**Rock Aggregates Plant Suitability Assessment System**

**—— Additional notes**

**Project Description**

This is a python based project designed to help a company assess potential sites for a rock aggregates plant in the UK. The project aimed to assess the suitability of different sites based on three influencing factors: geology, transport and population. Each factor is represented by a two-dimensional grid, with higher coefficients indicating higher suitability for the plant. However, as each factor has a different level of importance, they are multiplied by their respective weights and then added together to give an overall suitability score for each site.

The project provided a user-friendly interface that allowed the company to adjust the weighting of each factor according to their preferences and visualize the suitability of different locations on a map. This helps the company to make informed decisions on site selection and maximize profits in a highly competitive market.

**Project Design**

According to the project requirements, the project can be broken down into three modules, including GUI design, core code writing and code testing. Please refer to the source code and Readme.md for the code testing part.

**GUI Design**

According to the task requirements, the final presentation of the system needs to implement various operations on a GUI interface. Python can use a variety of GUI toolkits to create GUI applications, commonly used are Tkinter, PyQt and wxPython. Considering the practical needs of the project, Tkinter was chosen to implement the GUI interface. Tkinter is a graphical user interface (GUI) library that comes with Python. It is easy to learn, has a rich component library and can be used to quickly create lightweight desktop applications. Designing a prototype GUI interface is a very important part before building a GUI framework. Prototyping can help designers to simulate the expected effect, clarify the design ideas, and make the subsequent building process quick and effective without reworking.

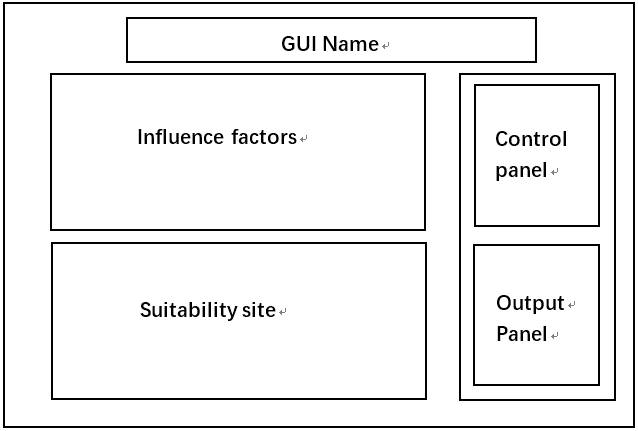


Fig. 1. Prototype of the GUI interface

The GUI interface consists of five sections: title, influence factor images, suitability image, control panel and output panel. Also, according to the project requirements, there are two buttons in the control panel, one for displaying images and one for calculating. There are also three weighting sliders to adjust the weight of the influence factors. In the output panel, there are three buttons for Save & Print, Clear and Quit. The GUI is built according to this design idea. First, create the base screen:

#Instantiating object

window = tk.Tk() #Using tkinter to define the GUI window.

window.iconbitmap("./icon/stone.ico") #Change icon.

window.title('Rock Aggregate Site Suitability Analysis') #Naming the window.

window.geometry("1100x720") #Set window size.

window.mainloop() #If there is no mainloop, it is a static window.

This is followed by the construction of frames. The frame is similar to a div box in html. It is a container that can be stored. First create the title frame, and place the title label in the frame. Then the content frame is created with a pack layout. The content frame is divided into a left frame and a right frame, so that the left and right frames can be centred in the grid layout. In the left frame, the factor frame and the suitability frame are created to display the influence factor image and the suitability image respectively. The right frame contains the control panel and the output panel.

#Title frame

title\_frame = tk.Frame(window)

title\_frame.pack()

#Title label

title\_label = tk.Label(title\_frame)

title\_label.pack()

#Content frame

content\_frame = tk.Frame(window)

content\_frame.pack()

#Left frame

left\_frame = tk.Frame(content\_frame)

left\_frame.grid(row=0, column=0)

#left content

factor\_frame = tk.LabelFrame(left\_frame)

factor\_frame.pack(side="top")

suitability\_frame = tk.LabelFrame(left\_frame)

suitability\_frame.pack(side="bottom")

#right frame

right\_frame = tk.Frame(content\_frame)

right\_frame.grid(row=0, column=1)

#right content

control\_frame = tk.LabelFrame(right\_frame)

control\_frame.pack(side="top")

output\_frame = tk.LabelFrame(right\_frame)

output\_frame.pack(side="bottom")

Once the framework is in place it is time to design the required elements. The required buttons and sliders are created in the control panel and the output panel respectively. After a series of styling changes the following effect is achieved:

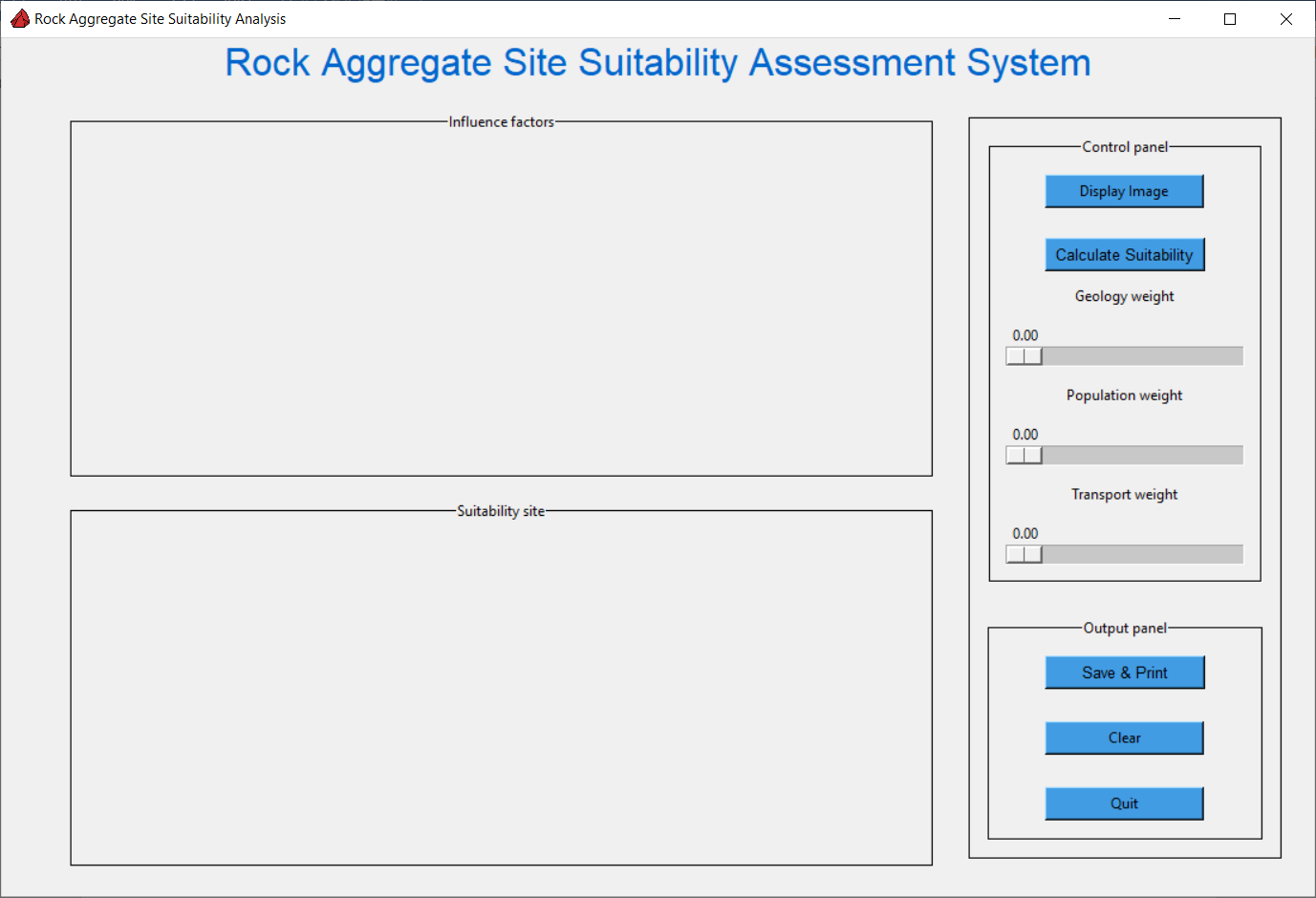


Fig. 2. GUI layout

The subsequent modification of the GUI is to bind functions to the buttons and sliders to enable interaction between the interface system and the user.

**Compiling the core code**

After building the GUI interface, the core code is compiled. To achieve the project requirements, the code needs to be divided into four parts: reading the raw data, converting the raster data into images and displaying them, calculating the suitability of the site and outputting the final results.

Firstly, the read\_data function is defined to read the data, with the file path as a parameter. The data, the number of rows and columns are iterated through and the data, the number of rows and the number of columns are returned as values. Finally the function is called to read the data.

geology\_data, num\_rows, num\_cols = read\_data('./input/geology.txt')

population\_data, \_, \_ = read\_data('./input/population.txt')

transport\_data, \_, \_ = read\_data('./input/transport.txt')

The next step is to define the plot\_data function for displaying the image, with data\_list and title\_list as parameters. The data\_list and title\_list are read from the raw data and then converted into a two-dimensional array and stored in the list. The function has the following steps:

1. Clear the canvas object in the current window.

2. Create a new Matplotlib graph.

3. Iterate through each dataset, creating subplots and plotting the corresponding dataset, setting the title and colour bars.

4. Adjust the spacing between the subplots.

5. Create a Tkinter canvas and display the Matplotlib graph in it.

6. Update the global canvas variables and return the Matplotlib graph and Tkinter canvas objects for external calls.

The most critical part of this function is iterating through the data and creating subplots:

fig\_f, ax1 = plt.subplots(1, len(data\_list), figsize=(10, 4)) # 1 row, Cols of data numbers

fig\_f.subplots\_adjust(wspace=0.3)

#Iteration makes the data correspond to

for i in range(len(data\_list)):

img1 = ax1[i].imshow(data\_list[i], cmap='viridis')

ax1[i].set\_title(title\_list[i])

fig\_f.colorbar(img1, ax=ax1[i], shrink=0.9, pad=0.05)

After testing, the function works fine and gives the following results.

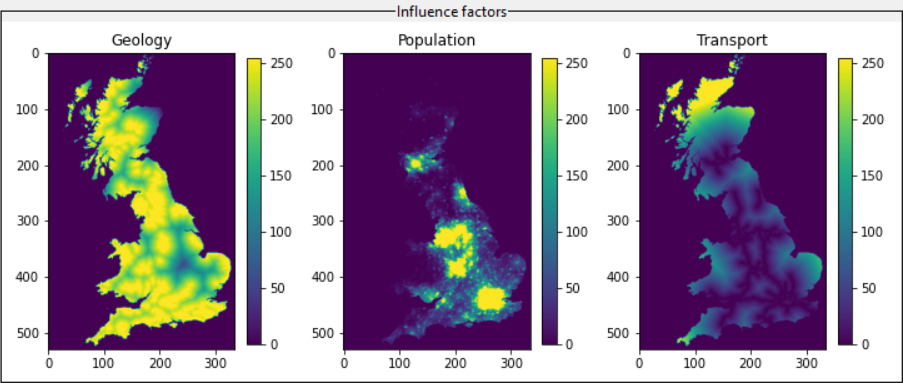


Fig. 3. Plot data

Then, the calculate function is defined to calculate the suitability results. This is achieved in the following steps:

1. Get the weights assigned to each factor from the slider values.

2. Check if all weights are non-zero and if their total equals 1.0.

3. Calculate the suitability score for each factor based on the assigned weights.

4. Normalize and scale the suitability score to a range of 0-255.

5. Create a new Matplotlib figure.

6. Display the suitability map on the Matplotlib figure.

7. Create a Tkinter canvas and display the Matplotlib figure in it.

8. Update global canvas variable and return the suitability score and Matplotlib figure objects for external calls.

The main part of the function is to iterate through the data, multiplying each data by the respective weight. This ensures that each value in the data has a corresponding weight change. Afterwards, the maximum and minimum values are obtained after the weights have been calculated, and the data is normalised to a range of 0-255.

suitability = [[0 for \_ in range(num\_cols)] for \_ in range(num\_rows)] #Initialization data

for i in range(num\_rows):

for j in range(num\_cols):

suitability[i][j] = (geology\_data[i][j] \* geology\_weight +

population\_data[i][j] \* population\_weight +

transport\_data[i][j] \* transport\_weight)

#Get maximum and minimum values --> normalized --> scaled to 0-255

min\_val = min(min(row) for row in suitability)

max\_val = max(max(row) for row in suitability)

scaled\_sum = [[int((val - min\_val) / (max\_val - min\_val) \* 255) for val in row] for row in suitability]

By assigning different weights to the impact factors, a suitability image is calculated. The image values are ranged from 0 to 255, with higher values representing higher suitability for the area.

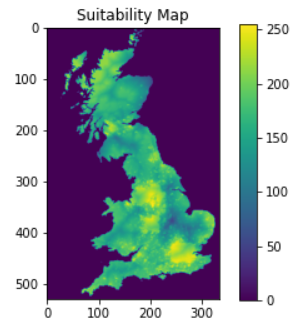


Fig. 4. Calculated data image

Finally, the result is exported. The download\_result function is defined and imported into the os module. The path is spliced and the calculated result data is downloaded and saved locally. Since the CSV file is saved in scientific notation, the output format is controlled by specifying the fmt parameter and using a comma delimited form when saving the file with np.savetxt().

script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

output\_dir = os.path.join(script\_dir, '..', 'output')

image\_path = os.path.join(output\_dir, 'suitabilityMap.png')

txt\_path = os.path.join(output\_dir, 'suitabilityMap.txt')

fig\_s.savefig(image\_path)

os.startfile(image\_path)

np.savetxt(txt\_path,scaled\_sum,delimiter=",", fmt='%d')

Once the main function is complete, define some small functions to make the code run more logically. In the clear canvas part. First declare two global variables current\_canvas1 = None; current\_canvas2 = None. Then define the clear function, if current\_canvas is not None, destroy the canvas and reassign current\_canvas to None. The quit function is very simple, it terminates the window loop, 'window.quit()', and 'window.destroy()', which releases all resources.

**Problems and solutions**

The biggest problem encountered in the code was in the image display part. The suitability image is affected by the impact factors image. When a colorbar is added to a suitability image, the image is off-centered on the canvas. After some study, the colourbar() takes up some space and causes the image to be out of the centre of the canvas. Use the 'add\_axes' function to add a new subplot to the suitability image and specify its position in the figure. Specify the left border position, the bottom border position, the width and the height of the colourbar respectively. This way the colourbar does not affect the position of the image.

**Future prospects**

The system can be applied to other GIS land suitability assessment projects. Users can download the source code and modify the read data section. Adapt it to the actual file required or allow the user to load the data file of their choice by modifying the functions. This makes it easy to apply the system to other similar land assessment projects, improving processing efficiency and accuracy. In addition, the system may be further optimised and extended in the future. For example, more types of data input and output formats are supported, as well as more interactive features. This will allow the system to meet the needs of a wider range of users and application situations. These will be released in future versions.