T.C. ESKİŞEHİR TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

BIM309 – Artificial Intelligence

HOMEWORK III - Report

Map Coloring Using Backtracking Search in Python

Kaouther MOUHEB

99926527616

I. Introduction

This is a brief report describing the Backtracking Search algorithm (BTS) used for solving Constraint Satisfaction Problems (CSPs) and an implementation of the approach on a Map Coloring Problem using the Python programming language on the map of South America.

II. Constraint Satisfaction Problems

A CSP is formally defined by three elements:

- A finite set of **variables** $\{x_1, x_2, \dots, x_n\}$
- A non-empty **domain** of possible values for each variable
- A set of **constraints** $\{C_1, C_2, \dots, C_m\}$. These are rules that limit the values of the domain that a variable x can legally take.

A **state** in a CSP is an assignment of values to some or all variables. CSPs use the **factored representation** of a state.

The solution is to find a **complete assignment** that assigns a **valid** value to each variable x without violating any of the constraints in the CSP definition. Such an assignment is called **consistent.**

III. Backtracking Search

Backtracking Search is an **uninformed search** algorithm that can be used to solve CSPs. It is a **DFS** with two improvements:

1. Assign **one variable** at a time
2. **Check constraints** before each assignment

In BTS, we step back whenever a new assignment violates the previous ones.

This algorithm can be optimized using a set of heuristics such as:

- **Minimum Remaining Values (MRV):** in each step, choose the variable that has the fewest legal values left in its domain.
- **Least Constraining Values (LCV):** for each variable, start with the value that gives the most options for the next steps, that is, the least constraining value.
- **Forward Checking (FC):** After each assignment, check if the neighbors of the assigned variables have at least 1 legal option left, if not, terminate with a failure.

Applying inference to maintain arc-consistency as a pre-processing step or after each assignment can be used to further optimize this algorithm.

The complexity of BTS is the same as DFS ($O(b^d)$). It can be improved by decomposing the problem to a set of less complicated sub-problems if possible.

Pseudocode of BTS:

Version optimized with MRV, LCV and FC

backtracking(assignment, csp)

- *If the assignment is complete* ***return assignment***
- *curr_var = the variable with the fewest legal options* *#mrv*
- *For each value in the ordered set of possible values for curr_var* *#LCV*
    - *If assigning value to curr_var satisfies csp.constraints*
        - *add { curr_var: value} to assignment*
        - *if one of curr_var.neighbors has no more possible values*
            return failure *#FC*
        - *result = backtracking(assignment, csp)* *#recursive call*
        - *if result is not failure* ***return result***
    - *remove { curr_var: value} from assignment* *#Backtracking*
- ***Return failure***

IV. Implementation:

In this work, we implemented a BTS approach to solve the Map coloring CSP on the map of South America using 4 colors.

• The CSP class:

This class represents the formal definition of a constraint satisfaction problem. Its member variables are:

- **Variables**: here, the variables are the 13 countries of South America
- **Domain**: The domain for all variables is the same set of colors {red, blue, green, yellow}
- **Constraints**: in our problem, the constraint is that no two adjacent countries have the same color. To check if an assignment {country = color} is possible we simply check if any of the country's neighbors are already assigned this same color.

Other than the 3 main elements of a CSP already mentioned this class contains:

- A dictionary that associates each country with a list of its **neighbors**
- A list to keep track of countries that **have not been assigned yet.**
- A function that checks the **remaining possible colors for a country** with respect to an assignment by subtracting the colors that violate the constraint (assigned to a neighbor of the country in question).

• We defined a function that **checks if color is consistent with a country** for the current assignment. This function is then passed as a constraint argument to the CSP instance.

• We defined a **minimum remaining value (MRV)** function that finds the country that has the fewest number of possible values among the unassigned set of countries in a CSP instance.

• A **BTS** function was created to start the backtracking by calling the backtracking function with an empty assignment and the CSP instance (**initial state**)

• Finally, a **recursive backtracking function** was created. This function takes a CSP as an argument and applies the BTS algorithm on it with **MRV** and **FC** optimization. It returns a dictionary that assigns a color to each country if a solution is found or a 'failure' message if no solution is available.

• For robustness, we created a function to check the final result before trying to plot the map that raises an exception in case one of the following problems has been encountered:

- One of the given countries has not been assigned a color (Incomplete result)
- A country returned in the result does not exist in the given list of countries
- A color returned in the result does not exist in the given list of colors
- Two neighbors have the same color in the result

• In the main function, we created the dictionary that associates each country with its neighbors then we initialized the CSP class according to the given map of South America. After that, we called the BTS function with the CSP object as an argument. The functions necessary to plot the map were given, we only installed the **plotly** and **pandas** dependencies.

• After getting the result, if a solution was found we check it then plot it using the given function. Otherwise, we print an error message in the console.

V. Sample Results:

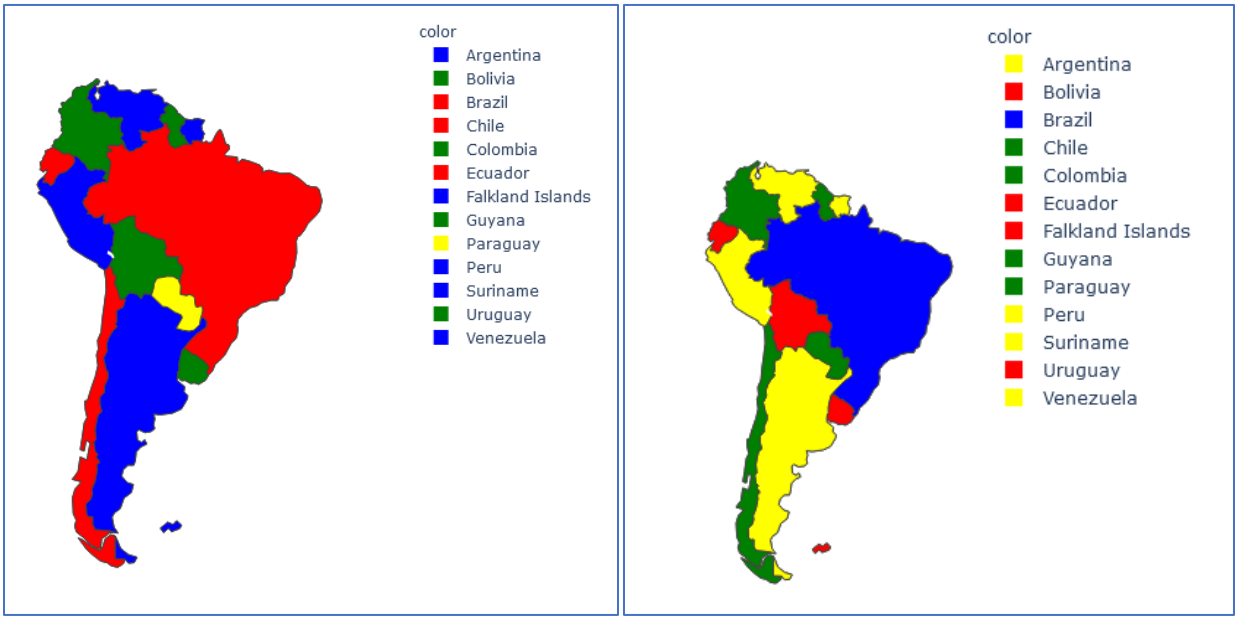

Figure 1: Map coloring results (sample 1)

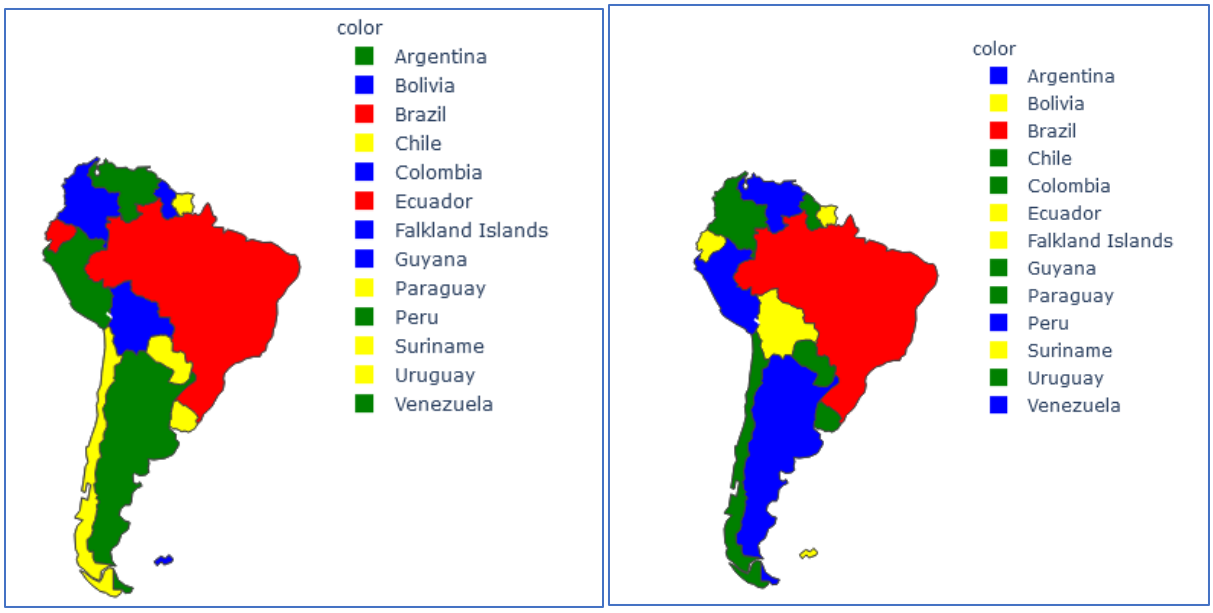

Figure 2: Map coloring results (sample 2)

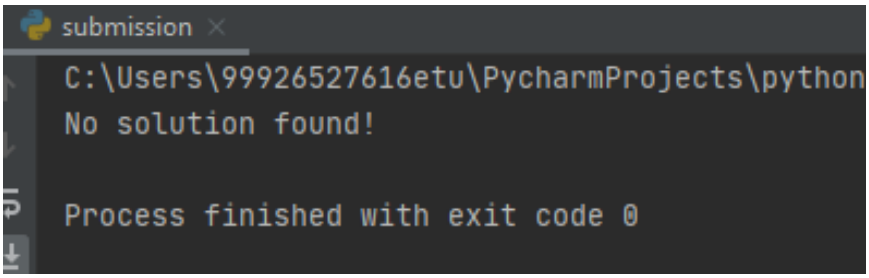

Figure 3: Error message when no solution is found

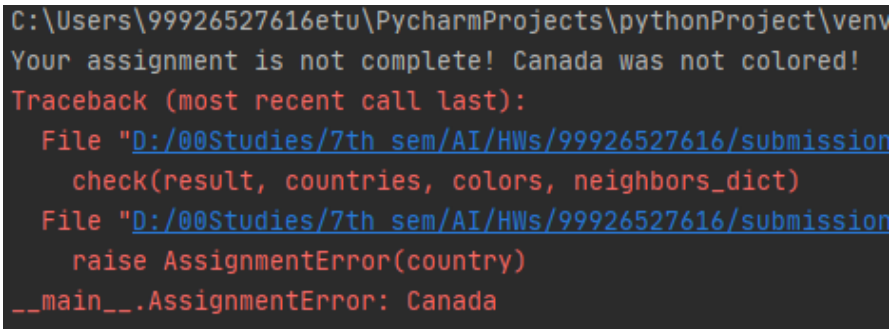

Figure 4: Example of a raised error if the result is wrong

References:

• "Artificial Intelligence: A Modern Approach" (3rd edition), Stuart Jonathan Russel and Peter Norvig, 2010
• Lecture notes