Natural computing

Assignment 3

Stijn Voss, s4150511
Kevin Jacobs, s4134621,
Jaap Buurman, s0828122

March 8, 2016

1 Exercise 1

Consider an illustrative example of a PSO system composed of three particles. Consider the following update rule for each particle i and dimension d:

$$v(i;d) = wv(i;d) + r_1(x^*(i;d) - x(i;d)) + r_2(x^*(d) - x(i;d)).$$

To facilitate calculation, we will ignore the fact that r_1 and r_2 are random numbers and fix them to 0.5 for this exercise. The space of solutions is the two dimensional real valued space R^2 and the current state of the swarm is as follows

- *Position of particles:* $x_1 = (5,5); x_2 = (8,3); x_3 = (6,7);$
- *Individual best positions:* $x_1 = (5,5); x_2 = (7,3); x_3 = (5,6);$
- *Social best position:* $x = (5,5);$
- *Velocities:* $v_1 = (2,2); v_2 = (3,3); v_3 = (4,4).$

(a) What would be the next position of each particle after one iteration of the PSO algorithm with $w = 2$?

(b) And using $w = 0.1$?

(c) Explain what is the effect of the parameter w.

(d) Give an advantage and a disadvantage of a high value of w.

(a) We will first update the velocity of the three particles seperately:

$v_1 = wv_1 + r_1(x_1^* - x_1) + r_2(x^* - x_1) = 2 * (2,2) + 0.5 * ((5,5) - (5,5)) + 0.5 * ((5,5) - (5,5)) = 2 * (2,2) = (4,4)$

$v_2 = wv_2 + r_1(x_2^* - x_2) + r_2(x^* - x_2) = 2*(3,3) + 0.5*((7,3) - (8,3)) + 0.5*((5,5) - (8,3)) = 2*(3,3) + 0.5*(-1,0) + 0.5*(-3,2) = (6,6) + (-0.5,0) + (-1.5,1) = (4,7)$

$v_3 = wv_3 + r_1(x_3^* - x_3) + r_2(x^* - x_3) = 2*(4,4) + 0.5*((5,6) - (6,7)) + 0.5*((5,5) - (6,7)) = 2*(4,4) + 0.5*(-1,-1) + 0.5*(-1,-2) = (8,8) + (-0.5,-0.5) + (-0.5,-1) = (7,6.5)$

We can now use these velocities to update the position of the particles via the following way:

$x_1^(t+1) = x_1 + v_1 = (5,5) + (4,4) = (9,9)$

$x_2^(t+1) = x_2 + v_2 = (8,3) + (4,7) = (12,10)$

$x_3^(t+1) = x_3 + v_3 = (6,7) + (7,6.5) = (13,13.5)$

(b) We will first update the velocity of the three particles seperately again:

$v_1 = wv_1 + r_1(x_1^* - x_1) + r_2(x^* - x_1) = 0.1*(2,2) + 0.5*((5,5) - (5,5)) + 0.5*((5,5) - (5,5)) = 0.1*(2,2) = (0.2,0.2)$

$v_2 = wv_2 + r_1(x_2^* - x_2) + r_2(x^* - x_2) = 0.1*(3,3) + 0.5*((7,3) - (8,3)) + 0.5*((5,5) - (8,3)) = 0.1*(3,3) + 0.5*(-1,0) + 0.5*(-3,2) = (0.3,0.3) + (-0.5,0) + (-1.5,1) = (-1.7,-0.7)$

$v_3 = wv_3 + r_1(x_3^* - x_3) + r_2(x^* - x_3) = 0.1*(4,4) + 0.5*((5,6) - (6,7)) + 0.5*((5,5) - (6,7)) = 0.1*(4,4) + 0.5*(-1,-1) + 0.5*(-1,-2) = (0.4,0.4) + (-0.5,-0.5) + (-0.5,-1) = (-0.6,-1.1)$

We can now use these velocities to update the position of the particles via the following way:

$x_1^(t+1) = x_1 + v_1 = (5,5) + (0.2,0.2) = (5.2,5.2)$

$x_2^(t+1) = x_2 + v_2 = (8,3) + (-1.7,-0.7) = (6.3,2.3)$

$x_3^(t+1) = x_3 + v_3 = (6,7) + (-0.6,-1.1) = (5.4,5.9)$

(c) The parameter w determines how much the old velocity dictates the new velocity of each particle.

(d) High values of w will increase the velocity of the particles and thus will make them explore the space faster. This turns the algorithm in a global search. Smaller values in contrast will constrain the velocities and thus will make the search more local.

While a global search is important to find the global optimum rather than a local optimum, with too high values of w will prevent the algorithm from converging.

2 Exercise 2

Code an ACO to solve Sudoku‘ s. First, you need to think how to represent Sudoku: there are three conditions an optimal solution must fulfill, each line must contain all integers {1, 2, 3, 4, 5, 6, 7, 8, 9} once and only once, the same applies to columns and 3x3 subgrids. You can fix one of them so that you will prevent violations. The ACO will have 9x9x9 pheromone matrix, where you update pheromones for the values appearing in the best solutions inversely proportionally to the fitness value. You have to consider also how the old pheromones will fade. Our python implementation can be found in the appendix.

2.1 Implementation

Pheromone
Since the exercise description talks about a 9x9x9 pheromone matrix we assume that every ant decides which digit it will choose purely based on the place of the digit it is selecting and the pheromone for that position, but not it previous decisions. In our case we use an 81x9 matrix where each digit is numbered by $9 * row + col$. We also use a single list of 81 digits to represent the sudoku puzzle when generating the solution. We reshape it to a 9x9 matrix when we are calculating the costs.

Cost
Our cost function is based on the number of duplicate digits it found in the rows, columns and sub-grids. For a sequence of 9 digits we sum the frequency of all the different digits in and substract the number of different digits from it. (thus if all digits only occur once, the total will be zero). We then sum these values for all possible rows, cols and 3x3 sub grids. We than take the square root(to make sure the cost differences for improvements are not to small) and add 1(to make sure the cost is always ≥ 1)

Another way of doing it(and perhaps better) would be to only consider the digits the ant could pick while satisfying the sudoku constraint. The cost would than be the number of digits it still had to fill when it had no options left. The pheromone would than be based on the numbers it had selected before the one it is selecting now

3 Exercise 3

Consider the benchmark Sudoku problems are available at `http://lipas.uwasa.fi/timan/sudoku/`*. Run your ACO on the following benchmark instances s10a.txt, s10b.txt,*

s10c.txt, s11a.txt, s11b.txt, s11c.txt. Report and discuss the results. We executed our ACO-implementation for each file 10 times for 5000 iterations each. We show the minimum cost, average cost and maximum cost of the 10 executions found in figure 1. We

	average	**min**	**max**
r10a.txt	7.65	7.35	7.87
r10b.txt	7.71	7.61	7.81
r10c.txt	7.64	7.62	7.75
r11a.txt	7.71	7.61	7.87
r11b.txt	7.85	7.68	7.94
r11c.txt	7.81	7.61	7.87

Figure 1: Benchmark of ACO sudoku algorithm

initiated our pheromone matrix equally($\frac{1}{9}$for each digit), we use 10 ants at a time and $\alpha = .5$ and $\beta = 1$ and our pheromone evaporation coefficient is set to 0.025. Hyperparameters are optimized a bit by selecting different settings by hand on the s10a.txt problem.

4 Appendix

This file is also provided as python file as attachement.

```
from collections import Counter
import numpy as np
import sys
import os
ALPHA = .5
BETA = 1
N_ANTS = 10
P = .025
Q = 1.0
ITERATIONS = 5000
def calc_duplicates(r):
    # for every i in r sum(#i - 1) if #i > 0
    c = Counter(r)
    return sum(c.values()) - len(c.values())

def calc_cost(solution):
    shape = solution.shape
    # sum duplicates in rows and cols
    s_rows = sum([calc_duplicates(r) for r in solution])
    s_cols = sum([calc_duplicates(solution[:,i]) for i in range(0, shape[1])])
```

```
    # make Sodoku 3x3 sub-grid by splitting first vertically and then horizon
    grid = np.vsplit(solution,3)
    grid = np.array([np.hsplit(s,3) for s in grid]).reshape(9, 3, 3)

    # now the array exists of the 9x 3x3 arrays
    s_grid = sum([calc_duplicates(g.flatten()) for g in grid])

    return (s_rows + s_cols + s_grid + 1) ** .5 # sum by one to prevent divisi

def calc_costs(solutions):
    return [calc_cost(s) for s in solutions]

def generate_solution(path,index,p):
    # stop recursion when complete solution is found
    if index == 81:
        return np.array(path).reshape(9,9)

    # replace only zeros, zeros are used to represent non filled digits
    if path[index] == 0:
        # un normalized probabilities for each of the 9 possible choices
        probs = np.multiply(np.power(p[index], ALPHA), np.power(float(1)/float
        # normalized:
        probs = np.divide(probs, np.sum(probs))
        # choose one according to the found probability distribution
        path[index] = np.random.choice(range(1,10), 1, p=probs)
    return generate_solution(path,index+1,p)


def generate_solutions(p, sudoku):
    return [generate_solution(sudoku, 0, p) for i in range(0, N_ANTS)]


def update_pheromone(p,s,l):
    d = np.zeros((81,9),np.float)
    for k in range(0,len(l)):
        lk = l[k]
        sk = s[k]
        for pos,digit in enumerate(sk.flatten()):
            d[pos,digit-1] = Q/lk

    return np.multiply(p,1-P) + d


def ACO(sudoku):
```

```
    p = np.multiply(np.ones((81, 9), np.float), float(1)/float(9))
    L = 1000000
    i = ITERATIONS
    while L > 1 and i > 0:
        # make copy since digits are replaced
        path = np.copy(sudoku)
        s = generate_solutions(p, path)
        l = calc_costs(s)
        p = update_pheromone(p, s, l)

        l.append(L)
        L = min(l)
        i -= 1

    return L

def main(file):
    # init pheromone equally
    sudoku = np.loadtxt(os.getcwd() + '/' + file).reshape(81,1)
    values = []
    for i in range(0,10):
        values.append(ACO(sudoku))

    print min(values),np.average(values),max(values)




if __name__ == '__main__':
    a = sys.argv
    if len(a) > 1:
        main(a[1])
    else:
        print("No soduku puzzle given")
```