sheet02-programming

May 6, 2024

Exercises for the course Artifical Intelligence Summer Semester 2024

G. Montavon Institute of Computer Science Department of Mathematics and Computer Science Freie Universität Berlin

Exercise Sheet 2 (programming part)

[1]:
```
%matplotlib inline
import utils
import numpy
import itertools
```

0.1 Exercise 3: CSPs and local consistency (10 + 10 P)

In this exercise, we consider the same geographical problem as in the lecture, but with the full network of cities (20 cities). The map depicting the cities and the roads is shown here (from the book by Russell & Norvig):

We would like to find how to navigate the road network by formulating a CSP and reduce it to a locally consistent CSP to make it computationally solvable. We consider the following CSP:

$$X = \{X_1, \dots, X_n\}$$
$$D = \{D_1, \dots, D_n\}$$
$$C = \{C_1, \dots, C_{n+1}\}$$

with X_i the city we visit on the ith day, $D_i = \{\text{Arad}, \text{Bucharest}, \dots, \text{Zerind}\}$ the domain from which the variable X_i take its values, and

$$C_i(X) = \text{road}(X_i, X_{i+1}) \quad \text{for all} \quad i = 1 \dots n-1$$

where $\text{road}(A, B)$ indicates whether there is a road directly connecting cities A and B (or whether cities A and B are the same). The last two constraints C_n and C_{n+1} specify the starting point and target destination, which we define to be Arad and Urziceni, i.e.

$$C_n(X) = (X_1 = \text{Arad})$$
$$C_{n+1}(X) = (X_n = \text{Urziceni})$$

We set the number of days to $n = 7$. Data structures are available in the module `utils`, in particular, `utils.cities` is a vector of strings containing cities names, and `utils.roads` is a boolean matrix indicating whether cities of the given indices are directly connected by a road or are the same.

[2]:
```
cities = utils.cities
roads = utils.roads
days = utils.days
```

For example, if we want to check whether Sibiu and Fagaras are directly connected by a road, we write:

[3]:
```
_sibiu = numpy.argmax(cities=='Sibiu')
_fagaras = numpy.argmax(cities=='Fagaras')
print(_sibiu,_fagaras,roads[_sibiu,_fagaras])
```

```
15 5 True
```

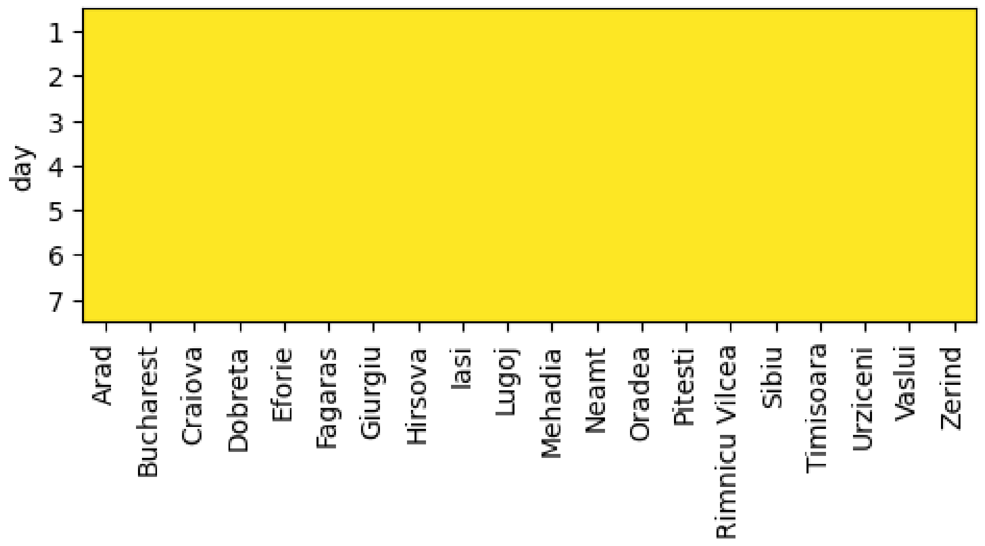
and similarly for Sibiu and Craiova, we write:

[4]:
```
_sibiu = numpy.argmax(cities=='Sibiu')
_craiova = numpy.argmax(cities=='Craiova')
print(_sibiu,_craiova,roads[_sibiu,_craiova])
```

```
15 2 False
```

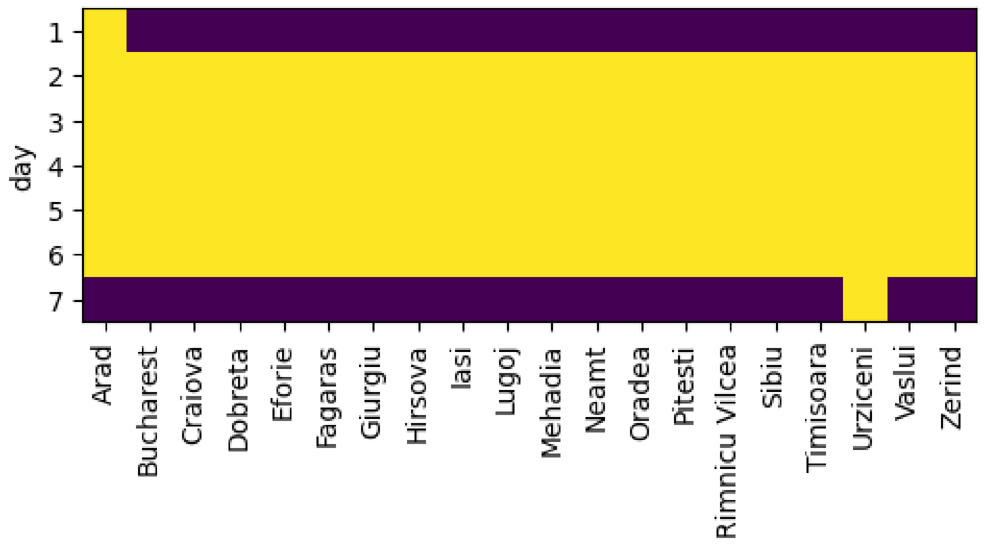
Searching a solution of the CSP above would require enumerating and testing in the worst case 20^7 sequences of cities. This is computationally infeasible. Hence, we would like to reduce the domain of each variable as much as possible before performing an enumeration. We will define a variable `domains` (a boolean matrix of size 7×20) which stores, for each variable and domain value, whether the domain value is still active (`True`) or has been pruned (`False`). Initially, all domain values are active. The function `utils.show` takes the domain matrix as input and renders it graphically, with the yellow color denoting active values and the blue color denoting pruned values.

[5]:
```
domains = numpy.full([days,len(cities)],True)
utils.show(domains)
```



As a first step, we implement node consistency, which consists of removing domain values based on the unary constraints of the CSP. There are two unary constraints, C_n and C_{n+1}, and they act on the domain of the first and last time step respectively. Domain values can be easily pruned based on these constraints by running the following code:

[6]:
```
domains[0] *= (cities == 'Arad')
domains[-1] *= (cities == 'Urziceni')
utils.show(domains)
```



This already results in a reduction of computations by a factor 400. However, further (and much stronger) pruning can be reached by reducing the CSP to an arc-consistent version of it.

Task

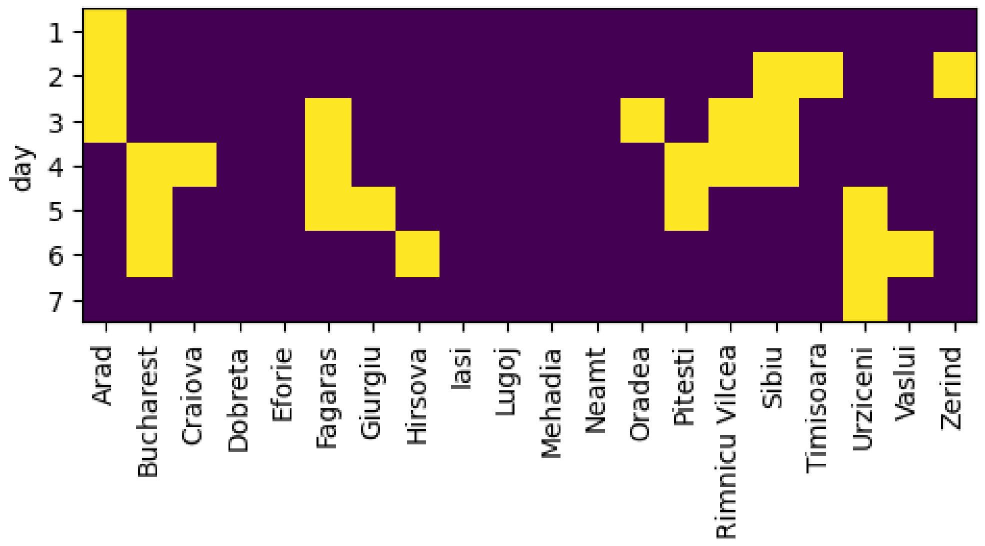
- **Write code that prunes domain values in a way that the CSP becomes arc-consistent. In this special example reaching arc-consistency can be interpreted as pruning cities at a given day i, if the city isn't reachable from the list of remaining cities at day $i-1$ or from the list of remaining cities at day $i+1$. The procedure should select variables iteratively in some meaningful order and terminate once no further domain reduction can be achieved for any variable.**

[21]:
```
domains = numpy.full([days,len(cities)],True)
domains[0] *= (cities == 'Arad')
domains[-1] *= (cities == 'Urziceni')

for i in range(1, days - 1):
    available_cities = cities[domains[i - 1]]
    for city in cities:
        _city = numpy.argmax(cities==city)
        available = False
        for available_city in available_cities:
            _available_city = numpy.argmax(cities==available_city)
            if roads[_city,_available_city]:
                available = True
        domains[i, _city] = available

for i in range(days - 2, 0, -1):
    available_cities = cities[domains[i + 1]]
    for city in cities:
        _city = numpy.argmax(cities==city)
        available = False
        for available_city in available_cities:
            _available_city = numpy.argmax(cities==available_city)
            if roads[_city,_available_city]:
                available = True
        domains[i, _city] = domains[i, _city] and available

utils.show(domains)
```



The domain reductions have now made the list of candidate solutions much shorter, specifically, there remains only $1 \cdot 4 \cdot 5 \cdot 6 \cdot 5 \cdot 4 \cdot 1 = 2400$ candidate solutions. While, not all candidate solutions are actual solutions, the complexity reduction allows us now to perform an exhaustive search as a last step.

Task:

- **Enumerate all 2400 candidate solutions and test for each candidate solution whether all constraints are satisfied. If that is the case, print the solution. (Hint: Enumerations of elements of products of domains can be obtained using the function `product` of the module `itertools`.)**

[20]:
```python
import itertools

eligible_indices = [ [i for i, x in enumerate(day) if x == 1] for day in␣
↪domains]

for solution in itertools.product(*eligible_indices):
    valid = True
    for i in range(len(solution) - 1):
        A, B = solution[i], solution[i + 1]
        if not roads[A, B]:
            valid = False
            break

    if valid:
        print('Valid solution:', [cities[index] for index in solution])
```

```
Valid solution: ['Arad', 'Arad', 'Arad', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
```

```
Valid solution: ['Arad', 'Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni',
'Urziceni']
Valid solution: ['Arad', 'Arad', 'Sibiu', 'Fagaras', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Arad', 'Sibiu', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Arad', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Bucharest',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Bucharest',
'Urziceni', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Fagaras',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Giurgiu',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Pitesti',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni', 'Hirsova',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni',
'Urziceni', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni', 'Vaslui',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Fagaras', 'Bucharest',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Fagaras', 'Bucharest', 'Urziceni',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Fagaras', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Fagaras', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Oradea', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Craiova', 'Pitesti',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest',
'Urziceni', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Pitesti',
'Bucharest', 'Urziceni']
```

```
Valid solution: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Rimnicu Vilcea', 'Pitesti',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Sibiu', 'Fagaras',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Sibiu', 'Fagaras', 'Bucharest', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Sibiu', 'Fagaras', 'Bucharest', 'Urziceni',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Sibiu', 'Fagaras', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti',
'Bucharest', 'Urziceni']
Valid solution: ['Arad', 'Sibiu', 'Sibiu', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Timisoara', 'Arad', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Zerind', 'Arad', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
Valid solution: ['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Fagaras', 'Bucharest',
'Urziceni']
```