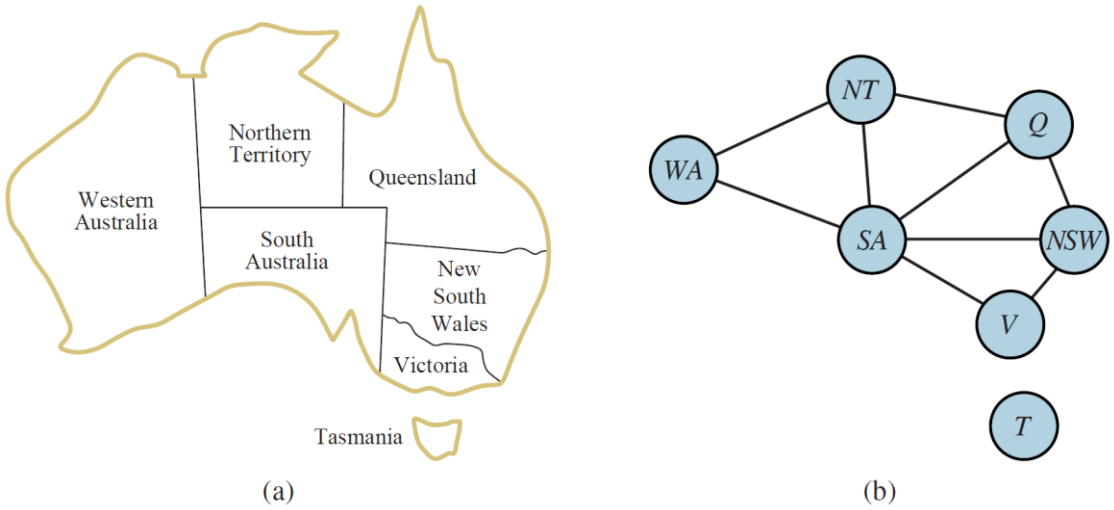
OptaPlanner CSP Problem

1. Problem Definition

본 과제에서 해결할 문제는 호주 map coloring 문제이다. 호주 map coloring 문제는 호주의 7개의 state를 3가지 색으로 칠하는 문제로 인접한 state의 색은 다른 색으로 칠해져야 하는 제약조건이 있다. 7개의 주는 Fig. 1 (a)와 같이 인접하여 있으며, 인접한 state끼리 그래프로 표현하면 Fig. 2 (b)와 같다.



<Fig. 1 (a) Australia map coloring problem, (b) Graph representation>

CSP 문제를 해결하기 위해선 변수(variable)를 정의하고 변수 값의 영역(domain)과 제약조건(constraint)을 정의하여야 한다.

1.1 Variables

본 CSP 문제에서 변수는 7개의 state가 된다. 7개의 state를 WA, NT, SA, Q, NSW, V와 T로 정의하고 각각 1, 2, ..., 7의 ID를 가지도록 정의하였다.

1.2 Domain

각 변수들이 가질 수 있는 domain은 RED, GREEN, BLUE의 3가지 색을 가질 수 있다고 정의하였다. 각각 1, 2, 3의 ID를 가진다.

1.3 Constraint

본 문제에서 각 state는 하나의 색을 가져야 하며, 인접한 주는 같은 색을 가지면 안되는 제약조건이 있다.

2. Implementation

2.1 Domain and Variables

OptaPlanner를 사용하여 다음과 같이 정의하였다. Variable인 7개의 state는 id와 name, neighbor, color를 가지는 planning entity인 ***State*** 클래스로 정의하였고, domain인 color는 id와 color을 가지는 problem fact인 ***Color*** 클래스로 정의하였다.

```
@problem_fact
class Color:
    def __init__(self, id, color):
        self.id = id
        self.color = color

    @planning_id
    def get_id(self):
        return self.id
```

```
@planning_entity
class State:
    def __init__(self, id, name, neighbor=None, color=None):
        self.id = id
        self.name = name
        self.color = color
        self.neighbor = neighbor

    @planning_id
    def get_id(self):
        return self.id

    @planning_variable(Color, ["colorRange"])
    def get_color(self):
        return self.color

    def set_color(self, new_color):
        self.color = new_color
```

2.2 Map, problem generation

또한 color로 색칠되어 있는 state를 담는 ***Map*** 클래스와 문제 생성과 state와 color를 포함한 map을 초기화하기 위한 ***generate_problem*** 함수도 정의하였다.

```
@planning_solution
class Map:
    def __init__(self, color_list, state_list, score=None):
        self.color_list = color_list
        self.state_list = state_list
        self.score = score

    @problem_fact_collection_property(Color)
    @value_range_provider("colorRange")
    def get_color_list(self):
        return self.color_list

    @planning_entity_collection_property(State)
    def get_state_list(self):
        return self.state_list

    @planning_score(HardSoftScore)
    def get_score(self):
        return self.score

    def set_score(self, score):
        self.score = score
```

```
def generate_problem():
    color_list = [
        Color(1, 'Red'),
        Color(2, 'Green'),
        Color(3, 'Blue')
    ]
    state_list = [
        State(1, 'Western Australia', [2,3]),
        State(2, 'Northern Territory', [1,3,4]),
        State(3, 'South Australia', [1,2,4,5,6]),
        State(4, 'Queensland', [2,3,5]),
        State(5, 'New South Wales', [3,4,6]),
        State(6, 'Victoria', [3,5]),
        State(7, 'Tasmania', []),
    ]

    state = state_list[0]
    state.set_color(color_list[0])

    return Map(color_list, state_list)
```

2.3 Constraints

각 state는 인접한 state와 다른 색깔로 칠해져야 한다. 이런 제약조건을 정의해주었다. 제약조건을 정의하기 위하여 각 state마다 인접한 state의 ID를 가지고 있는 리스트를 ***generate_problem*** 함수에서 정의해주었다. 이 리스트를 통해 서로 같은 색을 가지고 있는 state 중 리스트에 포함되어 있는 id의 state가 있을 경우 conflict가 발생하도록 ***isAdjacent*** 함수와 ***color_conflict*** 함수를 정의하여 Hard conflict로 정의하였다.

```
def isAdjacent(state1, state2):
    if state1.id in state2.neighbor:
        return True
    else: return False
```

```
def color_conflict(constraint_factory):
    return constraint_factory \
            .forEach(StateClass) \
            .join(StateClass,
                  [
                      Joiners.equal(lambda state: state.color),
                      Joiners.filtering(lambda state1, state2: isAdjacent(state1, state2))
                  ]) \
            .penalize("Color conflict", HardSoftScore.ONE_HARD)
```

3. Result

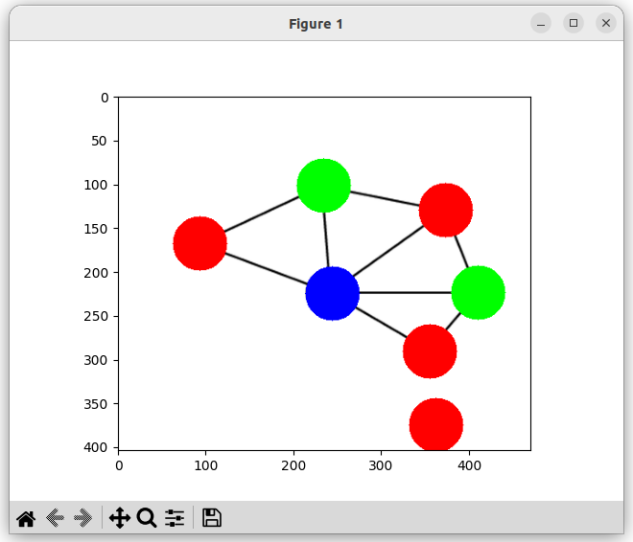
위 정의된 CSP 문제를 OptaPlanner를 이용하여 해결하고 간단한 plot 코드를 통해 시각화한 결과는 다음과 같다.

```
solver_config = optapy.config.solver.SolverConfig() \
    .withEntityClasses(get_class(State)) \
    .withSolutionClass(get_class(Map)) \
    .withConstraintProviderClass(get_class(define_constrains)) \
    .withTerminationSpentLimit(Duration.ofSeconds(10))

solution = solver_factory_create(solver_config)\
    .buildSolver()\
    .solve(generate_problem())
print(solution)
plot(solution)
```

```
(opta) leo@leo-GP66-Leopard-11UG:~/opta/mapcoloring$ python main.py
02:29:37.610 [main        ] INFO  Solving started: time spent (46), best score (-6init/0hard/0soft),
 environment mode (REPRODUCIBLE), move thread count (NONE), random (JDK with seed 0).
02:29:37.650 [main        ] INFO  Construction Heuristic phase (0) ended: time spent (87), best scor
e (0hard/0soft), score calculation speed (527/sec), step total (6).
02:29:47.563 [main        ] INFO  Local Search phase (1) ended: time spent (10000), best score (0har
d/0soft), score calculation speed (217514/sec), step total (160000).
02:29:47.564 [main        ] INFO  Solving ended: time spent (10000), best score (0hard/0soft), score
 calculation speed (215515/sec), phase total (2), environment mode (REPRODUCIBLE), move thread count
 (NONE).
Map(color_list=Color(id=1, color=Red),
Color(id=2, color=Green),
Color(id=3, color=Blue),
state_list=State(id=1, name=Western Austrailia, color=Color(id=1, color=Red)),
State(id=2, name=Northern Territory, color=Color(id=2, color=Green)),
State(id=3, name=South Austrailia, color=Color(id=3, color=Blue)),
State(id=4, name=Queensland, color=Color(id=1, color=Red)),
State(id=5, name=New South Wales, color=Color(id=2, color=Green)),
State(id=6, name=Victoria, color=Color(id=1, color=Red)),
State(id=7, name=Tasmania, color=Color(id=1, color=Red)),
score=0hard/0soft)
```

<Fig. 2 Result of map coloring problem>



<Fig. 3 Visualized result of map coloring problem>

4. Discussion

인접한 state끼리 다른 color를 가지게 된 것을 확인할 수 있다. 마지막 score에서도 hard conflict가 0인 것을 확인할 수 있다.

본 문제는 그래프 구조로 해결할 수 있는 그래프 구조에서 A, B 두 노드가 연결되어 있으면 A는 B의 neighbor가 되고 B는 A의 neighbor가 된다. 따라서 본 문제에서 A 노드에서 B노드와 겹치지 않다는 것을 확인하면, B노드 또한 A와 겹치지 않으므로 확인할 필요가 없어진다. 이 과정을 통해 planning 시간을 줄일 수 있는지 확인해보았다. ***generate_problem*** 함수의 neighbor에 해당하는 부분을 수정하여 실행한 결과는 다음과 같다.

```
    state_list = [
        State(1, 'Western Australia', [2,3]),
        State(2, 'Northern Territory', [3,4]),
        State(3, 'South Australia', [4,5,6]),
        State(4, 'Queensland', [5]),
        State(5, 'New South Wales', [6]),
        State(6, 'Victoria', []),
        State(7, 'Tasmania', []),
    ]
```

```
(opta) leo@leo-GP66-Leopard-11UG:~/opta/mapcoloring$ python main.py
02:30:37.022 [main        ] INFO  Solving started: time spent (44), best score (-6init/0hard/0soft),
 environment mode (REPRODUCIBLE), move thread count (NONE), random (JDK with seed 0).
02:30:37.041 [main        ] INFO  Construction Heuristic phase (0) ended: time spent (64), best scor
e (0hard/0soft), score calculation speed (1266/sec), step total (6).
02:30:46.977 [main        ] INFO  Local Search phase (1) ended: time spent (10000), best score (0har
d/0soft), score calculation speed (204920/sec), step total (151022).
02:30:46.978 [main        ] INFO  Solving ended: time spent (10000), best score (0hard/0soft), score
 calculation speed (203488/sec), phase total (2), environment mode (REPRODUCIBLE), move thread count
 (NONE).
Map(color_list=Color(id=1, color=Red),
Color(id=2, color=Green),
Color(id=3, color=Blue),
state_list=State(id=1, name=Western Austrailia, color=Color(id=1, color=Red)),
State(id=2, name=Northern Territory, color=Color(id=2, color=Green)),
State(id=3, name=South Austrailia, color=Color(id=3, color=Blue)),
State(id=4, name=Queensland, color=Color(id=1, color=Red)),
State(id=5, name=New South Wales, color=Color(id=2, color=Green)),
State(id=6, name=Victoria, color=Color(id=1, color=Red)),
State(id=7, name=Tasmania, color=Color(id=1, color=Red)),
score=0hard/0soft)
```

<Fig. 4 Result>

Fig. 2와 같은 결과를 보여주면서도 solving start과정에서 시간이 소폭 줄어든 것을 확인할 수 있었고 construction heuristic phase에서는 시간이 87ms에서 64ms로 유의미하게 줄어든 것을 확인할 수 있었다. Local search phase는 두 실험 모두 초기에 설정한 10s를 초과하여 종료되었다. 본 문제는 7개의 variable과 3개의 domain의 문제를 푸는 사소한 문제지만 큰 문제를 해결할 때에는 큰 시간의 차이가 존재할 것이다. 이를 통해 문제 정의의 중요성을 깨달을 수 있었다.

사용된 코드 전체는 다음에서 확인할 수 있다. <Github>

Reference.

1. 고민수, Deep Learning Bible - B. Artificial intelligence, wikidocs.net/book/9054
2. Christopher Chianelli, A new AI constraint solver for Python: OptaPy, optaplanner.org/blog