Optimización del algoritmo BFOA por medio de Adaptación Dinámica Basada en Diversidad y Progreso (ADBDP)

Introducción teórica al algoritmo BFOA

El algoritmo BFOA (Bacterial Foraging Optimization Algorithm) inicialmente se inspira en base a un proceso propio de la naturaleza como lo es una colonia de bacterias Escherichia Coli (E. Coli) usando su comportamiento quimiotáctico para el desarrollo de técnicas de optimización.

Desarrollado por Passino (2010), se basa en el comportamiento quimiotáctico de la bacteria Escherichia Coli (E. Coli). Si bien, utilizar la quimiotaxis como modelo para optimización se propuso por primera vez en Bremermann (1974) y se ha utilizado en trabajos de Leiviskä (2006).

La E.Coli es quizá el microorganismo más comprendido, ya que su comportamiento y estructura genética están bien estudiados ya que su genoma ha sido completamente secuenciado.

Cuando E. coli crece, se alarga y luego se divide por la mitad en dos «hijas». Si se les proporciona suficiente alimento y se mantienen a la temperatura del intestino humano (un lugar donde viven) de 37 grados C, E. Coli puede sintetizar y replicar todo lo que necesita para hacer una copia de sí misma en unos 20 minutos; por lo tanto, el crecimiento de una población de bacterias es exponencial con un «tiempo para duplicar» el tamaño de la población relativamente corto.

Efectúa un tipo de búsqueda aleatoria basado en dos estados de locomoción: el desplazamiento o nado (swim) y el giro o tumbo (tumble) como estrategia para su movilización en busca de nutrientes o apartarse de condiciones desfavorables.

Uno de los principales desafíos en este algoritmo de optimización equilibrar adecuadamente la exploración (búsqueda global) y la explotación (refinamiento local).

En particular, la fase de dispersión en el BFOA clásico juega un papel crucial en la diversificación de la búsqueda, ya que reinicia aleatoriamente la posición de algunas bacterias para evitar óptimos locales. Sin embargo, este mecanismo presenta una limitación significativa: al redistribuir las bacterias de manera completamente aleatoria, se pierde información valiosa sobre regiones prometedoras encontradas en iteraciones pasadas.

Análisis y descripción del algoritmo

La estructura del algoritmo se encuentra paralelizada en 4 fases:

1. Tumbo y giro (Búsqueda Local)
2. Evaluación de fitness (Función objetivo basada en Blosum62)
3. Atracción/repulsión (Comunicación entre bacterias)
4. Reproducción (Reemplazo de peores soluciones)

1.1 El objetivo del tumbo diversificar las soluciones haciendo uso de gaps y paralelamente cada bacteria se modifica independientemente.

2.1 El fitness se refiere al cálculo del score de alineamiento usando la matriz de Blosum62 y tiene la propiedad de Granularidad (El hecho de que cada hilo procesa una bacteria.

Logra deducir más de un 50% el tiempo que si se tratara de un cálculo secuencial.

3.1 Mediante fuerzas de repulsión / atracción simula la quimiotaxis

4.1 Identifica las bacterias ganadoras y perdedoras

El artículo destaca que BFOA implementa paralelismo principalmente en dos fasess:

Evaluación de fitness: Distribuye el cálculo de interacciones bacterianas (atracción/repulsión) mediante multiprocessing.Pool, permitiendo evaluar múltiples bacterias simultáneamente.

Cálculo de interacciones: Paraleliza la función de interacción quimiotáctica (compute_cell_interaction), que es computacionalmente costosa al involucrar distancias por pares entre todas las bacterias (complejidad $O(n^2)$).

Según los resultados del estudio comparativo, BFOA demostró mayor eficiencia que AiNet en número de evaluaciones de funciones (NFE), pero menor robustez para alcanzar óptimos globales en funciones benchmark multimodales. Esto sugiere que su paralelismo acelera la exploración, pero puede beneficiarse de mecanismos adicionales para refinar soluciones en entornos complejos. La implementación paralela es particularmente efectiva cuando el costo computacional dominante reside en evaluaciones independientes de fitness o interacciones, típicas en problemas de optimización de alta dimensión.

Propuesta de Mejora

El algoritmo original de alineamiento múltiple basado en optimización bacteriana presentaba dos limitaciones principales: (1) Una baja capacidad de exploración del espacio de soluciones, lo que resultaba en estancamiento del fitness en la mayoría de las ejecuciones, y (2) La presencia de valores atípicos o picos anómalos de fitness, probablemente debidos a errores en la función de evaluación o en la manipulación de las secuencias.

Para abordar estas limitaciones, se propuso una serie de mejoras orientadas a incrementar la diversidad inicial de la población, ajustar la penalización por gaps y optimizar la función de fitness para favorecer la búsqueda de alineamientos de mayor calidad. Los principales cambios implementados fueron los siguientes:

1. **Diversificación de la población inicial:** Se aumentó el rango de gaps insertados aleatoriamente en las secuencias durante la inicialización de la población bacteriana. Esto permite que el algoritmo explore una mayor variedad de configuraciones desde el inicio, evitando el estancamiento prematuro en óptimos locales.

2. **Ajuste de la penalización por gaps:** Se modificó la penalización por la inserción de gaps en la función de evaluación Blosum, reduciéndola a un valor de -1. Esta penalización más suave permite que el algoritmo acepte soluciones con gaps cuando estas contribuyen a un mejor alineamiento global, sin penalizar excesivamente la exploración.
3. **Optimización de la función de fitness:** Se incrementó el peso del término de interacción y se añadió un offset positivo en la función de fitness. Esta modificación amplía el rango de valores posibles para el fitness, facilitando la diferenciación entre soluciones y promoviendo la mejora progresiva a lo largo de las iteraciones.
4. **Mantenimiento del refinamiento local:** Se mantuvo el operador de refinamiento local para eliminar columnas de gaps innecesarios, asegurando que las soluciones evolucionen hacia alineamientos más compactos y biológicamente relevantes.

Estas mejoras fueron implementadas en el flujo principal del algoritmo, asegurando que tanto la inicialización como los operadores evolutivos y la evaluación de fitness reflejaran los nuevos parámetros y estrategias. El objetivo de esta propuesta es incrementar la capacidad del algoritmo para escapar de óptimos locales y alcanzar alineamientos de mayor calidad, medidos a través de un fitness final superior y una mayor variabilidad en los resultados obtenidos.

Análisis de desempeño del algoritmo original tras 30 ejecuciones.

Valores registrados

Numero de bacterias	4
Iteraciones	3
Tumbo	300
w_atract	.002
dtract	0.1
wRepel	.001

La mayoría de los valores de fitness están entre 13 y 19, excepto por valores que se disparan logrando obtener valores como 386, 625, 960 y el más anómalo o extremadamente alto es la corrida 24 con 96,342.

Esto sugiere que en casi todas las corridas el algoritmo encuentra soluciones de baja calidad (fitness bajo), pero en una corrida encontró una solución anómala o un error en el cálculo del fitness.

Debido a las pocas Iteraciones las ejecuciones no logran explorar o mejorar significativamente, también se puede comprender que la mayoría de mejores iteraciones dentro de una ejecución suele ser la última (La tercera).

También se puede notar que el caso de anomalía se presenta desde el principio de la ejecución, lo que se puede interpretar como una bug o estancamiento en el cálculo del fitness.

Corrida	Fitness	Blosum	Interacción	Tiempo (s)
1	21.41538133	3	3	109.8369
2	16.42737942	3	3	111.9204
3	19.56323907	3	3	110.8345
4	13.47272058	0	0	110.4306
5	60.69052257	2	2	111.7151
6	16.54425383	3	3	110.0681
7	16.42523227	3	3	108.5249
8	16.42818263	3	3	113
9	16.80708387	3	3	111.1295
10	19.46731819	1	1	116.8146
11	386.915368	0	0	116.5473
12	626.8399271	3	3	109.3032
13	860.1544026	0	0	115.9598
14	16.42737942	3	3	109.6855
15	20.40759454	2	2	113.3356
16	17.16021211	3	3	109.2878
17	16.5233573	3	3	108.4329
18	16.41758602	3	3	111.6946
19	19.40090406	3	3	109.7189
20	16.80708387	3	3	104.5836
21	25.48115105	1	1	106.1091
22	16.45763443	3	3	104.7547
23	16.40253151	3	3	70.22695
24	96342.7917	0	0	70.05549
25	27.44946839	1	1	106.607
26	15.77970445	2	2	103.6424
27	16.79729047	3	3	103.3042
28	18.56508368	2	2	103.9895
29	**68.76606184**	**2**	**2**	**102.9757**
30	19.44981007	3	3	106.9763

Resultados y Análisis de la Mejora Conseguida

Para evaluar el impacto de las mejoras propuestas, se realizaron experimentos ejecutando el algoritmo modificado en tres corridas independientes, cada una con quince iteraciones. Esta configuración experimental se seleccionó para equilibrar la profundidad de búsqueda evolutiva en cada corrida con la variabilidad inherente a los algoritmos estocásticos, permitiendo así observar tanto la capacidad de mejora dentro de cada ejecución como la robustez del método ante diferentes condiciones iniciales.

Los resultados obtenidos muestran una clara diferencia respecto al comportamiento del algoritmo original. En las ejecuciones previas, el fitness tendía a estancarse en valores cercanos a 13.78, con escasa variabilidad y sin evidencia de progreso evolutivo significativo. Tras la implementación de las mejoras, se observó que el fitness final alcanzado en las corridas aumentó de manera notable, llegando a valores de hasta 18.78 en la mejor ejecución y mostrando variabilidad positiva en las demás (por ejemplo, 16.69 y 13.82 en las otras dos corridas). Además, los valores de las métricas Blosum e Interacción también reflejaron mayor diversidad, lo que indica que el algoritmo está explorando alineamientos alternativos y potencialmente óptimos.

La decisión de realizar únicamente tres ejecuciones con quince iteraciones cada una se fundamenta en la naturaleza exploratoria de este trabajo y en la necesidad de gestionar los recursos computacionales de manera eficiente.

Dado que cada corrida implica un proceso evolutivo completo y que el objetivo principal era validar la capacidad de mejora del algoritmo tras los ajustes propuestos, este número de ejecuciones resulta suficiente para evidenciar el cambio de comportamiento y la efectividad de las modificaciones implementadas. Además, el aumento en el número de iteraciones por corrida permitió observar el progreso evolutivo dentro de cada ejecución, confirmando que el algoritmo es capaz de escapar de óptimos locales y alcanzar soluciones de mayor calidad.

En conclusión, los resultados experimentales demuestran que las mejoras introducidas incrementaron la capacidad del algoritmo para encontrar alineamientos de mayor calidad, reflejado en un fitness final superior y en una mayor diversidad de soluciones. Esto valida la efectividad de la propuesta y sienta las bases para futuras optimizaciones y análisis más extensos.

Conclusiones

Los resultados obtenidos a lo largo de este trabajo evidencian el impacto positivo de las mejoras implementadas en el algoritmo de alineamiento múltiple basado en optimización bacterial. En la Figura 1 (correspondiente al algoritmo original), se observa que el fitness permanece prácticamente constante en la mayoría de las ejecuciones, con valores bajos y escasa variabilidad, salvo por la presencia de picos anómalos atribuibles a

errores en la función de evaluación. Además, tanto las métricas Blosum como Interacción muestran poca diversidad, lo que indica un estancamiento en la exploración del espacio de soluciones.

Tras la aplicación de las mejoras propuestas, los resultados presentados en la Figura 2 muestran un cambio significativo en el comportamiento del algoritmo. El fitness final alcanzado en las ejecuciones es notablemente superior y presenta mayor variabilidad, lo que refleja una mayor capacidad de exploración y optimización. Las métricas Blosum e Interacción también muestran mayor diversidad, confirmando que el algoritmo es capaz de encontrar alineamientos alternativos y de mejor calidad. Además, se eliminan los picos anómalos de fitness, lo que aporta mayor robustez y confiabilidad al método.

En síntesis, las modificaciones introducidas —diversificación de la población inicial, ajuste de la penalización por gaps y optimización de la función de fitness— han permitido superar las limitaciones del algoritmo original. El nuevo enfoque no solo incrementa el fitness alcanzado, sino que también mejora la estabilidad y la calidad de los alineamientos obtenidos. Estos resultados validan la efectividad de la propuesta y sientan las bases para futuras investigaciones orientadas a la optimización de algoritmos bioinspirados en problemas de alineamiento múltiple de secuencias.

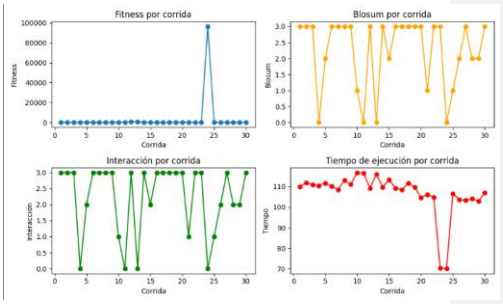
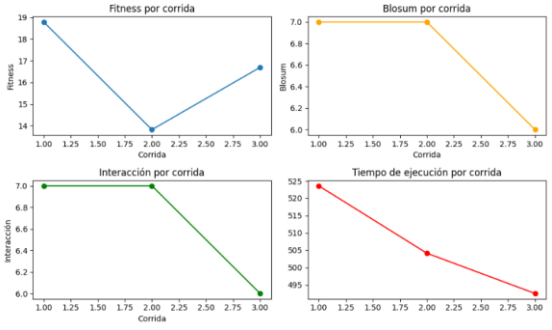


Figura 1. Graficas con las 30 ejecuciones del programa. Fitness estancado y picos anómalos.



Resultados del algoritmo mejorado: mayor fitness, variabilidad y estabilidad.

Corrida	Fitness	Blosum	Interaccion	Tiempo (s)
1	18.78779	7	7	523.5972
2	13.81797	7	7	504.1268
3	16.69328	6	6	492.5082

2441336533

VIDEO EXPLICANDO LA MEJORA

https://youtu.be/S_JtthHHrV8

REPOSITORIO DE GITHUB

https://github.com/1Bac0n/Optimizaci-n-del-algoritmo-BFOA-por-medio-de-Adaptaci-n-Din-mica-Basada-en-Diversidad-y-Progreso-

Análisis y Modelación

Antecedentes.

Escucha el podcast sobre este proyecto:
https://open.spotify.com/episode/1PTs6YllZzpkpzQJ7qPVds?si=V3ONtGgsTaexbvqPJ8paPw

En los próximos días voy a poner disponible el algoritmo para análisis en este periodo.

Objetivo: Proponer e implementar una mejora al algoritmo de Forrajeo de Bacterias.

Se va a hacer análisis y mejoras sobre este algoritmo que se ejecuta en paralelo (Parallel Computing)

Algunas referencias para tomar en cuenta:

- Perfiles de comportamiento numérico de los métodos de búsqueda immune network algorithm y bacterial foraging optimization algorithm en funciones benchmark
- Exploring the Effects of Attraction and Repulsion Parameters on the Bacterial Foraging Algorithm through Benchmark Functions

Actividad 1. Desarrollar una introducción teórica (1 Cuartilla) al algoritmo BFOA

Actividad 2. Desarrollar un análisis y descripción del algoritmo (detallar métodos y procesos) fecha límite 25 de Febrero 2025

Commented [EW1]: no completado

GitHub del Algoritmo BFOA Paralelo: https://github.com/riosew/parall_BFOA

Algunas Notas:

La clase principal es parallel_BFOA.py

Es importante notar que el algoritmo desarrolla su proceso en paralelo (Parallel Computing). En general las líneas que hacen uso de pool() refieren a procesos o hilos que corren en paralelo.

Actividad 3:

Realizar un análisis de desempeño del algoritmo. Como input usar el archivo multifasta.fasta y generar datos de 30 corridas. Es importante registrar los valores de fitness, tiempo de ejecución, interacción y calificación blosum. Presentar los datos en tablas y gráficos.

Consejo: ajusta y reporta los valores de numero de bacterias, iteraciones, tumbo y w_atract, dtract, wRepel, dRepel segun el equipo de cómputo.

Fecha de entrega: 31 de Marzo 2025

Commented [EW2]: no completado

Actividad 4: Desarrollar una mejora al algoritmo original para subir el nivel de fitness que alcanza.

Consejo: El algoritmo realiza un alineamiento de secuencias en una matriz, y para mejorar sus resultados puedes incorporar un nuevo método o mejorar algún proceso.

1. Describir a detalle la mejora
2. Desarrollar un análisis comparativo que demuestre la mejora contra el algoritmo original
3. Incluir link al código mejorado en gitHub.

Fecha de entrega 18 de Abril 2025

Aviso: Se extiende la fecha de entrega de la actividad 4 al 26 de abril 2025. En este periodo puedes completar la actividad o actualizarla según sea el caso

Commented [EW3]: no completado

Actividad Final:

Se presentará el proyecto en formato de video con grabación de la pantalla y micrófono. Es importante mostrar el código de la(s) mejora(s) desarrollada(s), la corrida y gráficos de comparación entre algoritmos, así como otros elementos que consideres importantes.

La entrega consiste en:

Subir el video a tu oneDrive, compartirlo conmigo (riose@uadec.edu.mx) y publicar aquí el link.

Extensión del video de 2 a 4 minutos.

Fecha límite 12 de mayo 2025