dots,x_m) = \Bigg(& \underset{x_1\in\Omega_1}{\operatorname{argmin}} f(x_1,\dots,x_m), \\ & \dots \\ & \underset{x_i\in\Omega_i}{\operatorname{argmin}} f(x_1,\dots,x_i,\dots,x_m), \\ & \dots \\ & \underset{x_m\in\Omega_m}{\operatorname{argmin}} f(x_1,\dots,x_m) \Bigg) \end{aligned} \tag{4.1}$$

En caso contrario, la función f se dice no separable. Como cabe esperar, las funciones separables son sumamente sencillas y difíciles de encontrar, en el sentido de que aquellos sistemas objeto de optimización en problemas prácticos suelen ser complejos y tienden a tener dependencias entre sus variables. Es por eso que se considera interesante la siguiente definición.

Definición 4.2: una función objetivo $f : \Omega \to \mathbb{R}$ se dice (r, s)-separable si existen $I_1, I_2, \dots, I_r \subseteq \{1, 2, \dots, m\}$ tales que:

1. $\cup_{i=1}^{r} I_i = \{1, 2, \dots, m\}$.
2. $I_i \cap I_j = \emptyset \iff i \neq j$.

3. El número de elementos de I_i, que se notará como s_i, es menor o igual que s para todo i.

4. Escribiendo $I_i = \{j_{i,1}, j_{i,2}, ..., j_{i,s_i}\}$, existen r funciones $g_i : \Omega_{I_i} \to \mathbb{R}$ cumpliendo:

$$f(x) = \sum_{i=1}^{r} g_i(x_{j_{i,1}}, x_{j_{i,2}}, ..., x_{j_{i,s_i}}) \qquad \forall x \in \Omega$$

donde Ω_{I_i} corresponde al subespacio de Ω formado por aquellas dimensiones con los índices dados por I_i.

Nótese que en la condición número 4 la formulación de f como una suma no es restrictiva. Esta podría cambiarse por un producto o por combinaciones de sumas y productos, simplemente modificando algunas de las g_i por su logaritmo neperiano y considerando la exponencial de la suma de dichas funciones en la en la reconstrucción de f. Intuitivamente, la idea importante es que se puede representar f como una operación sencilla de r funciones independientes más sencillas. Un tamaño s lo más pequeño posible es deseable, al acotar la complejidad de resolver el "subproblema" con una mayor dimensionalidad que, esperablemente, supondrá un cuello de botella.

Para acotar algo más la definición anterior, se dirá que la función f es **exactamente** (r, s)-separable si es (r, s)-separable y no existen $r' > r$ ni $s' < s$ tales que f sea (r', s')-separable. Si una función es (r, s)-separable con $r > 1$ y $s < m$, se dirá que f es **parcialmente separable**.

4.2.1.2. Tipos de descomposición

Los primeros cronológicamente y los más sencillos métodos de descomposición se conocen a día de hoy como **estáticos**. Las variables que optimizará cada subpoblación se fijan de forma previa al proceso de optimización y no se modifican durante el mismo. La elección de estos grupos puede venir precedida de un conocimiento experto del problema, aunque no es común tener información perfecta sobre su separabilidad.

El algoritmo *CCGA* propuesto en [PDJ94b] es de este tipo y ha demostrado utilidad en problemas totalmente separables. Sin embargo, en ese mismo artículo, sus autores experimentaron con una función bidimensional no separable, observando malos resultados.

Los siguientes métodos de descomposición se conocen como **agrupamiento aleatorio de variables**, y se caracterizan por ir modificando cada cierto tiempo los grupos aleatoriamente. De esta forma, variables dependientes que originalmente caigan en grupos distintos, eventualmente pueden ser optimizadas juntas.

Si no se conoce el número óptimo de grupos ni de variables por grupo, ambos valores suponen hiperparámetros a optimizar. Se pueden dejar ambos fijos al comienzo del algoritmo, o bien diseñar un método específico para adaptar estos parámetros a lo largo de la ejecución.

Existen muchos ejemplos de algoritmos coevolutivos cooperativos con agrupamiento aleatorio, aunque en los últimos años el interés ha ido migrando hacia las técnicas basadas en aprendizaje de conexiones, de las que se hablarán a continuación. Las referencias enumera-

das en [MLZ+19] exponen algunos de los trabajos más relevantes en agrupamiento aleatorio de variables.

La definición de grupos tanto estáticos como aleatorios ignora, salvo uso de conocimiento previo del problema, la epistasis entre variables. Los métodos de **aprendizaje de conexiones**, o como es más conocido, *linkage learning*, buscan basar el agrupamiento de las variables en un análisis de la estructura del problema. Este puede tomar la forma de un agrupamiento estático de los parámetros de acuerdo a un estudio previo de su interacción [OLMY14][OYM+17], bien de forma dinámica utilizando la evolución de las soluciones y de sus valores objetivo para encontrar patrones en su evolución conjunta [PGCP99][YGS+09], o bien una mezcla de ambas metodologías.

A su vez, existen tres formas de categorizar, *grosso modo*, los métodos de detección de dependencias [YGS+09][MLZ+19]:

1. **Perturbacionales**. La interacción se mide a partir de perturbaciones de 2 ó más variables, particularmente observando cómo estas afectan por separado y en conjunto al valor de la función objetivo.

   La definición cuantitativa de *interacción* usada por los métodos perturbacionales no es única. Hasta nuestro conocimiento, existen 2 definiciones de uso extendido. Se dará su definición para problemas continuos pero puede adaptarse fácilmente a funciones objetivo discretas.

   Se denotará como $\Delta_i f$ la perturbación de f dado un salto en la variable i-ésima de longitud Δx_i, y $\Delta_{i,j} f$ a la correspondiente con saltos Δx_i y Δx_j en las variables i-ésima y j-ésima respectivamente. En notación matemática, estas pueden definirse de la siguiente forma:

   $$\Delta_i f(x) = f(x_1, ..., x_m) - f(x_1, ..., x_i + \Delta x_i, ..., x_m)$$

   y

   $$\Delta_{i,j} f(x) = f(x_1, ..., x_m) - f(x_1, ..., x_i + \Delta x_i, ..., x_j + \Delta x_j, ..., x_m)$$

   Las definiciones para *interacción* quedan entonces como:

   a) Dos variables de índices i, j son interdependientes si $\exists x = (x_1, ..., x_m) \in \Omega, \Delta x_i, \Delta x_j$ tales que $\Delta_i f(x) \cdot (\Delta_{i,j} f(x) - \Delta_j f(x)) < 0$.

   b) Dos variables de índices i, j son interdependientes si $\exists x \in \Omega$ tales que $\Delta_i f(x) + \Delta_j f(x) \neq \Delta_{i,j} f(x)$.

   Cuando se aplican las definiciones anteriores en un análisis previo de la estructura del problema, es común tomar al menos un par de valores para cada par de variables y estudiar su conexión viendo si se cumple alguna de las condiciones anteriores. Este análisis por muestreo puede ser impreciso si se quiere limitar el número de evaluaciones de la función objetivo dedicadas al mismo (que crece en orden $O(m^2)$ con respecto al número de variables). Por ello, puede ser recomendable comprobar ambas condiciones para aumentar la probabilidad de encontrar posibles dependencias ocultas.

La segunda definición da, además, una forma de medir la interacción:

$$\Delta^{i,j} = \Delta_i f(x) + \Delta_j f(x) - \Delta_{i,j} f(x)$$

En caso de querer comparar las interacciones entre distintas parejas de variables, se sugiere que puede ser conveniente "normalizar" la perturbación observada sobre f dividiendo entre la longitud del salto. Esto es, sustituir $\Delta_i f(x) + \Delta_j f(x) - \Delta_{i,j} f(x)$ por $\frac{\Delta_i f(x)}{\Delta x_i} + \frac{\Delta_j f(x)}{\Delta x_j} - \frac{\Delta_{i,j} f(x)}{||(\Delta x_i, \Delta x_j)||}$, donde la norma usada es dependiente de la forma en que se apliquen las perturbaciones. De esta forma, se le da mayor peso a aquellas no linealidades generadas de forma abrupta en saltos cortos. Normalmente, se puede tomar $||(\Delta x_i, \Delta x_j)|| = \sqrt{(\Delta x_i)^2 + (\Delta x_j)^2}$.

2. **Basados en un análisis estadístico.** Los valores de las variables y el valor de la función objetivo se tratan como variables aleatorias. En este tipo de métodos de análisis de dependencias, las muestras tomadas a lo largo de la ejecución del algoritmo se usan para estimar medidas de dependencia como el coeficiente de correlación de Pearson, la entropía o la información mutua, entre otras.

   En [YGS+09], T. Yu, D.E. Goldberg, K. Sastry, C.F. Lima y M. Pelikan desarrollan un algoritmo genético basado en la matriz de estructura de dependencias (*DSMGA*), originada en el ámbito de la organización empresarial. Como indica su nombre, esta matriz no es más que una estructura de datos en la que se almacena, para cada par de variables, la medida de dependencia escogida (booleana o numérica) entre ambas. Por simplicidad, sus autores limitan su estudio a usar un valor binario por cada pareja de variables, 1 cuando se estima que estas son dependientes, 0 en caso contrario. La forma de construir la *DSM* es a partir de la información mutua entre cada dos genes, que posteriormente se aprovecha para agrupar las variables y así explotar la modularidad del problema. Nótese, sin embargo, que este trabajo no se aplica sobre el marco de los *CCEA*, sino que se centra en un *cruce* que respete los *building blocks*, o conjuntos de variables ligadas, con el objetivo de respetar dichos bloques y evitar destruirlos durante el cruce.

   Para una lista más extensa de trabajos que hacen uso de este tipo de técnicas, se recomienda consultar [MLZ+19] (Sección III, subsección C.2).

3. **Basados en un modelado de la distribución.** Este tipo de algoritmos usan la población actual para estimar la distribución de probabilidad de muestreo de soluciones. Esto es, el objetivo de estos métodos es aproximar la función de densidad con la que se ha trabajado en el capítulo 2, pero contando únicamente con un número finito de candidatos.

   Algunos de los ejemplos más conocidos de este tipo de algoritmos son los *Estimation of Distribution Algorithms* (*EDA*) [QM04] y el *Bayesian Optimization Algorithm* (*BOA*) [PGCP99]. Estos métodos tratan las interdependencias de forma implícita y por ello tienden a ser mejores que los algoritmos evolutivos clásicos cuando se quiere limitar el número de iteraciones y el problema no es demasiado complejo. Sin embargo, para su aplicación en evolución coevolutiva es necesario poder obtener información sobre la estructura de la función objetivo, y esta suele estar codificada en el modelo de la

distribución generado. Tanto la construcción del modelo en cada iteración como el análisis del mismo son procesos computacionalmente costosos.

Una versión mejorada del *BOA*, el *Hierarchical BOA* (*hBOA*)[PGT03], es capaz de encontrar mejores resultados en problemas con dependencias más complejas, si bien sigue sin solucionar los últimos problemas comentados.

Cabe destacar que, en la práctica, puede existir un solapamiento entre las variables que evoluciona cada subpoblación. El estudio realizado en [GSOG+16] muestra empíricamente que un esquema coevolutivo cooperativo con dominios solapados es capaz de obtener mejores resultados sobre algunas funciones test multiobjetivo ZDT que uno con dominios no solapados.

Para aquellas funciones con dependencias más complejas que no son separables por bloques, puede ser conveniente incorporar mecanismos que analicen la jerarquía de dependencias y los solapamientos entre distintos bloques. El *hBOA* es uno de los primeros algoritmos en afrontar este problema de una forma relativamente exitosa aunque, como hemos dicho, los modelos de distribución generados por este tipo de algoritmos no son fácilmente aprovechables desde el marco coevolutivo. El *DSMGA* también está diseñado para afrontar dependencias jerárquicas y solapadas, y la estructura de datos que representa dichas dependencias es fácilmente interpretable, aunque no es tan escalable como el *hBOA* [YGS+09].

4.2.2. Selección de colaboradores

En la práctica, la coevolución cooperativa no implica descomponer la función objetivo en suma de otras funciones más sencillas (aunque hay otros algoritmos que parten en su desarrollo de esta posibilidad [QM04]). Para evaluar la función a optimizar a pesar de haber descompuesto el espacio de búsqueda, es necesario que la población asociada a cada componente cuente con unos parámetros de referencia que representen al resto de componentes. Estas son las variables que se dejan fijas mientras cada población optimiza los parámetros a los que está restringida por la descomposición realizada. El nombre que se le da a estos "representantes" del resto de componentes es el de **colaboradores**, y la forma de seleccionarlos supone un papel crucial durante la coevolución.

Matemáticamente, se puede definir el concepto de colaboradores de la componente k-ésima con conjunto de índices I_k como el conjunto de valores a los que se fijan las variables x_i con $1 \leq i \leq m$, $i \notin I_k$. De nuevo, en coevolución cooperativa cada subpoblación evoluciona restringiéndose a alterar un subconjunto del total de variables, que se representa como I_k. Todas las demás se fijan a un valor concreto o se alteran cada cierto tiempo a partir de las soluciones parciales del resto de componentes mediante un proceso de comunicación preestablecido: el **esquema de colaboración**.

En primer lugar, se puede considerar el método más trivial de los posibles: cada vez que se deseen mezclar dos o más subpoblaciones se construyen todas las combinaciones posibles de soluciones de cada uno de los subproblemas (entendiendo por "solución" los valores de las variables evolucionadas por esa subpoblación en uno de sus individuos). Este método garantiza alcanzar el óptimo global si cada población es suficientemente grande [Pan10],

pero es computacionalmente inviable.

Las siguientes opciones más sencillas serían las de copiar siempre la solución correspondiente al individuo con mejor valor objetivo de la subpoblación (***single best collaborator***), o bien tomar una de sus soluciones de forma aleatoria como colaborador para cada individuo del resto de poblaciones (***random collaborator***).

Otro método, que añade capacidad de exploración al *single best collaborator* es la del ***elite collaborators*** o *K best collaborators*. Consiste en elegir como colaborador, para cada individuo, uno de los K mejores individuos de la subpoblación correspondiente de forma aleatoria. Un mayor valor de K reduce la presión selectiva, al permitir seleccionar colaboradores en apariencia peores. Empezar con un valor de K alto e ir reduciéndolo progresivamente puede ser una buena estrategia para favorecer la exploración al principio e ir centrando paulatinamente la búsqueda hacia las mejores soluciones.

También puede considerarse la **selección por torneo**. Análogamente a la selección por torneo en los *EA* tradicionales, esta consiste en muestrear un conjunto de colaboradores de entre los disponibles y seleccionar el mejor de todos. Igual que con la élite de colaboradores se puede ajustar la exploración/explotación dinámicamente, en este caso aumentando de forma progresiva el número de competidores.

Finalmente, pueden combinarse siempre los individuos con el mismo índice dentro de cada subpoblación, esto es, la i-ésima solución de una subpoblación con la i-ésima solución de todas las demás (donde el índice i es fijo, independiente de su valor objetivo). Este esquema suele conocerse como **colaboradores enlazados**.

Cualquier combinación de los métodos anteriores puede ser de interés. Por ejemplo, sería factible asignar el mejor colaborador a la mitad mejor valorada de una subpoblación y tomar uno aleatorio para el resto.

En [MLZ+19] se referencian numerosos estudios explicando los detalles de cada uno de estos esquemas de colaboración y comparando cuándo unos pueden ser más convenientes que otros, por lo que se recomienda su lectura para profundizar en el tema.

4.3. Modelos y técnicas para el análisis en coevolución cooperativa

En esta sección se definirá el algoritmo $CC(1+1)EA$, que adapta al $(1+1)EA$ al marco de la coevolución cooperativa, y se mostrarán algunos de los resultados formales obtenidos en [JW04][Jan13] sobre este algoritmo.

También se hablará del modelo simplificado formulado en [Pan10] y de algunos resultados que obtiene de su análisis.

Definición 4.3: sean $m, l, k \in \mathbb{N}$ tales que $l \cdot k = m$, y sea Ω un espacio de dimensión m. Sean $I_1, ..., I_k$ conjuntos de índices disjuntos, cada uno conteniendo l índices distintos. Dados $x \in \Omega$ y la descomposición $\Omega = \prod_{i=1}^{k} \Omega_{I_i}$, se notará $x^{(1)}...x^{(k)} \equiv x$, con $x^{(i)} \in \Omega_{I_i}$ a la *partición canónica* de x.

Esta partición es la que realizará nuestro algoritmo sobre el espacio de búsqueda. En la sección 4.2.1.1 se dijo que una función era separable cuando podía descomponerse en suma de funciones independientes definidas cada una sobre conjuntos de variables disjuntos, en un sentido parecido al que se acaba de comentar. Para evitar confusión, se llamarán **piezas** a los conjuntos de variables independientes del dominio de la función objetivo cuando se esté hablando de funciones separables, y se llamarán **componentes** a los grupos de variables generados por el algoritmo, y que no tienen por qué coincidir con las piezas. Se dirá, por tanto, que las componentes **coinciden** con las piezas cuando cada componente se encuentre dentro de alguna pieza, y que ambas coinciden **exactamente** cuando las componentes y las piezas son las mismas.

Estamos en posición de estudiar el $CC(1+1)EA$, que se entenderá como la adaptación natural del $(1+1)EA$ a la coevolución cooperativa.

Algoritmo 3: $CC(1+1)EA$

Input: $f \in \mathcal{F}$
Output: $x \in S$ donde f alcanza su máximo
se genera una solución inicial ϕ_0 aleatoria;
se inicia un contador de generaciones pasadas $t \leftarrow 0$;
while $f(\phi_t) < f_{max}$ **do**
  for $a \in 1, ..., k$ **do**
    $\phi_{t.m}^{(a)} \leftarrow$ mutación de $\phi_t^{(a)}$ con probabilidad de mutación $\min\{\frac{1}{l}, \frac{1}{2}\}$;
    if $f(\phi_{t.m}) \geq f(\phi_t)$ **then**
      $\phi_{t+1} \leftarrow \phi_{t.m}$;
    else
      $\phi_{t+1} \leftarrow \phi_t$;
    $t \leftarrow t+1$

Igual que el $(1+1)EA$, el $CC(1+1)EA$ genera una solución inicial aleatoria y la muta iterativamente, aceptando solo aquellas mutaciones mejores a la solución actual. En este caso, sin embargo, la mutación se hace sobre una sola de las partes de la solución total como si se tratase en una solución en sí misma independiente del resto. La evaluación del candidato mutado se hace, sin embargo, sobre la solución completa, aunque se da por válida una solución igual de buena que la anterior para evitar posibles estancamientos en una sola de las componentes. La probabilidad de mutación se toma como $\min\{\frac{1}{l}, \frac{1}{2}\}$ para conservar cierto grado de aleatoriedad en el caso $l = 1$.

Se procede a enunciar y comentar algunos resultados obtenidos en [Jan13] que se consideran interesantes e incluso anti-intuitivos. Sus demostraciones son relativamente técnicas y no aportan, a nuestro entender, mayor entendimiento sobre el tema que estamos tratando, así que se omitirán en nuestra exposición. Por simplicidad, estos resultados se refieren a funciones con dominio $\Omega = \{0,1\}^m$.

La principal motivación para los *CCEA* era la reducción del espacio de búsqueda, que en condiciones adecuadas debería traducirse en un menor número esperado de evaluaciones de la función objetivo antes de la convergencia, lo que en la sección 2.2 se nota con la variable aleatoria T o *first hitting time*. Concretamente, se denominará $T_{(1+1)EA,f}$ al número de iteracio-

nes del $(1+1)EA$ hasta alcanzar la solución óptima de f, y $T_{CC(1+1)EA,f}$ al correspondiente para el $CC(1+1)EA$.

Para hablar del orden de crecimiento de T, se usará la siguiente notación estándar: $\Omega(T)$ para el orden de complejidad de la cota inferior ajustada de T, $\omega(T)$ para el orden de complejidad de una cota inferior no ajustada de T, $O(T)$ para la cota superior y $\Theta(T)$ para el orden de ambas cuando la cota inferior y la superior coincidan en orden.

Hasta ahora se ha hablado de la separabilidad de la función objetivo como factor fundamental a la hora de sacar partido de los algoritmos coevolutivos cooperativos. Va a comprobarse que este puede no ser el único factor de importancia, o que al menos no garantiza un mejor comportamiento respecto a los EA clásicos (al menos, si se compara el $CC(1+1)EA$ con el $(1+1)EA$).

Se empezará remarcando el hecho de que toda función lineal, esto es, de la forma $f(x_1,...,x_m) = w_0 + \sum_{i=1}^{m} w_i \cdot x_i$, es por definición $(m,1)$-separable.

Teorema 4.4: ([Jan13], teorema 6.20) Dada una función lineal f cualquiera,

$$\mathbb{E}[T_{CC(1+1)EA,f}] = \Omega(m \cdot \ln m)$$

independientemente del número de componentes tomadas.

Es decir, la partición que se haga del dominio es prácticamente irrelevante en cuanto al tiempo de convergencia esperado.

Hasta cierto punto, podría discutirse que el resultado anterior es poco relevante al ser la función lineal tan sumamente sencilla de optimizar. Además, este ni siquiera tiene en cuenta las posibilidades de paralelización que se podrían aplicar sobre el $CC(1+1)EA$ en caso de saber que la función a optimizar es lineal. Véase el siguiente ejemplo.

Definición 4.5: sean $k,l,m \in \mathbb{N}$, tal que $k \cdot l = m$. Dado $x \in \{0,1\}^l$, notando $x_i = x[i]$, definimos:

$$g_l(x) = \begin{cases} l+i & \text{si } x = 1^i 0^{l-i}, i \in \{0,1,...,l\} \\ l+i & \text{si } x = 0^{l-i}1^i, i \in \{4\} \cup \{6,9,12,...,3\lfloor \frac{l}{3} \rfloor\} \\ l - \sum_{i=1}^{l} x[i] & \text{en otro caso} \end{cases} \tag{4.2}$$

$$f_{k,l} = \sum_{i=1}^{k} g_l(x[(i-1)\cdot l+1]x[(i-1)\cdot l+2]...x[i\cdot l])$$

Claramente, la función f es (k,l)-separable por definición.

Teorema 4.6: ([Jan13] teorema 6.23) sean $k,l,m \in \mathbb{N}$, tal que $k \cdot l = m$. Si $l = \Omega(\ln m)$

$$\mathbb{E}[T_{(1+1)EA,f_{k,l}}] = O(m \cdot l).$$

Teorema 4.7: ([Jan13] teorema 6.24) sean $l \in \mathbb{N}$, $k = l^4$ y $m = k \cdot l$. Supóngase que la partición canónica del $CC(1+1)EA$ coincide perfectamente con las piezas de f. Aunque en cada

componente se aplique la probabilidad de mutación óptima que maximice la probabilidad de convergencia, se cumple que:

$$\mathbb{E}[T_{CC(1+1)EA,f_{k,l}}] = \omega(m \cdot l^{4/3}).$$

Incluso en un problema separable y aplicando la descomposición perfecta el *CCEA* no parece tener garantizados buenos resultados. Por último, se verá que descomponer el problema puede llegar a impedir encontrar la solución óptima.

Definición 4.8: sean $m, l \in \mathbb{N}$, $l \leq m$. Se define $JUMP_l \colon \{0,1\}^m \to \mathbb{R}$ como:

$$JUMP_l(x) = \begin{cases} m - \sum_{i=1}^{m} x_i & \text{si } m - l < \sum_{i=1}^{m} x_i < m \\ l + \sum_{i=1}^{m} x_i & \text{en otro caso} \end{cases}$$

La función anterior no es, en general, separable, puesto la forma de evaluarla depende de la suma de todos los elementos del vector x. Por tanto, no se espera que el $CC(1+1)EA$ obtenga mejores resultados que el $(1+1)EA$. Se concluye exponiendo el siguiente resultado:

Teorema 4.9: (teoremas 5.13 y 6.25 de [Jan13])

1. $\mathbb{E}[T_{(1+1)EA,JUMP_m}] = \Theta(m^m + m \ln m)$
2. Para cualquier partición con 2 ó más componentes, el algoritmo $CC(1+1)EA$ nunca encuentra el óptimo de $JUMP_m$ con probabilidad $1 - 2^{-\Omega(m)}$.

El autor al que se referencia, Thomas Jansen, continúa el resultado anterior concluyendo que, "en general, los *CCEA* no son optimizadores globales, puesto que pueden no encontrar la solución óptima incluso dado tiempo infinito". Nos gustaría señalar que esta limitación se debe al modelo utilizado y que un esquema de colaboración más complejo sí puede garantizar la convergencia al óptimo global, como demuestra Liviu Panait en [Pan10] a partir del modelo que se comentará a continuación.

Inicialmente, este modelo asume un dominio finito con solo dos componentes, poblaciones infinitas e infinitos colaboradores. De esta forma, dentro de cada componente la mejor solución será aquella que tenga el valor objetivo más alto para alguno de sus colaboradores.

Formalmente, la población está representada para cada componente como un vector de longitud igual al número de posibles soluciones factibles, en el que cada elemento corresponde a la probabilidad de observar cada una de dichas soluciones dentro de la población total. Se asume que la primera componente puede tomar n valores distintos y la segunda m valores distintos. La población total queda, entonces, como un elemento del siguiente conjunto:

$$\left\{ x \in [0,1]^n | \sum_{i=1}^{n} x_i = 1 \right\} \times \left\{ y \in [0,1]^m | \sum_{i=1}^{m} y_i = 1 \right\}$$

Numerando cada una de las soluciones factibles en ambas componentes, la función objetivo puede representarse como una matriz $A \in \mathcal{M}_{n,m}$ en la que el elemento a_{ij} representa el valor

objetivo de combinar la i-ésima posible solución en la primera componente con la j-ésima posible solución de la segunda. Nótese que "mirar" el valor a_{ij} equivale en la práctica a evaluar la función objetivo combinando las subsoluciones correspondientes x_i e y_j.

La evolución de la población se modela de la siguiente forma:

$$
\begin{aligned}
u_i^{(t)} &= \max_{j=1,\dots,m} a_{ij} \\
w_j^{(t)} &= \max_{i=1,\dots,n} a_{ij} \\
x_i^{(t+1)} &= \frac{x_i^{(t)} \left(\left(\sum_{k:\, u_k^{(t)} \leq u_i^{(t)}} x_k^{(t)} \right)^K - \left(\sum_{k:\, u_k^{(t)} < u_i^{(t)}} x_k^{(t)} \right)^K \right)}{\sum_{k:\, u_k^{(t)} = u_i^{(t)}} x_k^{(t)}}
\end{aligned} \tag{4.3}
$$

$$
y_j^{(t+1)} = \frac{y_j^{(t)} \left(\left(\sum_{k:\, w_k^{(t)} \leq w_j^{(t)}} y_k^{(t)} \right)^K - \left(\sum_{k:\, w_k^{(t)} < w_j^{(t)}} y_k^{(t)} \right)^K \right)}{\sum_{k:\, w_k^{(t)} = w_j^{(t)}} y_k^{(t)}} \tag{4.4}
$$

Los números $u_i^{(t)}$ y $w_j^{(t)}$ corresponden al valor objetivo asociado a las soluciones de índices i y j en la primera y segunda componente, respectivamente. Este valor se calcula, como se ha indicado anteriormente, como el valor más alto que puede tomar dicha solución considerando todos sus posibles colaboradores. La definición de $x_i^{(t+1)}$ e $y_i^{(t+1)}$ modela el cambio en la distribución de soluciones aplicando selección por K-torneo a la generación actual. Claramente, el algoritmo convergerá a una solución de índices por componente i, j y se satisface que $\lim_{t\to\infty} x_i^{(t)} = \lim_{t\to\infty} y_j^{(t)} = 1$.

Lema 4.10: ([Pan10], Lema 1)

Asúmase que la población inicial del procedimiento anterior, $(x^{(0)}, y^{(0)})$, se toma aleatoriamente a partir de una distribución uniforme. Entonces

$$
P\left[\min_{i=1,\dots,n} x_i^{(0)} = 0\right] = P\left[\max_{i=1,\dots,n} x_i^{(0)} = 1\right] = P\left[\min_{j=1,\dots,m} y_j^{(0)} = 0\right] = P\left[\max_{j=1,\dots,m} y_j^{(0)} = 1\right] = 0.
$$

El lema anterior simplemente afirma que, tomando una muestra (infinita) uniforme sobre todas las soluciones factibles en cada componente, la probabilidad de que se de el "caso extremo" de tener posibles soluciones sin representar es nula. Este hecho es importante puesto que, igual que se vio en el capítulo 2 al demostrar la convergencia al óptimo para algoritmos evolutivos con poblaciones infinitas bajo selección, esta solo puede garantizarse cuando la solución óptima se encuentre en la población inicial. No se mostrará su demostración, que puede consultarse en el apéndice de la fuente referenciada.

Teorema 4.11: ([Pan10], Teorema 1)

Si la matriz A tiene un único máximo global con índices (i^*, j^*) y tomando una población inicial aleatoria uniforme, el proceso coevolutivo converge a dicho máximo con probabilidad 1.

Demostración: se observa que $u^{(t)}$ y $w^{(t)}$ son constantes a lo largo del tiempo. Por tanto, $x^{(t+1)}$ depende únicamente de $u^{(t)}$ y de $x^{(t)}$. Análogo para $y^{(t+1)}$. Por tanto, siguiendo este esquema con infinitos colaboradores ambas componentes evolucionan de forma independiente. Se demostrará que $x_{i^*}^{(t)} \xrightarrow{t\to\infty} 1$, obteniéndose el resultado análogo para y_{j^*}.

Se tiene que $u_{i^*}^{(t)} > u_i^{(t)}$ para todo $i \neq i^*$, por lo que $x_{i^*}^{(t+1)} = 1 - (1 - x_{i^*}^{(t)})^K$, de lo que se obtiene $x_{i^*}^{(t)} = 1 - (1 - x_{i^*}^{(0)})^{K^t}$. Ahora bien, por el lema 4.10 se sabe que $0 < x_{i^*}^{(0)} < 1$ con probabilidad 1, por lo cual, casi seguramente,

$$\lim_{t\to\infty} x_{i^*}^{(t)} = \lim_{t\to\infty} 1 - (1 - x_{i^*}^{(0)})^{K^t} = 1 - \lim_{t\to\infty} (1 - x_{i^*}^{(0)})^{K^t} = 1. \qquad \blacksquare$$

El teorema anterior puede adaptarse, de hecho, al caso con múltiples óptimos (ver apéndice de [Pan10]). Da que, dados colaboradores infinitos, se puede evaluar cada subsolución de una forma "objetiva", esto es, su valor pasa a ser fijo durante todo el algoritmo y deja de ser necesario considerar sus colaboradores en cada paso del proceso iterativo.

La hipótesis de infinitos colaboradores es tan inaplicable como la de poblaciones infinitas. Se vio en su momento, sin embargo, utilidad en asumir poblaciones infinitas para estudiar la convergencia de los *EA* y también se entiende que el teorema anterior pone de manifiesto la importancia del esquema de colaboración en los *CCEA*.

Para ajustar el modelo analizado a la forma de un algoritmo aplicable en la práctica, se redefine la evaluación de cada subsolución como aquella que tenga el máximo valor objetivo para alguno de entre N colaboradores muestreados aleatoriamente de la otra subpoblación (ver teorema 2 de [Pan10]).

$$\begin{aligned} u_i^{(t)} &= \sum_{j=1}^{m} a_{ij} \frac{y_j^{(t)} \left(\left(\sum_{k:\, a_{ik} \leq a_{ij}} y_k^{(t)} \right)^N - \left(\sum_{k:\, a_{ik} < a_{ij}} y_k^{(t)} \right)^N \right)}{\sum_{k:\, a_{ik} = a_{ij}} y_k^{(t)}} \\ w_j^{(t)} &= \sum_{i=1}^{n} a_{ij} \frac{x_i^{(t)} \left(\left(\sum_{k:\, a_{kj} \leq a_{ij}} x_k^{(t)} \right)^N - \left(\left(\sum_{k:\, a_{kj} < a_{ij}} x_k^{(t)} \right)^N \right)}{\sum_{k:\, a_{kj} = a_{ij}} x_k^{(t)}} \end{aligned} \tag{4.5}$$

La selección por K-torneo se conserva tal como la se ha considerado hasta ahora.

Teorema 4.12: ([Pan10], teorema 4)

Si la matriz A tiene un único óptimo global, para todo $\varepsilon > 0$ y $K \in \mathbb{N}, K \geq 2$, existe un número de colaboradores $N_\varepsilon \geq 1$ tal que el *CCEA* modelado por el proceso coevolutivo anterior converge al óptimo global con probabilidad mayor a $(1 - \varepsilon)$ para cualquier número de colaboradores $N \geq N_\varepsilon$.

Pensándolo con detenimiento, este resultado es una progresión natural del teorema anterior. Si infinitos colaboradores garantizan convergencia casi segura al óptimo, es de esperar que más de estos, aunque un número finito, aumente nuestras posibilidades de alcanzarlo.

El teorema 4.12, cuya demostración puede consultarse en la fuente original, garantiza que incrementar el número de colaboradores aumenta las probabilidades de encontrar el óptimo tanto como se desee, lo que le da mucho más peso al esquema de colaboración del que se le asociaba hasta entonces.

4.4. Problemas de los algoritmos coevolutivos cooperativos

Esta sección buscará exponer aquellas patologías de los CCEA que no se encontraban en los algoritmos evolutivos tradicionales.

4.4.1. Sobregeneralización relativa

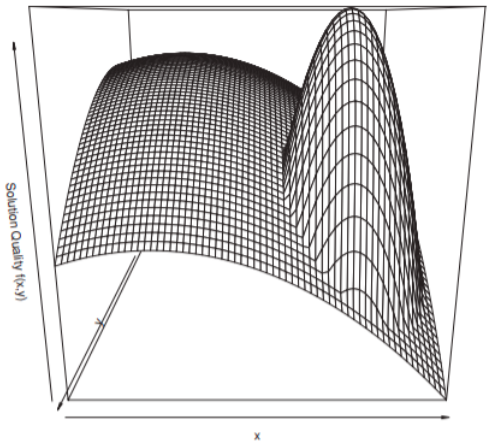
En [WP06] se observó que, comparando un algoritmo evolutivo con su versión coevolutiva-cooperativa, esta última tenía una mayor tendencia a soluciones robustas, independientemente de que estas fuesen óptimas o no. En el experimento realizado en [SS04] puede observarse que en problemas con una separabilidad no trivial, para una agrupación estática de las variables de decisión, la ejecución de un CCEA estándar convergía a óptimos locales.

En este apartado se continuará exponiendo los resultados de [Pan10], en tanto que formalizan este problema e intentan darle una explicación rigurosa.

De acuerdo con [Wie03], la **sobregeneralización relativa** sucede *"cuando las poblaciones en el sistema son atraídas hacia áreas del dominio en las cuales muchas estrategias posibles se comportan bien en relación a los colaboradores"*. En otras palabras, si una subsolución para una componente tiende a alcanzar buenos valores objetivo para la mayor parte de colaboradores provenientes de otra población, mientras que el óptimo global de dicha componente, en promedio, obtiene resultados peores, es probable que el algoritmo se estanque en el óptimo local.

En particular, si en el modelo utilizado en la sección anterior se sustituye la forma de evaluar cada solución, el máximo del valor de combinarla con sus posibles colaboradores, por la media de los valores de sus posibles combinaciones, [Wie03] demuestra que el algoritmo puede tender a converger a soluciones subóptimas incluso con poblaciones infinitas. Este es, de hecho, el modelo que [Pan10] analiza en primer lugar y que modifica para dar lugar a los resultados expuestos en la segunda mitad de la sección 4.3.

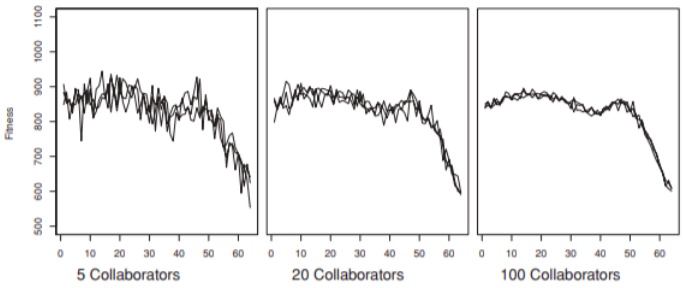
Panait ilustra el resultado obtenido por Wiegand a partir de la siguiente función bimodal:



(Imagen obtenida de Liviu Panait, *Theoretical convergence guarantees for cooperative coevolutionary algorithms* [Pan10].)

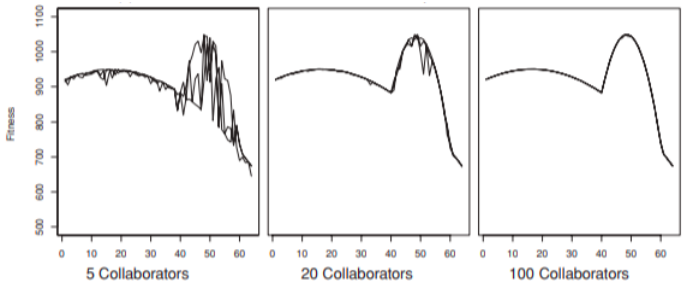
Visualmente, se entiende que el óptimo local a la izquierda es más robusto y cada componente obtiene, en promedio, mejores resultados para un valor cualquiera de sus colaboradores. El extremo de la derecha es claramente el óptimo global, pero la región donde esta diferencia se hace patente es mucho más pequeña.

Implementando (una versión adaptada de) el modelo de Wiegand a modo de algoritmo, evaluando cada individuo con el valor promedio de sus combinaciones con los colaboradores posibles, la población tiende a distribuirse de la siguiente forma:



(Imagen obtenida de Liviu Panait, *Theoretical convergence guarantees for cooperative coevolutionary algorithms* [Pan10].)

El resultado más alto parece encontrarse en el extremo más robusto. Ahora se ve como se distribuye la población si se considera el valor máximo:



(Imagen obtenida de Liviu Panait, *Theoretical convergence guarantees for cooperative coevolutionary algorithms* [Pan10].)

Puede argumentarse que un modelo como el formulado por Wiegand puede tener sentido a la hora de analizar el comportamiento de los *CCEA*, puesto que cabe esperar que la evolución de la población dependa de todos los posibles colaboradores. El trabajo realizado por Panait demuestra, teórica y experimentalmente, que si se desea encontrar el óptimo global la estrategia del mejor colaborador siempre será superior dado un número suficiente de colaboradores.

Se harán dos comentarios más sobre este problema observado en los *CCEA* conocido como sobregeneralización relativa. El primero es que esta solo se da en funciones no separables, o cuando la descomposición realizada por nuestro algoritmo no coincide con dicha separabilidad. Descomponer correctamente el problema, utilizando cualquiera de las técnicas disponibles para ello (sección 4.2.1.2), elimina completamente la sobregeneralización relativa.

En segundo lugar, este fenómeno es similar a la tendencia a óptimos robustos que pueden observarse en algunas funciones engañosas (sección 3.1) lo que, en ciertas situaciones prácticas, puede ser incluso deseable. Como ya se comentó anteriormente, muchos problemas prácticos presentan ruido, ya sea ruido determinista por las limitaciones en su modelado, o ruido estocástico en la evaluación de la función objetivo. Un método que tienda a obtener resultados robustos puede ser preferido en este tipo de situaciones. En [WP06] se discute en mayor profundidad la tendencia de los *CCEA* hacia soluciones robustas.

4.4.2. Otros problemas en los algoritmos coevolutivos cooperativos

En este apartado se comentarán otros problemas intrínsecos a los algoritmos coevolutivos cooperativos, tomando [MLZ+19] como principal referencia.

El primero se conoce en la litaratura como el **efecto Reina Roja**, o *Red Queen effect*. Se da cuando la interacción entre distintos individuos/poblaciones altera sistemáticamente su *fitness landscape*, dificultando la convergencia [CM95]. En la coevolución cooperativa, esto se traduce en que soluciones peores de una población obtengan puntualmente valores objetivo mejores al resto de individuos de su grupo por ser aleatoriamente combinadas con los mejores colaboradores. Esto puede influir negativamente en la evolución de la población durante las siguientes iteraciones, que tenderá a copiar esta peor solución. El efecto Reina Roja es producto de combinar una descomposición incorrecta del problema con cualquier esquema

de colaboración de tipo exploratorio, como el de colaboradores aleatorios o enlazados. Las formas de prevenir este fenómeno, en general, son una correcta descomposición del problema y un esquema de colaboración que favorezca los mejores colaboradores.

Otro problema observado en los *CCEA*, aunque no solamente en estos, es la **pérdida de gradiente**. En los algoritmos poblacionales, esta se da cuando el conjunto de soluciones pierde toda su diversidad antes de alcanzar el óptimo. Los algoritmos coevolutivos son más sensibles a este fenómeno, puesto que la pérdida de gradiente en una de sus poblaciones afecta a las demás. Cualquier operador evolutivo que fomente la exploración debería ayudar a evitar esta situación, aunque existen técnicas específicas para combatir la pérdida de gradiente [MLZ+19].

5 Arquitecturas Distribuidas y Estudio Experimental

En este capítulo exploraremos los algoritmos coevolutivos cooperativos en un sentido más práctico. Trataremos las características propias a los algoritmos evolutivos que los hacen aptos para ser paralelizados, y diseñaremos un experimento en el que adaptaremos un *CCEA* del estado-del-arte para minimizar la energía de una máquina de Boltzmann, con el objetivo de analizar cómo varía la eficacia del algoritmo en función de la separabilidad del problema. Finalmente, expondremos nuestras conclusiones sobre los resultados.

5.1. Paralelismo en algoritmos evolutivos: arquitecturas distribuidas

En esta sección, daremos una visión general de cómo los algoritmos evolutivos pueden aprovechar el paralelismo de las computadoras modernas, lo que supone un punto muy fuerte a su favor. Hablaremos sobre las arquitecturas distribuidas, de su importancia, sobre los principales esquemas de distribución de algoritmos evolutivos [GCZ+15] y, en particular, de cómo los *CCEAs* pueden adaptarse fácilmente a las mismas gracias a la modularidad inherente a su diseño, aunque no sin ciertos inconvenientes.

Al igual que en los algoritmos coevolutivos, el paralelismo nace de la posibilidad de descomponer un problema en múltiples subproblemas, aunque en un sentido más general y abstracto.

Un algoritmo, cómputo o proceso se dice **paralelo** si puede descomponerse en distintos elementos que pueden resolverse o ejecutarse de forma independiente y, por tanto, al mismo tiempo. Se usa como opuesto de **algoritmo secuencial**, en el que cada uno de sus elementos (si es que esa descomposición es siquiera posible) debe resolverse antes de empezar el "paso" siguiente. Ilustremos ambas definiciones con 3 ejemplos:

- Saber si una palabra específica aparece dentro de un texto es una tarea totalmente paralelizable. Distintas máquinas o *workers* pueden dividirse el total de palabras y comprobar al mismo tiempo si esa en particular se encuentra en la parte que tengan asignada. Reducir la respuesta del conjunto de trabajadores a la solución final es extremadamente sencillo.

- Recorrer ordenadamente una lista enlazada, una estructura de datos en la que todos los elementos se encuentran desordenados y en la que la posición de cada uno viene dada por su elemento anterior, es una tarea totalmente secuencial.

Cabe destacar que la capacidad para paralelizar un problema es altamente dependiente del grado de abstracción con la que lo estemos aproximando y de las herramientas (*hardware*

y *software*) de las que dispongamos para resolverlo. Un ejemplo claro sería el primero de los dos casos expuestos. Si, por cualquier motivo, el texto al que nos referimos se encontrase representado como una lista enlazada, la tarea no podría paralelizarse.

La mayor parte de los problemas prácticos son relativamente complejos y pueden descomponerse en múltiples pasos, muchos de los cuales son paralelizables. A día de hoy, los superordenadores que encabezan la lista del TOP500, formada a partir de sus resultados en el *benchmark* LINPACK con tareas comunes en el cálculo científico, cuentan con millones de núcleos, lo que se traduce en millones de hebras trabajando en paralelo.

La ley de Amdahl nos informa de la **ganancia** (o *speedup*) de paralelizar la resolución de un problema utilizando k trabajadores:

$$S = \frac{T_{\text{secuencial}}}{T_{\text{paralelo}}} = \frac{1}{(1 - f_{\text{paralelo}}) + \frac{f_{\text{paralelo}}}{k}}$$

donde T hace referencia al tiempo de cómputo total y f_{paralelo} nota la proporción del tiempo secuencial que puede repartirse en k trabajos paralelos. Esta fórmula es clara: para una mayor ganancia es necesario contar con un k grande (muchos trabajos) y/o una proporción de cálculo paralelizable alta. Mientras que el valor de k depende totalmente de los recursos de los que dispongamos, f_{paralelo} está ligada a la tarea que queramos resolver. Destacamos que una ganancia en velocidad puede aprovecharse reduciendo el tiempo de cómputo total o realizando más trabajo en el mismo tiempo.

Una de las ventajas de los algoritmos evolutivos es que, si bien tienden a requerir más tiempo y recursos computacionales que otros algoritmos específicos al problema que estemos tratando, muchos de ellos son muy fáciles de paralelizar. Por ejemplo, podemos evaluar la función objetivo al mismo tiempo sobre todos los individuos de la población actual que, salvo en algunos algoritmos coevolutivos, no dependen unos de otros. También es posible definir múltiples poblaciones a modo de "islas", que evolucionan independientemente aunque pueden comunicarse entre sí y compartir miembros de forma puntual.

Una de las ramas principales de la computación paralela se centra en las **arquitecturas distribuidas**, caracterizadas por que los distintos trabajadores no comparten nada salvo una red de paso de mensajes para su comunicación. A priori, la computación distribuida cuenta con más inconvenientes que la no distribuida. Tiene un uso de recursos menos eficiente, puesto que cada nodo necesita copiar la información que necesite para resolver su parte del problema. Además, la comunicación a través del paso de mensajes dificulta la sincronización entre trabajadores.

Las principal ventaja de las arquitecturas distribuidas es su **escalabilidad**. Un sistema se dice escalable cuando es relativamente sencillo aumentar sus prestaciones conservando la mayor parte del sistema original. Una arquitectura no distribuida centrada en una *CPU* con 8 núcleos no es escalable (al menos en lo que a *hardware* se refiere). Si queremos duplicar la velocidad de cómputo nuestras opciones son aumentar las prestaciones de cada núcleo o duplicar el número de núcleos. A día de hoy, ambas aproximaciones obligan a cambiar nuestra *CPU* por otra distinta.

De acuerdo con [GCZ+15], los *EA* distribuidos (*dEA*) pueden clasificarse como:

1. **Islas:** cada nodo de cómputo cuenta con, al menos, una población que evoluciona de manera convencional denominada como "isla". Las islas se encuentran conectadas en forma de red y cada cierto tiempo, de forma síncrona o asíncrona, cada isla comparte individuos con las islas vecinas (proceso de migración). Este mecanismo no solo permite aprovechar el *hardware* de cálculo paralelo sino que aumenta la diversidad de la búsqueda, dado que cada población evoluciona de manera parcialmente independiente al resto. La frecuencia de comunicación entre islas es un parámetro a ajustar. Una mayor frecuencia homogeiniza las distintas poblaciones, facilitando la convergencia a costa de reducir la diversidad.

2. **Maestro-trabajador:** un trabajador central aplica los operadores de selección, cruce y mutación, y delega la evaluación de los elementos de la población al resto de trabajadores. Es posible aumentar el trabajo de los esclavos para hacer el algoritmo más eficiente, como añadir una búsqueda local que cada nodo aplica sobre los individuos que tiene asignados.

3. **Modelos celulares o masivamente paralelos:** similares a las islas, pero con una única población sobre la que se definen "vecindades" de individuos. Cada solución solo compite y se cruza con aquellas dentro de su vecindad. Las distintas vecindades se comunican siguiendo una red de conexiones. Se benefician de un paralelismo masivo como el que puede obtenerse mediante cálculo en *GPU*, más que de una arquitectura distribuida (al contrario que los modelos de islas).

4. **Jerárquicos o híbridos:** entran en esta categoría aquellos que son combinaciones de cualquiera de los modelos anteriores. Tiene sentido formular, por ejemplo, esquemas de islas con maestros y trabajadores combinando la diversidad y escalabilidad de los primeros con la eficiencia de cálculo de los segundos, esquemas de islas de modelos celulares aprovechando distintos tipos de paralelismo, o de islas de islas, como los estudiados en [HLM99], que permiten combinar distintos tipos de comunicación dando mayor versatilidad al diseño del algoritmo.

5. ***Pool:*** todos los trabajadores tienen acceso a una estructura de datos con la población completa, de la cual evolucionan independientemente un subconjunto de individuos con cierto solapamiento. Dicho solapamiento permite una comunicación indirecta entre las distintas subpoblaciones. Este modelo es tan escalable como se quiera, al no necesitar definir explícitamente una jerarquía de paso de mensajes entre los trabajadores. Para implementar exitosamente este modelo, sin embargo, es necesaria una infraestructura adecuada que permita la sincronización de lectura y escritura de individuos y la transmisión de datos de manera eficiente. El desarrollo de bases de datos y otras herramientas *software* y *hardware* para computación de altas prestaciones en los últimos años ha facilitado la puesta en práctica de este tipo de modelos.

6. ***Modelos multiagente:*** son la adaptación a arquitecturas distribuidas de algoritmos coevolutivos competitivos. Tienen interés en aquellos problemas que puedan descomponerse en funciones objetivo más sencillas que cada trabajador pueda optimizar localmente, de forma que la competición entre los agentes haga tender al sistema hacia un equilibrio que se interpreta como el resultado final.

7. ***Coevolutivos Cooperativos:*** en una arquitectura distribuida, la evolución de cada

subpoblación puede hacerse en un nodo de cómputo distinto. La colaboración entre poblaciones puede llevarse a cabo de una forma similar a la realizada en los modelos de islas, salvo por el hecho de que en estos casos las islas envían soluciones completas, mientras que los *CCEA* distribuidos comunican soluciones parciales.

Como hemos explicado al principio de esta sección, la distribución del cómputo suele conllevar un sobrecoste en la comunicación entre trabajadores. De la exposición realizada sobre los *CCEA* en el capítulo 4, deducimos que este sobrecoste crecerá con la frecuencia de la colaboración entre subpoblaciones, la cual estamos obligados a aumentar cuando nos enfrentamos a problemas complejos y con epistasis alta. En conclusión, los *CCEA* distribuidos requieren la labor adicional de equilibrar una frecuencia de colaboración adecuada con las limitaciones de transmisión de información entre nodos impuestas por la latencia de la red de comunicación.

5.2. Campos aleatorios de Markov y máquinas de Boltzmann

Para hacer el contexto de nuestro experimento lo más asequible posible, empezaremos explicando el concepto de máquina de Boltzmann como caso particular de campo aleatorio de Markov [KS80] y cómo podemos definir un objeto de este tipo que nos permita modelar un problema con la separabilidad deseada.

Sea $G = (V, E)$ un grafo, donde V y E son, respectivamente, los conjuntos de vértices o nodos y de lados del grafo. Dos nodos son vecinos si existe un lado que los une. El conjunto de los vecinos de un vértice v, su vecindad, se denotará N_v.

Asignaremos a cada nodo una variable aleatoria X_v, con $v \in V$, que puede tomar valores en el espacio de estados S_v. Decimos entonces que el conjunto de procesos estocásticos asociados a los vértices del grafo forma un campo aleatorio de Markov si

$$P[X_v = \omega | \{X_{v'} : v' \in V \setminus \{v\}\}] = P[X_v = \omega | N_v] \qquad \forall v \in V, \forall \omega \in S_v.$$

Una máquina de Boltzmann es un tipo de modelo basado en energías (*EBM*), los cuales reciben su nombre por simular procesos propios de la mecánica estadística en los que múltiples agentes interactúan entre ellos para aportar a la energía total del sistema y evolucionar hasta alcanzar un punto de equilibrio [HS86]. En particular, una máquina de Boltzmann es un campo aleatorio de Markov donde cada vértice representa una unidad del sistema que puede estar encendida o apagada, cada lado es la energía aportada al sistema cuando ambas unidades se encuentran encendidas, y la probabilidad de observar una unidad como encendida depende, siguiendo la distribución de Boltzmann, de la energía que aporte al sistema.

La forma específica de esta distribución no es de interés para nuestro experimento. En la literatura sobre máquinas de Boltzmann existen dos tipos de trabajos bien distinguidos. El primero nace del marco de los modelos generativos en aprendizaje automático y se caracteriza principalmente por contar con unos datos de entrada de la red y otros de salida. La finalidad entonces es optimizar los pesos de la red de forma que, dado el mismo vector

de entrada, la máquina pueda generar un vector de salida similar al original con una alta probabilidad. El segundo tipo de experimento consiste en dar unos pesos fijos para la red, que codifican la estructura de una función objetivo que se desea optimizar representada como la energía del sistema. La forma de optimizarla es encontrando aquella configuración de estados que minimice (o maximice) la energía. Nuestro experimento se centrará únicamente en la optimización de la configuración de la red, por lo que el muestreo de la distribución final no nos interesa.

Por lo general, en la optimización de la máquina de Boltzmann se utiliza un algoritmo de enfriamiento simulado en el que la distribución de Boltzmann nos da la probabilidad de activar o desactivar una unidad. Algunos incluso adaptan este método al marco de la coevolución cooperativa, separando las variables del espacio de búsqueda [ORDP98]. Nosotros afrontaremos este problema haciendo uso de los *CCEA* poblacionales, que han sido el centro de nuestra discusión durante la mayor parte de este trabajo, considerando la energía global como función objetivo. Si tenemos m unidades, esta puede representarse como una función $E\colon \{0,1\}^m \to \mathbb{R}$ con la siguiente forma:

$$E(S) = \sum_{i=1}^{m} \sum_{j=1}^{m} W(i,j)S(i)S(j) \tag{5.1}$$

donde S es un vector con la configuración de las unidades y $W(i,j)$ es el peso del lado que conecta los nodos i y j.

Ahora solo falta justificar el interés en usar máquinas de Boltzmann para probar los algoritmos coevolutivos cooperativos. Como hemos visto, el problema de optimización a resolver consiste en encontrar aquella combinación de unidades encendidas y apagadas que optimice la energía del sistema. Dado que la contribución de dos variables, vistas en conjunto, depende únicamente de la energía asociada a su interacción, podemos definir explícitamente las conexiones entre variables a partir de los pesos del grafo. Si el peso del lado que conecta dos nodos tiene un valor absoluto bajo, así debería ser la dependencia que nuestro algoritmo estime entre ambas variables y estas se optimizarán en poblaciones separadas. Además, tras el análisis de conexiones resultará sencillo comprobar visualmente si nuestro algoritmo descompone correctamente el problema.

5.3. Aprendizaje de conexiones y agrupamiento de variables

En esta sección expondremos un método de optimización coevolutiva cooperativa basado en un análisis de las conexiones entre variables previo al proceso de optimización. Para ello, usaremos las definiciones cuantitativas de "interacción" entre dos variables dadas en la sección 4.2.1.2. Las dependencias calculadas servirán, mediante un algoritmo de agrupamiento, para dividir el dominio del problema en distintas poblaciones que evolucionarán independientemente.

5.3.1. Estimación de las dependencias

La **medida de dependencia** utilizada será:

$$\Delta^{i,j} = |\Delta_i f(x) + \Delta_j f(x) - \Delta_{i,j} f(x)|$$

que se calculará tomando una solución inicial aleatoria e invirtiendo el valor de los bits i y j a modo de perturbación. Cada variable x_i solo puede tomar dos valores, 0 o 1, así que Δx_i es siempre igual a ± 1. Deseamos medir la magnitud de la interacción y su signo es irrelevante, por lo que que cuantificaremos la dependencia para cada par de variables tomando valor absoluto.

El motivo de cuantificar las interacciones, en lugar de agrupar directamente aquellas variables con interacción distinta de 0, es que esto nos facilita controlar en mayor medida el detalle de la descomposición dando lugar a un método más robusto. Esto nos permitirá, como veremos en las secciones siguientes, insertar un pequeño ruido a los pesos de la máquina sin alterar notablemente los grupos resultantes.

Haremos un inciso en el coste computacional del proceso de cuantificación de dependencias anterior. Recordamos que $\Delta_i f(x) = f(x_1, ..., x_i + \Delta x_i, ..., x_m) - f(x_1, ..., x_i, ..., x_m)$. Para calcular $\Delta^{i,j}$ necesitaremos, por tanto, 4 evaluaciones de f (en x, perturbando x_i, perturbando x_j y perturbando ambas). Por otro lado, no nos interesa medir la dependencia de una variable consigo misma, y al ser esta simétrica entre cada dos variables solo necesitamos examinar $\frac{m(m-1)}{2}$ parejas de variables. Es posible realizar un análisis más completo tomando varias soluciones iniciales y midiendo la dependencia con distintas perturbaciones aleatorias para cada par de variables (aunque en nuestro caso nos limitaremos al caso más sencillo), por lo que el número de evaluaciones de la función objetivo crece en orden

$$O(3\frac{m(m-1)}{2} + 1) \equiv O(m^2),$$

donde recordamos que m es la dimensión del espacio de soluciones.

Un orden de complejidad cuadrático puede obligarnos a realizar más evaluaciones de la función objetivo de las que podríamos desear. En general, las metaheurísticas permiten detener el proceso de optimización cuando se ha encontrado una solución suficientemente buena o si se alcanza un número máximo de iteraciones preestablecido, devolviendo la mejor solución encontrada hasta entonces. Debemos tener en cuenta, por tanto, que el análisis de dependencias que consideraremos en nuestro experimento no será siempre la mejor opción al requerir un número mínimo de evaluaciones antes de comenzar el proceso de optimización. Para saber más sobre tipos alternativos de descomposición revísese la sección 4.2.1.2.

Por otro lado, un análisis de este tipo puede dar una información relativamente completa sobre el problema con el que se trabaje, especialmente si se sospecha que este pueda compartir estructura con otros problemas similares. Obsérvese que, en caso de buscar el óptimo de una función separable, el proceso de agrupación puede acelerarse notablemente considerando únicamente aquellas variables independientes a las previamente estudiadas (todas las dependientes a una variable dada serán automáticamente agrupadas juntas).

Además, recordemos que en general el tiempo esperado para encontrar la solución óptima será exponencial en el número de variables. Si la cantidad de evaluaciones de la función objetivo disponibles es suficiente, añadir un paso de complejidad cuadrática no debería afectar al comportamiento asintótico del coste total.

Otra ventaja del análisis de dependencias considerado, siempre y cuando el límite de evaluaciones no suponga un problema, es que este es totalmente paralelizable. Distintos análisis completos pueden ejecutarse independientemente contribuyendo a la matriz de dependencias final, o un mismo estudio puede distribuirse a distintos agentes dividiendo las parejas de variables de forma equitativa.

5.3.2. Agrupamiento de variables

A continuación hablaremos del **método de agrupamiento, o *clustering*, de las variables** a partir de las dependencias previamente estimadas. Un algoritmo de *clustering* nos permite formar distintos grupos en los que separar un conjunto de elementos a partir de una medida de "distancia" definida entre ellos, generalmente de forma que los elementos de cada grupo se encuentran "cerca" unos de otros y "lejos" de los del resto de grupos (aunque existen algoritmos con una concepción de *cluster* diferente, como por ejemplo DBSCAN [EKSX96]). En la literatura existen numerosas técnicas de agrupamiento. Algunas de las más conocidas son el K-medias y diversas variantes del *clustering* jerárquico ([HTF01], capítulo 14).

En este caso las interacciones entre variables pueden considerarse como una **"distancia inversa"**, en tanto que deseamos agrupar aquellas variables con una mayor dependencia. Teniendo en cuenta que hemos tomado todas las interacciones como positivas y que algunas pueden ser exactamente iguales a 0, podemos sumar un escalar positivo, por ejemplo 1, a cada elemento de la matriz de dependencias y luego invertir cada uno de ellos. Así obtenemos una nueva matriz donde el valor asociado a cada pareja de variables está correctamente definido en el intervalo $(0, 1]$ y es inversamente proporcional a la dependencia entre dichas variables. Este servirá como medida de distancia entre ambas.

El algoritmo de agrupamiento que usaremos será el *K-medoids* ([HTF01], sección 14.3.10). Este no solo es más robusto que el K-medias cuando la distancia considerada no es la euclídea, sino que permite operar directamente a partir de la matriz de distancias entre los elementos del conjunto de datos.

Comentaremos también los inconvenientes de utilizar el *K-medoids* y posibles soluciones o alternativas para paliar cada uno de ellos:

- Asume que es posible separar el dominio en grupos de variables con poca interacción intergrupal. Esto no es necesariamente posible, por ejemplo, cuando existe un solapamiento entre distintos "bloques" o, más generalmente, cuando hay una jerarquía de dependencias entre todas las variables del dominio. Dependiendo del caso, puede ser más conveniente aplicar un método de *clustering* jerárquico o alguna adaptación de un algoritmo clásico que admita solapamiento entre grupos [BTL16].
- Es obligatorio establecer el número de *clusters* (K) antes de iniciar el algoritmo. De

nuevo, esto puede combatirse cambiando el enfoque del experimento usando un método jerárquico, aunque en ese caso seguiría siendo necesario fijar valor mínimo y máximo para K. En nuestro caso, solucionaremos este problema repitiendo el proceso para varios valores de K y aplicando el comúnmente utilizado **método del codo**. Este consiste en visualizar para cada valor de K algún índice que mida la "bondad" del agrupamiento, habitualmente el coeficiente Silhouette o el índice Calinski-Harabasz. Estas métricas tienden a mejorar al aumentar el número de *clusters*, dado que esto trivialmente reduce la distancia media de los puntos dentro de cada grupo. Sin embargo, si el conjunto de puntos puede dividirse claramente en un cierto número de grupos, estos índices no mejorarán en gran medida una vez se alcance el valor óptimo de K. Visualizando dichos índices ordenados en un gráfico, este fenómeno se manifiesta como un aplanamiento repentino de los resultados con la apariencia de un "codo".

- Su coste de cómputo es de orden $O(m^2)$, mayor al del K-medias. Esto no debería suponer un problema si lo comparamos con el coste del resto del experimento, puesto que no necesita evaluaciones adicionales de la función objetivo y su cálculo en *hardware* moderno para el tamaño de los problemas considerados en este tipo de estudios (menos de 10000 dimensiones) es relativamente rápido. Además cabe destacar que, en caso de buscar agrupar un número más elevado de variables, existen métodos de *clustering* alternativos que permiten sacrificar precisión en los resultados por eficiencia computacional. Un ejemplo es CLARA (*Clustering Large Applications*) [Pop14], que adapta el *K-medoids* tomando subconjuntos aleatorios del total de puntos a agrupar con un tamaño significativamente menor y aplicando *clustering* sobre estos.

En esta sección hemos descrito una opción de implementación de algoritmos coevolutivos cooperativos dividida en dos fases diferenciadas: análisis de dependencias y agrupamiento de variables. El método de *clustering* escogido asume una estructura del problema favorable a la descomposición, puesto que no consideraremos casos más complicados en el estudio empírico de las próximas secciones. Nos gustaría hacer hincapié en que ante la falta de conocimiento experto del problema un análisis exploratorio de la estructura de la matriz de dependencias es fundamental para aplicar un algoritmo del tipo *CCEA* siguiendo el esquema anterior en un contexto general, tanto para la elección de un método de agrupamiento de variables adecuado como, directamente, para decidir si probar una metaheurística basada en la descomposición del dominio antes que cualquier otra de las descritas en la sección 5.1.

5.4. Planteamiento del experimento

El estudio práctico que realizaremos a continuación consistirá en la optimización de una máquina de Boltzmann con interacciones predefinidas entre unidades. Estas se organizarán en bloques, de forma que las unidades del mismo bloque tendrán una interacción base de 1 y las de distintos bloques una de 0. Sobre estos pesos añadiremos un ruido gaussiano obtenido a partir de una distribución normal con media 0 y desviación típica 0.1. De esta forma, la función a optimizar no será estrictamente separable (en el sentido definido en el apartado 4.2.1.1). Esperamos que esto haga conveniente usar métodos de agrupamiento de variables robustos ante interacciones débiles, como el expuesto en la sección anterior.

Consideraremos dos variaciones del mismo problema: una en la que supondremos la separabilidad de la función objetivo inicialmente desconocida, en la que aplicamos análisis de conexiones para estimar la matriz de dependencias y aplicar *K-medoids* para agrupar sus variables, y otra en la que aumentaremos notablemente la dimensión del dominio del problema pero en la que asumiremos conocida su descomposición. En ambos casos contaremos con 8 bloques disjuntos con el mismo número de variables entre sí. La máquina de Boltzmann correspondiente al primer experimento tendrá 800 unidades, mientras la segunda contará con 8000.

Recordamos que la estimación de las dependencias puede mejorarse, en general, repitiendo el cálculo de la matriz considerando distintas perturbaciones sobre las variables. En el caso de la máquina de Boltzmann, sus unidades solo pueden estar "encendidas" o "apagadas", esto es, las variables de la función objetivo solo toman los valores 0 o 1, luego no existe más de una perturbación posible.

En los dos casos compararemos los resultados obtenidos (mejor solución encontrada hasta el momento) de un *CCEA* sencillo con los de su *EA* clásico análogo. Los detalles e hiperparámetros escogidos para el diseño de estos algoritmos son los siguientes:

- Optimización iterativa mediante evaluación de la población, selección, cruce y mutación, en ese orden.
- Selección por torneo ternario para los dos problemas y ambos algoritmos.
- Cruce uniforme: dadas dos soluciones muestreada aleatoriamente con la misma probabilidad de la población previamente seleccionada, cada variable de la solución resultante se escoge o bien del primer progenitor con probabilidad 0.5, o bien del segundo.
- Mutación del 1 % de la población tomado aleatoriamente. Mutar una solución particular consiste en, por cada unidad de la máquina de Boltzmann asociada, "apagarla" si está "encendida" o "encenderla" si está "apagada" con probabilidad $\frac{1}{\text{nº de unidades}}$.
- Almacenamos la mejor solución encontrada durante el proceso iterativo, pero esta no permanece necesariamente en la población.
- El número de individuos por población/subpoblación es de 100 en el primer caso de estudio y 250 en el segundo.
- El número máximo de iteraciones es de 250 para el primer problema y 500 para el segundo. En el caso del *CCEA* dividimos dicho número entre el total de subpoblaciones para no superar la cantidad de evaluaciones de la función objetivo realizadas por el *EA* clásico.
- El *CCEA* tiene tantas subpoblaciones como grupos de variables se consideren. Estas no se comunican entre sí hasta concluir el proceso, pero a la hora de evaluar la mejor solución encontrada hasta el momento se genera un individuo nuevo tomando las variables correspondientes a la mejor solución encontrada en cada subpoblación (nos permitirá analizar la evolución conjunta de las soluciones para compararlas con las del *EA*).

El código utilizado para experimento ha sido implementado en C++ usando la biblioteca estándar para la gestión de las estructuras de datos y se ha ejecutado en el ordenador personal del autor de esta memoria, con Ubuntu 20.04 64-bits como sistema operativo y las siguientes especificaciones técnicas:

Memoria	15,5 GiB
Procesador	Intel® Core™ i7-10750H CPU @ 2.60GHz × 12
Gráficos	NVIDIA Corporation / GeForce RTX 2060/PCIe/SSE2
Capacidad del disco	1,0 TB

Dado que solo le daremos importancia al número de iteraciones y a la bondad de los resultados, si bien el *CCEA* posee la ventaja de ser directamente distribuible al no haber comunicación entre subpoblaciones hasta el final del proceso, no tendremos en cuenta el tiempo de cálculo ni la ganancia de una posible paralelización en nuestro trabajo. Un análisis de estas características se alejaría del tipo de estudio presentado en el resto del proyecto y los resultados obtenidos serían altamente dependientes de la implementación y del *hardware* utilizado.

5.5. Resultados

A continuación se expondrán y analizarán los resultados de los experimentos descritos en la sección anterior. La implementación del código utilizado para llevar a cabo estos experimentos se encuentra recopilado en el siguiente repositorio de Github. Se hace mención especial a la biblioteca `libnpy` y a su creador Leon Merten Lohse, al haber sido de gran utilidad para leer y almacenar los resultados generados desde nuestro código en C++.

5.5.1. Caso 1: 800 variables

En el primer experimento se aplica un análisis de dependencias entre variables. Se espera que la matriz de dependencias calculada se asemeje a la matriz de pesos, generada aleatoriamente siguiendo los pasos descritos en la sección anterior, que define la estructura del problema. Ambas matrices se visualizan en la figura 5.1.

Se observa que aunque estas no coinciden exactamente, puesto que en la práctica la matriz de dependencias se corresponde con los valores absolutos de la matriz de pesos, en el caso separable (o "prácticamente separable") ambas mantienen una estructura con bloques claramente diferenciados en la diagonal, que corresponde con los diferentes grupos de variables mutuamente dependientes.

Sumando 1 a cada elemento de la matriz e invirtiendo el resultado (elevando este a -1) se

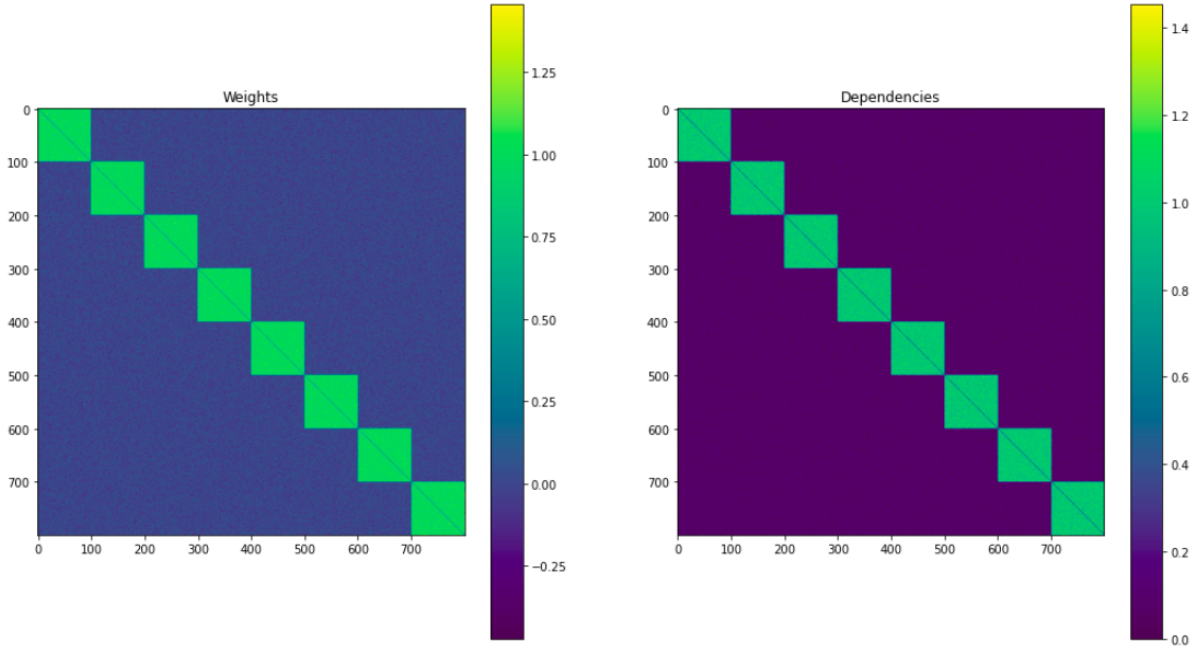


Figura 5.1: Comparación entre los pesos de la máquina de Boltzmann y las dependencias estimadas por nuestro algoritmo. El elemento (x, y) de cada matriz corresponde al peso/dependencia de la conexión entre la unidad/variable x y la y.

obtiene la matriz de la figura 5.2, que se tomará como matriz de distancias entre variables en la aplicación del *K-medoids*.

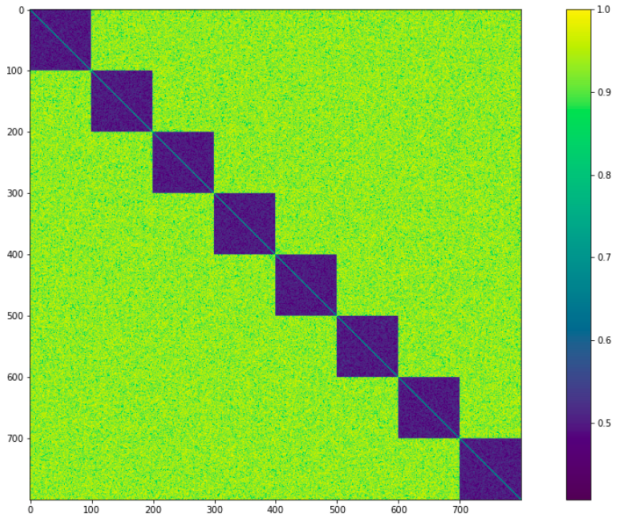


Figura 5.2: Matriz de distancias generada a partir de las dependencias entre cada dos variables.

A continuación se realiza el agrupamiento con *K-medoids*, para lo que se ha hecho uso de la

biblioteca `Biopython`, particularmente del método `kmedoids` en su módulo `Bio.Cluster`. Este devuelve, además del resultado del agrupamiento, la suma de las distancias *intracluster* que se usarán como métrica del error para aplicar el método del codo.

Se repite el agrupamiento con valores de K entre 2 y 14. Ante un problema desconocido en el que no conocemos *a priori* el número de grupos adecuado, este rango podría elegirse inicialmente de forma arbitraria y aumentarse gradualmente si fuese necesario. Los errores alcanzados tras repetir el algoritmo para los valores de K seleccionados pueden verse en la figura 5.3. Se percibe un claro aplanamiento del error, o una parada en su decrecimiento,

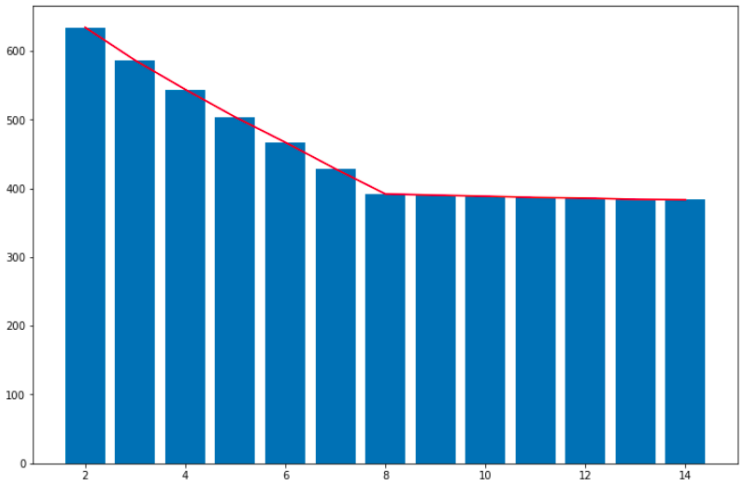


Figura 5.3: Error (suma de distancias *intracluster*) para el agrupamiento realizado en función del valor de K.

al alcanzar los 8 *clusters*. Se aplicará la descomposición definida por dicho agrupamiento que devuelve, de hecho, los 8 grupos de 100 variables altamente dependientes, tal como se esperaba.

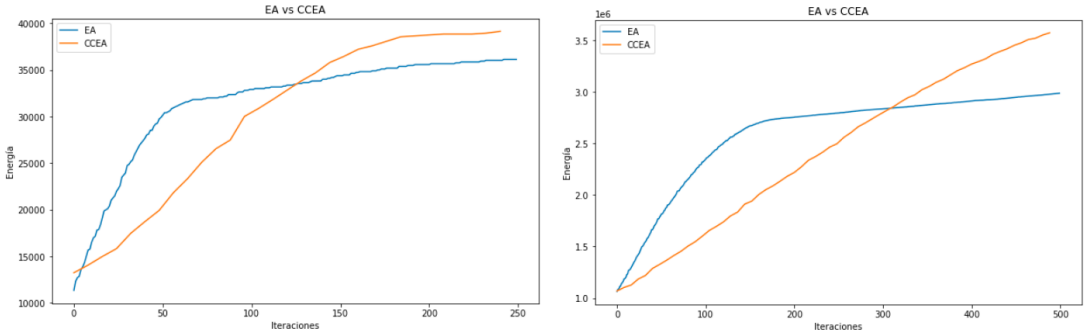
Se pasará a comparar la eficiencia del *CCEA* en comparación con la del *EA* clásico. La figura 5.4 *a)* muestra la evolución de la solución óptima encontrada por ambos algoritmos con respecto al número de iteraciones. Para la visualización de dicha gráfica se considera una iteración del algoritmo coevolutivo por cada 8 de las del clásico, una por cada subpoblación.

Sin necesidad de conocer exactamente la mejor solución se sabe que activar todas las unidades de la red, tal como la hemos definido, debería ser la solución óptima o acercarse a esta con una probabilidad muy alta. Se tomarán, por tanto, la suma de todos los pesos de la máquina de Boltzmann como estimación de la energía máxima de la red, que da un resultado de 39.652'97, muy cercano al valor alcanzado por el *CCEA*.

5.5.2. Caso 2: 8000 variables con descomposición conocida

El segundo experimento se centra solamente en comparar los resultados de ambos algoritmos asumiendo que se conoce de antemano la descomposición correcta del problema. Por

tanto, se obvia la fase de análisis de dependencias y se analiza directamente los resultados en función del número de iteraciones.



a) *b)*

Figura 5.4: Comparación de los algoritmos evolutivo y coevolutivo cooperativo considerando la mejor solución encontrada hasta el momento en función del número de iteraciones para: a) el caso con 800 variables y b) el caso con 8000 variables.

En este caso la suma de todos los pesos de la red, la energía alcanzada encendiendo todas las unidades, resulta en 3.996.690'75. Se observa que la gráfica de la figura 5.4 *b)* no muestra valores tan altos, ni siquiera para el *CCEA*, aunque la tasa de mejora de este último se conserva relativamente alta en las últimas iteraciones así que esto podría paliarse aumentando el tiempo de ejecución del algoritmo. En cualquier caso, pasadas las 300 iteraciones el *CCEA* devuelve resultados mucho mejores a los del *EA* y estos crecen más rápidamente.

En ambos experimentos se aprecia la misma tendencia: el algoritmo evolutivo clásico encuentra mejores soluciones las primeras iteraciones pero se estanca rápidamente, mientras que el *CCEA* conserva la misma tasa de crecimiento durante más iteraciones. Lo primero puede deberse a que el *CCEA* distribuye su avance entre sus subpoblaciones. Para un problema relativamente sencillo, como podría considerarse el que se está tratando en este caso, al principio del proceso es más "fácil" encontrar soluciones mejores a la actual puesto que se parte de una "mala" solución, y aplicar una población distinta para cada parte del dominio resulta contraproducente. Por otro lado, el mejor comportamiento a largo plazo del *CCEA* nace de su menor complejidad computacional, tal como se justificó en la sección 4.1. Cabe destacar que ambos fenómenos podrían tenerse en cuenta en un algoritmo combinado que analice el progreso de la optimización y torne al esquema coevolutivo cuando se observe un decrecimiento del ratio de mejora, como el que puede apreciarse en la figura 5.4 *a)* alrededor de la iteración 50, por ejemplo.

En general, se aprecia que la evolución dirigida por el *CCEA* obedece las pautas predichas por la teoría, y que descomponer un problema de optimización como el estudiado ayuda a encontrar mejores soluciones a largo plazo.

Para demostrar que los resultados anteriores son significativos, el proceso de optimización de ambos experimentos ha sido repetido un total de 10 veces. La tabla 5.1 muestra los resultados finales de los dos algoritmos (*EA* y *CCEA*) para ambos problemas. Puede observarse a simple vista que el *CCEA* alcanza consistentemente mejores soluciones que el *EA* en los

	Experimento 1 (800 variables)		Experimento 2 (8000 variables)	
	EA	*CCEA*	*EA*	*CCEA*
1	36147.145	**39156.688**	2987395.0	**3573788.8**
2	35900.05	**38869.617**	3020216.0	**3567484.5**
3	36095.562	**38968.695**	3003468.8	**3543065.5**
4	35899.684	**38572.645**	3081094.2	**3552216.2**
5	36188.88	**38575.707**	3034023.8	**3550586.0**
6	36178.625	**38089.547**	2938966.5	**3566709.5**
7	35757.566	**38573.996**	2995374.5	**3571347.8**
8	36949.027	**38483.3**	2976987.0	**3556413.0**
9	36154.07	**38672.645**	2963986.8	**3559064.0**
10	35807.727	**38953.832**	2993968.0	**3570172.8**
p-value	0.000157		0.000157	

Tabla 5.1: Resultados finales de aplicar cada algoritmo de optimización sobre los dos problemas considerados. En negrita se marcan las mejores soluciones encontradas. Se ha realizado un test de Kruskal-Wallis para comprobar la significación estadística de la diferencia entre los resultados de los dos algoritmos. El *p*-valor asociado puede verse en la última fila de la tabla, en ambos casos por debajo del 0.01 usado habitualmente para descartar la hipótesis de que ambas muestras (energías alcanzadas por la red) sigan la misma distribución.

problemas considerados.

5.6. Conclusiones

En esta sección se resumirán los elementos principales expuestos en este trabajo y se establecerá una relación entre estos y los resultados obtenidos experimentalmente en este capítulo. También se comentarán algunas ideas y sugerencias para la investigación sobre algoritmos coevolutivos cooperativos en el futuro.

Se ha empezado en el **capítulo 1**, definiendo lo que es un problema de optimización y justificando el uso de algoritmos iterativos y estocásticos de propósito general, las metaheurísticas, entre los cuales los algoritmos evolutivos son una de las variantes más estudiadas y utilizadas. Se vio que estos se caracterizan por ser un tipo de algoritmos poblacionales con una gran versatilidad para incluir mecanismos tanto de "exploración" como de "explotación".

Si se pretende utilizar algoritmos evolutivos para resolver problemas reales lo primero que nos interesa es entender el funcionamiento de cada uno de los operadores que lo componen y saber bajo qué condiciones puede garantizarse que nuestro algoritmo iterativo tenderá a la solución óptima. En el **capítulo 2** se expone un marco teórico desarrollado por Xiaofeng Qi y F. Palmieri que responde en gran medida a estas cuestiones. También se dedica una sección para hablar sobre el *drift analysis*, una técnica ampliamente utilizada para el estudio teórico de la convergencia de metaheurísticas.

El **capítulo 3** es fundamental en el desarrollo de este trabajo, en tanto que recopila los principales obstáculos que pueden encontrarse al afrontar un problema de optimización. La

maldición de la dimensionalidad y los teoremas NFL son la base y la motivación tanto para la definición de los algoritmos coevolutivos cooperativos como para entender qué puede salir mal en su aplicación. En general, en este proyecto se ha tratado de hacer hincapié en el hecho de que no es posible tener un algoritmo realmente "de propósito general", y que siempre que se desee mejorar nuestro rendimiento en la resolución de un problema será explotando conocimiento experto sobre el mismo. Este conocimiento puede obtenerse en el caso de los *CCEA* mediante un análisis de las dependencias entre variables que ayude a descomponer el espacio de soluciones, como se explica posteriormente en la sección 5.3, pero también es posible la aplicación de métodos analíticos. La sección 3.4 trata de mostrar un ejemplo de cómo aplicar herramientas analíticas, en este caso nacidas del *drift analysis*, para estudiar de forma conjunta problemas y algoritmos concretos.

El **capítulo 4** se centra en los algoritmos coevolutivos cooperativos, el tema principal de este proyecto. Estos añaden dos capas adicionales de complejidad al diseño del algoritmo: la descomposición del problema y la selección de colaboradores. No se limita a comentar los distintos mecanismos estudiados en la literatura para implementar ambos elementos del diseño en un *CCEA* particular, sino que en la sección 4.3 se exponen y analizan varios modelos matemáticos existentes en la literatura para entender su comportamiento más en profundidad, incluyendo diversos fenómenos que surgen al descomponer la función objetivo y que dificultan el proceso de búsqueda.

Finalmente, el **capítulo 5** aporta una nueva motivación para el uso de los algoritmos coevolutivos cooperativos, que es su paralelizabilidad. Se habla de las arquitecturas distribuidas y de las principales alternativas a los *CCEA* que existen a día de hoy para diseñar metaheurísticas adaptadas a arquitecturas paralelas. Además, se muestra un ejemplo práctico de problema separable y se comparan los resultados de aplicar un algoritmo evolutivo tradicional con un *CCEA* para resolverlo. Se ha planteado este experimento en torno al concepto de máquina de Boltzmann, al permitir esta definir problemas de optimización con la estructura y la separabilidad que se desee de una forma muy sencilla.

Como ya se ha comentado anteriormente, el hecho de que no siempre convenga descomponer la función a optimizar lleva a enfatizar la importancia de tener cierta información del problema o de la estructura de la función objetivo antes de elegir un método para resolverlo. Es por esto que se considera conveniente dedicar tiempo a estudiar las dependencias entre variables antes de aplicar un *CCEA*, como se ha hecho patente en la metodología seguida en nuestro experimento.

Para concluir, se sugieren algunas ideas que podrían incorporarse a la futura investigación sobre algoritmos coevolutivos cooperativos.

Una primera opción sería añadir complejidad a la máquina de Boltzmann, en un experimento similar al realizado en este capítulo pero que incluya distintos niveles de interacción que obliguen a aplicar algún tipo de agrupamiento jerárquico de las variables y un esquema de colaboración entre subpoblaciones. En general, en lo que a estudios empíricos se refiere, podría ser de interés analizar clases de problemas "de la vida real" desde el punto de vista de su separabilidad a partir de un análisis de dependencias como el visto en este capítulo, por ejemplo. Tener una referencia sobre qué tipo de estructuras de dependencias tienden a formarse en cada familia de problemas constituiría cierto conocimiento experto útil a la hora de elegir o descartar el uso de *CCEAs* en casos particulares. Cabe destacar que, si bien este

proyecto no se ha centrado en detalles sobre los costes computacionales de la comunicación en los algoritmos coevolutivos y su relación con la arquitectura donde se ejecuten, este ya es un tema investigado exhaustivamente en la literatura y defendemos su importancia en el estudio de estos algoritmos.

Por otro lado, los estudios teóricos sobre *CCEAs* desde el punto de vista analítico y matemático son relativamente escasos. Los ejemplos y referencias mostradas en esta memoria sobre *drift analysis* corresponden exclusivamente a metaheurísticas generales y a *EAs* clásicos porque, al menos a nuestro conocimiento, los esfuerzos por integrar los *CCEAs* y el análisis de la deriva son prácticamente inexistentes. Dados los resultados expuestos en este trabajo, consideramos justificado el interés en el estudio teórico de los algoritmos evolutivos y creemos que combinar las últimas metodologías de análisis de los *EAs* con el marco coevolutivo cooperativo es una opción prometedora aún inexplorada. Además, los modelos teóricos futuros sobre coevolución cooperativa deberían tornarse más detallados y cercanos a su uso práctico, incluyendo diversos esquemas de colaboración. Representaciones de este estilo podrían combinarse con enfoques como el visto en [SS04] (sección 5) para su adaptación directa a arquitecturas distribuidas.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[AP93] Peter J. Angeline and Jordan B. Pollack. Competitive environments evolve better solutions for complex tasks, 1993. [Citado en pág. 41]

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. [Citado en pág. 42]

[Bre10] H. Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Universitext. Springer New York, 2010. [Citado en pág. 2]

[BTL16] Said Baadel, Fadi Thabtah, and Joan Lu. Overlapping clustering: A review. In *2016 SAI Computing Conference (SAI)*, pages 233–237, 2016. [Citado en pág. 65]

[BVA07] Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. A review of particle swarm optimization. part i: background and development. *Natural Computing*, 6(4):467–484, July 2007. [Citado en pág. 3]

[Cas15] Noe Casas. A review of landmark articles in the field of co-evolutionary computing, 2015. [Citado en pág. 41]

[CDEL18] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation*, 22(5):707–719, 2018. [Citado en pág. 24]

[CM95] Dave Cliff and Geoffrey F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Proceedings of the Third European Conference on Advances in Artificial Life*, page 200–218, Berlin, Heidelberg, 1995. Springer-Verlag. [Citado en pág. 56]

[CNByM01] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-yates, and José Luis Marroquín. Searching in metric spaces, 2001. [Citado en pág. 31]

[CWM12] Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz, editors. *Variants of Evolutionary Algorithms for Real-World Applications*. Springer Berlin Heidelberg, 2012. [Citado en pág. 29]

[CWYT10] Wenxiang Chen, Thomas Weise, Zhenyu Yang, and Ke Tang. Large-scale global optimization using cooperative coevolution with variable interaction learning. In Robert Schaefer,

Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, pages 300–309, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. [Citado en pág. 30]

[CZH11] Yu Chen, Xiufen Zou, and Jun He. Drift conditions for estimating the first hitting times of evolutionary algorithms. *International Journal of Computer Mathematics*, 88(1):37–50, 2011. [Citado en pág. 13]

[CZY+18] Bin Cao, Jianwei Zhao, Po Yang, Zhihan Lv, Xin Liu, Xinyuan Kang, Shan Yang, Kai Kang, and Amjad Anvari-Moghaddam. Distributed parallel cooperative coevolutionary multi-objective large-scale immune algorithm for deployment of wireless sensor networks. *Future Generation Computer Systems - The International Journal of eScience*, 82:256–267, May 2018. [Citado en pág. 42]

[DGL96] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31 of *Stochastic Modelling and Applied Probability*. Springer, 1996. [Citado en pág. 7]

[DJW12] Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, Feb 2012. [Citado en pág. 23]

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996. [Citado en pág. 3]

[DWT12] Alexandre Devert, Thomas Weise, and Ke Tang. A study on scalable representations for evolutionary optimization of ground structures. *Evolutionary Computation*, 20(3):453–472, 2012. [Citado en pág. 32]

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996. [Citado en pág. 65]

[FKP19] Mahdi Fathi, Marzieh Khakifirooz, and Panos M. Pardalos, editors. *Optimization in Large Scale Problems*. Springer International Publishing, 2019. [Citado en pág. 42]

[GCZ+15] Yue-Jiao Gong, Wei-Neng Chen, Zhi-Hui Zhan, Jun Zhang, Yun Li, Qingfu Zhang, and Jing-Jing Li. Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, 34:286–300, 2015. [Citado en págs. 59 and 60]

[GdS77] H. Goldstein and C.E. de Salamanca. *Mecánica clásica*. Ciencia y Tecnica. Aguilar, 1977. [Citado en pág. 2]

[GKD89] D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Syst.*, 3, 1989. [Citado en pág. 30]

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. [Citado en pág. 41]

[GSOG+16] Pablo García-Sánchez, Julio Ortega, Jesús González, Pedro A. Castillo, and Juan J. Merelo. Addressing high dimensional multi-objective optimization problems by coevolutionary islands with overlapping search spaces. In Giovanni Squillero and Paolo Burelli, editors, *Applications of Evolutionary Computation*, pages 107–117, Cham, 2016. Springer International

Publishing. [Citado en pág. 47]

[HCY15] J. He, T. Chen, and X. Yao. On the easiest and hardest fitness functions. *IEEE Transactions on Evolutionary Computation*, 19(2):295–305, 2015. [Citado en págs. xii, 24, 35, 36, 37, 38, and 39]

[HLM99] F. Herrera, M. Lozano, and C. Moraga. Hierarchical distributed genetic algorithms. *International Journal of Intelligent Systems*, 14(11):1099–1121, 1999. [Citado en pág. 61]

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992. [Citado en pág. 4]

[HS86] G. E. Hinton and T. J. Sejnowski. *Learning and Relearning in Boltzmann Machines*, page 282–317. MIT Press, Cambridge, MA, USA, 1986. [Citado en pág. 62]

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. [Citado en pág. 65]

[HTZ+19] Wenjing Hong, Ke Tang, Aimin Zhou, Hisao Ishibuchi, and Xin Yao. A scalable indicator-based evolutionary algorithm for large-scale multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 23(3):525–537, 2019. [Citado en pág. 2]

[HY01] Jun He and Xin Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001. [Citado en pág. 24]

[HY02] Jun He and Xin Yao. From an individual to a population: an analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511, 2002. [Citado en pág. 13]

[HY04] Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004. [Citado en págs. 21, 22, and 24]

[HY16] Jun He and Xin Yao. Average drift analysis and population scalability, 2016. [Citado en págs. 21 and 24]

[HYT21] Wen-Jing Hong, Peng Yang, and Ke Tang. Evolutionary computation for large-scale multi-objective optimization: A decade of progresses. *International Journal of Automation and Computing*, 18(2):155–169, January 2021. [Citado en pág. 2]

[IYAN13] Hisao Ishibuchi, Masakazu Yamane, Naoya Akedo, and Yusuke Nojima. Many-objective and many-variable test problems for visual examination of multiobjective search. In *2013 IEEE Congress on Evolutionary Computation*, pages 1491–1498, 2013. [Citado en pág. 2]

[Jan13] Thomas Jansen. *Analyzing Evolutionary Algorithms*. Springer Berlin Heidelberg, 2013. [Citado en págs. 48, 49, 50, and 51]

[Joh10] Daniel Johannsen. Random combinatorial structures and randomized search heuristics, 2010. [Citado en pág. 23]

[JP96] Hugues Juille and Jordan B. Pollack. Co-evolving intertwined spirals. In *in Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 461–468. MIT Press, 1996. [Citado en pág. 41]

[JW04] T. Jansen and R. P. Wiegand. The cooperative coevolutionary (1+1) ea. *Evolutionary*

Computation, 12(4):405–434, 2004. [Citado en pág. 48]

[K̈] Mario Köppen. The curse of dimensionality. [Citado en págs. 31 and 32]

[KBLB20] Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4):1853–1859, 2020. [Citado en pág. 2]

[KM17] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining*, pages 314–315. Springer US, 2017. [Citado en pág. 31]

[KS80] Ross Kindermann and J. Laurie Snell. *Markov Random Fields and Their Applications*. American Mathematical Society, 1980. [Citado en pág. 62]

[KS05] Frances Y. Kuo and Ian H. Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52:1320–1329, 2005. [Citado en pág. 32]

[Len18] Johannes Lengler. Drift analysis, 2018. [Citado en págs. 21, 22, 23, 24, and 25]

[LS17] Johannes Lengler and Angelika Steger. Drift analysis and evolutionary algorithms revisited, 2017. [Citado en pág. 21]

[MLZ+19] X. Ma, X. Li, Q. Zhang, K. Tang, Z. Liang, W. Xie, and Z. Zhu. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, 2019. [Citado en págs. xv, 2, 42, 45, 46, 48, 56, and 57]

[MPS+20] Daniel Molina, Javier Poyatos, Javier Del Ser, Salvador García, Amir Hussain, and Francisco Herrera. Comprehensive taxonomies of nature- and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis and recommendations, 2020. [Citado en pág. 4]

[Mü97] Heinz Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3):303–346, 1997. [Citado en pág. 14]

[NCFL13] Mariela Nogueira, Carlos Cotta, and Antonio J. Fernández-Leiva. An analysis of hall-of-fame strategies in competitive coevolutionary algorithms for self-learning in rts games. In Giuseppe Nicosia and Panos Pardalos, editors, *Learning and Intelligent Optimization*, pages 174–188, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. [Citado en pág. 41]

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006. [Citado en pág. 2]

[OHY09] Pietro S. Oliveto, Jun He, and Xin Yao. Analysis of the $(1+1)$-ea for finding approximate solutions to vertex cover problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1006–1029, 2009. [Citado en pág. 27]

[OLMY14] M. N. Omidvar, X. Li, Y. Mei, and X. Yao. Cooperative co-evolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393, 2014. [Citado en págs. 43 and 45]

[ORDP98] Julio Ortega, Ignacio Rojas, Antonio F. Diaz, and Alberto Prieto. *Neural Processing Letters*, 7(3):169–184, 1998. [Citado en págs. xv and 63]

[oST] National Institute of Standards and Technology. Digital library of mathematical functions. volume of n-dimensional ball. https://dlmf.nist.gov/5.19. [Citado en pág. 32]

[OYM+17] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao. Dg2: A faster and more accurate differential grouping for large-scale black-box optimization. *IEEE Transactions on Evolutionary Computation*, 21(6):929–942, 2017. [Citado en pág. 45]

[Pan10] Liviu Panait. Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evol. Comput.*, 18(4):581–615, December 2010. [Citado en págs. xv, 47, 48, 51, 52, 53, 54, 55, and 56]

[PB98] Jordan B. Pollack and Alan D. Blair. *Machine Learning*, 32(3):225–240, 1998. [Citado en pág. 41]

[pClYS+07] Ying ping Chen, Tian li Yu, Kumara Sastry, David E. Goldberg, Ying ping Chen, Tian li Yu, Kumara Sastry, and David E. Goldberg. A survey of linkage learning techniques in genetic and evolutionary algorithms, 2007. [Citado en pág. 30]

[PDJ94a] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature — PPSN III*, pages 249–257, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. [Citado en págs. xv and 41]

[PDJ94b] Mitchell A. Potter and Kenneth A. De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature — PPSN III*, pages 249–257, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. [Citado en págs. 42 and 44]

[PGCP99] M. Pelikan, D. Goldberg, and E. Cantú-Paz. Boa: the bayesian optimization algorithm. 1999. [Citado en págs. 30, 45, and 46]

[PGT03] M. Pelikan, D.E. Goldberg, and S. Tsutsui. Hierarchical bayesian optimization algorithm: toward a new generation of evolutionary algorithms. In *SICE 2003 Annual Conference (IEEE Cat. No.03TH8734)*, volume 3, pages 2738–2743 Vol.3, 2003. [Citado en pág. 47]

[PL02] Liviu Panait and Sean Luke. A comparison of two competitive fitness functions. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, page 503–511, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. [Citado en pág. 41]

[Pop14] Shraddha K. Popat. Review and comparative study of clustering techniques. 2014. [Citado en pág. 66]

[QM04] Qingfu Zhang and H. Muhlenbein. On the convergence of a class of estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):127–136, 2004. [Citado en págs. 7, 8, 12, 13, 29, 46, and 47]

[RB97] Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evol. Comput.*, 5(1):1–29, March 1997. [Citado en pág. 41]

[SHC+17] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. [Citado en pág. 2]

[SHS+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017. [Citado en pág. 41]

[SP97] Rainer Storn and Kenneth Price. *Journal of Global Optimization*, 11(4):341–359, 1997. [Citado en pág. 5]

[SS04] R. Subbu and A. C. Sanderson. Modeling and convergence analysis of distributed coevolutionary algorithms. *Trans. Sys. Man Cyber. Part B*, 34(2):806–822, April 2004. [Citado en págs. xv, 41, 42, 54, and 74]

[Sto96] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing*, pages 519–523, 1996. [Citado en pág. 5]

[SY00] T. Schnier and Xin Yao. Using multiple representations in evolutionary algorithms. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 479–486 vol.1, 2000. [Citado en pág. 29]

[WCH14] Xingwei Wang, Hui Cheng, and Min Huang. Qos multicast routing protocol oriented to cognitive network using competitive coevolutionary algorithm. *Expert Systems with Applications*, 41(10):4513–4528, 2014. [Citado en pág. 41]

[WCT12] Thomas Weise, Raymond Chiong, and Ke Tang. Evolutionary optimization: Pitfalls and booby traps. *Journal of Computer Science and Technology*, 27(5):907–936, September 2012. [Citado en págs. xv, 27, and 31]

[WHB98] D. Wiesmann, U. Hammel, and T. Back. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):162–167, 1998. [Citado en pág. 29]

[Wie03] R. P. Wiegand. An analysis of cooperative coevolutionary algorithms a dissertation submitted in partial fulfillment of the requirements for the degree of doctor of philosophy at george mason university. 2003. [Citado en págs. xv and 54]

[WM97] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. [Citado en págs. xv, 32, 33, and 34]

[WP06] R. Paul Wiegand and Mitchell A. Potter. Robustness in cooperative coevolution. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, page 369–376, New York, NY, USA, 2006. Association for Computing Machinery. [Citado en págs. 54 and 56]

[XP94a] Xiaofeng Qi and F. Palmieri. Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. part i: Basic properties of selection and mutation. *IEEE Transactions on Neural Networks*, 5(1):102–119, 1994. [Citado en págs. xv, 7, 8, 9, and 16]

[XP94b] Xiaofeng Qi and F. Palmieri. Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space. part ii: Analysis of the diversification role of crossover. *IEEE Transactions on Neural Networks*, 5(1):120–129, 1994. [Citado en págs. xv, 7, 16, 17, 18, and 21]

[YGS+09] Tian-Li Yu, David E. Goldberg, Kumara Sastry, Claudio F. Lima, and Martin Pelikan. Dependency structure matrix, genetic algorithms, and effective recombination. *Evolutionary Computation*, 17(4):595–626, 2009. [Citado en págs. 30, 45, 46, and 47]