Nature-Inspired Computing (NIC) Methods in Wind Generator Design Project Documentation

Janne Kemppainen & Eero Järviluoma

Aalto University School of Electrical Engineering

Abstract

This document is the documentation for the project Nature-Inspired Computing (NIC) Methods in Wind Generator Design for the course AS-0.3200 Automaatio- ja systeemitekniikan projektityöt. NIC methods include various different algorithms for efficiently finding near-optimal solutions for many optimizing problems.

I. The Goal

The goal of this project is to implement two nature-inspired algorithms for optimizing wind generator design. The generator design module is provided by VTT and it is handled as a black box. The methods used in this project are Genetic Algorithms (GA) and Particle Swarm Optimization (PSO). We describe the basics of the algorithms, our implementations and the findings.

At the end of this project we should atleast have ready implementations for both of the algorithms for optimization of one objective. Depending on the workload and difficulty of implementation multiobjective optimization is also considered.

The aim for the workload was 4cr which equals approximately 100 hours.

We acknowledge the computational resources provided by Aalto Science-IT project.

II. Structure

III. The Optimization Problem

Our optimization problem is a black box model of a wind turbine generator developed by VTT. The model has 14 inputs and 6 outputs. Two of the inputs are integer values and the rest are floating point numbers. In addition to this there are 3 constraints that should be met for the design.

An ideal generator would produce exactly 3 MW power and have a maximised torque density, minimum weight, high efficiency, high power factor and minimised costs. All the goals can't be optimised simultaneously as they require different parameter combinations. The optimal solution is a compromise where improving some value would deteriorate the others.

IV. Particle Swarm Optimization

Implementing the Particle Swarm Optimization algorithm was assigned to Janne. The algorithm was first proposed by James Kennedy and Russel Eberhart in 1995 and in our project is their version of the algorithm is implemented with some minor modifications.

The algorithm itself is quite simple. The solution space is first initialized with particles set to random locations with zero velocity. Then the fitness function is evaluated for each particle and the best coordinates are stored. Then the velocities are calculated with the following formula: [1]

```
vx[][]=vx[][]+
 2*rand()*(pbestx[][]-presentx[][])+
```

```
2*rand()*(pbestx[][gbest]-presentx[][])
```

The positions and velocities are updated until a set criterion is fulfilled. The velocity function has a constant 2 for the stochastic part so that the particles would have the possibility to move over the previous best value. This value yields the best results as there is no good way to individually guess which one of the velocity increments should be weighted more. [1]

Our optimization problem is a hybrid of discrete and continuous parameters whereas the original algorithm was developed for continuous case only. Therefore we have to make some special arrangements for the two discrete parameters. Pole pair number is an integer between 20 and 80. The first idea was to approximate handle this parameter as continuous and always round the result. The number of slots per pole per pahse has the possible values of 1, 2 and 3 so the idea was to calculate the optimization for each of the three possible parameter values.

In the implementation it was decided, however, that in the eyes of the algorithm itself all the parameters would be continous. The simulation model doesn't allow for non-integer parameters so before each run the corresponding values are rounded to the nearest integer before running the model.

The implementation began with defining the required variables for the swarm and the limitations. The easy part was to fill the solution space with random particles. After implementing the basic algorithm some problems emerged.

The first disturbing observation was that the computer resources weren't used to their full potential as the calculations were performed mostly on one core at a time. After some research the Matlab Parallel Computing Toolbox, and especially the parfor loop, was found. This allowed for the parallelization of the generator simulation and somewhat improved the performance. Not everything is parallelizable though as the particles have to be synchronized.

The next problem was how to enforce the modeling limitations. The initial attempt was to enforce the parameters to stay inside the defined problem space but this is against the nature of the algorithm. As a result the particles would often hit the hard limits and gather near the edges. The decision was made to check the boundaries but skip the calculations and assign a bad fitness value if any of the parameters were out of bounds.

After the change in handling the boundaries it was noticed that the swarm diverges for some reason. In spite of penalizing out of bounds values the particles seemed to wander far from the desired area. This had to be looked into.

After finding the document by Engelbrecht [2] it became evident that the particle velocities exploded. Using a simpler model $f(x, y) = -x^2 - y^2$ yielded similar results, so the next step was to introduce velocity clamping. By changing the fraction of the maximum allowed velocity related to the size of the parameter space it is possible to adjust the particle convergence speed and the tendency of the particles wandering off.

There was also the fundamental mistake that the best particle index was not updated properly so the best value remained to be the one that happened to be the best one on the initialization. This was observed as one of the particles would not move at all.

Running the provided simulation model was inconsistent performance wise. Usually the simulation was calculated in much less than one second but sometimes the calculation could take several seconds, though. This might happen because of nearly singular matrices in the model but the issue is out of our reach.

To manage the large amount of computations in a reasonable time we chose to take advantage of the Aalto University Triton computing cluster. The usage of Triton is described in its own chapter.

V. Genetic Algorithm

Genetic Algorithm was implemented by Eero. The algorithm is modeled from DarwinâĂŹs theory of evolution. A good overview and discussion on the genetic algorithm is presented in [3]. In terms of actual implementation, the Genetic Algorithm is based on 5 different steps [4]:

1. New Population
2. Fitness evaluation
3. Testing whether the solution is satisfactory
4. Creating a new population
5. Looping from step 2.

In step 1, a random new population is generated. Population size may vary, but a too large population will lead to unacceptable waste of calculating time and offers very little in return when compared to smaller population sizes. A good population size is around 20-40 candidates.

In steps 2 and 3, the fitness of each candidate is evaluated. Fitness describes how well the candidate satisfies our criteria.Our fitness evaluation is based on the simulation model provided to us by VTT. If the best candidate passes the pre -set goal in terms of fitness, the looping is stopped and algorithm is complete.

In step 4 a new population is created. First, 10 survivals are selected based on "pie chart roulette", that is, most fit candidates are most likely to survive, but the process is also based on randomness. Next, the survivals produce offspring, and stand a chance to create corssovers with each other. This means that the survivals' offspring inherit attributes from both parents. The best candidate is passed on to the next generation as is. This is called elitism. Finally, a random, low-chance, mutation may occur. In our case this means that a random parameter of a random candidate will be incremented or decremented by a small amount.

The methods used in step 4 and how often they occur are based on empirical evidence. Good chance for crossover is determined to be around 0.7, for example. Mutation chance needs to be low, or else the process loses it's properties and becomes essentially a random-walk process. However, a low mutation chance is required to make the system evolve.

Finally, in step 5 the process is repeated from step 2. The process is repeated maximum in our case maximum of 100 times, to make the calculations run smoothly and in reasonable timeframe.

Currently the program code for the algorithm is complete, but fails to compile, due to errors with the provided simulation model. Further evaluation of the selected methods and parameters, such as crossover and mutation chance, are needed. After the code is complete, the results are analyzed and parameters tuned for best possible results.

VI. Computing in the Cluster

The calculation capabilities of the home computers proved to be inadequate when we decided to run multiple iterations of the algorithms so that we could investigate the consistency of the results. For example running the PSO algorithm with a swarm size of 50 and 150 simulation time steps took several hours to produce 70 results. The performance felt insufficient so we chose to search for alternative methods.

The next attempt was to try to utilize the new Aalto servers Brute and Force. However, their Matlab license allows only for 12 parallel workers and as the server CPU cores run only at a bit over 2 GHz the performance was very similar to the home computer.

Disappointed with the performance of the available resources we came upon the Aalto Triton computing cluster. Running some 400 compute nodes it was even more than we were looking for.

The cluster is running Scientific Linux and the computation tasks are delivered as batch jobs. We had to adapt the Matlab scripts to the requirements of the environment. The algorithms run as functions that among other parameters take the batch ID so that the com-

putation results can be stored to different files. The batch system runs multiple instances of Matlab on different compute nodes simultaneously.

After the calculations the saved .mat files are transferred for further analysis. A script reads the output files one by one and then analyzes the results.

VII. Analyzing the Results

We performed many and more calculations which to analyze. For each algorithm we selected the goal of 10,000 iterations as a baseline. The time allocated for the task was four hours with a grace period of approximately one hour.

VIII. Time and Teamwork Management

As mentioned before we had some challenges allocating time for the project. The deadlines for the different phases have been delayed. Here we summarize the time usage up to date.

References

[1] Kennedy, J. and Eberhart, R. *Particle Swarm Optimization*, Proceedings of IEEE international conference on neural networks. Vol. 4. No. 2. 1995.

[2] Engelbrecht, A. *Particle Swarm Optimization: Pitfalls and Convergence Aspects.* Department of Computer Science, University of Pretoria, South Africa

[3] Mitchell, M. *An Introduction to Genetic Algorithms.* First MIT paperback edition, fifth printing, 1999.

[4] Obitko, M., *Introduction to genetic algorithms*, Hochschule für Technik und Wirtschaft Dresden (FH), `http://www.obitko.com/tutorials/genetic-algorithms/index.php`, accessed 3.3.2014