Some algorithms for solving graph path finding problems

Kenneth Assogba & Alexis Squarcioni

Python Project

Sorbonne University - 2019-2020

Methodology

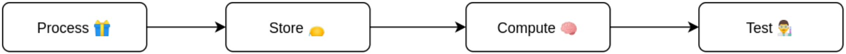


Figure – How we have worked

Plan

1. Process input file
2. Store data
3. Compute
4. Results
5. Conclusion

formatForClean()

```
def formatForClean(filename):
    f = open(filename, 'r')
    data = []
    for line in f.readlines():
        data.append(line)
    data = [line.rstrip("\n") for line in data if line]
    nx, ny, nd, *_ = map(int, data[0].split())

    grilleList = data[1:nx+1]
    grilleList = [[int(d, base=16) for d in str(number)] for number in grilleList]
    grilleList = [["{0:04b}".format(ch) for ch in line] for line in grilleList]
    grilleList = [item for sublist in grilleList for item in sublist]

    robotsLines = data[nx+1:nx+1+nd]
    robots = {}
    for line in robotsLines:
        couleur, x, y, *_ = line.split()
        x = int(x)
        y = int(y)
        position = x*ny+y
        robots[couleur] = position

    f.close()
    return nx, ny, nd, grilleList, robots
```

Cleaning class

```python
class Cleaning(object):
    """
    The Cleaning class defines the main storage point for room to clean.
    Each room has seven fields :
    - **nx** - number of rows of the room
    - **ny** - number of columns of the room
    - **dim** - nxny
    - **grilleList** - list of string who tell walls position
    - **robots** - dict that contain robots colors and their positions
    - **casesPropre** - used to control if the room is clean
    - **graph** - stores the possible deplacements from each box
    """

    def __init__(self, nx, ny, grilleList, robots):
        self.nx = nx  # int
        self.ny = ny  # int
        self.dim = nx*ny  # int
        self.grilleList = grilleList  # list
        self.robots = robots.copy()  # dict
        self.casesPropre = [0]*self.dim  # clean box = 1 else 0
        self.graph = {i: self.voisinCaseList(i) for i in range(self.dim)}
```

Robot and cleaned cases

```
# Robots data
    robotsLines = data[nx+1:nx+1+nd]
    robots = {}
    for line in robotsLines:
        couleur, x, y, *_ = line.split()
        x = int(x)
        y = int(y)
        position = x*ny+y
        robots[couleur] = position
# robots = {'B': 0, 'R': 12}
```

Figure – robots

Robot's position is represent as dict having for key : robotColor and for value : robotPosition.

Robot and cleaned cases

```
self.casesPropre = [0]*self.dim  # clean box = 1 else 0
for key in robots:  # a method exist for that
            position = robots[key]
            self.casesPropre[position] = 1  # Robots clean their starting boxes
# casesPropre = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
```

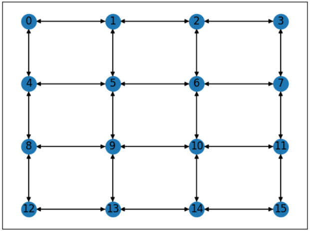
Figure – cleanedCases

Cleaned cases is represent by a list. Cases are numbered from 0 with row-major order when value 1 is for cleaned case and else 0 :

Graph -> without walls without robot

```python
# we generate a dict with the keys from 0 to dim-1
# and we simply connect the related boxes
self.graph = {i: self.voisinCaseList(i) for i in range(self.dim)}
```

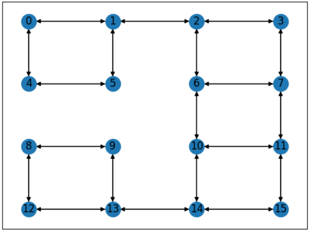
```python
graph = {0: [-1, 4, 1, -1], 1: [0, 5, 2, -1],
         2: [1, 6, 3, -1], 3: [2, 7, -1, -1],
         4: [-1, 8, 5, 0], 5: [4, 9, 6, 1],
         6: [5, 10, 7, 2], 7: [6, 11, -1, 3],
         8: [-1, 12, 9, 4], 9: [8, 13, 10, 5],
         10: [9, 14, 11, 6], 11: [10, 15, -1, 7],
         12: [-1, -1, 13, 8], 13: [12, -1, 14, 9],
         14: [13, -1, 15, 10], 15: [14, -1, -1, 11]}
```



Graph -> with walls and without robot

```python
# we place the internal walls
for i, murs in enumerate(grilleList):
    for j, bin in enumerate(murs):
        if bin == '1':
            self.graph[i][j] = -1
```

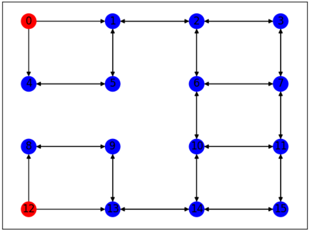
```python
graph = {0: [-1, 4, 1, -1], 1: [0, 5, 2, -1],
         2: [1, 6, 3, -1], 3: [2, 7, -1, -1],
         4: [-1, -1, 5, 0], 5: [4, -1, -1, 1],
         6: [-1, 10, 7, 2], 7: [6, 11, -1, 3],
         8: [-1, 12, 9, -1], 9: [8, 13, -1, -1],
         10: [-1, 14, 11, 6], 11: [10, 15, -1, 7],
         12: [-1, -1, 13, 8], 13: [12, -1, 14, 9],
         14: [13, -1, 15, 10], 15: [14, -1, -1, 11]}
```



Graph -> with walls and robots

```python
# we place the robots
for key in robots:
    position = robots[key]
    caseVoisinRobot = self.voisinCaseList(position)
    for idc, case in enumerate(caseVoisinRobot):
        if case != -1:
            self.graph[case][(idc+2) % 4] = -1
```

```python
graph = {0: [-1, 4, 1, -1], 1: [-1, 5, 2, -1],
         2: [1, 6, 3, -1], 3: [2, 7, -1, -1],
         4: [-1, -1, 5, -1], 5: [4, -1, -1, 1],
         6: [-1, 10, 7, 2], 7: [6, 11, -1, 3],
         8: [-1, -1, 9, -1], 9: [8, 13, -1, -1],
         10: [-1, 14, 11, 6], 11: [10, 15, -1, 7],
         12: [-1, -1, 13, 8], 13: [-1, -1, 14, 9],
         14: [13, -1, 15, 10], 15: [14, 11]}
```



Epsilon-greedy

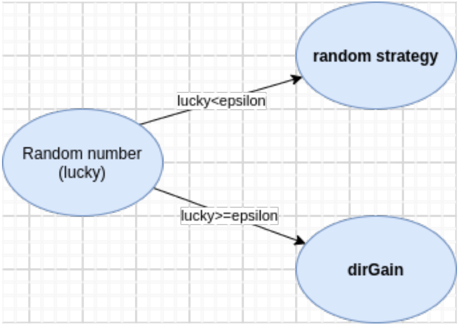
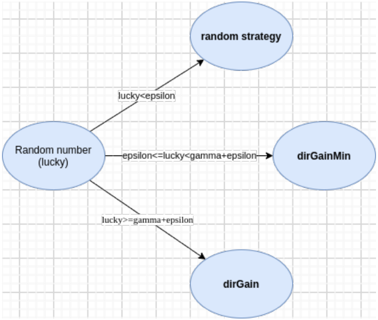


Figure – Epsilon-greedy method

Epsilon-Gamma-greedy




```
if lucky < greed1:                                # ->random
    navigation = random.choice(list(directionPossible.keys()))
elif lucky >= greed1 and lucky < greed2:          # ->dirGainMin
    navigation = self.dirGainMin(positionRobotJoueur, directionPossible)
else:                                             # ->dirGain
    navigation = self.dirGain(positionRobotJoueur, directionPossible)
```

Results

	nx*ny*nd	IW	Depl	iter	ϵ	γ	Time(s)
0	4*3*2	0	6	200	0.1	0	0.02
1	4*4*2	0	6	200	0.1	0	0.02
2	4*4*2	4	10	10000	0.1	0	1.71
3	6*6*3	8	16	200000	0.15	0.2	67.81
4	6*6*2	6	**12***	60000	0.15	0.8	14.77
5	6*6*2	6	12	20000	0.1	0.1	4.97
6	6*7*2	6	**14***	200000	0.2	0.1	64.42
7	6*7*2	6	14	40000	0.15	0	11.87

Perspective

One more thing

- **All units tests write and passed**
- fig.py to have smooth visualisation of our graphs

Perspective

- Judicious choice of the robot to move
- Improve *setParameters* to obtain parameters based on the grid's data
- implementing an ant colony algorithm

Conclusion

Conclusion

- Strong algorithm with good compute time
- Used of different Python data structures and packages
- Clean code with good documentation and readme
- Code optimisation (use of cProfile per example)