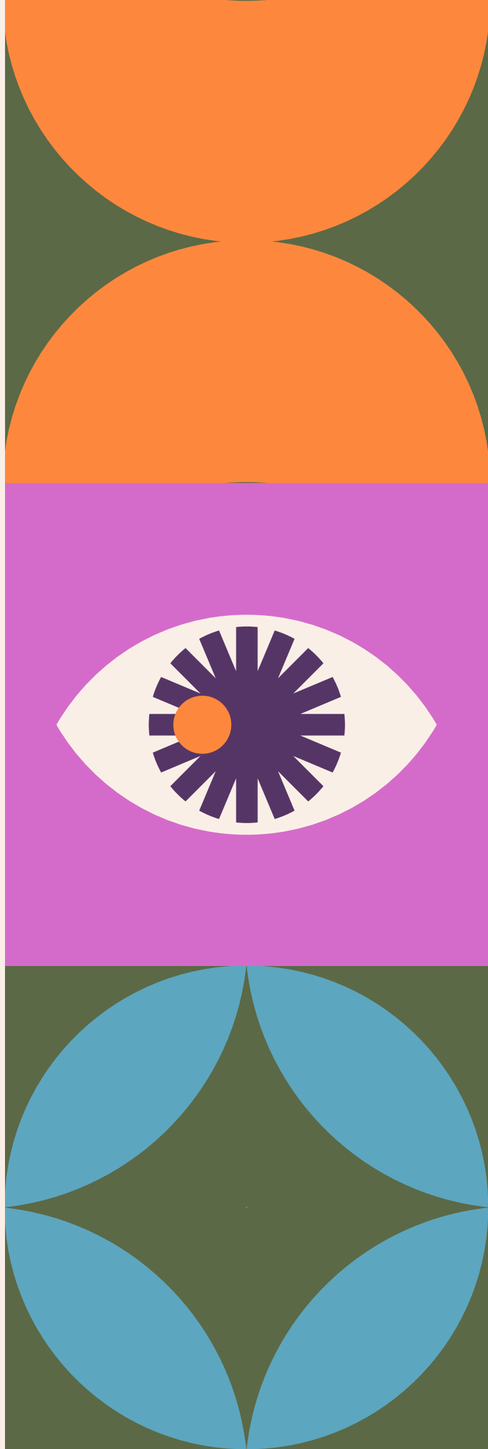
TUGAS 3 INDIVIDU

Map Colouring

Constraint Satisfaction Problem

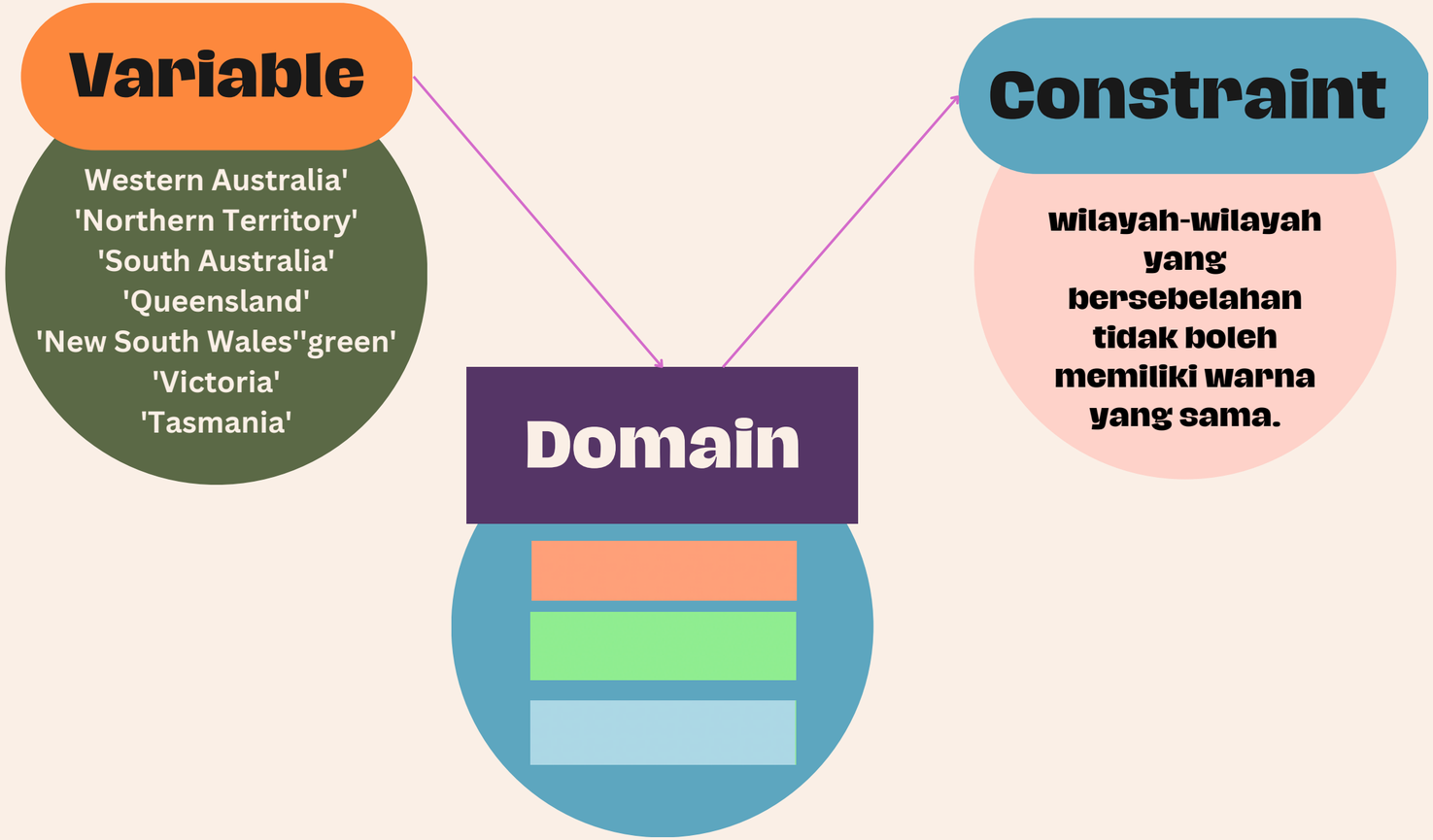


LAURIVASYA GADHING SYAHAFIDH - 5025211136

Constraint Satisfaction Problem

CSP adalah teknik algoritmik yang digunakan untuk menyelesaikan masalah di mana harus memilih nilai dari sejumlah variabel sedemikian rupa sehingga nilai tersebut memenuhi serangkaian batasan atau aturan yang telah ditentukan sebelumnya. Map coloring adalah salah satu contoh dari permasalahan CSP, di mana tujuannya adalah untuk memberi warna pada suatu peta sehingga tidak ada dua wilayah yang bersebelahan memiliki warna yang sama.

Application of Constraint-satisfaction Frameworks

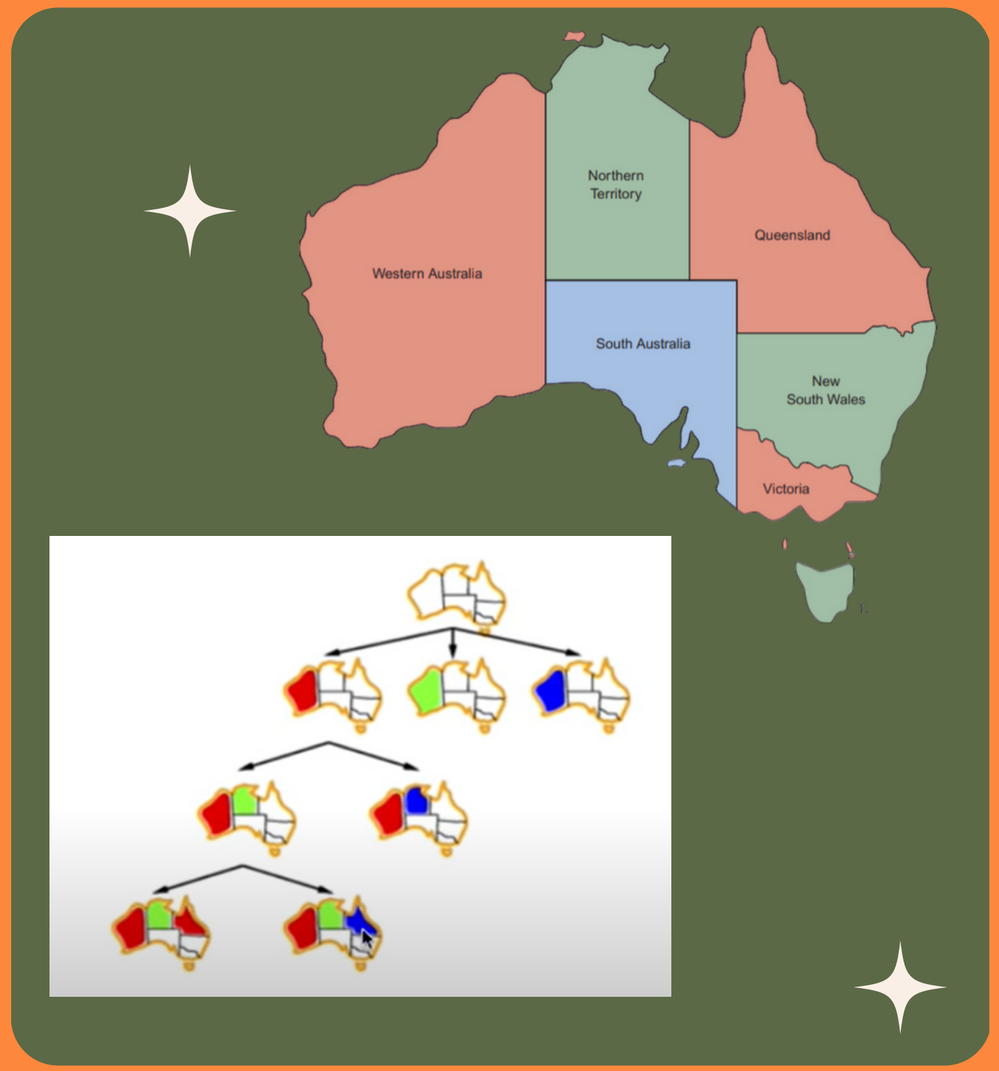


Secara umum, CSP terdiri dari tiga komponen yaitu variabel, domain, dan constraint.

Variabel merepresentasikan objek yang ingin diberi nilai, domain adalah kumpulan nilai yang mungkin untuk setiap variabel, dan constraint adalah batasan-batasan atau keterkaitan antar variabel yang harus dipenuhi agar solusi dapat ditemukan.

Untuk memecahkan masalah map coloring dengan menggunakan CSP, kita dapat menggunakan algoritma backtracking.

Pada awalnya, semua wilayah belum memiliki warna. Kemudian, kita akan memilih sebuah wilayah dan mencoba memberikan warna pada wilayah tersebut. Jika warna yang dipilih tidak melanggar constraint (yaitu wilayah-wilayah yang bersebelahan tidak memiliki warna yang sama), maka kita akan mencoba memberikan warna pada wilayah selanjutnya. Jika warna yang dipilih tidak dapat dipakai, maka kita akan mencoba warna yang lain. Jika tidak ada warna yang dapat dipakai, maka kita akan mundur ke wilayah sebelumnya dan mencoba warna yang berbeda. Proses ini akan terus berlanjut sampai semua wilayah terisi warna atau tidak ada solusi yang dapat ditemukan.



Pada gambar disamping, terdapat 7 wilayah yang ingin diwarnai menggunakan 3 warna yang berbeda. Wilayah-wilayah yang bersebelahan ditandai dengan garis Algoritma backtracking akan mencoba memberikan warna pada wilayah-wilayah tersebut sehingga tidak ada wilayah yang bersebelahan memiliki warna yang sama. Pada akhirnya, semua wilayah terisi warna dan solusi yang ditemukan adalah

{'Western Australia': 'salmon', 'Northern Territory': 'light green', 'South Australia': 'light blue', 'Queensland': 'salmon', 'New South Wales': 'light green', 'Victoria': 'salmon', 'Tasmania': 'light green'}

Class Constraint(Generic[V, D], ABC)

- Pada bagian awal, terdapat pengimportan beberapa module yang akan digunakan dalam program, seperti modul untuk tipe data generic (Generic), untuk tipe variabel (TypeVar), untuk abstraksi kelas (ABC), dan modul untuk plot grafik (matplotlib).
- Kemudian, didefinisikan tipe variabel untuk variabel (V) dan domain (D) dengan fungsi TypeVar().
- Setelah itu, didefinisikan kelas Constraint yang memiliki atribut variables dan method abstract satisfied(). Atribut variables adalah list dari variabel yang terlibat dalam suatu constraint, sedangkan method satisfied() adalah method yang akan mengembalikan nilai boolean True jika constraint tersebut telah terpenuhi, dan False jika tidak.

```python
from typing import Generic, TypeVar, Dict, List, Optional
from abc import ABC, abstractmethod
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon

V = TypeVar('V') # variable type
D = TypeVar('D') # domain type

class Constraint(Generic[V, D], ABC):
    def __init__(self, variables: List[V]) -> None:
        self.variables = variables

    @abstractmethod
    def satisfied(self, assignment: Dict[V, D]) -> bool:
        ...
```

Class CSP(Generic[V, D]):

- Setelah itu terdapat Class CSP (Constraint Satisfaction Problem) yang memiliki atribut variables, domains, dan constraints. Variabel variables adalah list dari variabel yang akan dibatasi (constrained), domains adalah dictionary yang menyimpan domain dari setiap variabel, dan constraints adalah dictionary yang menyimpan constraint yang terdapat pada setiap variabel.
- Kelas CSP juga memiliki beberapa method, seperti add_constraint() untuk menambahkan constraint ke dalam dictionary constraints, consistent() untuk memeriksa konsistensi dari suatu variabel, dan backtracking_search() untuk melakukan pencarian solusi dengan menggunakan algoritma backtracking.

```python
class CSP(Generic[V, D]):
    def __init__(self, variables: List[V], domains: Dict[V, List[D]]) -> None:
        self.variables: List[V] = variables # variables to be constrained
        self.domains: Dict[V, List[D]] = domains # domain of each variable
        self.constraints: Dict[V, List[Constraint[V, D]]] = {}
        for variable in self.variables:
            self.constraints[variable] = []
            if variable not in self.domains:
                raise LookupError("Every variable should have a domain assigned to it.")

    def add_constraint(self, constraint: Constraint[V, D]) -> None:
        for variable in constraint.variables:
            if variable not in self.variables:
                raise LookupError("Variable in constraint not in CSP")
            else:
                self.constraints[variable].append(constraint)

    def consistent(self, variable: V, assignment: Dict[V, D]) -> bool:
        for constraint in self.constraints[variable]:
            if not constraint.satisfied(assignment):
                return False
        return True

    def backtracking_search(self, assignment: Dict[V, D] = {}) -> Optional[Dict[V, D]]:
        if len(assignment) == len(self.variables):
            return assignment

        unassigned: List[V] = [v for v in self.variables if v not in assignment]

        first: V = unassigned[0]
        for value in self.domains[first]:
            local_assignment = assignment.copy()
            local_assignment[first] = value
            # if we're still consistent, we recurse (continue)
            if self.consistent(first, local_assignment):
                result: Optional[Dict[V, D]] = self.backtracking_search(local_assignment)
                # if we didn't find the result, we will end up backtracking
                if result is not None:
                    return result
        return None
```

Subclass MapColoringConstraint

- Selanjutnya, didefinisikan subclass MapColoringConstraint dari kelas Constraint. Kelas ini merepresentasikan constraint untuk kasus Map Coloring, yaitu dua negara yang bersebelahan tidak boleh memiliki warna yang sama. Kelas ini memiliki dua atribut, yaitu place1 dan place2 yang merepresentasikan nama negara pada peta yang akan dibatasi warnanya. Kelas MapColoringConstraint mengimplementasikan method satisfied(), yaitu method yang didefinisikan pada kelas parent Constraint.
- Pada bagian main program, pertama-tama dilakukan inisialisasi variabel variables dan domains yang merepresentasikan list dari nama negara pada peta dan domain dari setiap variabel. Variabel variables merepresentasikan list dari nama negara yang akan dicari warnanya. Variabel domains merepresentasikan dictionary yang berisi list warna-warna yang mungkin untuk setiap variabel.

```python
class MapColoringConstraint(Constraint[str, str]):
    def __init__(self, place1: str, place2: str) -> None:
        super().__init__([place1, place2])
        self.place1: str = place1
        self.place2: str = place2

    def satisfied(self, assignment: Dict[str, str]) -> bool:
        if self.place1 not in assignment or self.place2 not in assignment:
            return True
        return assignment[self.place1] != assignment[self.place2]

if __name__ == "__main__":
    variables: List[str] = ["Western Australia", "Northern Territory", "South Australia",
    "Queensland", "New South Wales", "Victoria", "Tasmania"]

    domains: Dict[str, List[str]] = {}
    for variable in variables:
        domains = {v: ["salmon", "light green", "light blue"] for v in variables}
```

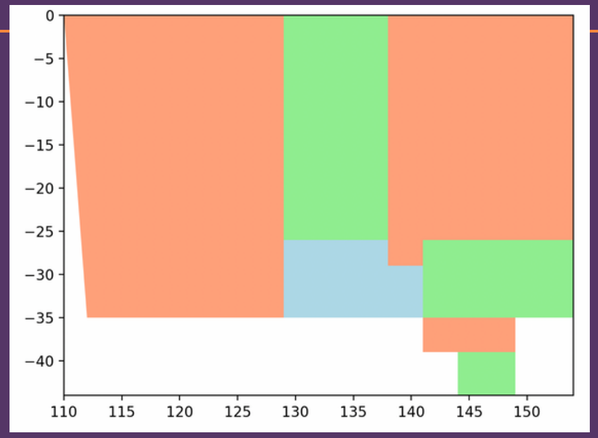
Objek CSP(Constraint Satisfaction Problem)

- Selanjutnya, dibuat objek csp dengan kelas CSP dan ditambahkan constraint-constraint yang sudah ditentukan sebelumnya menggunakan method add_constraint(). Constraint yang ditambahkan adalah constraint MapColoringConstraint yang merepresentasikan constraint untuk kasus Map Coloring, yaitu dua negara yang bersebelahan tidak boleh memiliki warna yang sama.

```
    csp: CSP[str, str] = CSP(variables, domains)
    csp.add_constraint(MapColoringConstraint("Western Australia", "Northern Territory"))
    csp.add_constraint(MapColoringConstraint("Western Australia", "South Australia"))
    csp.add_constraint(MapColoringConstraint("South Australia", "Northern Territory"))
    csp.add_constraint(MapColoringConstraint("Queensland", "Northern Territory"))
    csp.add_constraint(MapColoringConstraint("Queensland", "South Australia"))
    csp.add_constraint(MapColoringConstraint("Queensland", "New South Wales"))
    csp.add_constraint(MapColoringConstraint("New South Wales", "South Australia"))
    csp.add_constraint(MapColoringConstraint("Victoria", "South Australia"))
    csp.add_constraint(MapColoringConstraint("Victoria", "New South Wales"))
    csp.add_constraint(MapColoringConstraint("Victoria", "Tasmania"))

    # Solve the problem
solution: Optional[Dict[str, str]] = csp.backtracking_search()
if solution is None:
    print("No solution found!")
else:
    print(solution)
```

- Selanjutnya, variabel map_coordinates dan colors didefinisikan. map_coordinates adalah sebuah dictionary yang menyimpan koordinat dari setiap negara pada peta. Setiap negara direpresentasikan sebagai sebuah tuple berisi koordinat titik-titik sudut yang membentuk negara tersebut. colors adalah sebuah dictionary yang menyimpan warna-warna yang akan digunakan dalam map coloring. Setiap warna direpresentasikan dengan kode heksadesimal.
- Setelah itu, dilakukan plotting dengan menggunakan library Matplotlib. Pertama-tama, objek Polygon dibuat untuk setiap negara pada peta, dengan warna negara dipilih berdasarkan solusi yang telah ditemukan pada proses pencarian solusi. Objek Polygon kemudian ditambahkan ke dalam objek axis menggunakan method add_patch().
- Setelah semua negara ditambahkan ke dalam objek axis, batas-batas sumbu x dan y ditentukan menggunakan method set_xlim() dan set_ylim(). Terakhir, plot yang telah dibuat ditampilkan menggunakan method show().



{'Western Australia': 'salmon', 'Northern Territory': 'light green', 'South Australia': 'light blue', 'Queensland': 'salmon', 'New South Wales': 'light green', 'Victoria': 'salmon', 'Tasmania': 'light green'}

```python
# Define the map coordinates
map_coordinates = {
    "Western Australia": ((110, 0), (129, 0), (129, -35), (112, -35)),
    "Northern Territory": ((129, 0), (138, 0), (138, -26), (129, -26)),
    "South Australia": ((129, -26), (141, -26), (141, -35), (129, -35)),
    "Queensland": ((138, 0), (154, 0), (154, -29), (138, -29)),
    "New South Wales": ((141, -26), (154, -26), (154, -35), (141, -35)),
    "Victoria": ((141, -35), (149, -35), (149, -39), (141, -39)),
    "Tasmania": ((144, -39), (149, -39), (149, -44), (144, -44))
}

# Define the colors
colors = {
"salmon": "#FFA07A",
"light green": "#90EE90",
"light blue": "#ADD8E6"
}

# Create the figure and axis objects
fig, ax = plt.subplots()

# Plot each state
for state, coords in map_coordinates.items():
    color = solution[state]
    poly = Polygon(coords, facecolor=colors[color])
    ax.add_patch(poly)

# Set the x and y limits
ax.set_xlim([110, 154])
ax.set_ylim([-44, 0])

# Show the plot
plt.show()
```