Smart Agriculture Crop Layout Optimization

Overview

This project leverages Genetic Algorithms (GA) to optimize crop layouts while considering key farming factors like sunlight, water availability, economic value, and crop compatibility. It helps in maximizing yield and profit by optimizing land usage.

Features

- Optimized Crop Layout Generation using AI
- Random Layout vs. Optimized Layout Comparison
- Sunlight & Water Heatmaps for better decision-making
- Real-time Fitness & Profit Analysis
- Interactive Crop Selection Feature
- Download Optimized Layout as CSV

Technology Stack

- Python: Data Processing & AI Algorithms
- Streamlit: Interactive Web UI
- Matplotlib & Seaborn: Visualizations
- Pandas & NumPy: Data Processing
- Genetic Algorithms (GA): Crop Layout Optimization

Installation & Setup

1. Clone the Repository:

```
git clone https://github.com/yourusername/smart-agriculture-optimization.git
cd smart-agriculture-optimization
```

2. Install Dependencies:

```
pip install -r requirements.txt
```

3. Run the Streamlit App:

streamlit run main.py

code

```
import pandas as pd

import numpy as np

import random

import matplotlib.pyplot as plt

import streamlit as st

import seaborn as sns

import time


# Load dataset

file_path = 'synthetic_agriculture_field_large.csv'

dataset = pd.read_csv(file_path)


# Add price per unit for cost & profit analysis

dataset["Price_per_unit"] = np.random.randint(50, 200, size=len(dataset))  # Random price for each crop


# Define category mappings

category_mapping = {

    'Sunlight': {'high': 2, 'medium': 1, 'low': 0},

    'Water': {'high': 2, 'medium': 1, 'low': 0},

    'Soil_type': {'loamy': 2, 'clay': 1, 'sandy': 0},

    'Water_efficiency': {'high': 2, 'medium': 1, 'low': 0},

    'Temperature_range': {'hot': 2, 'moderate': 1, 'cool': 0},

    'Pest_resistance': {'high': 2, 'medium': 1, 'low': 0},
```

```
    'Crop_rotation': {'yes': 1, 'no': 0},

    'Harvest_frequency': {'biannual': 2, 'annual': 1}

}


for column, mapping in category_mapping.items():

    dataset[column] = dataset[column].map(mapping)


st.title('🌱 Smart Agriculture Crop Layout Optimization 🌱')


rows = st.number_input("Enter number of rows:", min_value=5, max_value=50, value=10,
step=1)

cols = st.number_input("Enter number of columns:", min_value=5, max_value=50, value=10,
step=1)


population_size = st.slider('Population Size', 50, 500, 100)

generations = st.slider('Generations', 10, 500, 200)

mutation_rate = st.slider('Mutation Rate', 0.01, 0.2, 0.02)

crossover_rate = st.slider('Crossover Rate', 0.5, 1.0, 0.9)


compatibility_scores = {('Beans', 'Corn'): 1, ('Lettuce', 'Wheat'): -1}

sunlight_zones = np.random.choice([0, 1, 2], size=(rows, cols))

water_zones = np.random.choice([0, 1, 2], size=(rows, cols))


def fitness(layout):

    score = 0

    economic_value = 0

    total_profit = 0
```

```
    grid = np.array(layout).reshape(rows, cols)

    for i in range(rows):
        for j in range(cols):
            crop_index = grid[i, j]
            crop_info = dataset.iloc[crop_index]

            if crop_info['Sunlight'] == sunlight_zones[i, j]: score += 2
            if crop_info['Water'] == water_zones[i, j]: score += 2

            economic_value += crop_info['Economic_value']
            total_profit += crop_info['Economic_value'] * crop_info['Price_per_unit']

            if j < cols - 1:
                neighbor_crop_index = grid[i, j + 1]
                neighbor_crop_name = dataset.iloc[neighbor_crop_index]['Crop']
                crop_name = dataset.iloc[crop_index]['Crop']
                score += compatibility_scores.get((crop_name, neighbor_crop_name), 0)

    return score, economic_value, total_profit

# Generate a Completely Random Layout
def generate_random_layout():
    return np.array(random.choices(range(len(dataset)), k=rows * cols)).reshape(rows, cols)

# Genetic Algorithm Functions
```

```python
def generate_population(pop_size):
    return [random.choices(range(len(dataset)), k=rows * cols) for _ in range(pop_size)]

def tournament_selection(population, tournament_size=3):
    selected = random.sample(population, tournament_size)
    return sorted(selected, key=lambda x: fitness(x)[0], reverse=True)[:2]

def crossover(parent1, parent2):
    point = random.randint(1, rows * cols - 1)
    return parent1[:point] + parent2[point:]

def mutate(child):
    for i in range(len(child)):
        if random.random() < mutation_rate:
            child[i] = random.choice(range(len(dataset)))
    return child

def genetic_algorithm():
    population = generate_population(population_size)
    best_fitness = []
    best_value = []
    total_profit = []

    for generation in range(generations):
        new_population = []
        while len(new_population) < population_size:
```

```python
            parent1, parent2 = tournament_selection(population)
            child = crossover(parent1, parent2) if random.random() < crossover_rate else parent1
            child = mutate(child)
            new_population.append(child)

        population = new_population
        best_individual = max(population, key=lambda x: fitness(x)[0])
        best_fit, best_econ_value, best_profit = fitness(best_individual)
        best_fitness.append(best_fit)
        best_value.append(best_econ_value)
        total_profit.append(best_profit)

    return np.array(best_individual).reshape(rows, cols), best_fitness, total_profit

if st.button("Generate Optimized Layout"):
    optimized_layout, fitness_progress, profit_progress = genetic_algorithm()

    # Display Heatmaps BEFORE the layout
    st.write("🌞 **Sunlight Heatmap**")
    plt.figure(figsize=(8, 6))
    sns.heatmap(sunlight_zones, cmap="YlOrBr", annot=True, fmt=".0f")
    st.pyplot(plt)

    st.write("💧 **Water Availability Heatmap**")
    plt.figure(figsize=(8, 6))
```

```
sns.heatmap(water_zones, cmap="Blues", annot=True, fmt=".0f")
st.pyplot(plt)

# Display Optimized Layout
st.subheader("✅ **Optimized Crop Layout**")
fig, ax = plt.subplots(figsize=(10, 10))
crop_colors = {'Wheat': 'gold', 'Lettuce': 'green', 'Corn': 'yellow', 'Beans': 'brown'}

ax.set_xlim(0, cols)
ax.set_ylim(0, rows)
ax.set_xticks([])
ax.set_yticks([])
ax.set_aspect('equal', 'box')

for i in range(rows):
    for j in range(cols):
        crop_index = optimized_layout[i, j]
        crop_name = dataset.iloc[crop_index]['Crop']
        ax.add_patch(plt.Rectangle((j, i), 1, 1, color=crop_colors[crop_name], edgecolor='black'))
        ax.text(j + 0.5, i + 0.5, crop_name, ha='center', va='center', fontsize=8, color='black')

st.pyplot(fig)

# Compare with Random Layout
st.subheader("📊 **Comparison: Random vs Optimized Layout**")
```

```
random_layout = generate_random_layout()

fig, axes = plt.subplots(1, 2, figsize=(16, 8))


def plot_layout(ax, layout, title):

    ax.set_xlim(0, cols)

    ax.set_ylim(0, rows)

    ax.set_xticks([])

    ax.set_yticks([])

    ax.set_aspect('equal', 'box')

    ax.set_title(title)


    for i in range(rows):

        for j in range(cols):

            crop_index = layout[i, j]

            crop_name = dataset.iloc[crop_index]['Crop']

            ax.add_patch(plt.Rectangle((j, i), 1, 1, color=crop_colors[crop_name],
edgecolor='black'))

            ax.text(j + 0.5, i + 0.5, crop_name, ha='center', va='center', fontsize=8, color='black')


plot_layout(axes[0], random_layout, "🌾 Random Layout (Before Optimization)")

plot_layout(axes[1], optimized_layout, "✅ Optimized Layout (After AI Processing)")


st.pyplot(fig)

# 📊 Calculate fitness, economic value, and profit for the random layout

random_fitness, random_economic_value, random_total_profit = fitness(random_layout)
```

```
# 📊 Calculate fitness, economic value, and profit for the optimized layout
optimized_fitness, optimized_economic_value, optimized_total_profit =
fitness(optimized_layout)

# 📈 Display insights before vs. after optimization
st.subheader("📊 **Insights: Before vs. After Optimization**")

col1, col2 = st.columns(2)

with col1:
    st.write("### ❌ **Before Optimization (Random Layout)**")
    st.write(f"🔷 **Fitness Score:** {random_fitness}")
    st.write(f"💰 **Total Economic Value:** {random_economic_value}")
    st.write(f"📈 **Total Profit:** ${random_total_profit}")

with col2:
    st.write("### ✅ **After Optimization (AI-Optimized Layout)**")
    st.write(f"🔷 **Fitness Score:** {optimized_fitness} (**↑ {((optimized_fitness -
random_fitness) / random_fitness) * 100:.2f}%**)")
    st.write(f"💰 **Total Economic Value:** {optimized_economic_value} (**↑
{((optimized_economic_value - random_economic_value) / random_economic_value) *
100:.2f}%**)")
    st.write(f"📈 **Total Profit:** ${optimized_total_profit} (**↑ {((optimized_total_profit -
random_total_profit) / random_total_profit) * 100:.2f}%**)")
```

```
# Save Optimized Layout for Download
optimized_layout_df = pd.DataFrame(optimized_layout)
optimized_layout_df.to_csv("optimized_layout.csv", index=False)
st.download_button("Download Optimized Layout", "optimized_layout.csv")
```

License

Developer & Contact Info

Developed by: P subanaveen

Contact: subanaveen_p@srmap.edu.in

GitHub Repository: https://github.com/subanaveen/Insyde.Io