Biologically inspired models: Coursework 2

1762355

King's College London

1 Problem of your choice

1.1 Setting of the problem

In this coursework I consider the problem of exchanging gifts in a social network during a major festivity, *e.g* Easter eggs. The aim of the user "A" is to make gifts to all of their friends by recycling as many gifts as possible. In other words, "A" wants to minimise the number of gifts to be bought. In the following I will refer to this variable as cost. If A's friends do not know each other, then the cost is always trivially 1. In fact "A" needs to buy only the present for the first friend, then A will give the gifts received from the the first friend to the second friend, and so on.

To make the problem more realistic, I consider a constrain arising from social relationships. Assume that person 'A' exchange a gift with friend '1', then 'A' cannot give a present to friend '2' if friend '2' knows friend '1'. In fact it would be socially embarrassing if the recycled gift is recognised by the the first owner in common friend's hands. 'A' would definitely prefer to buy a new gift instead.

To sum up, given the network of friend relationship, the problem is to find the optimal path taken by 'A' through all of their friends, with the aim to reduce the number of gifts needed. On the contrary, the topology of friendships reduces the opportunity to recycle a gift. Furthermore, in some situations, 'A' may have several choices of gifts to give, so I define a decision variable that dictates the choice.

The problem resembles the salesman travel, with a penalty coming from the network topology rather than the physical distance. However, this is not a pure permutation problem. Let N be the number of friends 'A' visits, the cost of a path depends on the order of visiting N nodes and the choice of the gift given. We can take up to N-1 decisions (because when visiting the first friend, 'A' has to buy a gift). Hence the cost function depends on $2N - 1$ variables.

Create social interactions The solution of the problem strongly depends on the topology chosen because it gives a constrain on the optimal solution. The graph of social interactions consist of N nodes, which are $\mathbb{V} = \{1 \dots N\}$. For any node i , the list of k friends of node i is selected randomly from $\mathbb{V}$. In order to make the problem non trivial, I investigate the case where the number of connections, *i.e.* constraints, is high. Hence k is chosen to be $k = round(N * 0.8)$

. Because of non zero chance of taking the node i , the mean connectivity is smaller than 0.8. The code is listed below

```
function neighbours = create_social_interactions(N)
% This function creates the social network of who knows who.
%   Considering the total number of friends as N,
%each of them can have up to n_neighbours friends.

n_neighbours=ones(N,1)*round(N*0.8);
neighbours=cell(N,1);
for i=1:N
    %take randomly n_neighbours as friends
neighbours{i} = randperm(N,n_neighbours);
end
save('neighbours.mat','neighbours');
end
```

The code above is run only once to assign the topology of interaction. The topology used in the coursework is shown in figure 1.

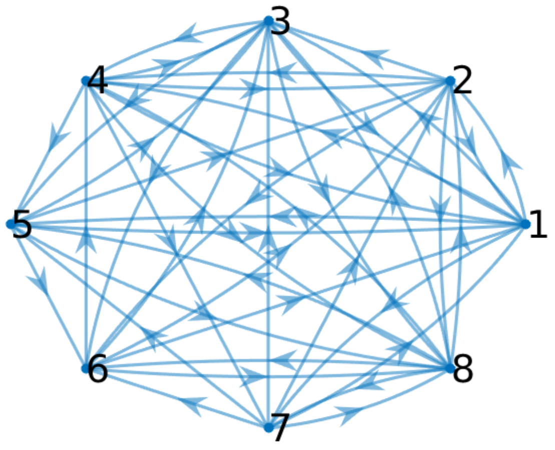
1.2 Cost of making presents

The state of the problem is given by one array of $2N-1$ variables $[s_1, \dots s_{2N-1}]$ where each of the variables s_i can take any real value between $[0.5, N+0.5]$. Let $x = [s_1, \dots s_N]$ be the array containing the first N variables from s. The entries of vector x are converted to integers. In this way they describe the order in which 'A' visits all of their friends. For examples in the case $N = 3$ if $x = \{2, 3, 1\}$ 'A' visits first friend 2, then friend 3, and lastly friend '1'. In each visit 'A' receives the gift from the friends they meet.

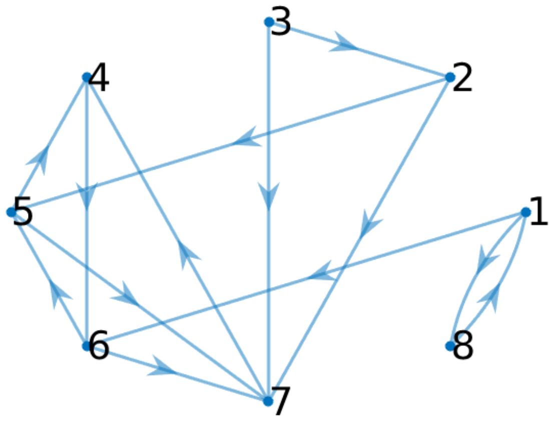
The problem explicitly requires to go through all friends. This constrain is imposed by verifying that the number of unique elements in x is equal to N: if this does not happen I assign the maximum cost, *i.e.* N.

I now illustrate the behaviour in the case $x = \{2, 3, 1\}$.The flowchart is shown in Figure 2. After visiting the first friend, 'A' owns the gift from friend 2. When 'A' meets friend 3, 'A' checks if friend '3' knows friend 2. If '3' and '2' do not know each other, then 'A' exchanges the gift received from friend'2' with friend 3, *i.e.* at the end of transaction, 'A' owns only the gift from friend 3. If '3' and'2' know each other, then 'A' does not gives the gift they received from friend'2'to friend 3. Instead 'A' buys a new gift. Thus the number of gifts bought by 'A' is 2. After the exchange , the list of gifts owned by 'A' is $\{2, 3\}$.

When 'A' visits friend '1' 'A' checks if the gifts that they owned (either '2'or $\{2, 3\}$) comes from people known by friend '1'. If they do, than 'A' buys a new gift, so the cost raises by one and the list of gift owned is either $\{2, 1\}$ or $\{2, 3, 1\}$. If 'A' owns only one gift that can be exchanged with friend '1', the algorithm follows as above. If the list of gifts owned by 'A' before the exchange is $\{2, 3\}$ and friend '1' does not know neither of people $\{2, 3\}$, then 'A' has to choose which gift to give to friend '1'.



(a) Frienship network



(b) Non-frienship network

Fig. 1. (**a**) Directed network of social interactions. The outgoing links from node i connects to the nodes which are known from i . For example the direct link from '7' to '6' indicates that '7' knows '6' but not vice versa. (**b**) Directed network of non-interaction. In our problem this representation is more intuitive. The outgoing links from node i points to the nodes i can exchange gifts with. For example '6' can receive a gift from '5' .

The kind of decisions are taken by using the last $N-1$ variables of the state vector s. These variables are shifted by 0.5 and divided by N , thus they are within the range $r \in [0,1]$. These new quantities can be used as indicator to choose the element to pick. In the example discussed above, if 1' does not know neither of people $\{2,3\}$, '1' has two possible choices. If the decision variable r is greater than 0.5 the gift chosen is 3, otherwise it is 2.

In the general case, the rescaled decision variable r is multiplied by the number of possible choices, *e.g.*'2'in the example above, and by use of the function ceil they provide the index of the gift to choose.

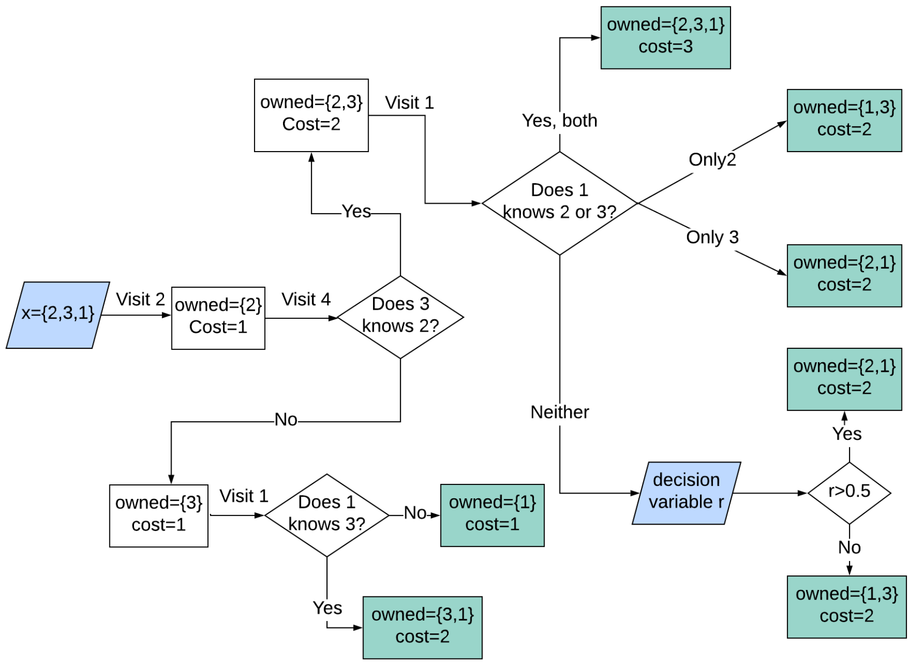


Fig. 2. Flowchart of the example discussed in the text. The input is coloured in blue, while the final output is shown in green.

The implementation of the cost function in Matlab is listed below:

```
function [cost,gift_owned,gifted] =make_presents(state)
    global neighbours
    state=round(state);
    % state is a vectors which contains N+ (N-1) variables
    N=(length(state)+1)/2;
    x=state(1:N); %the path is assigned by  the first N variables of the state
    %x represents the order of visiting people
    if (length(unique(x))<N)% check if the path visit all friends
        cost=N; %   I penalise  if not do so
        return;
    end
    decision=state(N:end)-0.5; % this variable remove ambiguity
    %whenever I can choose among different presents
    decision=decision/N; % I want it normalised between [0,1]
    cost=1; % in the first step I need to buy a presents,
    gift_owned=x(1);% I receive the gift from the first friend
    gifted=zeros(N,1);% this is  not used in the evalutation of the cost.
    %0 means that the gift is bought, otherwise it indicated  who  the gift comes from
    for i=2:N
        indices=find(~ismember(gift_owned,neighbours{x(i)}));
        % gives the indices of the presents that can be gifted
        if  isempty(indices)
        % If  all  presents  owned  would be spotted by the receiver
            cost=cost+1;
            gift_owned(end +1)=x(i);
        else
            %index of the selected gift is:
            indx=indeces(ceil(decision(i-1)*length(indeces)));
            % I pick one gift among the list of possibles.
            % choice is made accordingly  to  decision variables.
            gifted(i)=gift_owned(indx);% this is the gift I  make to x(i)
            gift_owned(indx)=x(i); % I receive the gift
        end
    end
end
```

1.3 Best solution using exhaustive search

The best solution of this problem has minimum cost equal to 2. To prove it, I use the exhaustive search on the permutations of the nodes for a fixes state of decision variables (for instance all of them set to 1).

$$stat = [\text{permutations of (1:8)}, 1, 1, 1, 1, 1, 1, 1]$$

In this setting the minimum of the cost functions is found to be 2. In this particular case the decision variables do not play any role, because whenever there is only one gift available there is no ambiguity on the choice to do. Furthermore, if along the path the cost becomes 2, it cannot decrease.

This consideration is essential because exhaustive search we can deal with the number of permutations of 8 objects, $8! = 40320$, in a reasonable computing time. However, the set of all possible disposition with repetition of 7 variables contains $7^7 = 823543$ different decision variables. Therefore, all possible cases would be $8! * 7^7 \sim 10^{10}$ which are way to much for the actual computing resources.

Analyse the best solution I discuss one of the optimal solutions which is shown in Figure 3. One of the optimal solution consists in visiting the nodes in the order $[5, 6, 4, 7, 2, 3, 1, 8]$. The list of gift is $g = [0, 5, 6, 4, 7, 2, 0, 1]$, where the i entry indicates that node i receives the gift that at the beginning it belonged to $g(i)$ (0 means the gift is new). After buying the first gift to 5, 'A' exchange the gift 5 with 6 since 6 does not know 5. Then 'A' exchanges one gift each time with 4, 7,2,3 . This is equivalent to find a path in the network of Figure 3. So far 'A' has the gift 3, which cannot be exchanged with anybody because '3' is known by everybody. Therefore 'A' has to buy a new gift for visiting 1 and then 8.

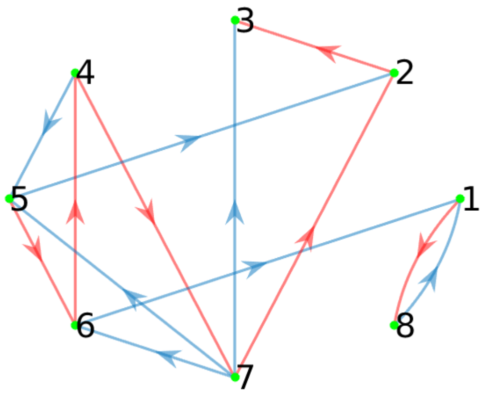


Fig. 3. Direct graph of non-interaction with direction reversed. Arrows point in the opposite directions compared to figure 1(b). Red arrows highlight the path of the optimal solution.

2 Comparison among different algorithms

The cost function has been minimised using four different biologically inspired methods, namely:

1. Binary Genetic Algorithm (BGA),
2. Continuous genetic algorithm,
3. Evolution Strategy (ES),
4. Particle Swarm Optimisation (PSO).

The network used in all computations is shown in figure 1. It consist in $N = 8$ nodes. Therefore, in the case of CGA, ES, and PSO the number of variables is $2N - 1 = 15$. Each variable is selected in the range $[0.5, 8.5]$. The parameters of different algorithms are chosen such a way to keep the time for execution similar $4 - 5$ s.

Binary genetic algorithm The range $[0.5, 8.5]$ is mapped by 4 bits. The precision is smaller than 1, in fact: $(8.5-0.5)/(2^4-1) \approx 0.53$. The population consists of 12 chromosomes, each of them composed of $4*(2N-1) = 60$ genes. The rate of mutation is set to $\mu = 0.05$. In the population of 12 chromosomes, only the best 6 are kept for mating. Parents are chosen using the weighted random pairing . The probability to be selected is determined by the ranking of the cost function. For a given couple, two offspring are generating using the single point crossover. After crossover, the mutation is implemented to all chromosomes but the best (elitism). The fraction μ of genes belonging to the last 11 chromosomes are flipped, *i.e.* $0.05*11*4*15$ =33 chromosomes. The minimisation procedure is interrupted after 5000 iterations.

Continuous genetic algorithm I use the same value of parameters as for the binary genetic algorithm. The population size is set to 12 each of them composed of $(2N - 1) = 15$ genes. Only the best 6 chromosomes are kept for mating. Mates are chosen by using weighted random pairing, as in BGA. Two offsprings are obtained through blending between the parents. One crossover point is chosen for a couple of parents. Calling ma, pa the corresponding genes of the two parents, the new offsprings' genes are obtained as:

$$ma - \beta(ma - pa)$$

$$pa + \beta(ma - pa)$$

where β is a random variable in $[0, 1]$ which is extracted for each couple . Mutations are implemented with elitism, therefore the fraction μ of genes in the worst 11 chromosomes are randomly selected for mutations, i.e. $ceil(0.05*11*15) = 9$ genes. Each of these genes takes a random value that is sampled uniformly in the range [0.5,8.5]. Algorithm is stopped after 5000 iterations.

Evolution strategy I employ the 1+1-ES. Given one individual x, a mutated individual y is obtained as:

$$y = x + \sigma\mathbb{N}(0,1)$$

where σ is set to 1, and $\mathbb{N}(0,1)$ is a standard Gaussian random variable with mean 0 and variance 1. The parameter σ is called in this context strategy parameter, and σ parametrise the size of the searching area. In this implementation σ is kept constant, but different choices are possible. For instance one might adapt σ according to the fraction of improved mutations. Finally, if the cost associated to y is smaller than then the cost of x , the new solution is accepted. The algorithm is run for 30000 iterations.

Particle Swarm optimisation I consider a population of 12 particles, each of them composed by $2N-1 = 15$ elements. The algorithm employs the global best PSO with inertia weight. The total number of iteration is $T = 5000$. The inertia weight is a linear decreasing function on the time: $w(t) = (T-t)/T$. Let $\mathbf{v}(\mathrm{t})$ be the velocity of one particle at time t and let assigned the constants $c_1 = 1$, $c_2 = 3$, $C = 1$. The velocity in the next time step is:

$$\mathbf{v}(t+1) = C\left[w(t)\mathbf{v}(t) + c_1 r_1(\mathbf{y}(t) - \mathbf{x}(t)) + c_2 r_2(\hat{\mathbf{y}}(t) - \mathbf{x}(t))\right]$$

$r_1$and r_2 are random variable extracted in [0,1]; $\mathbf{y}(t)$ and $\hat{\mathbf{y}}(t)$ are the best solutions with respect of the single particle and of the entire swarm respectively. $\mathbf{x}(t)$ is the position of the particle at time t. I impose the limits on the variables by using periodic boundary conditions in 0.5 and 8.5, I use the code:

```
par = mod(par-0.5,N)+0.5;
```

2.1 Comments on results obtained

In the following whenever I refer to an algorithm, *e.g. BGA* , I denote also the choice of parameters associated to it. The following consideration are preliminary since a larger number of iteration might be useful for statistical purposes.

BGA managed to find the best solution 4 times out of 10 repetition, finding the second best solution in the other cases. The algorithm is found to be quite reliable, in fact the standard deviation is small compared to the mean value.

Similar behaviour is observed for CGA, although the best solution is found only 2 times over 10 repetitions. This means that the algorithm is not as reliable as BGA in finding the best solution, in fact it find the second best solution more frequently compared to BGA. On the other hand, CGA seems to depend less on the different repetition, as the lower value of variance shows.

ES manages to find the optimal solution 4 times over 10 iteration, as for BGA. On the other hand, 1 time over 10 iteration the algorithm does not manage to make any improvement from the initial condition, but the cost equates the

Table 1. (Top) Best cost for 10 repetitions of the Binary Genetic Algorithm (BGA), Continuous genetic algorithm, Evolution Strategy (ES), and Particle Swarm Optimisation (PSO). (Bottom) Statistics over the 10 realisations.

Repetition	**BGA**	**CGA**	**ES**	**PSO**
1	3	3	3	2
2	2	3	3	2
3	2	3	2	2
4	3	2	3	2
5	3	3	2	2
6	3	3	8	2
7	3	3	3	2
8	3	3	3	2
9	2	2	2	2
10	2	3	2	2
Statistics				
Best	2	2	2	2
Worstμ	3	3	8	2
Mean σ	2.6	2.8	3.1	2
Standard deviation	0.5	0.4	1.8	0

maximum. Consequently, the algorithm is least reliable among the four, with the worst average performances.

PSO outperforms all the other approaches in terms of reliability. It manages to find the best solution in all attempts.

With regards of sensibility, I have shown in table 2 the average and the standard deviation of the quantity analysed in table 1. In 10 repetitions all the four methods are able to find the global minimum of the cost function. However, these methods do not have the same reliability. In fact the worst solution strongly differs among the different methods, as it observed from the high value of the standard deviation respect to the average $\frac{\textbf{Std.deviation}}{\textbf{average}} = 0.68$. The behaviour of the mean is not sensible with respect to the different methods, since $\frac{\textbf{Std.deviation}}{\textbf{average}} = 0.17$. Finally, σ quantify the reliability of the algorithm, which changes consistently over different algorithm.

Table 2. Average and standard deviation over different methods of **Best Worst Mean St. Dev.** from Table 1.

	Best	**Worst**	**Mean** μ	**St. Dev.** σ
Average	2	4	2.6	0.68
Std.deviation	0	2.7	0.47	0.78
$\frac{\textbf{Std.deviation}}{\textbf{average}}$	0	0.68	0.17	1.15

A Permutation problem

The methods discussed here are not specifically designed to dial with permutation problems. In fact they are not able to find a meaningful solution as $N \sim 15$. In order to explore higher values of N with CGA it is advisable to look at different approaches for mutation and crossover. I implemented a CGA algorithm which deals in a different way the variables indicating the path from the decision variables. The variables indicating the path undergoes to cycling crossover and mutation through reciprocal exchange. In this way the algorithm reduces the space where to search.