Laboratorio 6: Algoritmos Genéticos para Resolución de Problemas Complejos

Profesores:
Daniel Angel Fuertes
Germán Adolfo Montoya,

Profesor de Laboratorio:
Juan Andrés Mendez

20 de noviembre de 2025

1 Objetivo

El objetivo de este laboratorio es que los estudiantes comprendan y apliquen los conceptos fundamentales de los algoritmos genéticos en escenarios prácticos de complejidad creciente. Al finalizar, los estudiantes serán capaces de:

- Formular problemas de optimización adaptados a la estructura de algoritmos genéticos
- Diseñar representaciones cromosómicas adecuadas para diferentes tipos de problemas
- Implementar y personalizar operadores genéticos según las características de cada problema
- Analizar el rendimiento y convergencia de algoritmos evolutivos
- Aplicar algoritmos genéticos a problemas reales en dominios como robótica, redes y logística

2 Fundamentación Matemática

2.1 Algoritmos Genéticos

Los algoritmos genéticos son técnicas de búsqueda y optimización basadas en los principios de evolución natural y genética. Los componentes fundamentales incluyen:

- **Representación cromosómica**: Codificación de soluciones potenciales como secuencias de genes (binarios, enteros, reales, permutaciones, etc.)
- **Función de aptitud (fitness)**: Evaluación de la calidad de cada solución

- **Selección**: Mecanismos para escoger individuos que participarán en la reproducción (rueda de ruleta, torneo, ranking, etc.)
- **Operadores genéticos**:
  - Cruce (crossover): Combinación de material genético de progenitores
  - Mutación: Modificación aleatoria para mantener diversidad
  - Elitismo: Preservación de mejores soluciones entre generaciones

2.2 Estructura General de un Algoritmo Genético

El pseudocódigo básico de un algoritmo genético es:

Pseudocódigo de un Algoritmo Genético

```
Evaluar aptitud de cada individuo en P(0)
t = 0

MIENTRAS no se cumpla criterio de parada HACER
    Seleccionar padres de P(t)
    Aplicar operadores geneticos (cruce, mutacion) para crear P(t
       +1)
    Evaluar aptitud de cada individuo en P(t+1)
    t = t + 1
FIN MIENTRAS

Retornar mejor solucion encontrada
```

3 Problemas

3.1 Problema 1: Corridor Crawler (Laberinto Determinístico)

3.1.1 Descripción del Problema

En este problema, los estudiantes asumen el rol de ingenieros de robótica contratados para automatizar una instalación postal subterránea. La tarea inmediata consiste en desarrollar un robot de tipo micromouse que pueda salir de un laberinto fijo de 20×20 lo más rápido posible.

El objetivo es evolucionar una estrategia de navegación (conjunto de reglas o secuencia de movimientos) que permita al robot encontrar la salida del laberinto de manera eficiente, minimizando el número de pasos y evitando colisiones con paredes.

3.1.2 Datos del Problema

- **Entorno**: Laberinto de 20x20 celdas con paredes internas, una entrada y una salida. La estructura del laberinto es fija (determinística).
- **Robot**:
  - Ocupa una celda
  - Puede moverse en cuatro direcciones (norte, sur, este, oeste)
  - Puede detectar si hay paredes en las celdas adyacentes
  - No tiene memoria de las celdas visitadas previamente (excepto lo codificado en su "genética")
- **Restricciones**:
  - El robot no puede atravesar paredes
  - Existe un límite máximo de 500 pasos por simulación para evitar bucles infinitos
  - El robot comienza siempre en la misma posición inicial
  - La salida está siempre en la misma posición

3.1.3 Instrucciones

Resolver los casos `maze_case_base.txt`, `maze_case_heavy.txt`, que se encuentran en bloque neon.

1. **Diseño de la Representación**:
   - Diseñe una codificación cromosómica que represente una estrategia de navegación. Puede elegir entre:
     - Secuencia directa de movimientos
     - Conjunto de reglas basadas en la detección de paredes circundantes
     - Tabla de decisiones o máquina de estados simple
   - Justifique la representación elegida en términos de su expresividad y eficiencia

2. **Implementación del Algoritmo Genético**:
   - Desarrolle una función de aptitud (fitness) que tenga en cuenta:
     - Distancia recorrida (negativa)
     - Penalización por colisiones
     - Bonificación por alcanzar la salida
     - Tiempo/pasos utilizados
   - Implemente operadores de selección, cruce y mutación adaptados al problema
   - Incluya elitismo ($\leq 2\,\%$ de la población)
   - Utilice un tamaño de población ≤ 200 y ≤ 300 generaciones
3. **Entorno de Simulación**:
   - Se proporciona un simulador básico de mundo de rejilla en Python (150 líneas)
   - El estudiante debe implementar únicamente el núcleo del algoritmo genético
   - El simulador realiza el seguimiento del robot en el laberinto y calcula la aptitud

Notas para la Implementación:

- La implementación debe ser eficiente para mantener tiempos de ejecución cortos (menos de 2 minutos en un portátil estándar)
- Compare diferentes estrategias de representación y operadores de cruce
- Analice la convergencia del algoritmo a lo largo de las generaciones
- Visualice la trayectoria final del mejor individuo

3.2 Problema 2: Nomad Crawler (Generalización de Laberintos Estocásticos) Bono Opcional

3.2.1 Descripción del Problema

Extendiendo el problema anterior, ahora el cliente desea que el mismo robot sea capaz de manejar cualquier diseño de almacén que cambia diariamente. Para ello, los individuos de cada generación serán evaluados en K laberintos generados aleatoriamente, y la aptitud será el tiempo promedio de salida.

Este problema introduce el concepto de robustez y la necesidad de evitar el sobreajuste a un único entorno. Los estudiantes deben desarrollar un controlador genético que funcione bien en diversos laberintos, no solo en uno específico.

3.2.2 Datos del Problema

- **Entorno**:
  - Laberintos de 15x15 celdas generados aleatoriamente
  - K=10 laberintos diferentes por evaluación
  - Cada laberinto tiene una entrada y una salida en posiciones aleatorias
  - Garantía de que existe al menos un camino entre entrada y salida
- **Robot**: Igual que en el Problema 1, con las mismas capacidades
- **Restricciones**:
  - Límite de 300 pasos por laberinto
  - Si el robot no encuentra la salida en ese tiempo, se considera fallido

Se proporciona un generador de laberintos aleatorios en Python que produce laberintos diferentes en cada llamada, con complejidad O(N log N).

3.2.3 Instrucciones

1. **Adaptación de la Representación**:
   - Refine o rediseñe la representación cromosómica del Problema 1 para hacerla más generalizable
   - Considere estructuras que puedan adaptarse a laberintos variables sin conocimiento previo
2. **Implementación de Evaluación Estocástica**:
   - Modifique la función de aptitud para evaluar cada individuo en K laberintos distintos
   - Calcule el tiempo promedio de salida y métricas adicionales como:
     - Tasa de éxito (porcentaje de laberintos resueltos)
     - Desviación estándar de los tiempos (indicador de robustez)
   - Implemente un presupuesto de replicación: cada individuo se evalúa K veces

- Calcule intervalos de confianza para las estimaciones de aptitud

3. **Exploración de la Diversidad**:
   - Implemente mecanismos para mantener la diversidad en la población:
     - Técnicas de compartición de aptitud (fitness sharing)
     - Nichos para preservar diferentes estrategias
     - Selección por torneo con diversidad
   - Compare la efectividad de estas técnicas para evitar el sobreajuste

Notas para la Implementación:

- Al generar nuevos laberintos, utilice una semilla fija para la fase de entrenamiento para permitir comparaciones justas
- Para la evaluación final, el autoevaluador utilizará semillas diferentes para prevenir el sobreajuste
- Considere el equilibrio entre exploración (diversidad) y explotación (convergencia)
- Analice la robustez de las soluciones frente a diferentes tipos de laberintos

4 Entregables y Criterios de Calificación

4.1 Entregables

Para cada problema, se deberá entregar:

1. **Código Fuente**:
   - Archivo Python (*.py) con la implementación completa
   - Funciones claramente documentadas
   - Scripts adicionales para visualización (si aplica)
   - Instrucciones de ejecución
2. **Informe Técnico (PDF)**:
   - Formulación del problema y representación elegida
   - Descripción de los operadores genéticos implementados
   - Análisis de resultados con gráficas relevantes
   - Comparativas de diferentes configuraciones
   - Conclusiones y limitaciones encontradas

4.2 Criterios de Calificación

Criterio	**Puntuación**
Diseño de la representación cromosómica	20 %
Implementación de operadores genéticos	20 %
Función de aptitud y manejo de restricciones	15 %
Calidad de las soluciones obtenidas	15 %
Experimentación y análisis de resultados	15 %
Visualización y presentación del informe	10 %
Eficiencia computacional	5 %

Cuadro 1: Criterios de calificación del laboratorio

5 Referencias

1. Talbi, E. G. (2009). Metaheuristics: from design to implementation. John Wiley & Sons.
2. Luke, S. (2013). Essentials of metaheuristics (2nd ed.). Lulu.
3. Fortin, F. A., De Rainville, F. M., Gardner, M. A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research, 13, 2171-2175.
4. Toth, P., & Vigo, D. (Eds.). (2002). The vehicle routing problem. Society for Industrial and Applied Mathematics.

5. Michalewicz, Z. (2013). Genetic algorithms + data structures = evolution programs. Springer Science & Business Media.

6. Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.

7. Eiben, A. E., & Smith, J. E. (2003). Introduction to evolutionary computing. Springer.

6 Apéndice: Guía de Implementación

6.1 Estructura Base de Código

A continuación se presenta una estructura básica recomendada para la implementación:

6.2 Consejos para Afrontar los Problemas

1. **Diseño de la representación**: La clave del éxito en algoritmos genéticos es elegir una representación adecuada.
   - Para el Problema 1 (Corridor Crawler), considere representaciones basadas en reglas como "Si hay pared a la izquierda, entonces girar a la derecha", etc.
   - Para el Problema 2 (Nomad Crawler), enfóquese en representaciones más generales y adaptables.
2. **Operadores genéticos**: Diseñe operadores específicos para cada tipo de representación.
   - Para representaciones basadas en reglas, utilice cruce de un punto o dos puntos.
   - Para permutaciones, utilice operadores que preserven el orden (OX, PMX).
   - Para árboles de decisión, considere cruce de subárboles.
3. **Estrategias de inicialización**: No subestime la importancia de una buena población inicial.
   - Incorpore conocimiento del dominio cuando sea posible.
   - Combine soluciones aleatorias con algunas heurísticas simples.
   - Asegure diversidad suficiente en la población inicial.
4. **Depuración y validación**: Implemente funciones para verificar la validez de las soluciones.
   - Verifique restricciones en cada paso (después de cruce, mutación, etc.)
   - Implemente visualizaciones simples para entender el comportamiento.
   - Use poblaciones pequeñas durante la fase de depuración.