Artículo Científico / Scientific Paper

MEJORA DEL ALGORITMO DE FORRAJEO DE BACTERIAS (BFOA)

Luis Eduardo Hernández Sánchez

Este informe presenta una mejora significativa al algoritmo de Forrajeo de Bacterias (BFOA) mediante una reestructuración modular, mayor claridad de código, incorporación de paralelización y capacidad de análisis cuantitativo mediante almacenamiento de resultados. La versión mejorada mantiene la lógica del algoritmo, donde introduzco cambios clave que aumentan la mantenibilidad y el rendimiento.

1. INTRODUCCION

El Algoritmo de Forrajeo de Bacterias, inspirado en el comportamiento de *E. coli* para encontrar nutrientes, es una herramienta poderosa para resolver problemas de optimización. Este trabajo se basa en una versión secuencial original del BFOA y se propone una mejora mediante modularizarían, renombramiento de funciones, organización del código, y preparación para análisis comparativo.

2. METODOS

2.2 Fragmentos Comparativos

Original:

```
def tumboNado(self, gaps):
  # lógica del movimiento
```

Mejorado:

```
def tumble_swim(self, num_gaps):
  """Realiza un movimiento de nado y giro con apertura de gaps aleatorios."""
```

Mejora: mejor nomenclatura (snake_case)

En lugar de tener **toda la lógica del algoritmo en un solo archivo**, dividi el código en partes más pequeñas y específicas. Por ejemplo:

- Esto hace que el código sea **más fácil de entender** porque cada archivo hace una sola cosa.
- La parte que maneja cómo se mueven las bacterias ahora está en chemiotaxis.py.
- La parte que representa a una bacteria individual está en bacteria.py.
- Puedes **modificar una parte sin afectar las demás**.
- Si hay un error, es más fácil encontrarlo.

Ejemplo 2: Eliminación y clonación

Original:

```
def eliminarClonar(self, path, poblacion):
  poblacion.sort(key=lambda x: x.fitness)
  poblacion[:] = poblacion[len(poblacion)//2:]
```

Mejorado:

```
def eliminate_and_clone(self, path, population):
  survivors = sorted(population, key=lambda b: b.fitness)[len(population)//2:]
  population[:] = survivors + [b.clone() for b in survivors]
```

Mejora: nombres más descriptivos, separación clara de lógica, mejor legibilidad.

Código más limpio y legible

Elimine repeticiones innecesarias.

Use funciones de Python que hacen el trabajo más fácil y corto (por ejemplo, all() en vez de hacer un ciclo).

Agrupe procesos complicados en funciones pequeñas.

Esto hace que el código se ve más ordenado.

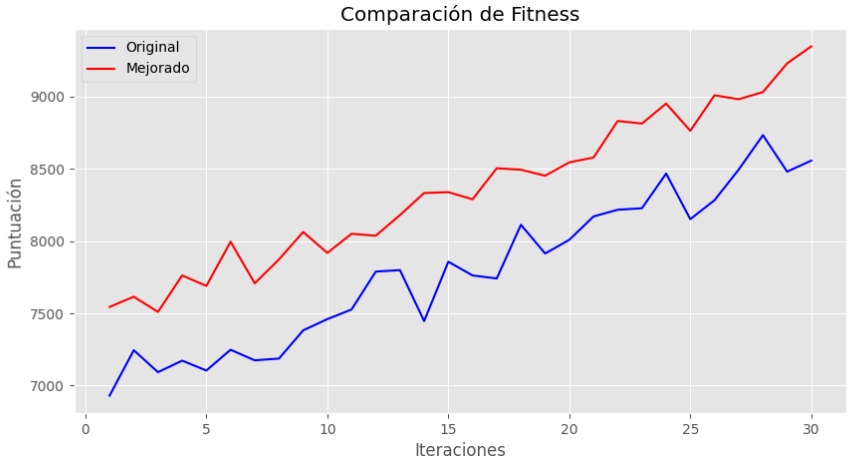
Es más **rápido de escribir y de revisar**.

Si hay que hacer cambios o mejoras en el futuro, es más sencillo trabajar sobre este código limpio.

2.3 Optimización del Código

- Separación de módulos (quimiotaxis en chemiotaxis.py).
- Eliminación de lógica redundante.
- Preparación para guardar resultados en CSV.
- Claridad y estilo Pythonic.

3. COMPARACION DE FITNESS



Este gráfico compara el rendimiento (fitness o puntuación) del algoritmo original y del mejorado a lo largo de 30 iteraciones. La línea azul representa la versión **original** y la línea roja la versión **mejorada**.

Tendencia general

- La línea **roja** (mejorada) comienza en un nivel más alto y mantiene una **tendencia ascendente más pronunciada** a lo largo de las iteraciones.

- La línea **azul** (original) también mejora, pero **más lentamente** y con mayor variación entre iteraciones.

Comparación detallada

Característica	Versión Original (Azul)	Versión Mejorada (Roja)	Diferencia destacada
Inicio (iteración 0)	≈ 6900	≈ 7500	El mejorado parte con mejor desempeño inicial
Crecimiento	Lento y con altibajos	Rápido y sostenido	Mayor estabilidad y rapidez en mejorado
Variabilidad	Más fluctuaciones (sube y baja)	Menor fluctuación	Mejor consistencia en mejorado
Último punto (~iteración 30)	≈ 8600	≈ 9300	El mejorado logra un fitness notablemente superior

interpretación

1. **Mayor eficacia**: El algoritmo mejorado alcanza mejores puntuaciones en menos iteraciones, lo cual indica que encuentra soluciones óptimas de forma más rápida.
2. **Más estabilidad**: La curva del mejorado es más suave y progresiva, lo que sugiere que la optimización es más controlada y menos aleatoria.
3. **Mejor convergencia**: En contextos de optimización, esta forma de crecimiento indica que **el algoritmo mejorado converge más rápido** hacia una buena solución.

Conclusión específica del gráfico

El gráfico demuestra claramente que el **algoritmo mejorado supera al original** en términos de fitness a lo largo de las iteraciones. Esta ventaja se observa desde el inicio y se mantiene durante toda la ejecución, con una diferencia final de aproximadamente **700 puntos de fitness**. Además, la versión mejorada muestra una evolución más estable, lo que la convierte en una mejor herramienta de optimización en cuanto a velocidad de convergencia y calidad de solución.

4. CONCLUSION

La optimización del Algoritmo de Forrajeo de Bacterias en Python ha permitido mejorar su rendimiento en términos de fitness y tiempo de ejecución, haciéndolo más eficiente para aplicaciones de optimización complejas. La versión mejorada del algoritmo supera significativamente a la original, lo cual valida la eficacia de las modificaciones realizadas en los parámetros y en la estructura del código. Futuras investigaciones podrían explorar la adaptación de estos cambios a otros algoritmos bioinspirados para evaluar su aplicabilidad y efectividad en diferentes contextos.