Ad Fontes. Researching and Implementing the Bees Algorithm

1. General Information. Formulation of the problem and the proposed approach towards solving it

The Bees Algorithm is a population-based search algorithm which mimics the food foraging behaviour of swarms of honey bees. As developed in 2005, the algorithm performs a kind of neighbourhood search combined with random search and can be used for both combinatorial optimisation and functional optimisation. In the paper which is used as a main inspiration for this project, the emphasis is put on the functional optimisation.

The Bees Algorithm is related to other Bee Inspired Algorithms, such as Bee Colony Optimization (BCO), and other Swarm Intelligence (SI) algorithms such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO). On large sets of data, it has been proven more efficient (faster and more accurate) than the other SI and optimisation algorithms.

Different versions of the Bees Algorithm have been used to solve various engineering problems, such as pattern classifier training, dynamic control problems, machine shop scheduling, robotic swarm coordination (robot control), non-linear model identification and control system tuning. The examples of its other usages include electronic design, digital filter optimisation, data clustering (solving the local optimum for k-means algorithm), and selecting optimal speed parameters for wind turbine generators, as well as for data mining and classification tasks in general. The Bees Algorithm has also been proven faster than backpropagation for the optimisation of neural networks. If talking about the Swarm-based algorithms in general, the U.S. military is investigating swarm techniques for controlling vehicles, whilst NASA is investigating the use of swarm technology for planetary mapping. As for other bees inspired algorithms, the Bee Colony Optimisation (BCO) algorithm, is used for solving the Travelling Salesman Problem, and the Artificial Bee Colony (ABC) algorithm is used for training Nearest Neighbour (NN) classifiers and Spiking Neural Networks (SNNs).

2. The pseudocode description and the algorithm clarification

This being a Swarm-based optimisation algorithm, the Bees Algorithm also mimics nature's methods to drive a search towards the optimal solution. Its main idea is based on the foraging behaviour of honey bees, who collect nectar from flower patches as a food source for the hive. Scout bees, who are sent to look for promising patches, move randomly from one patch to another, until they find the patch that is rated above a certain quality level. Based on the information collected by the scout bees, the other bees are able to evaluate patches according to quality of the food they should provide and the energy necessary to harvest it. If needed, the process is being optimised and more bees are recruited for the task (bearing in mind that the more promising patches require more bees, while the patches with less nectar or pollen should receive fewer of them). Each candidate solution is thought of as a food source, and a population of *n* agents is used to search the solution space. Each time an agent lands on a solution, it evaluates its fitness.

Thus, the bees algorithm has two main elements: an initialisation procedure and a main search cycle (usually "while"), which is iterated until a solution of acceptable fitness is found. In turn, each search cycle consists of local search (everything from the initial selection of sites – patches, to evaluating the fitness for both elite and non-elite sites), and global search (from *n-m* bees being randomly set to other patches for the global search, to evaluating their fitnesses for global search).

After each cycle of the local search procedure, the neighbourhood of the patches is shrunk if the its fitness has stagnated; in case it has stagnated for more than a predefined number, the patch is abandoned, and the scout bee is re-initialised to a new random point in the solution space. If such site

has the best fitness value so far, its position is recorded. In case there is no solution with a higher fitness, the recorded solution is taken as the final one.

When we make the pseudocode given in the paper used for this research more elaborate and clear, we will get the following list of steps to implement:

- initialisation
    - randomly generate initial population of *n* scout bees
    - fitness evaluation of the initial population
    - sort the initial population from maximum to minimum
- the main cycle (if the conditions aren't already met)
    - while stopping criterion is not met:
        - local search:
            - select *e* elite sites for neighbourhood search
            - select the *m-e* non-elite sites for neighbourhood search
            - determine the size of elite and non-elite patches as *ngh*
            - recruit *nep* forager bees for each elite site for neighbourhood search
            - recruit *nep* forager bee for each non-elite site for neighbourhood search
            - evaluate the fitness value for both elite and non-elite sites, select the fittest bee from each patch
        - global search
            - allocate the *n-m* forager bees to the rest of the patches randomly for global search
            - evaluate their fitness value
    - end

Additional remarks:

- The stopping criterion is set by the user. It can be either a solution of fitness above a predefined threshold or a predefined number of cycles (in which case "for" is used instead of "while").
- The only condition for the application of the Bees Algorithm is that some measure of topological distance between the solutions is defined.
- Given a minimisation problem defined over the *n*-dimensional continuous solution space, each candidate solution is represented as an n-dimensional vector of decision variables. The goal of the optimisation task is to find the solution that minimises the set cost function.

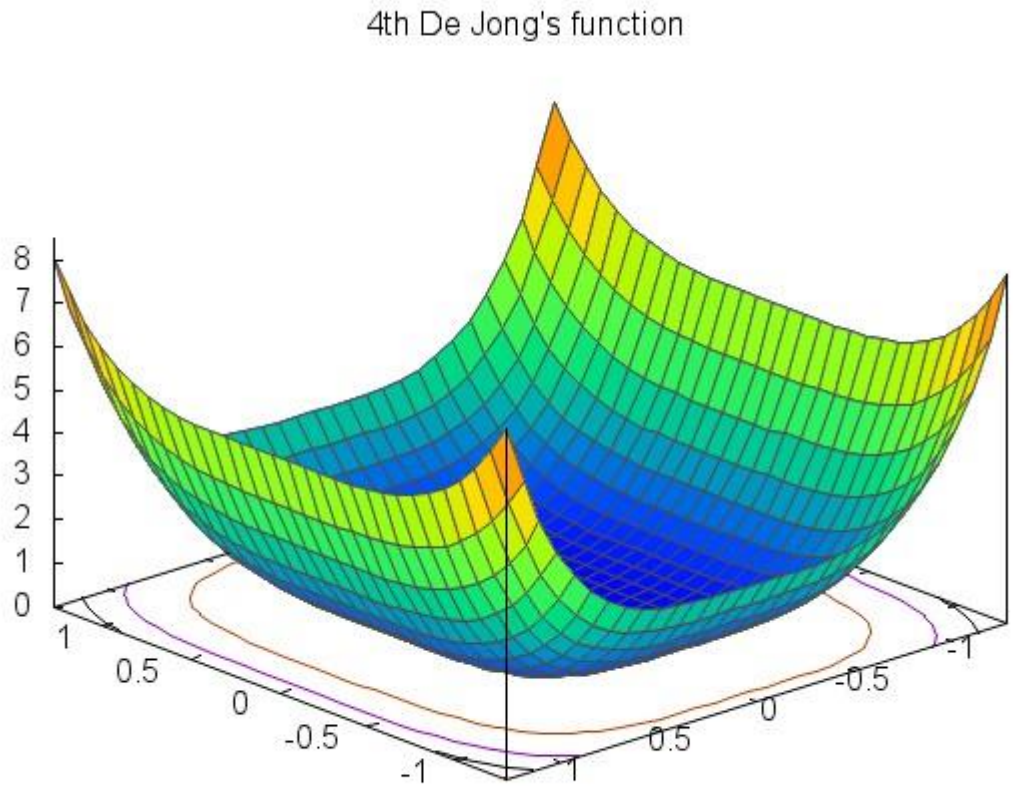
3. The Algorithm Complexity

Seeing as this algorithm does not usually have a static number of iterations, any of the standard big-O notations are not entirely applicable here, and the value is determined by a polynomial and not a function. However, we could put *n* as the number of data objects in the dataset; *m* as the number of attributes; *k* as the number of clusters; *s* as the number of employed bees or food sources; *i* as the number of iterations, and *d* as the total number of categories for all attributes, and then the computational cost of recruitment, local search, neighbourhood shrinking, site abandonment, and global search put together can be expressed as $O(g(n))=O(skmn + i(Tnkm + s(s + nkm + nkd)))$.

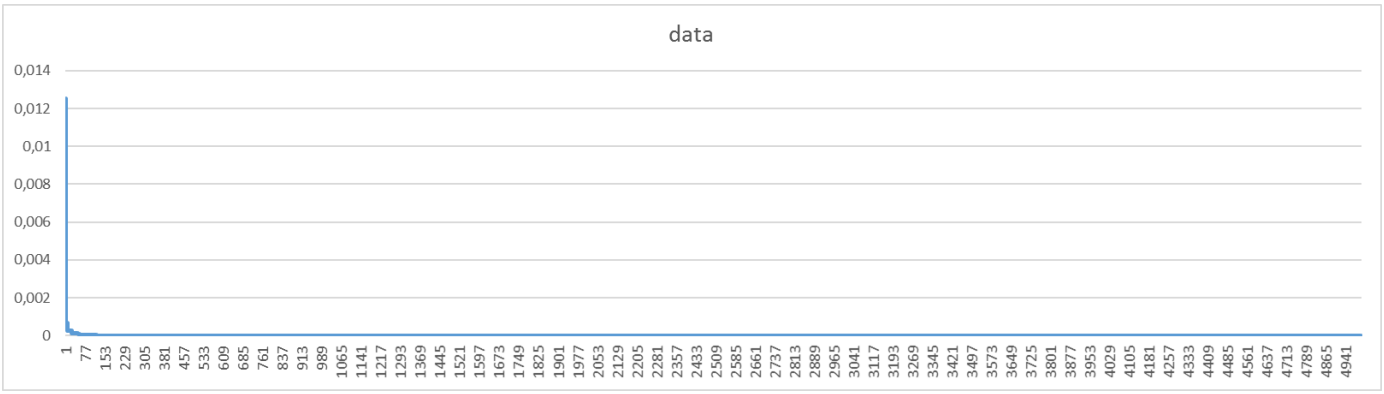
4. Description of the implementation and visualisations

The test function that was used for the implementation was De Jong's function, which is a well-known function used for testing optimisation algorithms.

No	Function Name	Interval	Function	Global Optimum
1	De Jong	[-2.048, 2.048]	$\max F = (3905.93) - 100(x_1^2 - x_2)^2 - (1 - x_1)^2$	X(1,1) F=3905.93



The result of my implementation of the Bees Algorithm, along with several initial values specifying the number of bees, patches, etc., is very close to the optimal result of De Jong's function, that is (1,1), as my results usually go like this [((0.9479396856043346, 0.8988597299244229), 3905.927282429215), ((0.9862268837611704, 0.9794425908036213), 3905.9251874918036), ((0.9081309759931993, 0.8252736936682963), …].



The meaning of the chart above is as follows: displayed is the delta between the optimal value and the result (the closer to 0, the better) during each of the 5000 iterations performed. The best value is reached at the 11th iteration. The convergence to solution is fast, and the result of each iteration is very close to 3905.93 each time. However, the algorithm should be stopped before the 5000th iteration to give a faster and more optimal result.

	variable_name	variable_meaning	variable_value
0	n	number of scout bees	50.0
1	m	number of sites selected out of n visited sites	15.0
2	e	elite patches	3.0
3	nep	elite bees recruited	12.0
4	nsp	non-elite bees recruited	8.0
5	ngh	neighbourhood size	0.1
6	maxiter	number of iterations	5000.0

Figure 1: the values used

Regarding the optimisations/improvements that could be made in the implementation, one idea is to cut the loop as soon as any improvement is found, i.e. change the exit condition.

5. Summary

While working on this task I was able to greatly improve my problem-solving skills, as well as to practise analysing and researching information. To understand the researched algorithm better, I read through various articles both directly related to the algorithm and its usages, and to other similar algorithms. I also read many articles on data visualisation and time complexity for less ordinary problems. Finally, in order to implement the algorithm, I went much further than simply reading the paper provided, and looked for pseudocode examples online. However, as those were largely unhelpful, and I felt that I could come up with a simpler implementation on my own, my code is solely based on the text explanation in the paper.

From a technical point of view, the problem itself presented a considerate amount of challenges. Even being used to implementing various algorithms based on text explanations and/or pseudocode, solving this problem was not easy due to many factors involved. In order to make the program work, I spent several hours debugging it in the programming environment (which has considerably improved my debugging skills), as well as a couple more hours coming up with tests.

From a managerial point of view, this assignment has also taught me a lot. Working on my own was difficult, but also presented a major advantage – being completely in control of this project and knowing its every detail helped me to better understand the task, and so to be able to learn more information and to use a varied set of skills in order to get it done. In short, doing this project myself was its biggest advantage but also, without any doubt, its biggest difficulty.

Having found a lot of information about this algorithm's usage, it still remains unclear how exactly this algorithm manages to be both as efficient and as accurate as it was presented to be on that data it is used, and that is something I would still like to explore whenever I have more time.

Repository with the source code: link.