

# Nature-Inspired Computing (NIC) Methods in Wind Generator Design Intermediate Report

JANNE KEMPPAINEN & EERO JÄRVILUOMA

Aalto University School of Electrical Engineering

## Abstract

*This document is the intermediate report 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. Here we summarize what we have achieved thus far and what are the main problems.*

## I. OVERALL THOUGHTS

**A**FTER the first meeting with the instructor we had a pretty clear vision of the project work. The work was divided between two different algorithms, namely Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), and the difficulty was tunable to fit the work requirements. Originally we planned to implement the Differential Evolution algorithm but we switched to PSO.

During the first few weeks we were preoccupied with the project work for the course System Dynamics and had no time to concentrate on this project. After finishing the other project there was more time available and we started looking deeper into this project.

Then there appeared some more obstacles as we both had decided to apply for exchange and we had to find information, plan studies, fill application forms etc. In addition to this, Eero had to allocate his time to apply for a summerjob as the application deadlines were closing in. Therefore we had to delay our initial deadlines.

## II. 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.

## III. 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 the aim is to implement their version of the algorithm with only some small 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 by 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. For this parameter we can use an approximation of the continuous case and always round the value to the nearest integer. However, the number of slots per pole per phase can only be 1, 2 or 3. Therefore we have to run the optimization three times with each value to find the optimal parameters.

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.

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.

## IV. GENETIC ALGORITHM

Genetic Algorithm was implemented by Eero. The algorithm is modeled from Darwin's 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 crossovers 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 its 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.

## V. 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.

**Janne** Project planning took 8 hours. If the first lectures are counted in, the total time for the first phase is 14 hours. Familiarization with the PSO algorithm took 1 hour and there have been 17.5 hours of programming and troubleshooting. Writing the intermediate report and presentation slides took 7 hours. Total time allocated for the course is 40 hours.

**Eero** Project planning was done in cooperation with Janne and thus took the same 14 hours, including lectures. Familiarization of algorithms took around 3 hours, and programming around 12 hours, bringing the total hours of implementation phase to 15 hours. Finally, 3 hours were spent on writing this document and presentation slides. Total time spent is thus 32 hours.

## VI. RISK MANAGEMENT

Here we discuss the risks that have realized. As mentioned in the project plan the System Dynamics project work required quite a lot of effort and prevented this project from advancing. Then there have been many other things distracting us from this particular project.

There have been no illnesses that would have hindered the project.

At this time, we have opted out of the multiobjective optimization as it would require a much more advanced approach to the optimization.

There has already been some need to rewrite program code as things haven't always worked out the way they were intended to.

The diverging of the PSO algorithm caused some headache until velocity clamping was introduced to the system.

Git hasn't caused any major problems yet.

#### 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