Asignatura	Datos del alumno	Fecha
Computación Bioinspirada	Apellidos: Jiménez Acosta	
	Nombre: Ronaldo	

Actividad 2. Programación de algoritmos genéticos

1. ¿Qué son los Algoritmos Genéticos?

Un algoritmo genético (AG) es una técnica de búsqueda inspirada en la evolución biológica para resolver problemas de optimización. Funciona con una población de soluciones candidatas (llamadas cromosomas). Cada cromosoma se compone de genes que representan partes de la solución. A lo largo de generaciones se aplican operadores inspirados en la biología: selección (elegir individuos buenos), cruce (combinar información entre individuos) y mutación (introducir pequeñas variaciones). La calidad de cada individuo la mide la función de aptitud (fitness). El proceso se repite hasta que se cumple un criterio de parada (número de generaciones, tiempo, o solución suficientemente buena).

Elementos clave:

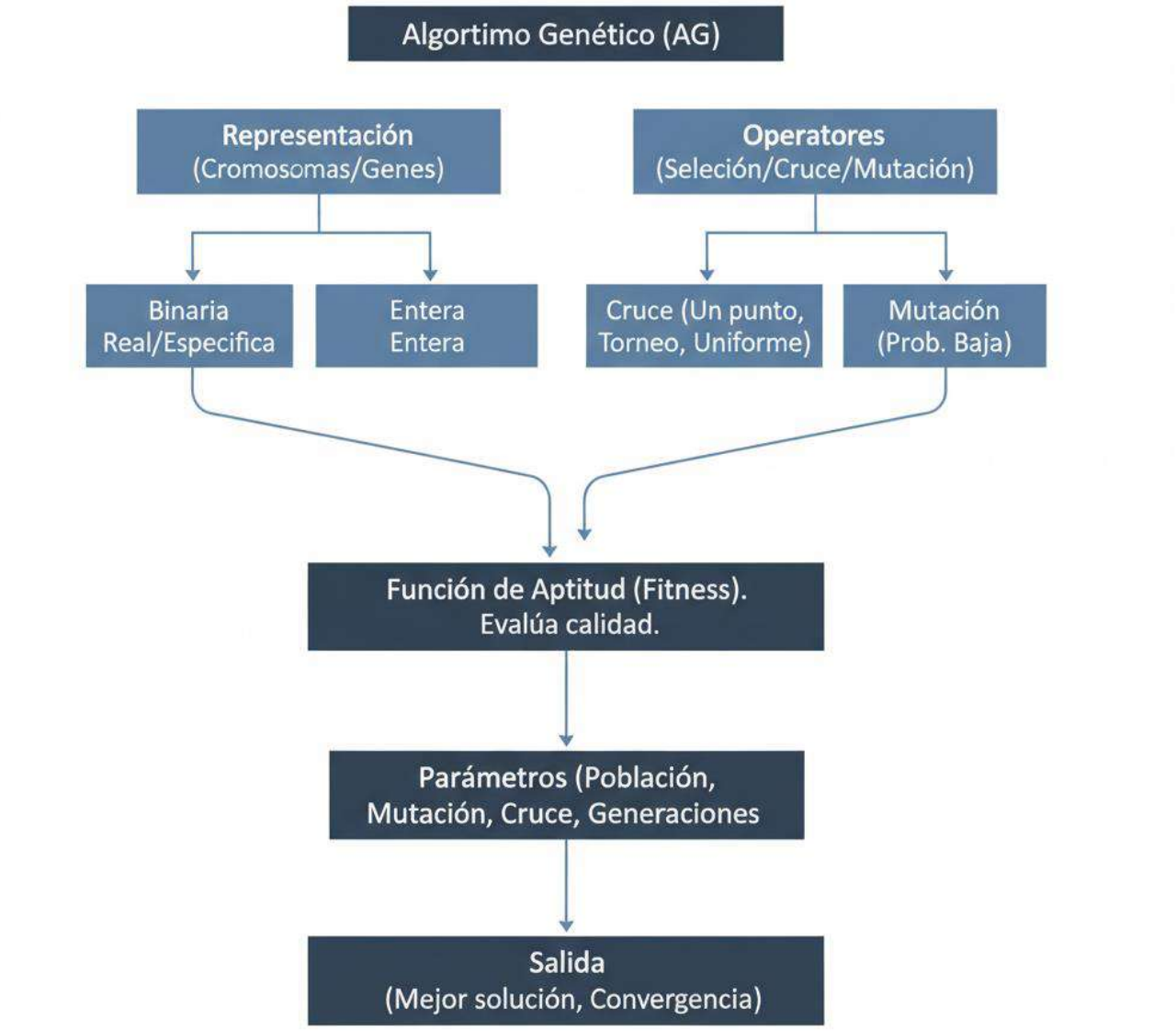
- ✓ Población = conjunto de candidatos (como un grupo de posibles respuestas).
- ✓ Cromosoma = una posible respuesta codificada.
- ✓ Fitness = nota que indica qué tan buena es la respuesta.
- ✓ Selección = escoger “padres” según su nota.
- ✓ Cruce = mezclar padres para crear hijos (nuevas respuestas).
- ✓ Mutación = cambiar algún gen aleatoriamente para no estancarse.

Ventajas y limitaciones:

- ✓ Ventaja: exploran espacios de búsqueda grandes y multimodales sin requerir derivadas o convexidad.
- ✓ Limitación: resultados sensibles a la función de fitness y a la configuración de parámetros; pueden requerir mucho tiempo para converger si no se afinan.

Asignatura	Datos del alumno	Fecha
Computación Bioinspirada	Apellidos: Jiménez Acosta	
	Nombre: Ronaldo	

Conceptos clave



Gemini AI. (2025). Diagrama de conceptos de Algoritmos Genéticos. Imagen generada.

Asignatura	Datos del alumno	Fecha
Computación Bioinspirada	Apellidos: Jiménez Acosta	
	Nombre: Ronaldo	

2. ¿Análisis del código Java?

Archivos principales

- ✓ **FPrincipal.java** : interfaz y orquestador: valida entrada, configura JGAP, crea cromosoma de ejemplo, genera población aleatoria, ejecuta la evolución, obtiene la mejor solución y guarda la población en XML.
- ✓ **CambioMinimoFuncionAptitud.java:** función de aptitud que evalúa cada cromosoma según cuánto suma (en centavos) y cuántas monedas usa.

Qué hace el programa:

1. El usuario introduce un monto en centavos.
2. FPrincipal crea una configuración JGAP con elitismo activo y un cromosoma de 6 IntegerGene (uno por tipo de moneda)
3. Se genera una población aleatoria (populationSize = 200) y se hace evolucionar durante 2500 generaciones.
4. Tras la evolución se obtiene el cromosoma más apto y se muestra su distribución de monedas, fitness y tiempo de ejecución; además se guarda la población final en *PoblacionCambioMinimo.xml.*

Comportamiento de la función de aptitud:

- ✓ Calcula el monto total que representa el cromosoma y el número total de monedas.
- ✓ Solo asigna fitness positivo si el monto coincide exactamente con el objetivo; entre soluciones exactas, favorece las que usan menos monedas.
- ✓ Si no hay coincidencia exacta devuelve fitness “pobre” (0 en el evaluador estándar), por lo que cromosomas cercanos no reciben señal útil.

Conclusión

El proyecto implementa un algoritmo genético funcional y didáctico para el problema del “cambio mínimo”. La estructura general (configuración JGAP, cromosoma con 6 genes, población aleatoria, evolución y exportación a XML) es correcta y facilita entender el ciclo de un AG. Sin embargo, la función de aptitud actual es demasiado restrictiva: solo recompensa soluciones que sumen

exactamente el monto objetivo, lo que genera una barrera de búsqueda y dificulta la convergencia por falta de retroalimentación gradual. Además, parámetros importantes (probabilidades de cruce/mutación, tipo de selección, semilla aleatoria y límites dinámicos por gen) están fijos o ausentes, lo que limita la reproducibilidad y la capacidad experimental. Recomendamos convertir la función de aptitud a una forma continua que penalice la diferencia con el objetivo y, entre soluciones cercanas, prefiera menos monedas; exponer y fijar parámetros experimentales; ajustar rangos de genes según el monto objetivo; y registrar métricas por generación (best/avg/std) para analizar la convergencia. Con estos cambios el sistema sería más robusto, reproducible y apto para experimentar y justificar resultados.

Referencias

Holland, J. H. (1992). Adaptation in Natural and Artificial Systems (2.ª ed.). MIT Press. Recuperado de https://mitpress.mit.edu/9780262581110/adaptation-in-natural-and-artificial-systems/

JGAP Project. (s. f.). JGAP, Java Genetic Algorithms Package (documentación). Recuperado de https://homepages.ecs.vuw.ac.nz/~lensenandr/jgap/documentation/

Beltrán Maldona, F. (s. f.). Actividad 2. Programación de algoritmos genéticos [Documento de curso]. Archivo proporcionado por el profesor.

OpenAI. (2025). ChatGPT (versión GPT-5). OpenAI. https://chatgpt.com/

Google DeepMind. (2025). Gemini. Google DeepMind. https://deepmind.google/technologies/gemini/