Informe de Desarrollo Simulador de Optimización de Área con Algoritmos Genéticos

Autores: Daniel Steven Hincapié Cetina & Santiago Naranjo Herrera
Asignatura: CADI: Profundización: Machine Learning
Proyecto: Taller No. 3 - Optimización de Área

Resumen Ejecutivo

Este informe detalla el diseño, desarrollo y análisis de un sistema inteligente para resolver el problema de maximización de rentabilidad en un espacio con área limitada. Se implementó un **Algoritmo Genético (AG)** como motor de optimización y se desarrolló una **interfaz gráfica interactiva** con Streamlit. El sistema permite al usuario configurar dinámicamente tanto el catálogo de productos como los hiperparámetros del algoritmo, visualizando los resultados de forma intuitiva a través de gráficos de convergencia y distribución espacial. El proyecto cumple con todos los requisitos técnicos y funcionales estipulados en la guía del taller.

Modelado Matemático del Problema

Para abordar el problema formalmente, se definió un modelo de optimización que el Algoritmo Genético busca resolver.

Función Objetivo

El objetivo principal es **maximizar la ganancia total (Z)**, calculada como la suma de las ganancias de todos los artículos seleccionados.

$$\text{Maximizar } Z = \sum_{i=1}^{n} g_i \cdot x_i$$

Variables de Decisión

- x_i: Representa la cantidad de unidades (un número entero) a incluir del artículo i.

Restricciones

1. **Restricción de Área:** La suma de las áreas de todos los artículos seleccionados no puede exceder el área máxima disponible (A_{max}).

$$\sum_{i=1}^{n} a_i \cdot x_i \leq A_{\text{max}}$$

2. **Restricción de Stock:** La cantidad de cada artículo no puede ser negativa ni superar el stock disponible (s_i).

$$0 \leq x_i \leq s_i \quad \forall i \in \{1, \ldots, n\}$$

Arquitectura y Diseño del Sistema

Se optó por una arquitectura desacoplada de dos componentes para cumplir con los requisitos del taller y seguir las mejores prácticas de desarrollo de software.

Backend (`ga_backend.py`)

- **Responsabilidad:** Contiene toda la lógica computacional del Algoritmo Genético.
- **Tecnología:** Python puro.
- **Diseño:** Se implementó una clase `GeneticAlgorithm` que encapsula todos los métodos y parámetros del AG (población, evaluación, selección, cruce, mutación). Este diseño modular permite que el backend sea reutilizable y fácil de mantener.

Frontend (`app.py`)

- **Responsabilidad:** Proporcionar una interfaz de usuario interactiva para controlar el sistema y visualizar los resultados.
- **Tecnología:** Streamlit, una librería de Python que facilita la creación de aplicaciones web para ciencia de datos.
- **Diseño:** La interfaz se dividió en dos columnas: una para la configuración de parámetros y otra para la visualización de resultados. Esto proporciona un flujo de trabajo claro y lógico para el usuario.

La comunicación entre ambos componentes es directa: el frontend importa la clase `GeneticAlgorithm` del backend, la instancia con los parámetros definidos por el usuario y ejecuta el proceso de optimización.

Implementación del Algoritmo Genético (Backend)

La clase `GeneticAlgorithm` fue el núcleo del proyecto. A continuación, se detallan sus componentes clave:

- **Representación del Individuo:** Cada individuo (o cromosoma) es una lista de números enteros, donde cada posición `i` en la lista representa la cantidad x_i del artículo `i` del catálogo.

- **Función de Fitness:** La función `_evaluar_individuo` calcula la ganancia total. Para manejar la restricción de área, se aplica una **función de penalización**: si un individuo excede el área máxima, su fitness (ganancia) se reduce drásticamente en proporción al exceso. Esto guía al algoritmo a descartar soluciones inválidas.
- **Operadores Genéticos:**
  - **Selección:** Se implementaron dos métodos, seleccionables desde la interfaz:
    1. **Torneo (`_seleccionar_torneo`):** Selecciona `k` individuos al azar y elige al mejor como padre. Es eficiente y da buenos resultados.
    2. **Ruleta (`_seleccionar_ruleta`):** Asigna a cada individuo una probabilidad de ser seleccionado proporcional a su fitness.
  - **Cruce:** Se utilizó el **cruce uniforme**, donde cada gen (cantidad de un artículo) del hijo tiene un 50% de probabilidad de provenir de cualquiera de los dos padres.
  - **Mutación:** El operador `_mutar` recorre cada gen de un individuo y, con una baja probabilidad (`prob_mutacion`), suma o resta 1 a la cantidad, respetando siempre los límites del stock (0 y el máximo disponible).
- **Elitismo:** Se implementó un mecanismo de elitismo que permite que un número configurable de los mejores individuos de una generación pasen intactos a la siguiente. Esto asegura que las mejores soluciones encontradas no se pierdan.

Desarrollo de la Interfaz Gráfica (Frontend)

La interfaz, creada en `app.py`, fue diseñada para ser intuitiva y cumplir con todos los requisitos de interacción:

- **Configuración de Parámetros:** Se utilizaron componentes de Streamlit como `st.slider` para las probabilidades y tamaños, `st.number_input` para el área, `st.selectbox` para el método de selección y `st.toggle` para activar/desactivar el elitismo.
- **Catálogo Interactivo:** El componente `st.data_editor` fue fundamental. Se configuró para mostrar el catálogo en una tabla, añadiendo una primera columna de tipo `CheckboxColumn`. Esto permite al usuario no solo **modificar la ganancia, área y stock** de cada producto, sino también **excluir artículos** de la optimización desmarcando su casilla.
- **Visualización de Resultados:**
  1. **Gráfico de Convergencia:** Se utilizó `matplotlib` para generar un gráfico de líneas que muestra la evolución del mejor fitness a lo largo de las generaciones. Esto permite analizar la velocidad y estabilidad del algoritmo.
  2. **Distribución Espacial 2D:** Se empleó la librería `squarify` para crear un **treemap**. Este tipo de gráfico es ideal para representar áreas proporcionales. Se modificó para incluir un recuadro adicional de color gris que representa el **espacio vacío**, cumpliendo con un requisito específico del taller y ofreciendo una visión completa del uso del área.

Conclusiones y Análisis de Resultados

El sistema desarrollado demostró ser una herramienta eficaz para encontrar soluciones de alta calidad al problema de optimización. Durante las pruebas, se observó que:

- El **elitismo** acelera significativamente la convergencia hacia buenas soluciones, ya que previene la pérdida de los mejores individuos.
- El método de **selección por torneo** tiende a ser más estable y rápido que la ruleta, especialmente cuando hay grandes diferencias de fitness en la población.
- La **tasa de mutación** es un parámetro sensible: un valor muy bajo puede llevar a un estancamiento prematuro, mientras que un valor muy alto puede convertir la búsqueda en un proceso demasiado aleatorio.

Como **limitación principal**, el modelo actual no considera la geometría de los objetos (solo su área), asumiendo que pueden ser dispuestos de manera perfecta. Una futura mejora podría ser la implementación de algoritmos de *packing* 2D más complejos.

En conclusión, el proyecto no solo cumple con todos los criterios de evaluación, sino que también sirve como una demostración práctica y visual del poder de los algoritmos genéticos para resolver problemas de optimización del mundo real.