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. 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 3 MW of 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.

For the constraints τ_u should be at least 7 mm, the output power should be greater than the target power and the temperature of the permanent magnets should stay below 100 °C. If any one of these constraints is violated the generator design should be considered invalid.

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 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.

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. Creating a new population
4. 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 step 2, 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.

In step 3 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 3 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.

The simulation results show that the used genetic algorithm managed to find a reasonable good result in practical timeframe. However, the algorithm suffered from 2 drawbacks. Firstly, the algorithm was quite slow when compared to the PSO algorithm. This is probably due to selection process, which requires multiple for-loops at each iteration. Additionally, parallel for loops were not used in the GA code, so the performance could probably have been improved without influencing the overall functionality of the algorithm. The cluster computations were performed with a single core assigned to each task so the performance is comparable in that regard, though. All in all, with the given simulation model, the PSO was almost 10 times faster than GA.

The second drawback with GA was that the solutions weren't quite as good as the ones found by PSO. We are quite sure this had to do with the roulette-wheel selection used in GA. If the fitness values of 2 diffirent candidates are very close to each other, say 0.978 and 0.977, the diffirence in fitness is negligible and the randomness of the "roulette-wheel" takes over. What happens is the solutions become basicly a random-walk process as long as the fitness values don't significantly differ from each other. Therefore the algorithm with this selection method is poorly suited for high end optimization and fine tuning. We could have tested diffirent selection methods, but didn't have time to implement them in the scope of this project work.

V. 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 in constant heavy use by other researchers so naturally we couldn't use all the possible resources although there was no need for that either.

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

We mainly used the 4 hour limit for our batch jobs as it was the maximum time setting for the task to be assigned to the short partition of the cluster. In the short partition we had a better chance of getting the recources quickly when compared to the main partition. Also the 4 hour limit with a one hour grace period was enough for most runs of the PSO algorithm with 10 000 iterations.

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.

VI. 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. In the analysis we only used the base versions of the algorithms.

As noted previously, we only considered the single goal optimization problem of the efficiency. A more practical result could be achieved if the optimization goal was altered in a way that the other generator properties are taken into account. Initially, we tried this approach by setting certain weights to the parameters but the results didn't have any real meaning and they weren't very easy to interpret.

The final results of the simulations are shown in Appendix A. The results show that the 2 algorithms find their solutions in roughly the same areas. Due to GA's mutation element, the results found by GA are much more spread out when compared to PSO. As mentioned earlier, the GA turns basicly into a random walk process with parameters that don't have a major impact on the fitness. This is especially obvious with parameters 13 and 14, which seem to have very little impact on the generator efficiency.

A better way of doing this multiple goal optimization could be setting some restrictions to the desired properties. As was seen in the comparison of the best generators (Appendix A, Table 1) the one that we got from the PSO algorithm was almost twice the price of the GA version. We could simultaneously optimize the generator efficiency and the power factor by simply taking the sum of these properties. For the other parameters we could set some desired conditions that should be fulfilled and give some penalty if the limits are exceeded. For example, we could set the maximum allowed cost to be 7M and the minimum torque density to 11. We did not have the time to implement this.

VII. Conclusions and Further Research

Here in our project the PSO algorithm outperformed the genetic algorithm in speed even with a larger swarm than the GA population. Also the achieved fitness value was better. This doesn't necessarily mean that the algorithm itself is any worse but perhaps the scripts weren't equally optimized perfomance and parameter wise.

The next step would be to try to improve the performance of the genetic algorithm by trying to get rid of as many for-loops as possible. Where possible, the PSO implementation already makes use of the Matlab matrix operations which are proved to be more efficcient when compared to the for-loop. After this a better optimization goal should be formed such that every goal would be optimized in a way. A more advanced approach would be to try to achieve Pareto optimality.

VIII. Time and Teamwork Management

The initial project plan had quite amibitous goals for the time management. Due to many reasons they were delayed to the official course deadlines. For example in March and April the project was on halt for almost 4 weeks because of other more urgent assignments.

The workload wasn't evenly distributed because Janne handled the cluster computing in addition to his own algorithm design. Also the available data had to be analyzed so that the feasibility of the results could be verified. This analysis later became the core of the achieved results. Otherwise there were no problems distributing the work as the algorithms required approximately the same amount of work. We could have spent much more time improving both algorithms, but had to keep the time spent on reasonable levels, keeping the scope of the project in mind.

The overall time usage for Janne was 107.5 hours and 77.5 hours for Eero. These are conservative estimates, as not every task was included in the calculations and some tasks probably took a bit longer than was reported. The time usage can be seen in more detail in Appendix B.

References

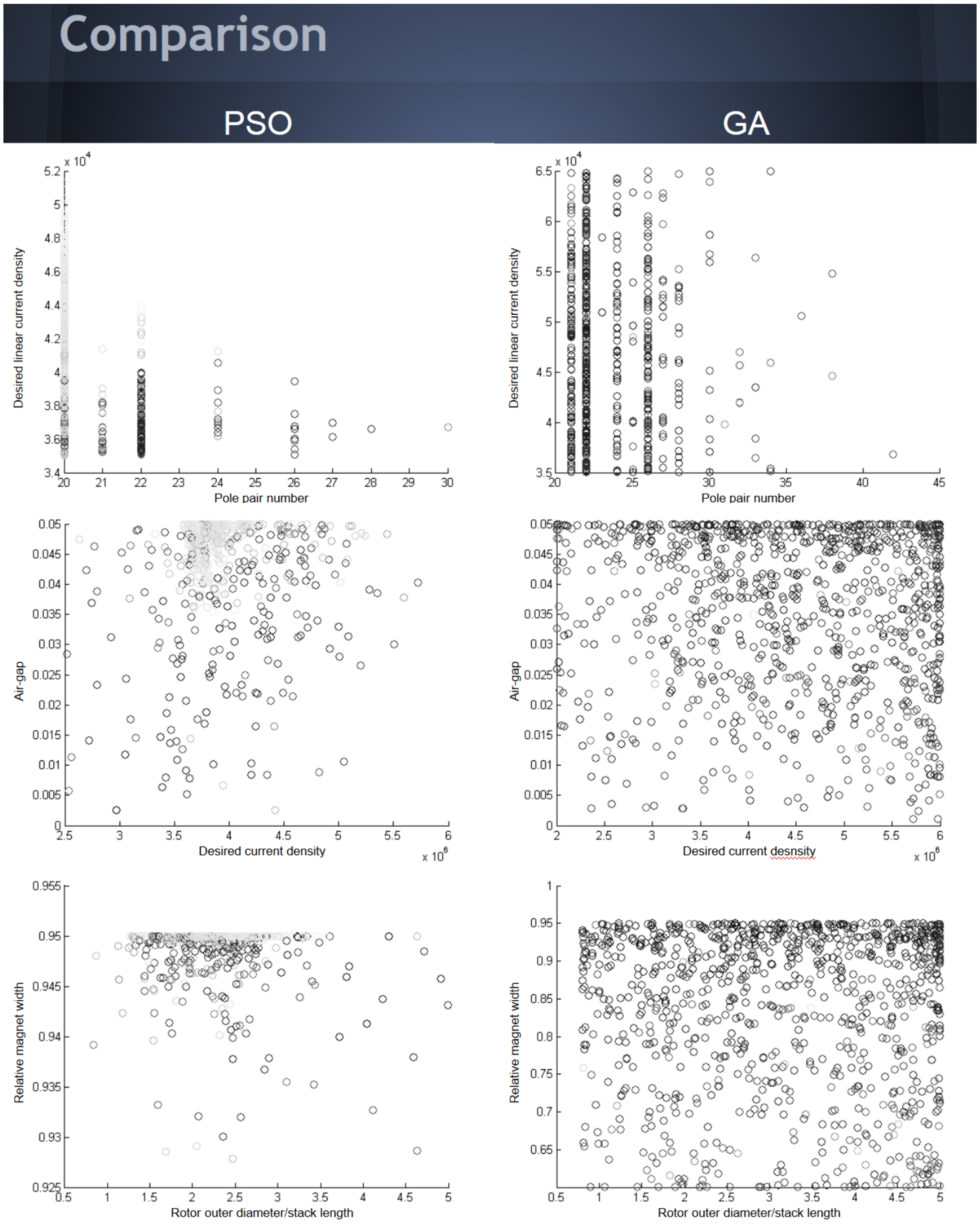
[1] Kennedy, J. and Eberhart, R. *Particle Swarm Optimization*, Proceedings of IEEE international conference on neural networks. Vol. 4. No. 2. 1995.

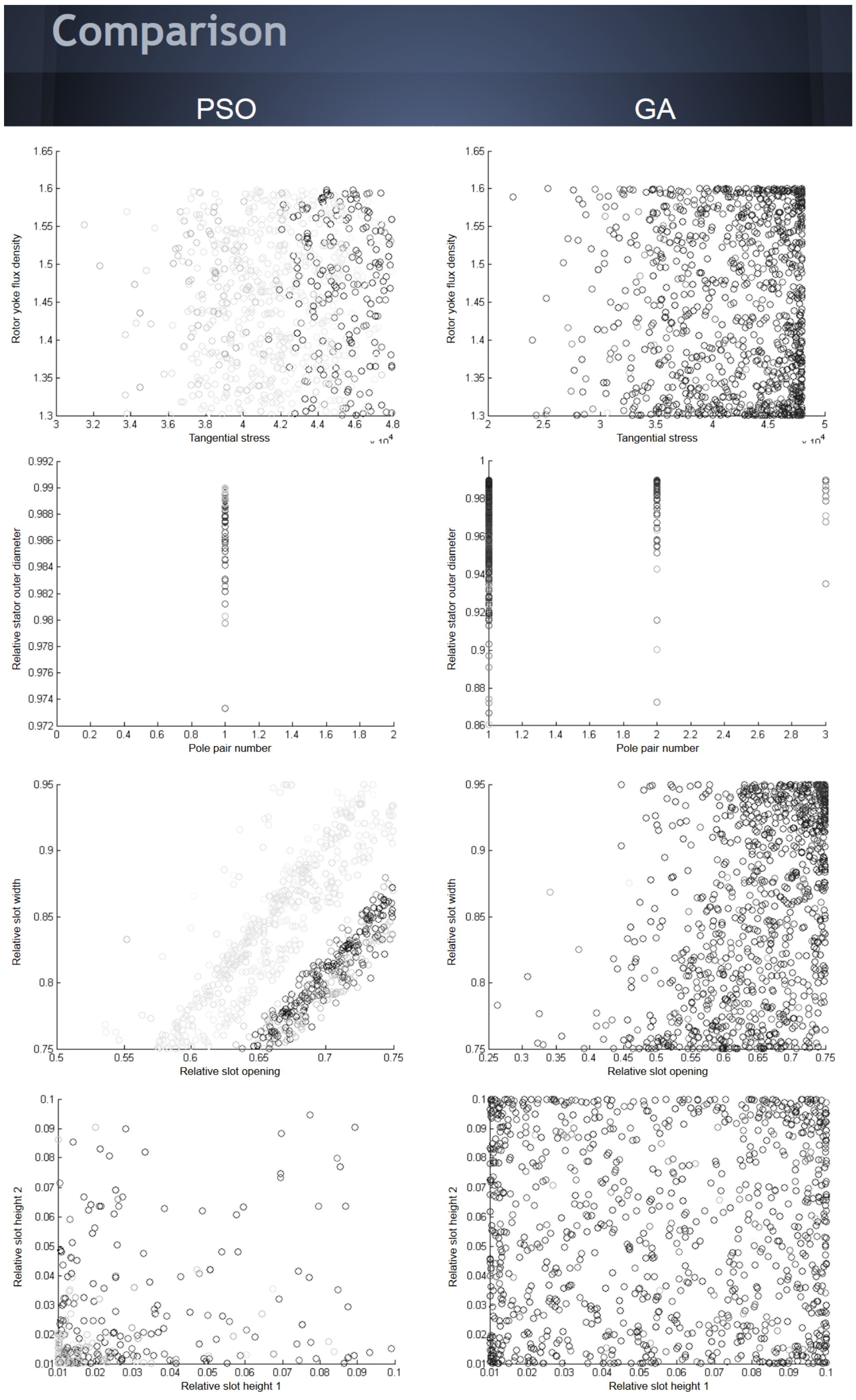
[2] Engelbrecht, A. *Particle Swarm Optimization: Pitfalls and Convergence Aspects.* Department of Computer Science, University of Pretoria, South Africa

[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

A. Result comparison





Comparison

PSO

GA

Table 1: *Comparison of the best generators*

	PSO	GA
Output power (3MW)	3.0107 MW	3.0625 MW
Torque density	1475489	2513175
Mass	1.6159e+05	1.4712+05
Efficiency	0.9891	0.9856
Power factor	0.9966	0.8206
Cost	9.0962e+06	5.3166e+06

B. Time management

lectures	12	
Janne	**total**	95.5
	total + lectures	107.5
date	time (h)	description
20.1.2014	2	Familiarization with the project and exploring different algorithms
21.1.2014	1	Checking the Matlab files
24.1.2014	2	Project planning
25.1.2014	2.5	Updating the plan to LaTeX
30.1.2014	0.5	First meeting
15.2.2014	1	Reading about the PSO algorithm
17.2.2014	7	Implementing PSO
17.2.2014	2	Studying parallelization
18.2.2014	1	Figuring out performance issues
24.2.2014	4	Matlab movie capture, wondering why the algorithm doesn't work
2.3.2014	5	Intermediate report writing
2.3.2014	1.5	Troubleshooting with simpler algorithm
3.3.2014	1	Preparing presentation slides
3.3.2014	1	Velocity clamping
3.3.2014	1	Finding suitable software to capture video from screen
4.3.2014	1	Intermediate report writing
4.3.2014	0.5	Video editing and upload to YouTube
15.3.2014	1	Started a clean attempt of PSO in order to improve code readability
15.3.2014	1	How to apply a function only to a subset of matrix elements
16.3.2014	1	Wondering one particle immobility
16.3.2014	1	Found the reason for the problem, there was a typo in the bestidx parameter
17.3.2014	3	Improve plotting, run simulation with large swarm, parametrize velocity calculation
17.3.2014	1	Print text to console, run multiple simulations
18.3.2014	1	Explore multidimensional data visualization
19.3.2014	2	Gather data for multiple runs of PSO
22.3.2014	1	Found out that the data from the 5 hour run was not saved. Started new calculation runs.
23.3.2014	1.5	Write script to analyze the optimal parameter distributions. Find info about the computing facilities in Aalto.
23.3.2014	0.5	Apply for a Triton account.
23.3.2014	1	Reading about various inertia handling methods.
24.3.2014	1	Ask for Triton account, discuss with Eero about the project.
25.3.2014	1.5	Read Triton manuals and about Bash scripting
26.3.2014	1.5	Read about SLURM and sbatch, project discussion

26.3.2014	1.5	Implementing PSO to be used in Triton
26.3.2014	1.5	Experiment with Bash scripting, script .mat file upload from Triton to Kosh
26.3.2014	1	Run tests and benchmark on home computer. Run hundred iterations on cluster.
26.3.2014	1	Every calculation produced the same results, set the random number seed manually. Start new calculation with 200 tasks.
27.3.2014	0.5	Basic analyzation of the data.
22.4.2014	0.5	Telling Eero what kind of function he should make so that it could be run in Triton, fixed random number generation
23.4.2014	2	Documentation
23.4.2014	0.5	For some reason the PSO_SLURM folder was deleted. Recovered missing files.
24.4.2014	0.5	Fixed documentation
24.4.2014	1.5	Run simulations for random inertia
25.4.2014	1	Some shell wizardry to find the failed (old) results and remove them as I forgot to clean the folder before running the job
26.4.2014	1	What to do with the data and is the collected data sufficient?
26.4.2014	0.5	Set up SSH tunneling so that the file transfering is easier
26.4.2014	1	Prepare and run new batch jobs
26.4.2014	1	The previous run was erroneous, made some modifications and started new batch jobs
27.4.2014	0.5	Check the simulation results
29.4.2014	1	Lunch discussion with Eero about the
29.4.2014	0.5	Write e-mail
29.4.2014	1	Prepare and run the GA simulations
2.5.2014	2	Change the fitness function to efficiency only, remove the saving of the development of the swarm from PSO_function, test, start new calculation in cluster
2.5.2014	2	Checking results and plotting plots
9.5.2014	1	Prepare and run GA with updated fitness function
9.5.2014	0.5	Start making presentation slides
10.5.2014	1	Changed GA to utilize parfor and started new runs. The previous run had only 1000 iterations and the convergence limit of 200.
10.5.2014	1	Prepare slides and make changes so that the values are stored to the disk after always after certain amount of iterations instead of at the end.
10.5.2014	1	Fight with the batch jobs
10.5.2014	1	Data analysis
10.5.2014	3	Visualizations and making slides
10.5.2014	0.5	Check and upload last GA runs
11.5.2014	2	Fighting with network problems
11.5.2014	3	Development of PSO fitness visualization
11.5.2014	2	Prepare slides
12.5.2014	1.5	Prepare slides
12.5.2014	0.5	Run GA, then run it again

22.5.2014	2.5	Documentation
25.5.2014	1.5	Documentaiton

Eero	**total**	**64.5**
total + lectures	77.5	
2.3.2014	32	intermediate report total
14.3.2014	3	Trouoble with the sum function, calculating fitness
17.3.2014	2	Fine-tuning the selection process
17.3.2014	2.5	Fixed a bug with crossover, first working model
24.3.2014	1	Miscellaneous discussion (including Triton)
26.3.2014	1.5	Modifying GA to handle negative fitness values
28.4.2014	6	Huge revamp to reduce convergence to single solution
29.4.2014	1	Lunch with Janne, discussion about data analysis
10.5.2014	4	Mutation chance analysis
11.5.2014	6	Mutation chance analysis, average fitness analysis
12.5.2014	1	Tweaking mutation chance, one more iteration on Triton
12.5.2014	2.5	Making slides
23.5.2014	4	Documentation