**defining ant class:**

cost: eta

pheromone: 1/tau

we just use Edge Cost & Edge Pheromone when we want to calculate probability of choosing next edge in choose_next_edge() func. so I set self.reachableEdge a n^2 dict like this Instead of calculating (1/costs[i][j]^bate) * (pheromone[i][j]^ alpha ) each time.

In [1]:

```
class Ant:
    def __init__(self):
        self.visited = []
        self.cost = 0                                                    # ant with the minimum cost has the maximum fitness
        self.reachableEdge = {(i,j): pow(1/costs[i][j],bate)*pow(pheromone[i][j], alpha) for i in range(n)for j in range(n)}
```

**import all we need:**

In [2]:

```
import time
import numpy as np
import random
```

**read the file and create a cost matrix**

**assign a random number to all edges of pheromone matrix**

**define evaporation, alpha, beta (parameters)**

**large evaporation: divergence and small evaporation: local optimal (it's not really evaporation. it's 1-evaporation)**

In [3]:

```
n = 4
numOfAnts = 5
optimalAnswer = 26
costs = np.loadtxt('job1.assign', usecols=range(n))
firstPheromone = random.random()
print("Pheromone of each edge is ", firstPheromone, "fist")
pheromone = np.array([[firstPheromone for _ in range(n)]for _ in range(n)])
evaporation = 0.5
alpha = 1
bate = 1
bestAnt = Ant()
bestAnt.cost = np.inf
```

```
Pheromone of earch edge is  0.1893536942243479 fist
```

**Division by sigma[(tau^alpha)*(ate^bate)] doesn't have any effect on choice. So I didn't do it.**

**if the ant visits [i,j], it can't visit [z,j] and [i,z] for all 0<z<n ( mentioned in prelude) so i poped them.**

**after each step, the cost get updated**

In [4]:

```
def move_ant(ant = Ant()):
    for _ in range(n):
        choice = (random.choices(population=list(ant.reachableEdge), weights=ant.reachableEdge.values() ,k=1))[0] #cum_weights is x2 faster than cum_weights! but it doesn't work is the true way
        ant.visited.append(choice)
        for i in range(n):
            ant.reachableEdge.pop((i, choice[1]), -1)
        for j in range(n):
            ant.reachableEdge.pop((choice[0], j), -1)
        ant.cost +=costs[choice[0], choice[1]]
```

In [5]:

```
def evaporate_SAC():
    for i in range(n):
        for j in range(n):
            pheromone[i][j] *= evaporation
def update_pheromone():
    for ant in ants:
        for visitedEdge in ant.visited:
            pheromone[visitedEdge[0]][visitedEdge[1]] += (1/ant.cost)
```

**let's check if it works fine:**

In [6]:

```
ants = np.array([Ant() for _ in range(numOfAnts)])
print('pheromone:',pheromone)
print('costs',costs)
print('visited edge after creating ant:',ants[0].visited)
print('reachable edges after creating ant:',ants[0].reachableEdge)
```

```
pheromone: [[0.18935369 0.18935369 0.18935369 0.18935369]
 [0.18935369 0.18935369 0.18935369 0.18935369]
```

```
 [0.18935369 0.18935369 0.18935369 0.18935369]
 [0.18935369 0.18935369 0.18935369 0.18935369]]
costs [[  5.  10.   3.   4.]
 [  8.   7.  49.  19.]
 [  6.  10.   9.  28.]
 [  6.  10.  39. 289.]]
visited edge after creating ant: []
reachable edges after creating ant: {(0, 0): 0.03787073884486958, (0, 1): 0.01893536942243479, (0, 2): 0.06311789807478263, (0, 3)
: 0.04733842355608697, (1, 0): 0.023669211778043486, (1, 1): 0.02705052774633541, (1, 2): 0.003864361106619344, (1, 3): 0.00996598
3906544625, (2, 0): 0.031558949037391315, (2, 1): 0.01893536942243479, (2, 2): 0.021039299358260877, (2, 3): 0.006762631936583853,
(3, 0): 0.031558949037391315, (3, 1): 0.01893536942243479, (3, 2): 0.004855222928829433, (3, 3): 0.0006552030942018959}
```

**run ant colony:**

In [7]:

```
i = 0
p = time.perf_counter()
while bestAnt.cost!=optimalAnswer:
    for ant in ants:
        move_ant(ant)
        if bestAnt.cost > ant.cost:
            bestAnt = ant
    print(bestAnt.cost)
    evaporate_SAC()
    update_pheromone()
    ants = np.array([Ant() for _ in range(numOfAnts)])
    i+=1
print('time:',time.perf_counter()-p)
print('itrarion:',i)
```

```
43.0
31.0
31.0
31.0
31.0
31.0
31.0
31.0
31.0
31.0
31.0
26.0
time: 0.005746691000240389
itrarion: 14
```

**show ans:(in each tuple the first element is the task and the second element is the agent)**

In [8]:

```
print(bestAnt.visited)
print('pheromone:',pheromone)
print('visited edge after creating ant:',ants[0].visited)
print('reachable edges after creating ant:',ants[0].reachableEdge)
```

```
[(2, 2), (1, 1), (0, 3), (3, 0)]
pheromone: [[1.57705888e-03 1.81556276e-05 7.46104406e-03 2.41521800e-01]
 [1.44437363e-01 6.83787384e-02 3.46264519e-02 3.13550569e-03]
 [6.60713793e-02 2.20559288e-03 1.77088285e-01 5.21280097e-03]
 [3.84922578e-02 1.79975572e-01 3.14022772e-02 7.07951904e-04]]
visited edge after creating ant: []
reachable edges after creating ant: {(0, 0): 0.00031541177535744045, (0, 1): 1.8155627583768067e-06, (0, 2): 0.002487014685975767,
(0, 3): 0.06038044999859335, (1, 0): 0.01805467031992267, (1, 1): 0.009768391196443793, (1, 2): 0.0007066622843551113, (1, 3): 0.00
01650266151991199, (2, 0): 0.011011896554143426, (2, 1): 0.00022055928823822504, (2, 2): 0.019676476153327262, (2, 3): 0.0001861714
6319583892, (3, 0): 0.0064153762992737555, (3, 1): 0.01799755716715991, (3, 2): 0.0008051865944973983, (3, 3): 2.449660567583059e-0
6}
```

**everything is okay(time and iterations :) )!**

**let's solve job2:**

**I changed the parameters into the best parameters(in README I told how)**

In [9]:

```
n = 100
numOfAnts = 15
optimalAnswer = 350
costs = np.loadtxt('job2.assign', usecols=range(n))
firstPheromone = random.random()
print("Pheromone of each edge is ", firstPheromone, "fist")
pheromone = np.array([[firstPheromone for _ in range(n)]for _ in range(n)])
evaporation = 0.3
alpha = 1
bate = 1
```

```
Pheromone of earch edge is  0.83976222585083 fist
```

In [10]:

```
ants = np.array([Ant() for _ in range(numOfAnts)])
bestAnt = Ant()
bestAnt.cost = np.inf
i = 0
p = time.perf_counter()
while bestAnt.cost>optimalAnswer:
    for ant in ants:
        move_ant(ant)
```

```
        if bestAnt.cost > ant.cost:
            bestAnt = ant
    print(bestAnt.cost)
    evaporate_SAC()
    update_pheromone()
    ants = np.array([Ant() for _ in range(numOfAnts)])
    i+=1
print('time:',time.perf_counter()-p)
print('itrarion:',i)
```

```
1842.0
1842.0
1765.0
1765.0
1765.0
1398.0
1396.0
783.0
568.0
515.0
515.0
485.0
444.0
396.0
396.0
396.0
396.0
396.0
396.0
396.0
396.0
396.0
396.0
305.0
time: 12.134056224000233
itrarion: 24
```

**show ans:(in each tuple the first element is the task and the second element is the agent) reachable edges should be empty:**

In [11]:

```
print('cost:', bestAnt.cost)
print('reachable edges:',bestAnt.reachableEdge)
print(bestAnt.visited)
```

```
cost: 305.0
reachable edges: {}
[(29, 46), (43, 48), (47, 79), (3, 74), (37, 29), (95, 6), (96, 44), (48, 95), (60, 37), (49, 25), (5, 81), (65, 65), (17, 60), (3
1, 77), (50, 32), (91, 58), (19, 26), (56, 41), (16, 10), (22, 31), (90, 55), (40, 14), (70, 63), (10, 21), (84, 12), (28, 92), (7
9, 35), (92, 11), (61, 43), (76, 69), (54, 22), (42, 86), (35, 93), (1, 85), (14, 1), (81, 80), (46, 3), (36, 15), (8, 73), (74, 8
8), (75, 28), (53, 20), (86, 39), (21, 72), (97, 54), (69, 90), (57, 94), (98, 51), (55, 89), (80, 62), (12, 87), (23, 24), (83, 5
0), (2, 61), (88, 66), (24, 5), (51, 82), (27, 4), (6, 49), (72, 53), (44, 91), (15, 84), (30, 9), (89, 16), (33, 98), (78, 33), (6
7, 36), (68, 67), (77, 42), (38, 13), (41, 18), (82, 0), (34, 7), (26, 64), (85, 8), (63, 59), (71, 38), (94, 78), (13, 97), (99,
76), (59, 52), (4, 40), (7, 56), (25, 19), (32, 17), (39, 71), (73, 23), (9, 99), (11, 30), (66, 45), (20, 70), (64, 47), (87, 2),
(58, 75), (62, 34), (52, 68), (93, 57), (45, 27), (0, 83), (18, 96)]
```

**lets solve job3:**

In [12]:

```
n = 200
numOfAnts = 15
optimalAnswer = 750
costs = np.loadtxt('job3.assign', usecols=range(n))
firstPheromone = random.random()
print("Pheromone of each edge is ", firstPheromone, "fist")
pheromone = np.array([[firstPheromone for _ in range(n)]for _ in range(n)])
evaporation = 0.3
alpha = 1
bate = 1
```

```
Pheromone of earch edge is  0.37349667618774196 fist
```

In [13]:

```
ants = np.array([Ant() for _ in range(numOfAnts)])
bestAnt = Ant()
bestAnt.cost = np.inf
i = 0
p = time.perf_counter()
while bestAnt.cost>optimalAnswer:
    for ant in ants:
        move_ant(ant)
        if bestAnt.cost > ant.cost:
            bestAnt = ant
    print(bestAnt.cost)
    evaporate_SAC()
    update_pheromone()
    ants = np.array([Ant() for _ in range(numOfAnts)])
    i+=1
print('time:',time.perf_counter()-p)
print('itrarion:',i)
```

```
6204.0
6204.0
6204.0
6204.0
6204.0
6204.0
6204.0
4214.0
3223.0
```

```
2052.0
1560.0
1194.0
1194.0
1073.0
1073.0
1051.0
993.0
993.0
993.0
993.0
993.0
893.0
893.0
893.0
886.0
886.0
873.0
722.0
time: 97.81328862200007
itrarion: 27
```

In [15]:

```
print('cost:', bestAnt.cost)
print(bestAnt.visited)
```

```
cost: 722.0
[(26, 67), (123, 192), (13, 195), (187, 100), (94, 173), (196, 177), (154, 36), (152, 19), (170, 12), (95, 23), (135, 57), (149, 2
5), (162, 82), (41, 32), (11, 128), (86, 85), (138, 144), (85, 190), (146, 98), (133, 2), (120, 47), (80, 175), (125, 51), (176, 6
), (165, 29), (9, 40), (108, 138), (173, 197), (193, 108), (25, 149), (5, 104), (134, 117), (136, 28), (145, 135), (84, 50), (76,
86), (172, 20), (72, 64), (96, 166), (20, 77), (48, 30), (166, 196), (160, 1), (62, 116), (54, 33), (92, 154), (147, 169), (198, 1
55), (178, 91), (199, 9), (68, 114), (27, 162), (112, 84), (153, 69), (31, 16), (65, 165), (97, 178), (179, 176), (194, 199), (98,
80), (161, 129), (53, 90), (24, 41), (158, 118), (148, 22), (182, 76), (192, 182), (14, 59), (46, 18), (10, 110), (164, 43), (57,
187), (70, 103), (99, 189), (50, 75), (23, 106), (74, 194), (93, 181), (67, 55), (122, 37), (60, 93), (45, 58), (189, 119), (73, 1
64), (37, 142), (104, 136), (81, 46), (28, 150), (64, 62), (144, 191), (100, 14), (56, 13), (88, 65), (163, 15), (3, 78), (19, 42)
, (180, 53), (77, 60), (190, 74), (63, 52), (186, 79), (8, 56), (171, 7), (111, 113), (140, 17), (150, 61), (118, 156), (36, 54),
(78, 88), (114, 120), (183, 167), (184, 185), (38, 94), (131, 143), (156, 141), (151, 134), (21, 45), (66, 152), (185, 21), (116,
172), (197, 112), (40, 159), (51, 157), (177, 131), (39, 170), (79, 4), (121, 97), (29, 87), (117, 133), (12, 158), (167, 147), (1
68, 126), (1, 89), (18, 27), (0, 8), (15, 174), (188, 10), (130, 71), (127, 48), (42, 183), (55, 39), (4, 5), (61, 95), (119, 81),
(35, 161), (181, 123), (89, 107), (142, 68), (191, 96), (139, 145), (126, 24), (22, 140), (58, 111), (107, 63), (44, 34), (115, 0)
, (109, 49), (71, 73), (141, 148), (101, 66), (124, 188), (2, 109), (110, 31), (83, 26), (75, 3), (129, 121), (47, 160), (16, 83),
(90, 168), (174, 153), (157, 139), (69, 44), (34, 179), (143, 180), (195, 163), (7, 92), (175, 70), (33, 127), (43, 115), (155, 14
6), (91, 198), (128, 132), (6, 122), (137, 38), (49, 151), (17, 11), (82, 101), (52, 72), (105, 184), (87, 130), (169, 99), (30, 1
24), (106, 125), (103, 35), (59, 137), (159, 105), (113, 193), (132, 171), (102, 102), (32, 186)]
```