



## **Constraint Satisfaction Problem**

### Map Coloring (Backtracking Search)

#### **Names: -**

1. Maram Abdullah Al-Nuaimi ~ 16S1967
2. Mohammed Amour Al-Marshudi ~ 16S194

## **Introduction: -**

**The focus of the project is on the color mapping problem of the CSP.**

The report demonstrates how the Constraint Satisfaction Problem (CSP) can solve a problem by covering a constraint value to reach the objective. Color mapping was the problem for CSP. In solving with CSP the state is defined by variables  $X_i$  with values from domain  $D_i$ .

Being more specific in each constraint satisfaction problem (CSP), Constraint is defining a set of conditions. For example, if the student gets  $>50$  the student will pass. (It is a condition but at the same time it is a constraint). Satisfaction is defining the constraint value. Problem is defining the issue to reach the constraint value. The goal in constraint satisfaction problems is not to arrive at a certain time or cost, it is to reach the desired outcome. The following concepts of constraint satisfaction problem are formalization, backtracking search, arc consistency, node consistency, and local search. Most algorithms for solving CSPs search systematically through the possible assignments of values to variables. Such algorithms are guaranteed to find a solution, if one exists, or to prove that the problem is not satisfied.

In this report, a generated build-it was constructed by a color mapping problem that involved coding steps in order to solve the problem using a finite domain, to be assigned to each variable in the problem, so that all constraints relating the variables are satisfied.

By using the Python Programming language, the Constraints Satisfaction Problems are implemented in Visual Studio Code. In map coloring CSP, each region must be colored in such a way that any two or more adjacent regions have different colors. Map-coloring constraints can be summarized as follows: Each region is assigned only by one color, out of all possible colors. As part of our case problem, we define the domain, variables, and constraints.

### **According to Thomaz(2022) research about Pros and Cons of CSP:**

#### **Pros of CSP:**

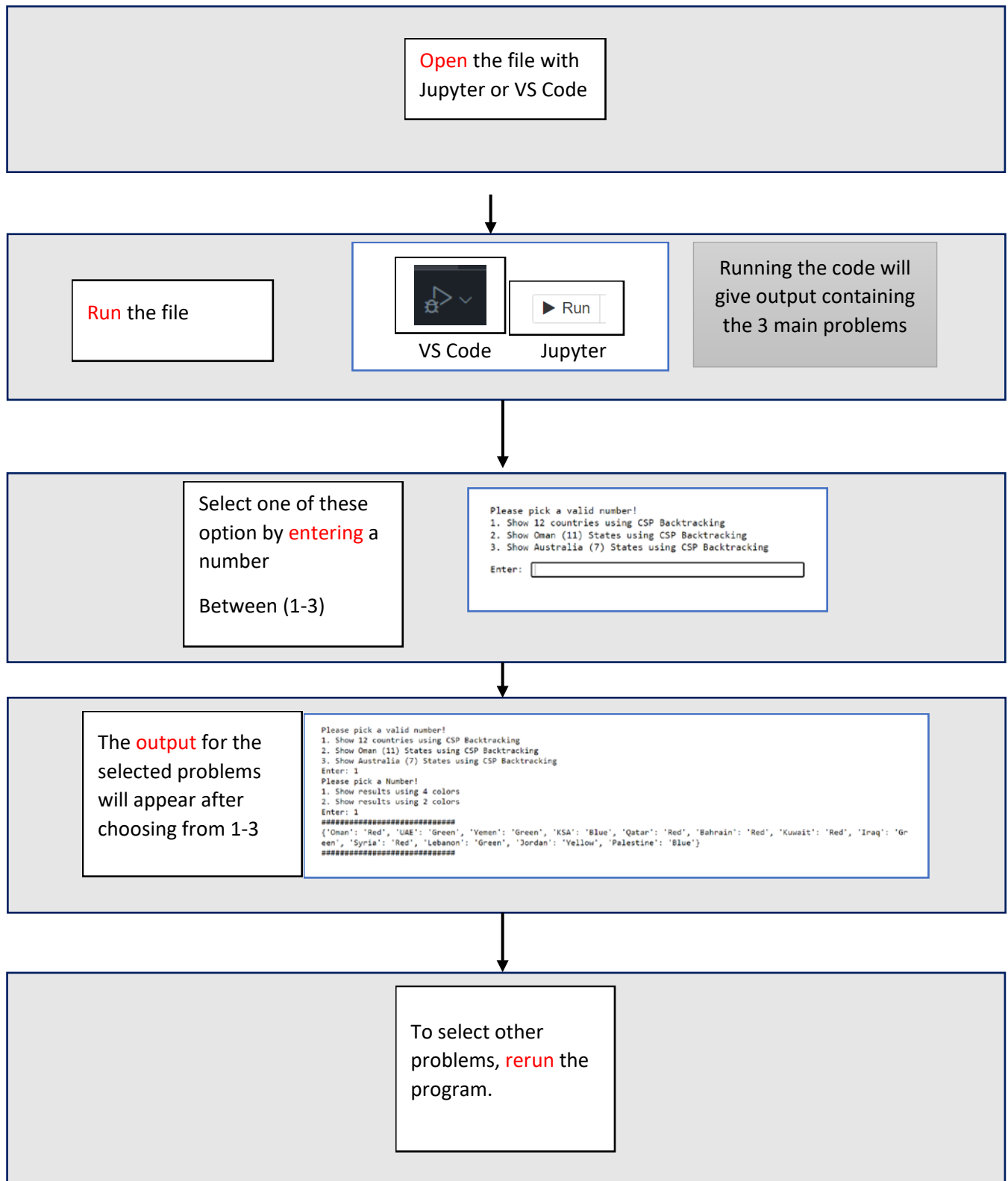
- Backtracking can almost solve any problems, due to its brute-force nature.
- Can be used to find all the existing solutions if there exists for any problem.
- It is a step-by-step representation of a solution to a given problem, which is very easy to understand.
- Very easy to write the code, and to debug.

#### **Cons of CSP:**

- It is very slow compared to other solutions.
- Depending on the data that you have, there is the possibility to perform a very large search with Backtracking and at the end don't find any match to your search params.
- Very-high computational cost, Backtracking is a recursive algorithm that consumes a lot from the memory and from the processor.

## Guideline: -

Detailed instructions for running the code.



```

print("Please pick a valid number!")
print("1. Show 12 countries using CSP Backtracking")
print("2. Show Oman (11) States using CSP Backtracking")
print("3. Show Australia (7) States using CSP Backtracking")
num=int(input("Enter: "))

```

(Figure 1)

First, in figure 1 there are three main coloring problems where the user can input a number 1 or 2 or 3. The problem number 1 is about color mapping problem to show the 12 countries(states), the problem number 2 to show 11 Oman states and number 3 to show the color mapping problem of 7 states.

```

if(num==1):
    print("Please pick a Number!")
    print("1. Show results using 4 colors")
    print("2. Show results using 2 colors")
    nums=int(input("Enter: "))
    if(nums==1):
        print("#####")

```

(Figure 2: Option 1)

There are two case problems of the 12 countries (states), the first case is to show the outcome when there are four domain colors. The second option is to show how the output will look when there are two domain colors. The constraint will break when there are two color domains with 12 variable states. This will result in a false result, which is zero weight, but if the user select the first option, with four colour domains, you will get a True output since it covers all the constraints. Therefore, the weight will be 1.

```

if(nums==1):
    print("#####")
    #domain
    colors=['Red','Green','Blue','Yellow'] #minimum color for these state=4
    #variables
    states=['Oman','UAE','Yemen','KSA','Qatar','Bahrain','Kuwait','Iraq','Syria','Lebanon','Jordan','Palestine'] #12 countries

```

(Figure 3: Option 1/Case 1)

(Figure 3) defines the domains and variables of the case problem 1 of the Gulf.

```

neighbors= {}
# connection between the border
#constraints
#adjacent regions must have different colors
|      #adjacents      #regions
neighbors['Oman']=['UAE','KSA','Yemen']
neighbors['UAE']=['Oman','KSA']
neighbors['Yemen']=['Oman','KSA']
neighbors['KSA']=['Yemen','Oman','UAE','Qatar','Kuwait','Iraq','Jordan']
neighbors['Qatar']=['KSA']
neighbors['Bahrain']=[]
neighbors['Kuwait']=['Iraq','KSA']
neighbors['Iraq']=['Kuwait','KSA','Syria','Jordan']
neighbors['Syria']=['Palestine','Lebanon','Jordan','Iraq']
neighbors['Lebanon']=['Palestine','Syria']
neighbors['Jordan']=['Palestine','KSA','Syria','Iraq']
neighbors['Palestine']=['Jordan','Syria','Lebanon']

```

(Figure 4: Option 1 / Case 1)

The neighbors [] method to get the nearest states to that specific state. As in (Figure 4), Oman's neighbors are UAE, KSA and Yemen. Likewise, UAE's neighbors are Oman and KSA. Each State have neighbors of their own.

```

colors_of_states={}

def promising(state,color):
    for neighbor in neighbors.get(state):
        colors_of_neighbor= colors_of_states.get(neighbor)
        if colors_of_neighbor== color:
            return False          # if the neighbor color and the state color are the same it will return False
        return True              # if the neighbor color and the state color are diffrent it will return True

def get_color_for_state(state):
    for color in colors:
        if promising(state, color):    #if promising which color will be allowed to the state // it takes us to def promising for checking wether the neighbor
            return color                # is having the same color of the state

# here we are assigning the color for the state(variable)
def main():
    for state in states:
        colors_of_states[state]=get_color_for_state(state)
    print(colors_of_states)
main()
print("#####")

```

(Figure 5: Option 1 / Case 1)

There are three main functions in Figure 5, and they are def promising, def get\_color\_for\_state and def main. When colors\_of\_neighbor and colors\_of\_states have the same color, def promising will return false, since two adjacent colors cannot have the same color. When the neighbor and state have different colors, It will return true and move to the second function, def get\_color\_for\_state, will color the state, and in the main function, the output will be printed. As shown in (Figure 6: Output).

```

Enter: 1
Please pick a Number!
1. Show results using 4 colors
2. Show results using 2 colors
Enter: 1
#####
{'Oman': 'Red', 'UAE': 'Green', 'Yemen': 'Green', 'KSA': 'Blue', 'Qatar': 'Red', 'Bahrain': 'Red', 'Kuwait': 'Red', 'Iraq': 'Green', 'Syria': 'Red', 'Lebanon': 'Green', 'Jordan': 'Yellow', 'Palestine': 'Blue'}

```

(Figure 6: Output/ Case 1)

```

elif(nums==2):
    print("#####")
    #domain
    colors=['Red','Green'] #minimum color for these state=4
    #variables
    states=['Oman','UAE','Yemen','KSA','Qatar','Bahrain','Kuwait','Iraq','Syria','Lebanon','Jordan','Palestine'] #12 countries

```

(Figure 7: Option 1 / Case 2)

The Difference between Case 1 and Case 2 are the domains, and the minimum color/domain of problem 1 is 4. If there are two domains in the problem, then there will be an error and some countries will print with "None" color and the output is shown in (Figure 8: Output/ Case 2).

```

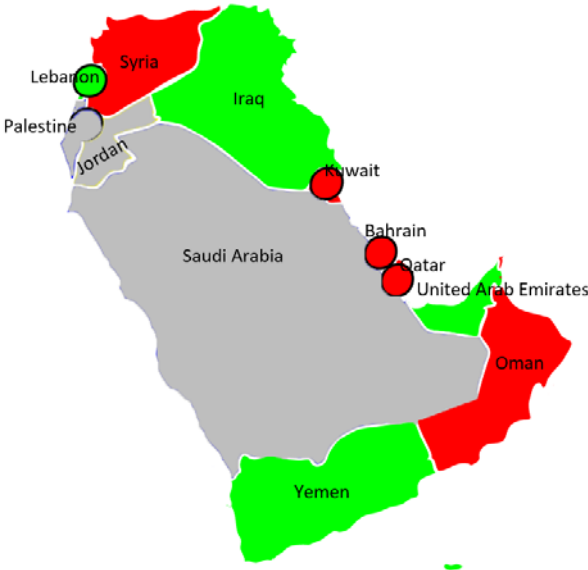

1. Show results using 4 colors
2. Show results using 2 colors
Enter: 2
#####
{'Oman': 'Red', 'UAE': 'Green', 'Yemen': 'Green', 'KSA': None, 'Qatar': 'Red', 'Bahrain': 'Red', 'Kuwait': 'Red', 'Iraq': 'Green', 'Syria': 'Red', 'Lebanon': 'Green', 'Jordan': None, 'Palestine': None}
#####
Here You will see the [ None ] values becuse minimum color is = 4
#####

```

(Figure 8: Output/ Case 2)

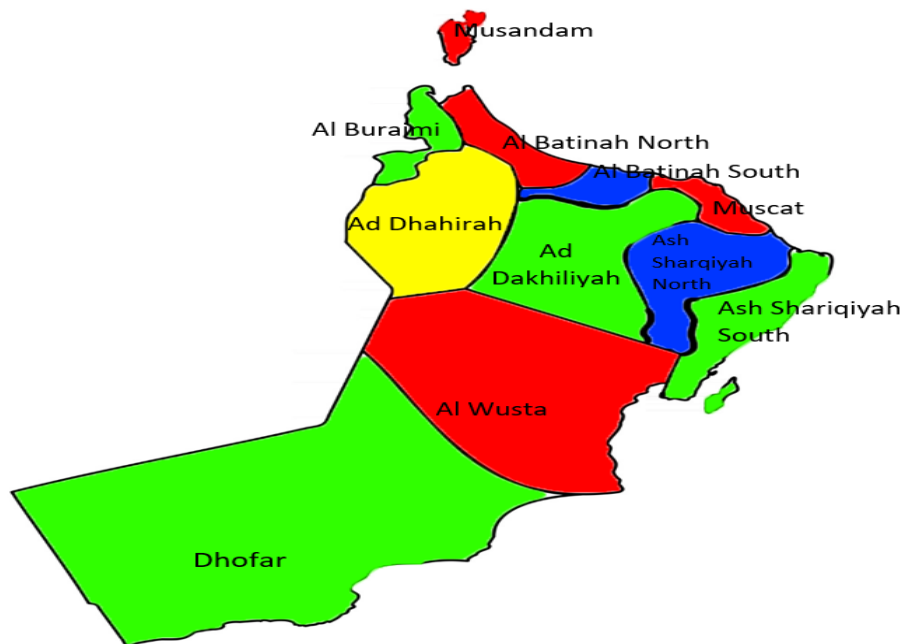


For more clarification problem of map coloring between Case 1 and Case 2 of the 12 countries (states).

	
<p><b>Domains:</b>  <code>colors=['Red','Green']</code></p> <p><b>Map Coloring:</b></p> <p>{'Oman': 'Red', 'UAE': 'Green', 'Yemen': 'Green', 'KSA': <b>None</b>, 'Qatar': 'Red', 'Bahrain': 'Red', 'Kuwait': 'Red', 'Iraq': 'Green', 'Syria': 'Red', 'Lebanon': 'Green', <b>Jordan': None, 'Palestine': None</b>}</p> <p><b>Constraints is not Satisfied</b></p>	<p><b>Domains:</b>  <code>colors=['Red','Green','Blue','Yellow']</code></p> <p><b>Map Coloring:</b></p> <p>{'Oman': 'Red', 'UAE': 'Green', 'Yemen': 'Green', 'KSA': 'Blue', 'Qatar': 'Red', 'Bahrain': 'Red', 'Kuwait': 'Red', 'Iraq': 'Green', 'Syria': 'Red', 'Lebanon': 'Green', 'Jordan': 'Yellow', 'Palestine': 'Blue'}</p> <p><b>Constraints is Satisfied</b></p>
<p><b>Variables:</b>  <code>states=['Oman','UAE','Yemen','KSA','Qatar','Bahrain','Kuwait','Iraq','Syria','Lebanon','Jordan','Palestine']</code></p> <p><b>Constraints:</b>          Adjacent regions must have different colors;</p> <p><b>Goal:</b> all regions are colored, but the adjacent region must have different color.</p> <p><b>The Type of constraint is</b> a Binary Constraint. It is a backtracking-search (CSP), the search technique that we have used is Uninformed (or blind) search.</p> <p><b>Initiating/ partial assignment:</b> 'Oman': 'Red' → “First variable: First Domain”</p>	

## Map coloring problem for the Oman states:

---



### Variables:

States=['Muscat','Ash Sharqiyah South','Ash Sharqiyah North','Ad Dakhiliyah','Al Batinah South','Al Batinah North','Ad Dhahirah','Al Buraimi','Musandam','Al Wusta','Dhofar']

### Domains:

Color =['Red','Green','Blue','Yellow']

### Constraints:

Adjacent regions must have different colors.

### Map coloring:

{Dhofar = green, Al Wusta = red, Ad Dhahirah = yellow, Ad Dakhiliyah = green, Ashariqiyah North = blue, Ash Shariqiyah south = green, Al Buraimi = green, Al Batinah North = red, Al Batinah south = Blue, Muscat = red , Musandam = red}

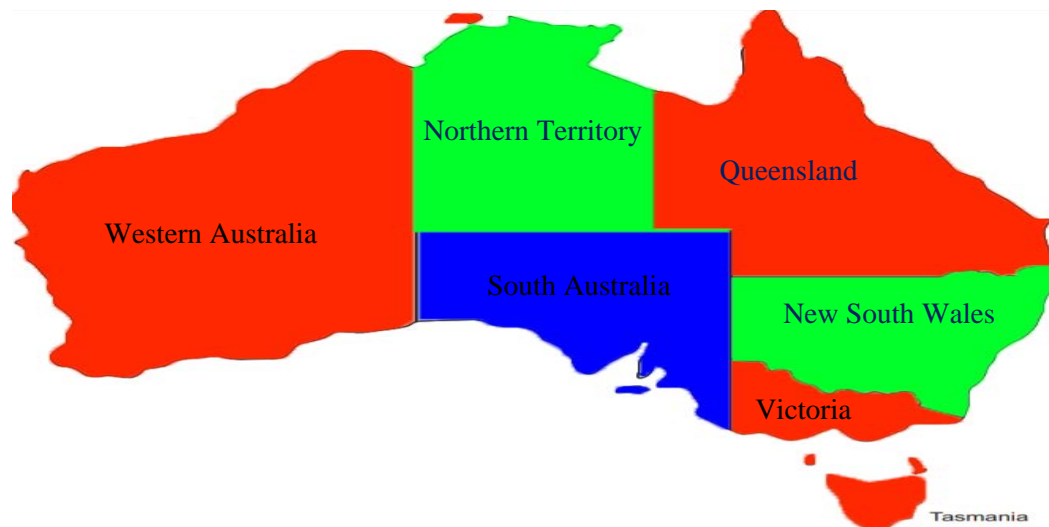
### Goal:

All regions are colored but the adjacent region must have different color.

[Constraints is Satisfied]

### Map coloring problem for the Australia states:

---



#### Variables:

States=['Western Australia', 'Northern Territory', 'South Australia', 'Queensland', 'New South Wales', 'Victoria', 'Tasmania']

#### Domains:

Colors=['Red','Green','Blue']

#### Constraints:

Adjacent regions must have different colors.

#### Map coloring:


{'Western Australia': 'Red', 'Northern Territory': 'Green', 'South Australia': 'Blue', 'Queensland': 'Red', 'New South Wales': 'Green', 'Victoria': 'Red', 'Tasmania': 'Red'}

#### Goal:

All regions are colored but the adjacent region must have different color.

[Constraints is Satisfied]

## References:

- I. Constraint Satisfaction Problems in Artificial Intelligence - TAE. (n.d.). <https://www.tutorialandexample.com/constraint-satisfaction-problems-in-artificial-intelligence>.
- II. Munipraveena Rela's Classroom. (2021, May 28). Python program for Graph coloring Problem /Artificial Intelligence Lab [Video]. YouTube. <https://www.youtube.com/watch?v=8deLBev5MFE>
- III. Thomaz, J. (2022, January 27). What is Backtracking? DEV Community . <https://dev.to/josethz00/what-is-backtracking-56cg>.

