# Seminario de Investigación II

# Introducción

El Algoritmo de Optimización por Forrajeo Bacteriano (BFOA, por sus siglas en inglés) es un enfoque innovador para resolver problemas de optimización complejos, inspirado en el comportamiento de forrajeo de la bacteria \*Escherichia coli\*. Este algoritmo emplea procesos naturales como la quimiotaxis, la reproducción y la eliminación-dispersión para explorar y explotar soluciones potenciales dentro de un espacio de búsqueda definido. Al imitar las estrategias adaptativas utilizadas por las bacterias en su búsqueda de nutrientes, BFOA presenta ventajas significativas en términos de robustez y capacidad de cómputo paralelo. Gracias a su eficacia en la resolución de desafíos de optimización diversos, ha ganado aceptación en múltiples campos, incluyendo el análisis de redes y la planificación de tareas.

El desarrollo de BFOA representa un avance importante en el ámbito de la inteligencia de enjambre y las técnicas de optimización, ya que se basa en principios biológicos para navegar a través de espacios de solución complejos en busca de resultados óptimos. Su fundamento teórico incluye un análisis matemático que explica sus mecanismos esenciales, en particular el movimiento quimiotáctico, el cual permite a las bacterias explorar su entorno de manera eficiente mientras evitan sustancias perjudiciales.

Uno de los atributos más destacados del BFOA es su insensibilidad a las condiciones iniciales y su capacidad para ejecutar procesos de cómputo en paralelo. Sin embargo, a pesar de sus fortalezas, también presenta ciertas limitaciones. Una de las más notables es su velocidad de convergencia, que suele ser lenta, además de enfrentar dificultades al abordar problemas de alta dimensionalidad o con paisajes de solución intrincados. Para superar estas deficiencias, los investigadores han desarrollado diversas estrategias para mejorar el rendimiento del algoritmo. Estas mejoras se han implementado principalmente a través de la hibridación con otras técnicas de optimización, las cuales pueden clasificarse en enfoques basados en población y enfoques basados en individuos. Estos métodos buscan refinar los parámetros o la estructura del algoritmo para incrementar su eficiencia y aplicabilidad en contextos más exigentes.

Además, en los últimos años, el BFOA ha sido ampliamente estudiado en combinación con técnicas de inteligencia artificial y aprendizaje automático, lo que ha permitido su implementación en problemas aún más desafiantes. Por ejemplo, se ha aplicado en la optimización de redes neuronales y en la selección de características para modelos predictivos, lo que demuestra su versatilidad en la mejora de procesos computacionales avanzados. Estas investigaciones continúan evolucionando, buscando integrar el BFOA con enfoques modernos que potencien su eficacia y amplíen su alcance en diversas disciplinas científicas y tecnológicas.

# Análisis y descripción del algoritmo

Clase fastaReader.

La clase está diseñada para leer archivos FASTA, un formato utilizado para representar secuencias de ADN, ARN o proteínas. Esta clase almacena: identificadores de las secuencias en self.names y las secuencias de ADN/ARN/proteínas en self.seqs.

Clase evaluadorBlosum.

Está clase fue diseñada para trabajar con la matriz de sustitución, que en la bioinformatica se usa para medir similitudes entre secuencias de aminoácidos. La matriz asigna puntuaciones, reflejando la probabilidad de que estos cambios ocurran de manera evolutiva. Métodos:

* \_\_init\_\_(self):

a. El constructor de la clase:

i. Carga la matriz utilizando la función bl.BLOSUM(62) de la biblioteca blosum.

ii. Asigna esta matriz al atributo self.matrix, permitiendo que otros métodos accedan a ella.

* showMatrix(self)

a. Este método imprime la matriz completa

* getScore(self, A, B):

a. Entrada: Recibe dos aminoácidos A y B, que deben ser caracteres válidos que representan aminoácidos

b. Busca la puntuación de sustitución en la matriz en la posición correspondiente a A y B.

c. Devuelve el valor de la matriz, que representa la probabilidad de sustitución entre los aminoácidos A y B de acuerdo con la matriz BLOSUM.

d. Salida: Devuelve un número entero, que es la puntuación de sustitución entre A y B.

Clase bacteria.

Esta clase representa una bacteria en el contexto del Algoritmo de Optimización por Forrajeo Bacteriano (BFOA). Está diseñada para gestionar la población de bacterias, evaluar su desempeño mediante la matriz BLOSUM y calcular interacciones entre ellas para mejorar la solución de optimización. Métodos:

* \_\_init\_\_(self, numBacterias)

Constructor de la clase. Crea varias listas administradas por multiprocessing.Manager() para manejar datos como puntuaciones BLOSUM, tablas de atracción y repulsión, interacciones, aptitud (fitness) y evaluaciones (NFE).

* resetListas(self, numBacterias)

Reinicia las listas de datos de la clase utilizando Manager() para volver al estado inicial con un nuevo número de bacterias.

* cuadra(self, numSec, poblacion)

Asegura que todas las secuencias en la población tengan la misma longitud. Si una secuencia es más corta, se rellena con caracteres de gap "-".

* limpiaColumnas(self)

Recorre la matriz y elimina las columnas donde todos los elementos sean gaps.

* deleteCulmn(self, pos)

Elimina una columna específica de la matriz.

* gapColumn(self, col)

Verifica si una columna está compuesta únicamente por gaps.

* tumbo(self, numSec, poblacion, numGaps)

Introduce un número determinado de gaps en posiciones aleatorias de la población.

* creaGranListaPares(self, poblacion)

Para cada bacteria en la población, genera una lista de pares únicos de caracteres en cada columna de sus secuencias.

* evaluaFila(self, fila, num)

Utiliza evaluadorBlosum para calcular la puntuación de una fila en la matriz BLOSUM.

* evaluaBlosum(self)

Usa Pool() de multiprocessing para evaluar en paralelo la puntuación BLOSUM de todas las bacterias.

* getColumn(self, bacterTmp, colNum)

Extrae una columna específica de una bacteria.

* obtener\_pares\_unicos(self, columna)

Genera una lista de pares únicos de caracteres en una columna.

* compute\_diff(self, args)

Calcula la diferencia cuadrática entre las puntuaciones BLOSUM de una bacteria y las demás.

* compute\_cell\_interaction(self, indexBacteria, d, w, atracTrue)

Evalúa la interacción entre bacterias utilizando atracción y repulsión.

* creaTablaAtract(self, poblacion, d, w)

Calcula la tabla de atracción entre bacterias.

* creaTablaRepel(self, poblacion, d, w)

Calcula la tabla de repulsión entre bacterias.

* creaTablasAtractRepel(self, poblacion, dAttr, wAttr, dRepel, wRepel)

Ejecuta en paralelo los métodos de atracción y repulsión.

* creaTablaInteraction(self)

Suma las tablas de atracción y repulsión para calcular la interacción total.

* creaTablaFitness(self)

Calcula la aptitud de cada bacteria sumando su interacción y puntuación BLOSUM.

* getNFE(self)

Devuelve la suma total de evaluaciones realizadas (NFE).

* obtieneBest(self, globalNFE)

Encuentra la mejor bacteria según su fitness.

* replaceWorst(self, poblacion, best)

Reemplaza la peor bacteria por una copia de la mejor.

Clase principal parallel\_BFOA.

El archivo parallel\_BFOA.py implementa el BFOA utilizando la clase bacteria.

* Inicialización

Define parámetros como número de bacterias, iteraciones y valores de atracción/repulsión.

1. numeroDeBacterias: Cantidad de bacterias en la población.
2. numRandomBacteria: Define cuántas bacterias se seleccionan al azar en ciertos procesos.
3. iteraciones: Número de ciclos de optimización que se ejecutarán.
4. tumbo: Cantidad de gaps a insertar en las secuencias durante la fase de mutación.
5. nado: Parece hacer referencia a otro mecanismo de optimización (aunque en este código no se usa).
6. dAttr y wAttr: Determinan la intensidad de la atracción entre bacterias.
7. hRep y wRep: Determinan la repulsión entre bacterias para evitar convergencia prematura.

* poblacionInicial()

Genera una población inicial de bacterias basadas en secuencias de un archivo FASTA.

* Ejecución del algoritmo

Realiza iteraciones en las que se aplican las operaciones de tumbo, cuadra, evaluación BLOSUM y ajuste de interacciones.

Se identifica la mejor bacteria y se reemplaza la peor con su copia.

* Resultados

Al final, imprime la mejor solución encontrada junto con el tiempo de ejecución.

# Referencias

* Li, J., Wang, Y., & Zhang, H. (2020). Advances in Bioinformatics: A Review on Algorithms for Genomic Data Analysis. \*Journal of Computational Biology\*, 27(10), 1457-1472. <https://doi.org/10.3390/ijms24010004>.
* (2013). *[PDF] A review of the bacterial foraging algorithm in constrained numerical ....* Retrieved from <https://www.uv.mx/personal/emezura/files/2013/03/PID2771049.pdf>. DOI:10.1109/CEC.2013.6557895
* (2023). *[PDF] bacteria foraging optimization algorithm (bfoa) for thd suppression in ....* Retrieved from <https://www.journalijisr.com/sites/default/files/issues-pdf/IJISRR-1342.pdf>. SSN: 2582-6131
* Mokhlis, H. (2024). *Improved Bacterial Foraging Optimization Algorithm with Machine ....* Retrieved from [**https://doi.org/10.3390/a17110510**](https://doi.org/10.3390/a17110510)
* Zafar, M., & Jhala, A. K. (2014). *Paper Detail - IJTRD.* Retrieved from <https://www.ijtrd.com/ViewFullText.aspx?Id=22745>.