NTNU – Trondheim
Norwegian University of
Science and Technology

Optimal Design of Tidal Power Generator Using Stochastic Optimization Techniques

Erlend L Engevik

Master of Energy and Environmental Engineering
Submission date: June 2014
Supervisor: Robert Nilssen, ELKRAFT
Co-supervisor: Astrid Røkke, ELKRAFT

Norwegian University of Science and Technology
Department of Electric Power Engineering

Optimal Design of Tidal Power Generator Using Stochastic Optimization Techniques

Erlend Løklingholm Engevik
Norwegian University of Science and Technology
Department of Electric Power Engineering
email: erlend.l.engevik@gmail.com

Tidal energy extraction has the potential to become an important source of renewable electricity production. For tidal energy to become competitive with traditional sources of electric energy the cost per kWh must be reduced. One way of achieving this is to minimize the cost of the generator.

It is expected that the student investigates the usefulness of Genetic Algorithms versus Particle Swarm Optimization as a tool for design of electrical machines. Important evaluation criteria may be convergence towards the global optimal solution, variance of the solution from optimization to optimization and the need for computational resources.

Various studies have been made to investigate how one can combine the global search capability of stochastic optimization algorithms and local optimization using gradient based methods. It is of great interest to increase the knowledge at the Department of Electric Power Engineering on how this can be achieved with the optimization features that are included in Matlab and its associated toolboxes.

The department possesses computational resources that should be utilized more often. One part of the master's thesis work should be devoted to investigating the applicability of multi-processor computers to optimization problems. It is of interest to see how much the computation time for optimization of electrical machine design can be reduced with the help of parallel processing tools.

Supervisors:
Professor Robert Nilssen
PhD candidate Astrid Røkke

Samandrag

To stokastiske optimeringsalgoritmar kalla Genetiske algoritmar (GA) og Partikkelsvermoptimering (PSO) vert nytta til å seinka kostnadane til ein synkrongenerator med permanentmagnetar og konsentrerte viklingar. Generatoren er tenkt nytta til elektrisitetsproduksjon frå tidevatn. Kostnadsreduksjon av elektriske maskinar er ein veg å gå for å gjera tidevassenergi meir konurransedyktig i forhold til tradisjonell elektrisitetsproduksjon.

Hybride optimeringsmetodar som kombinerer PSO eller GA med tradisjonelle, gradientbaserte algoritmar ser ut til passa godt til design av elektriske maskinar. Optimeringsresultat frå Matlab indikerer at hybrid-GA oppnår betre designforslag enn hybrid-PSO for denne typen problemstillingar. Hybrid-GA viser betre konvergens, mindre varians og kortare køyretid.

Hybrid-GA konvergerte mot ein gjennomsnittskostnad for generatoren som var 5,2 % lågare enn gjennomsnittskostnaden hybrid-PSO kom fram til. Variansen til resultata frå hybrid-GA er 98,6 % lågare enn variansen til hybrid-PSO. Overgangen frå vanleg GA til hybridversjonen reduserte snittkostnaden til generatoren med 31,2 %.

Parallellprosessering gjer det mogleg å redusera køyretida til kvar optimering med opp til 97 % for store problem. Køyretida for GA med 2500 individ vart redusert frå 12 timar til 21 minutt ved å gå frå ein einkjernemaskin til ein datamaskin med 48 prosessorkjernar. Tida det tok å optimera med 400 partiklar og 100 iterasjonar med PSO gjekk frå 18,5 timar til 74 minutt. Dette svarar til ein reduksjon på 93 %.

Summary

Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) are used to reduce the cost of a permanent magnet synchronous generator with concentrated windings for tidal power applications. Reducing the cost of the electrical machine is one way of making tidal energy more competitive compared to traditional sources of electricity.

Hybrid optimization combining PSO or GA with gradient based algorithms seems to be suited for design of electrical machines. Results from optimization with Matlab indicate that hybrid GA performs better than Hybrid PSO for this kind of optimization problems. Hybrid GA shows better convergence, less variance and shorter computation time than hybrid PSO.

Hybrid GA managed to converge to an average cost of the generator that is 5.2 % lower than what was reached by the hybrid PSO. Optimization results show a variance that is 98.6 % lower for hybrid GA than it is for hybrid PSO. Moving from a pure GA optimization to the hybrid version reduced the average cost 31.2 %.

Parallel processing features are able to reduce the computation time of each optimization up to 97 % for large problems. The time it took to compute a GA problem with 2500 individuals was reduced from 12 hours to 21 minutes by switching from a single-processor computer to a computer with 48 processor cores. The run time for PSO with 400 particles and 100 iterations went from 18.5 hours to 74 minutes, a 93 % reduction.

Optimal Design of Tidal Power Generator Using Stochastic Optimization Techniques

Erlend Løklingholm Engevik, *Student, Norwegian University of Science and Technology (NTNU)*

***Abstract*—Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) are used to reduce the cost of a permanent magnet synchronous generator with concentrated windings for tidal power applications. Reducing the cost of the electrical machine is one way of making tidal energy more competitive compared to traditional sources of electricity. Hybrid optimization combining PSO or GA with gradient based algorithms seems to be suited for design of electrical machines. Results from optimization with MATLAB indicate that hybrid GA performs better than Hybrid PSO for this kind of optimization problems. Hybrid GA shows better convergence, less variance and shorter computation time than hybrid PSO. Hybrid GA managed to converge to an average cost of the generator that is 5.2 % lower than what was reached by the hybrid PSO. Optimization results show a variance that is 98.6 % lower for hybrid GA than it is for hybrid PSO. Moving from a pure GA optimization to the hybrid version reduced the average cost 31.2 %. Parallel processing features are able to reduce the computation time of each optimization up to 97 % for large problems. The time it took to compute a GA problem with 2500 individuals was reduced from 12 hours to 21 minutes by switching from a single-processor computer to a computer with 48 processor cores. The run time for PSO with 400 particles and 100 iterations went from 18.5 hours to 74 minutes, a 93 % reduction.**

Keywords*—Hybrid Optimization, Genetic Algorithms, Particle Swarm Optimization, Electrical Machine Design, Parallel Processing.

I. Introduction

A. Background

Tidal energy extraction has the potential to become an important source of renewable electricity production [1] [2]. For tidal energy to become competitive with traditional sources of electric energy, the cost per kWh must be reduced. One way of achieving this is to minimize the cost of the generator.

This paper is a continuation of the work on optimization presented in [3]. Design of electrical machines using stochastic optimization runs back to the late 1960s [4]. Different stochastic optimization algorithms have been used during the design process, including Tabu Search [5], Simulated Annealing [6], Genetic Algorithms (GA) [7], Differential Evolution [8] and Particle Swarm Optimization (PSO) [9].

Various studies have been made to investigate how one can combine the global search capability of stochastic optimization algorithms and local optimization using gradient based methods. Hybrid algorithms have been used on design of both induction motors [10] and permanent magnet synchronous generators [11].

Several studies on hybrid optimization have been published using GA [12] [13] and PSO [14] [15] as global search method.

This paper will focus on Genetic Algorithms and Particle Swarm Optimization in combination with gradient based methods. Genetic algorithms exploit the principles of natural selection and survival of the fittest in order to select the best design configuration. Particle swarm optimization mimics the behaviour of social swarms who shares collective knowledge in the search for desirable features.

B. Scope and limitations

The scope of this paper is to investigate if either Particle Swarm Optimization or Genetic Algorithms is better suited for design of electrical machines. Different methods for improving the optimization process will be evaluated.

In the next section, the theoretical foundations of general optimization problems are outlined along with a description of Particle Swarm Optimization and Genetic Algorithms. In Section III, the details of the permanent magnet synchronous machine model used in this paper are introduced.

Implementation of Particle Swarm Optimization in MATLAB is described in Section IV. GA and parallel computing solutions available in MATLAB are presented. One method for hybrid optimization is also introduced and described.

Section V presents the results from optimizing the synchronous generator model with PSO, GA and hybrid optimization. Results from the use of parallel computing are given.

Genetic Algorithms and Particle Swarm Optimization are compared in Section VI based on their demand for computational resources and on how efficiently each method converges towards the optimal solution. The usefulness of hybrid optimization and the savings in computational resources when using parallel processing are discussed. Section VII draws conclusions from the work presented in this paper.

II. Optimization techniques

A. General formulation of optimization problems

A basic optimization problem can be formulated in the following way:

$$\min_{\mathbf{x}} \quad Z(\mathbf{x}) \tag{1}$$

Subject to

$$g_i(\mathbf{x}) \leq b_i \tag{2}$$

$$h_j(\mathbf{x}) = d_j \tag{3}$$

$$L \leq \mathbf{x} \leq U \tag{4}$$

for $i = 1 : p$ inequality constraints and $j = 1 : q$ equality constraints.

Eq. 1 is the objective function and is needed to measure the quality of a solution relative to other possible solutions. The feasible region is defined by the inequality constraints in Eq. 2, the equality constraints in Eq. 3 and the upper and lower limits of the independent variables given by Eq. 4.

Most engineering problems involve some sort of nonlinear formulation either in the formulation of objective functions or in constraints to the feasible region of solutions. Nonlinear optimization algorithms are the response to this challenge [3], but they have one important drawback.

Gradient based, nonlinear algorithms are generally unable to reach the global optimum as they get trapped in local optima [16]. They are however, from a given starting point, quite efficient at finding the nearest optimal solution as they use the gradient of the objective function to find the best search direction [16].

Integer variables are needed when electrical machines are to be optimized. Poles, stator slots and the number of base windings are some of the variables that must be integers. Gradient based methods can not optimize problems with integer variables [17]. Other optimization algorithms should be investigated.

Stochastic optimization algorithms like Genetic Algorithms and Particle Swarm Optimization are sophisticated search methods. They search through the feasible region to find good solutions [16]. The solution from such stochastic algorithms can not generally be guaranteed to be the optimal solution.

B. Particle Swarm Optimization

PSO has become increasingly popular in the field of electromagnetics and electrical machine design and was first suggested by Kennedy and Eberhart in [18] and [19]. This presentation is a modified version of what was presented in [3].

PSO emulates the behaviour of a flock of birds or a swarm of bees [19] where each individual in the population is searching through the solution space for the best possible position. Individuals are utilizing both their own memory and the collective memory of the swarm. Each individual is racing through the solution space with a given velocity and direction in search for a better location. The direction of each individual is determined by the best position recorded by the swarm and the best position recorded by the individual.

For each iteration every individual in the swarm is moved into a new position where a new fitness value is calculated. The fitness value is compared to the best recorded position of both the individual and the swarm. If a better position is found, then the memory is updated. This is repeated until the algorithm reaches a predefined stopping criteria. A graphical presentation of the algorithm can be found in Figure 1.

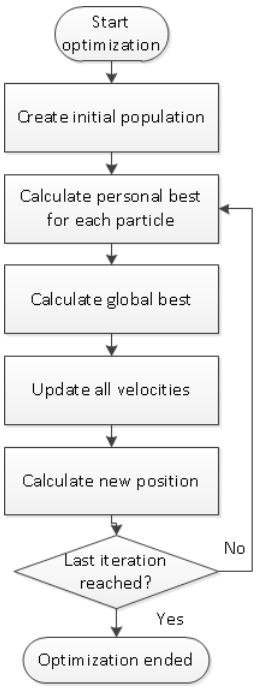

Fig. 1. Basic flow chart for Particle Swarm Optimization

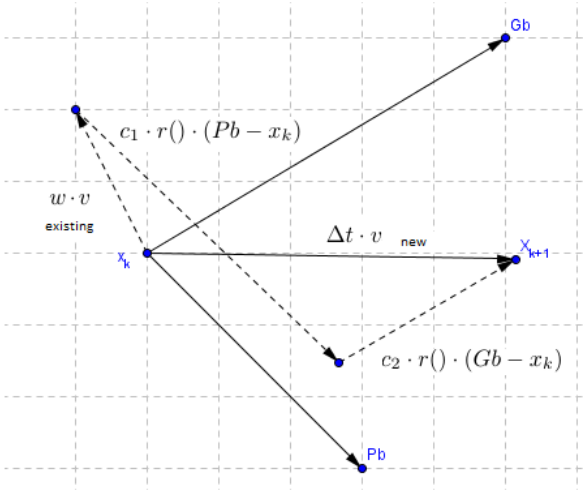

Fig. 2. Graphical representation of the position calculation

According to [20], the updated velocity and position along each dimension for each particle can be formulated using Eqs. 5 and 6 together with Table I. A graphic presentation is given in Figure 2.

$$v_n = w \cdot v_n + c_1 \cdot r() \cdot (x_n^{pb} - x_n) + c_2 \cdot r() \cdot (x_n^{gb} - x_n) \quad (5)$$

$$x_n = x_n + \Delta t \cdot v_n \quad (6)$$

TABLE I.
PSO PARAMETER VALUES

Δt	0.1
c_1	2
c_2	2
w	$\in [0.4, 0.9]$

A linearly decreasing inertial weight is proposed in [21] on the form of Eq. 7. The method of using a decreasing inertial weight is utilized in the standard PSO that is implemented for this paper.

$$w(i) = w_{max} - (\frac{w_{max} - w_{min}}{i_{max}}) \cdot i \quad (7)$$

w_{max} is the initial value of the inertial weight and w_{min} is the final value. i_{max} is the maximum number of iterations and i is the current iteration.

Boundary conditions can be implemented by applying different constraints to the solution space. Three different methods for dealing with boundary conditions have been proposed in [20]. In this paper, *absorbing walls* have been chosen. This method sets the velocity component of a particle in the direction of the wall to zero when the particle hits the wall.

There are many suggested ways of improving the performance of the PSO. A few of them are presented in [22], [23] and [24]. Two suggestions that will be investigated are called *PSO with centroid (CPSO)* [25] and *improved PSO (IPSO)* [26]. A detailed description on how to implement the IPSO and CPSO is presented in Appendix A.

C. Genetic Algorithms

Genetic Algorithms, or evolution programs, are sophisticated search methods. This section is based on modifications to the presentation of Genetic Algorithms in [3]. Along with methods like Simulated Annealing and Particle Swarm Optimization, Genetic Algorithms will search through a vast solution space to come up with a solution that is as close to the global optimum as possible. The method of Genetic Algorithms exploit the features of genetic evolution in nature. This process can be summarized, according to [27], by Figure 3.

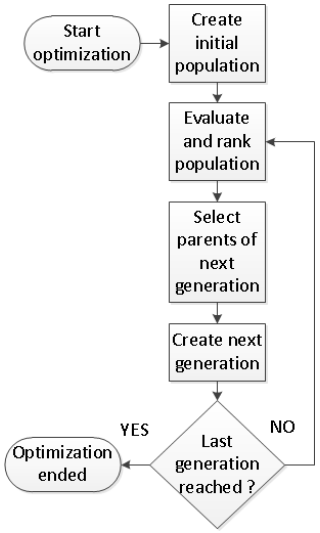

Fig. 3. Basic flow chart for genetic algorithms

There are several selection algorithms that can be utilized together with genetic algorithms. *Tournament selection* [28] will be used based on recommendations from [3].

Selection processes must be able to favour individuals with higher fitness values in order to converge on the global optima. According to [28] this is called *selection pressure*. Tournament selection has the advantage that it can be organized to give a relatively high selection pressure.

By utilizing a tournament selection algorithm you will get higher fitness values for each individual that will form the foundation for the next generation. Genetic algorithms will converge faster as a result of this. The drawback of this method is that it requires more computational resources. This can be solved by parallel processing.

Genetic operators are utilized in order to create new individuals for the next generation in GA. The most commonly used operators are *crossover-*, *mutation-* and *elite operators* [29].

Crossover operators use two parent individuals and creates an offspring for the next generation. These operators creates the individual by combining one part of one parent's vector of variable values and one part of the other parent's vector of variable values. The main concept is to generate a point in the vector containing the values of each variable where the variable values before this point is filled with the equivalent values from the first parent, and the variable values after this point is filled with the equivalent values from the other parent.

Mutation operators are often applied after the crossover operator has created the next generation. Their function is to randomly change one or more values in the vector of variable values of one individual. This is done to increase diversity and to improve the convergence process [30].

An *elite operator* can be used to send the most fit individual of one generation on to the next without any changes. A global optimum may be reached faster by implementing this operator.

III. Tidal power generator model

A permanent magnet synchronous generator with concentrated windings is optimized in this paper. It is designed for tidal power applications, and the machine model is implemented and optimized with gradient methods by Astrid Røkke for her PhD studies [17]. This paper will apply stochastic optimization techniques to the same model.

A. Variables and design specifications

TABLE II.
DESIGN SPECIFICATIONS

P_{shaft}	Turbine power	1.5 MW
n	Mechanical speed	80 rpm
N_p	Number of poles	$22 \cdot n_{base}$
N_s	Number of slots	$24 \cdot n_{base}$
N_{ph}	Number of phases	3
V	Line-to-line voltage (rms)	3300 V

This permanent magnet synchronous generator is designed to be used for tidal power applications. Rated mechanical speed is therefore set to be 80 rpm with a rated power of 1.5 MW. All design specifications are listed in Table II, where initially the number of poles is kept as listed.

The generator model is using the variables listed in Table III. All the geometric variables are presented graphically in Figure 4. In chapter 2.11 of [31], base windings are defined in the following way:

"The smallest independent symmetrical section of a winding is called a base winding. When a winding consists of several base windings, the current and voltage of which are due to geometrical reasons always of the same phase and magnitude, it is possible to connect these basic windings in series and in parallel to form a complete winding".

TABLE III.
INDEPENDENT MACHINE VARIABLES

D_{outer}	Outer diameter of stator lamination [m]
$d_{s,yoke}$	Thickness of stator yoke [m]
d_{slot}	slot depth [m]
w_{slot}	slot width [m]
l_m	length of magnets [m]
α_m	Magnet width / pole pitch ($\frac{w_m}{\tau_p}$)
$d_{r,yoke}$	Thickness of rotor yoke [m]
J	Current density [A/mm^2]
n_{base}	Number of base windings

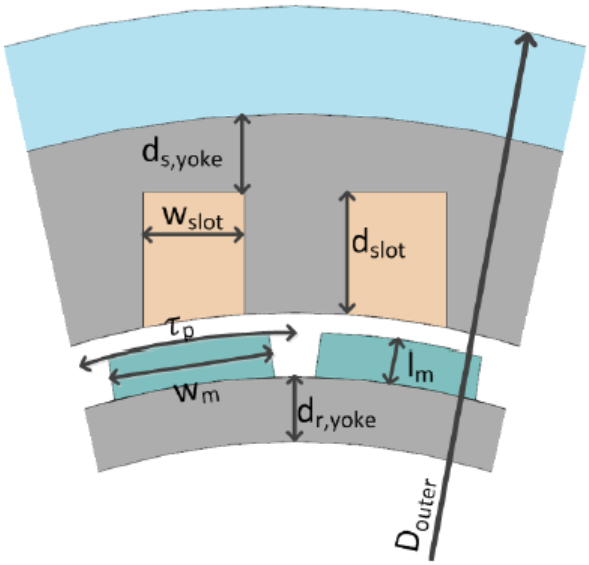

Fig. 4. Graphic representation of independent machine variables, taken from "Gradient based optimization of permanent magnet generator design" by A. Røkke [17]

B. Constraints and limitations

Boundaries are applied to the independent machine variables. These are presented in Table IV and are modelled as one vector of lower bounds and one vector of upper bounds.

TABLE IV.
CONSTRAINTS ON MACHINE VARIABLES

Parameter	Lower constraint	Upper constraint
Outer diameter	3 m	3.5 m
Stator yoke thickness	15 mm	60 mm
Slot depth	20 mm	75 mm
Slot width	15 mm	50
Magnet length	2 mm	30 mm
Magnet width / pole pitch	0.8	0.98
Rotor yoke thickness	10 mm	60 mm
Current density	1.5 A/mm^2	4 A/mm^2
Base windings	1	13

A set of parameter constraints on the machine is defined in Table V. Constraints are used to ensure that the machine meets

operational requirements on parameters like power factor, efficiency and torque production.

Variables, boundaries, constraints and machine details are implemented in a form that makes the model compatible to the functionality of the optimization toolboxes of MATLAB. This makes comparison between different optimization algorithms straight forward.

TABLE V.
PARAMETER CONSTRAINTS

Parameter	Lower constraint	Upper constraint
Machine length	0	6 m
Tooth width	5 mm	∞
Magnet width	0	$\frac{2\cdot\pi\cdot R_inner}{N_p}$ + 1 mm
Inner radius	0	∞
Winding width	1 mm	∞
Winding diameter	0	∞
Tooth flux density	0	1.5 T
Stator yoke flux density	0	1.1 T
Rotor yoke flux density	0	0.9 T
Current loading	0	35 kA/m
Air gap torque	$T_{required}$	∞
Efficiency (η)	0.97	1.0
Power factor	0.85	1.0
Frequency	0	200 Hz
Mech. time constant (τ_{mech})	0	30 μ s

C. Model details

A magnetic circuit model, see Figures 5 and 6, is used to analytically calculate the air gap flux density in the machine. Air gap length is modified using the Carter coefficient [32]. Air gap flux is used together with the stator current to calculate the working torque. Torque is calculated with Eq. 8. Stator current is calculated using winding theory presented in [32] and [31].

$$T = \frac{3}{2}\frac{N_p}{2}L\phi_m I_s\sqrt{2} \tag{8}$$

T is the produced torque, ϕ_m is the flux linkage from the rotor field and I_s is the stator current. The efficiency of the machine is found after copper-, magnet-,rotor- and stator yoke losses have been calculated. Other aspects of the machine that are modelled are the power factor and the mechanical torque relationship. For a more detailed description of the model of this machine, see [17].

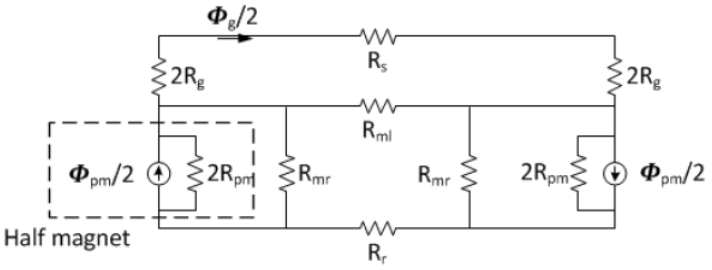

Fig. 5. No-load magnetic circuit model for air gap flux density calculations [17].

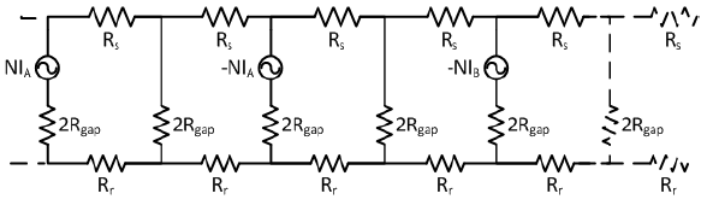

Fig. 6. Section of armature reaction circuit for calculation of electric loading and magneto-motive force [17].

D. Objective function

One of the main challenges for renewable electricity generation is cost reduction. Total cost is therefore an integral part of the objective function for optimizing a machine for tidal energy conversion.

$$\begin{aligned} Z(\mathbf{x}) = M_{s,yoke}\cdot C_{lamination} + M_{conductor}\cdot C_{copper} \\ +M_{r,yoke}\cdot C_{steel} + M_{pm}\cdot C_{pm} \end{aligned} \tag{9}$$

TABLE VI.
COST COEFFICIENTS

$C_{lamination}$	C_{copper}	C_{steel}	C_{pm}
4 €/kg	11 €/kg	6 €/kg	85 €/kg

M is the mass of each material and C is the cost per unit of mass in Eq. 9. A list of unit costs for each material is given in Table VI. Another possible objective function may be total mass instead of total cost.

IV. MATLAB BASED OPTIMIZATION TOOLS

A. Particle swarm optimization

The Particle Swarm Optimization algorithm that is implemented for this paper uses four input parameters. Population size and the number of iterations must be set to govern the

optimization process. Two vectors must be defined to set the upper and lower boundaries of the independent variables. Constraints are given directly by the script containing the objective function.

When the population for a PSO problem is created, two things need to be taken into consideration. First, all variables are randomly given a value within a given range. Second, each individual is evaluated to make sure that the suggested solution is within the feasible design space set by inequality constraints.

This paper has selected the following method for creating the initial population; a set of variable values are randomly drawn and then the solution is checked for feasibility. The script moves on to create another particle in the population if the suggested particle is feasible. Otherwise a new particle is drawn and checked until a feasible combination is found. Global best position and personal best position for each particle are calculated after the initialization is completed, and at the beginning of each iteration.

Velocity and position for the standard version of PSO is calculated in each iteration according to Eqs. 5 and 6. Inertial weight is calculated using Eq. 7 and the rest of the parameters of the standard PSO is given in Table I.

In centroid PSO Eq. 5 is substituted with Eqs. 19, 20 and 21 while improved PSO uses Eq. 13 to update velocity. Parameters and details are given in Appendix A.

The details of the code used in this report can be examined in Appendix C.

B. Genetic algorithms

The Global Optimization Toolbox of MATLAB [33] provides a complete package of genetic algorithm features. The toolbox is used to optimize the machine model presented in Section III and compare it to the implementation of Particle Swarm Optimization.

An objective function, a function with constraints, the size of the problem and boundaries for the independent variables must be supplied by the user. A detailed procedure for the setup of GA in MATLAB is presented in Appendix B.

C. Hybrid optimization

Population based, stochastic algorithms are good at escaping local minima but not as good at finding the exact global optimum. Gradient based optimization is, on the other hand, unable to escape local minima [16].

Combining stochastic and gradient methods into a hybrid solution will prevent both inability to converge, and premature convergence tendencies. The method outlined in this paper focuses on the use of MATLAB and its associated toolboxes [33] [34] as the vehicle for the optimization process.

In MATLAB, the standard optimizer for constrained nonlinear problems is called *fmincon* [34]. Four different algorithms are included for *fmincon*; *Trust Region Reflective Algorithm*, *Active Set Algorithm*, *Interior Point Algorihtm* and *SQP Algorithm*.

Constrained, nonlinear optimization with *fmincon* has some limitations. First, *fmincon* is gradient based and therefore made for problems with continuous objective functions, continuous constraints and continuous first derivatives. This excludes the option of using integer variables when this solver is applied. Stochastic optimization is not dependent on gradients. Integer variables are therefore not a problem. GA has implemented integer optimization options and PSO is easily modified to take integer constraints into account.

Astrid Røkke [17] has tested all four versions of *fmincon* on the generator model presented in Section III, and found that *Trust Region Reflective Algorithm* cannot use nonlinear constraints. It is therefore not applicable to models of electrical machines. Optimizing with *Interior Point Algorihtm* gave both more robust and faster answers than the remaining two algorithms [17]. All three methods reached similar optimization results.

Interior Point Algorithm solves a problem as a sequence of approximate minimization problems as defined in Eqs. 10, 11 and 12. This is done by changing the inequality constraints into equality constraints by introducing slack variables [34]:

$$\min_{x,s} \quad f_\mu(x) - \mu \sum_{i=1} \ln(s_i) \tag{10}$$

Subject to

$$h(x) = 0 \quad \text{and} \quad g(x) + s = 0 \tag{11}$$

$$\mu \to 0 \Rightarrow f(x) \to f(x)^* \tag{12}$$

Where μ is a positive constant and each s is a slack variable representing a non-binding constraint. The Interior Point Algorithm computes each step with one of two built-in methods. First, it tries to solve the KKT-conditions with a linear approximation of the reformulated problem. KKT-conditions are requirements for solutions of nonlinear optimization problems. For more details on nonlinear optimization and KKT-conditions, see [3]. The first step is called the Direct step or Newton step approach. Second, if the Newton step fails the algorithm will solve the problem with a Conjugate Gradient step using a trust region approach [34].

The method for hybrid optimization used in this paper runs either PSO or GA with a population of 25 individuals in order to search for the global optimal region. The output vector containing the values of the independent variables from stochastic optimization is fed as input to the *fmincon*-solver.

Variables that are set to be integer numbers are handled by setting the lower and upper boundary for that particular variable to the output from stochastic optimization. The rest of the boundaries and constraints are imported in the same way as with PSO and GA, see Figure 7.

```
[X fval exitflag output population scores] = ga(
    @iTool_fun,10,[],[],[],[],LB,UB,@iTool_con,
    options);


options = optimset('Algorithm','interior-point','
    MaxFunEvals',1500,'TolCon',1e-10,'TolX',1e-12,'
    ObjectiveLimit',0); % run interior-point
    algorithm
[X2,fval,exitflag,output] = fmincon(@iTool_fun,X
    ,[],[],[],[],LB,UB,@iTool_con,options);
```

Fig. 7. Setup of hybrid GA in MATLAB

GA, as implemented in MATLAB has options for hybrid optimization for unconstrained and constrained problems [33]. Constrained problems are handled in the same way as discussed in this section. Output from GA is sent to *fmincon* after the final step is reached in GA. Using the automatic option is not possible for problems with integer variables. Hybrid optimization is set in GA by adding *'HybridFcn'*,{*@fmincon,hybridopts*} to the options declaration for the *ga*-function [33]. *hybridopts* must be declared before it is used.

D. Parallel computing solutions

One challenge with stochastic optimization is the amount of computational resources needed. In order to efficiently work on problems with more than a handful of independent variables, it is desirable to utilize parallel processing tools.

One definition of parallel processing is to use more than one computer processor to perform work on a problem. Independent tasks are performed simultaneous on several processors [35]. MATLAB offers the possibility for parallel computing on machines with more than one processor core. This is done through the Parallel Computing Toolbox [36].

For a problem to be suitable for parallel computing, it must be made up of tasks that can be processed independently. The number of tasks must be large enough to defend the administration costs of distributing each task to independent processing units. Stochastic optimization seems to be well suited for parallel processing. The main reason for this is that the problem consists of independent sub-problems, i.e. particles or individuals. Numerous operations are executed iteratively.

In the PSO-algorithm that has been implemented for this paper, see Appendix C-B, only the initialization part has been modified for parallel computing. Initialization is done by drawing a random particle configuration and checking if it is a feasible solution. If the proposed particle is infeasible, a new random configuration is drawn. By using parallel computing, this can be changed into a structure called *parfor*, see Appendix C-E. With *parfor* each iteration of the for-loop is distributed to the available number of parallel workers [36].

Parallel computing with Genetic Algorithms in MATLAB is done by enabling the 'UseParallel','always' option for the *ga*-function, see Appendix B. To enable parallel computing in MATLAB, the command *parpool('local')* must be given. By telling MATLAB to use the setting 'local', it will use the available number of processor cores. One can specify the number of cores one will utilize by giving the number instead of 'local' when starting *parpool*, e.g. 12 or 24.

Parallel computing software using clusters of computers in cloud solutions are presently being tested at NTNU [37]. One can get access to up to 168 processor cores. Techila Distributed Computing Solution can be used to perform computationally intensive MATLAB applications on distributed work stations by utilizing unused computer processors, e.g. computer labs. This solution has been tested in the master's thesis of Anne-Siri Borander at the Department of Electric Power Engineering during the spring of 2014 [38]. Techila Distributed Computing does not need the Parallel Computing Toolbox [37].

V. Optimization results

A. Particle swarm optimization

The best design found using a hybrid Particle Swarm Optimization algorithm with gradient optimization in the final stage is presented in Table VII. This design proposal is the final result of optimizing with PSO.

TABLE VII.
Design proposal from Particle Swarm Optimization

	Cost	115 900 €
	Total weight	15056 kg
	Independent variables:	
D_{outer}	Outer diameter of stator lamination	3.5 m
$d_{s,yoke}$	Thickness of stator yoke	38 mm
d_{slot}	slot depth	62 mm
w_{slot}	slot width	30 mm
l_m	length of magnets	6 mm
α_m	Magnet width / pole pitch ($\frac{w_m}{\tau_p}$)	0.972
$d_{r,yoke}$	Thickness of rotor yoke	35 mm
J	Current density	2.57 A/mm^2
n_{base}	Number of base windings	4
	Machine parameters:	
l	Length	0.957 m
η	Efficiency	97 %
pf	Power factor	0.85
	Current loading	21016 A/m
w_{Cu}	Copper weight	920 kg
w_{magn}	Magnet weight	420 kg
w_{Fe}	Iron weight	9103 kg

Moving from 25 particles and 25 iterations to 25 particles and 200 iterations gave a 1.15 % reduction in the variance of the optimization results. Mean value increased with 0.3 %. Optimizing with 100 particles and 25 iterations instead of 25 particles and 25 iterations reduced both mean value and variance with 11.6 % and 19.1 % respectively. The above mentioned results are presented in Figure 8.

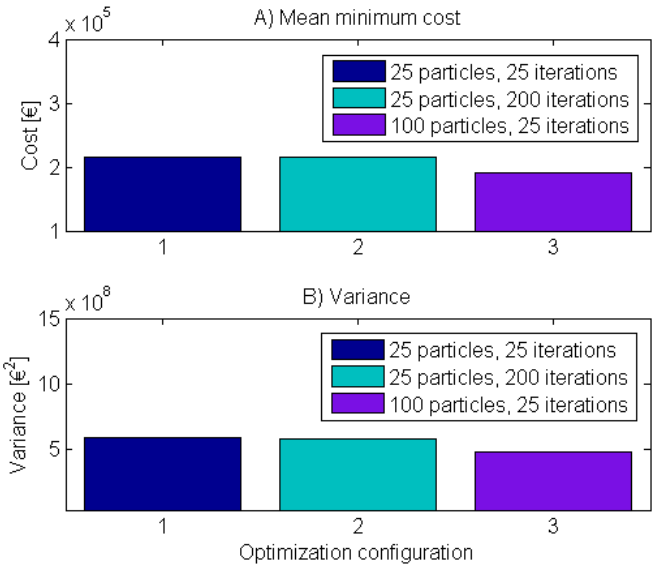

Fig. 8. Particle Swarm Optimization results after 100 runs of each configuration

Every configuration was run 100 times in order to get statistical data to analyse. Figures 9, 10, 11 and 12 display how the optimization results fluctuates from one run to the other. Variance is used to present the amount of fluctuation for each configuration.

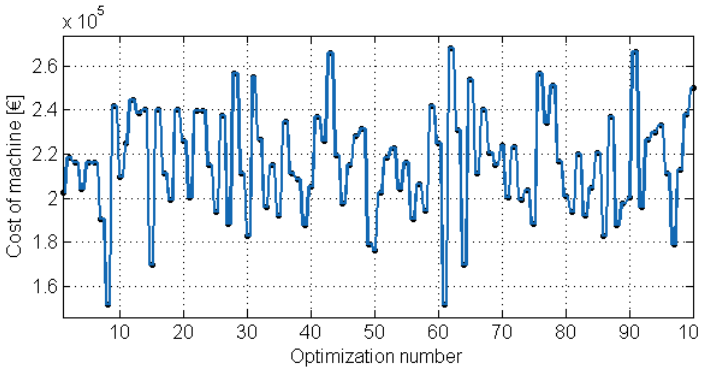

Fig. 9. Plot of standard PSO results

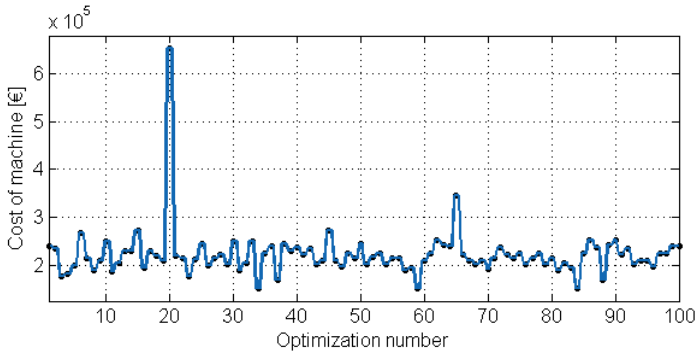

Fig. 10. Plot of improved PSO results

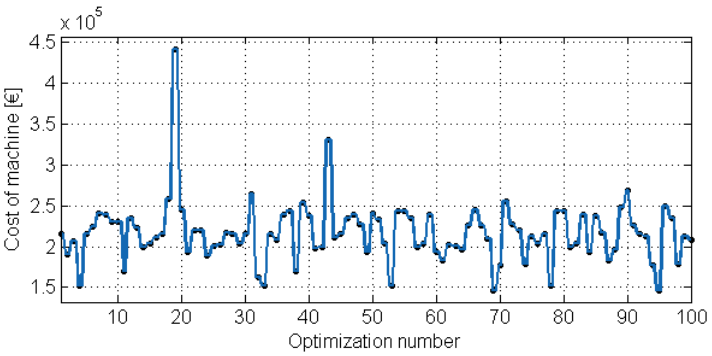

Fig. 11. Plot of centroid PSO results

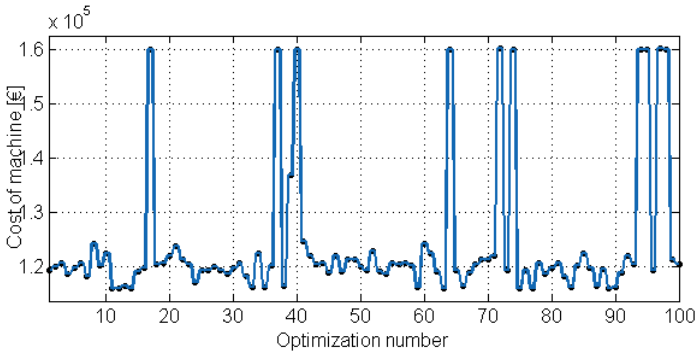

Fig. 12. Plot of hybrid PSO results

PSO has been evaluated in three different versions. Comparisons have been made using 25 particles and 200 iterations. The versions are labelled standard-, improved- and centroid PSO. Results are presented in Figure 13.

Improved PSO gave a mean value that was 3.45 % higher than the standard version and a variance that was 354.8 % higher. Centroid PSO gave a 0.39 % higher mean value and a variance that was 143 % higher than the standard version.

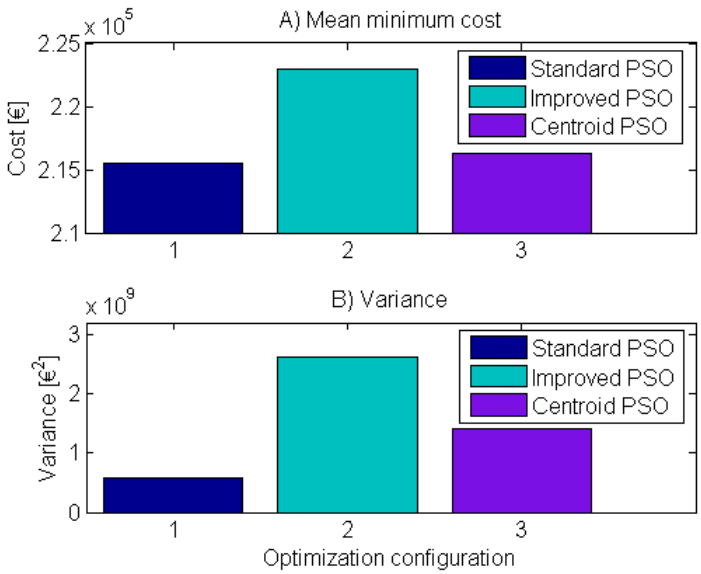

Fig. 13. Different versions of Particle Swarm Optimization

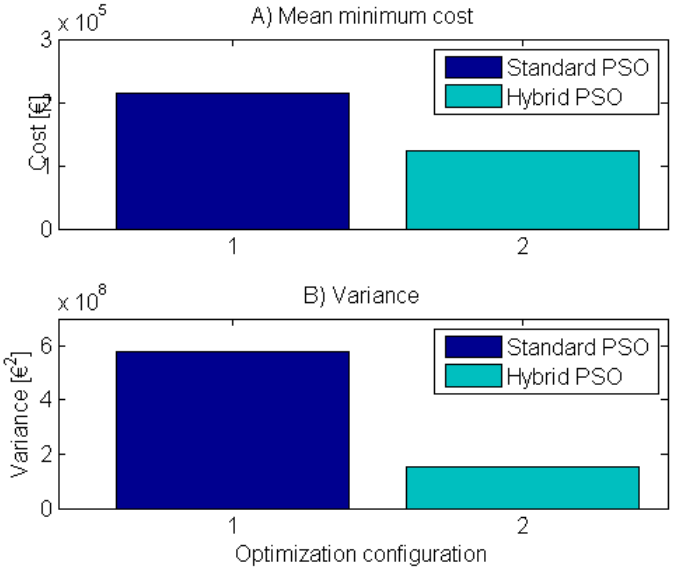

Fig. 14. Standard versus hybrid PSO

Including gradient optimization in the final stage of the optimization was tested. Data from testing the hybrid version are presented in Figures 12 and 14. Output from hybrid PSO indicates that a gradient step at the end of the optimization process improves the result. Mean value was reduced with 42.5 % and variance went down 73.2 %.

B. Genetic Algorithms

A hybrid version of the Genetic Algorithms similar to the hybrid version of PSO has been used to find an optimal design proposal for the permanent magnet synchronous machine. The design configuration is presented in Table VIII and is similar to the proposal from Particle Swarm Optimization presented in Table VII.

TABLE VIII.
DESIGN PROPOSAL FROM GENETIC ALGORITHMS

	Cost	115 889 €
	Total weight	15062 kg
	Independent variables:	
D_{outer}	Outer diameter of stator lamination	3.5 m
$d_{s,yoke}$	Thickness of stator yoke	38 mm
d_{slot}	slot depth	63 mm
w_{slot}	slot width	30 mm
l_m	length of magnets	6 mm
α_m	Magnet width / pole pitch ($\frac{w_m}{\tau_p}$)	0.9719
$d_{r,yoke}$	Thickness of rotor yoke	36 mm
J	Current density	2.55 A/mm^2
n_{base}	Number of base windings	4
	Machine parameters:	
l	Length	0.953 m
η	Efficiency	97 %
pf	Power factor	0.85
	Current loading	21054 A/m
w_{Cu}	Copper weight	924 kg
w_{magn}	Magnet weight	419 kg
w_{Fe}	Iron weight	9124 kg

100 optimizations with 25 individuals and 200 optimizations with 250 individuals have been run with three different crossover fractions. This was done to examine the possibility for correlation between population size, crossover fraction, mean value and variance.

GA with populations of 25 individuals seems, according to Figure 15, to give increasingly better results when crossover fraction is changed from 0.8 via 0.6 to 0.4. A change of crossover fraction from 0.8 to 0.6 reduced the mean value from 187 540 € to 184 370 € or 1.7 %. Variance was reduced 13.2 %. Changing the crossover fraction to 0.4 reduced the mean value to 174 330 €, a 7 % reduction from the initial value. Variance was 29 % lower for this crossover fraction, something that can be seen when comparing Figures 17, 18 and 19.

For the case with 250 individuals, it seems to be better to use a crossover fraction of 0.6 if focus is solely on mean value. A crossover fraction of 80 % gives a mean cost of 145 650 € with a 3.6 % and 2.1 % reduction when crossover fraction is lowered in steps of 20 percentage points. The first reduction in crossover fraction seems to increase the variance 408 % while the second reduction in crossover fraction moves the variance down to a 206.4 % increase from the initial value.

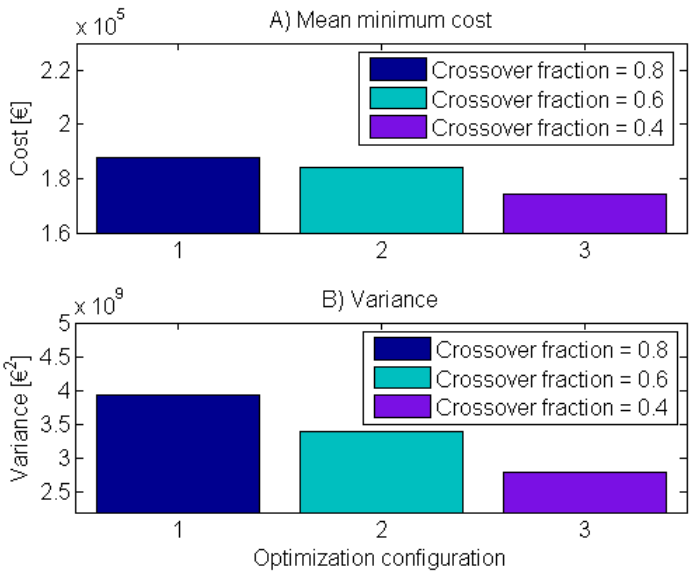

Fig. 15. 100 runs of GA with different configurations and 25 individuals

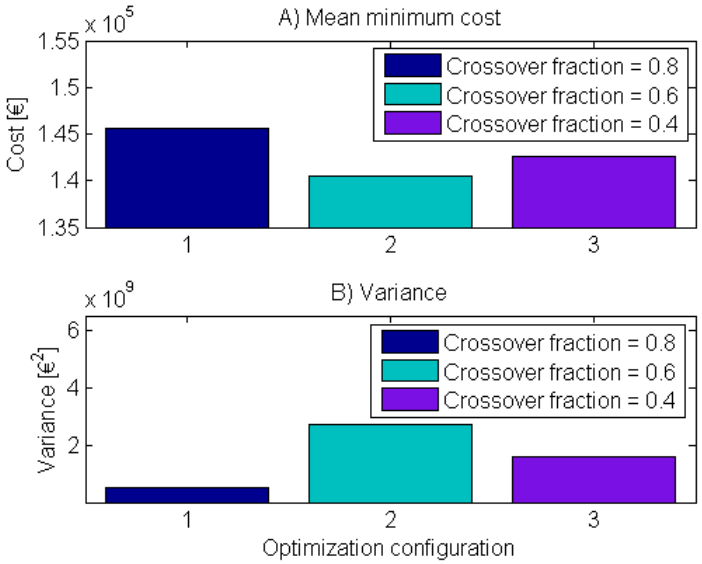

Fig. 16. 200 runs of GA with different configurations and 250 individuals

By comparing Figures 15 and 16, one can see that the mean value from a crossover fraction of 0.4 is reduced from 174 330 € to 142 610 € when population size is changed from 25 to 250. This corresponds to a 18.2 % reduction in the mean cost of the machine. Similarly, the variance of the 250 individual optimization is 41.1 % lower than when the number of individuals is 25.

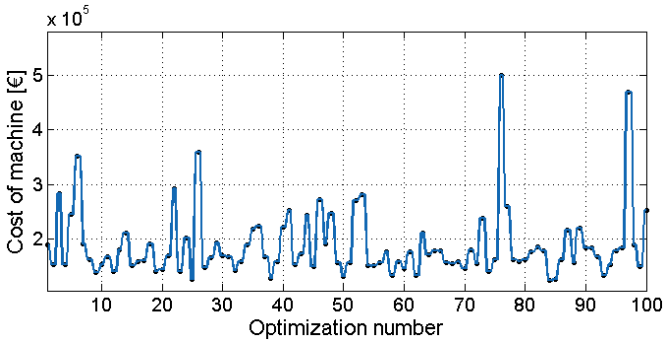

Fig. 17. Plot of GA with crossover fraction equal to 0.8 and 25 individuals

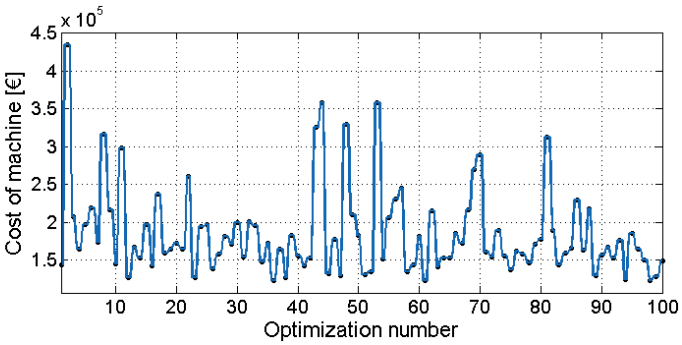

Fig. 18. Plot of GA with crossover fraction equal to 0.6 and 25 individuals

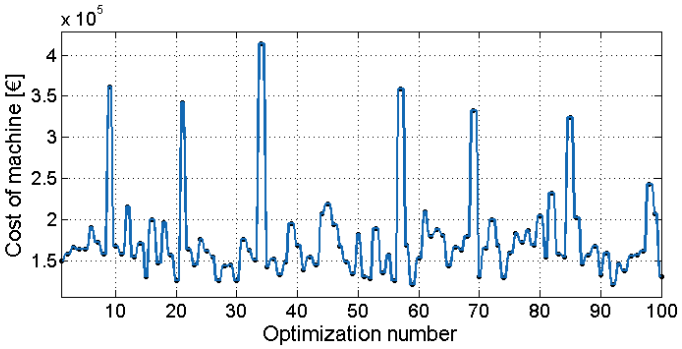

Fig. 19. Plot of GA with crossover fraction equal to 0.4 and 25 individuals

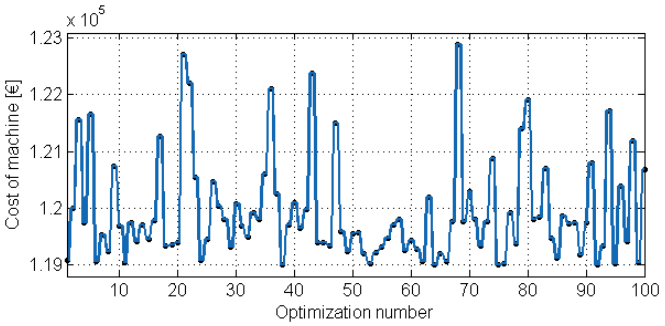

Fig. 20. Plot of Hybrid GA with 25 individuals

Optimizing with 25 particles and a crossover fraction of 40 %, see Figure 19, is compared to the hybrid version in Figure 20. The comparison is displayed in Figure 21 and shows that hybrid GA gives a mean machine cost that is 31.2 % lower than standard GA. Variance is 99.97 % smaller for hybrid GA compared to the initial results of the standard version.

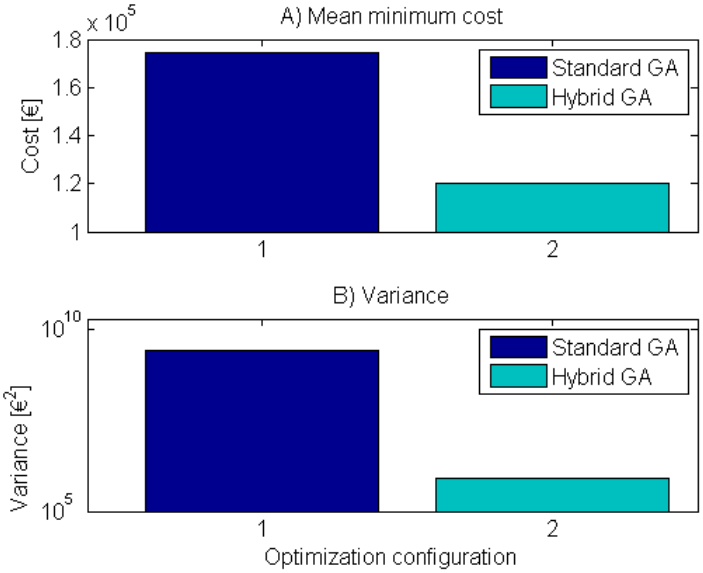

Fig. 21. Standard versus hybrid GA

In all optimizations presented to this point, the number of base windings has been set to three, based on the initial results from gradient optimization and PSO. A study of how sensitive the generator cost is to the number of base windings has been run. Results are presented in Figure 22, and one can see that generator cost is reduced substantially when the design is changed from one to two base windings. Further cost reduction is achieved when changing to four base windings as one can see from Figure 22-B. More than four base windings leads to a more expensive machine.

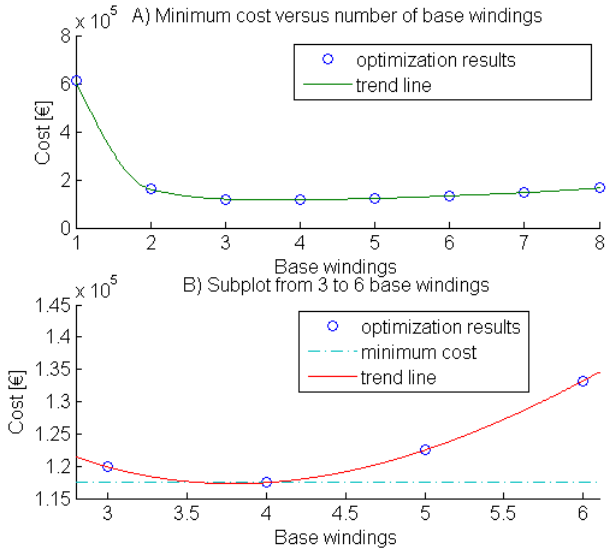

Fig. 22. Plot of mean minimum cost versus number of base windings

TABLE IX.
OPTIMIZING WITH PSO VS GA

	Mean:	Variance
Hybrid PSO	$1.2386 \cdot 10^5$	$1.5482 \cdot 10^8$
Hybrid GA	$1.1745 \cdot 10^5$	$2.1917 \cdot 10^6$

Hybrid PSO and hybrid GA with 25 individuals are compared in Table IX. Hybrid GA, with the number of base windings set to four, produced an average machine cost that is 5.2 % lower than the average generator from hybrid PSO. Results from hybrid GA have a variance that is 98.6 % lower than the results from hybrid PSO.

A final modification was made to the model and the GA in order to take advantage of the integer variable features that are implemented in GA by MATLAB. The number of base windings was allowed to vary between one and 13. In addition, the number of poles in the machine was allowed to vary between a given upper and lower boundary. Results from the last run of optimizations are presented in Figure 23. By opening these two variables for optimization, the mean result from the hybrid GA was reduced to 101 660 €, which corresponds to a 13.4 % reduction. The variance increased to $1.1578 \cdot 10^8$ €^{2}. Constraints on the rest of the machine parameters were kept the same.

The optimal design proposal presented in Table VIII has a harmonic spectrum that is presented in Figure 24 with a generator cost of 115 889 €. Optimization with number of poles and number of base windings as integer variables came up with a design proposal presented in Table X that costs 91 200 €. This machine has 84 poles and 96 slots while the former has 88 poles and 96 slots. The harmonic spectrum of the new design proposal is presented in Figure 25. There is no visible difference between the harmonic spectrum in Figure 25 and Figure 24, only that the absolute amplitude of the main

harmonic is 1821 for the 84 pole machine and 1434 in the 88 pole machine.

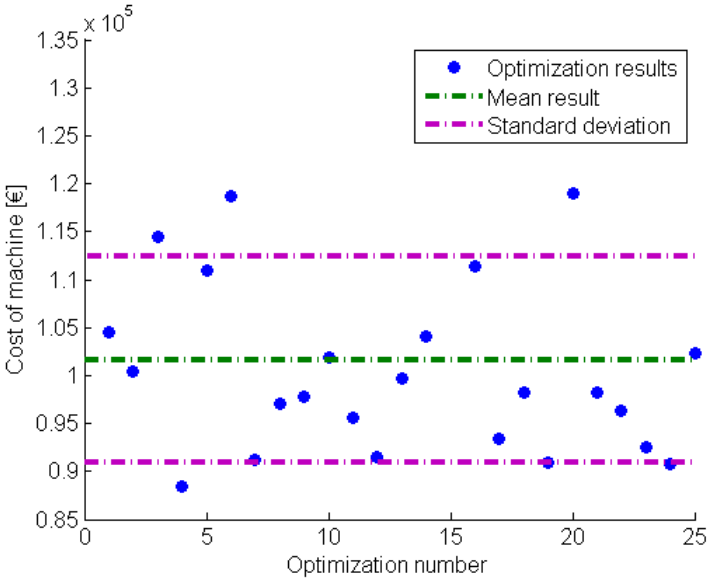

Fig. 23. Optimizing with number of poles and number of base windings as integer variables

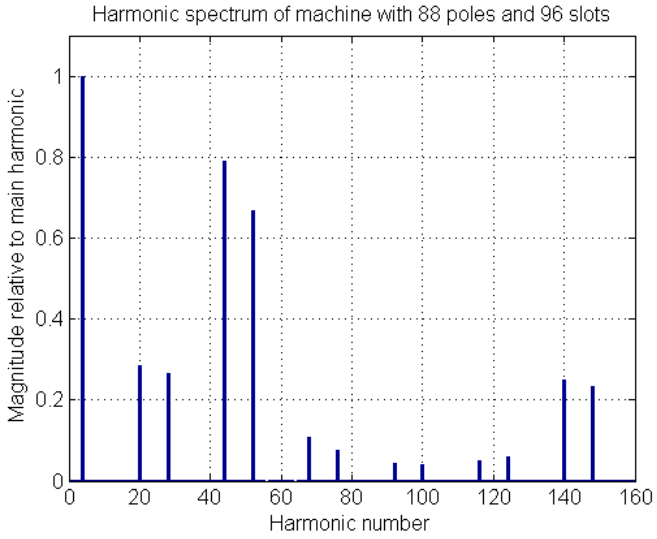

Fig. 24. Harmonic spectrum of machine with 88 poles and 96 slots

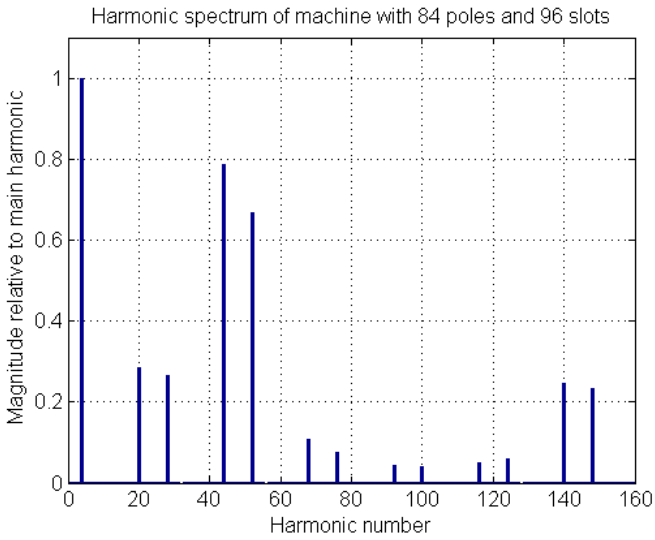

Fig. 25. Harmonic spectrum of machine with 84 poles and 96 slots

TABLE X.
DESIGN PROPOSAL FOR GENERATOR WITH 84 POLES AND 96 STATOR SLOTS

	Cost	91 201 €
	Total weight	10889 kg
	Independent variables:	
D_{outer}	Outer diameter of stator lamination	3.5 m
$d_{s,yoke}$	Thickness of stator yoke	29 mm
d_{slot}	slot depth	61 mm
w_{slot}	slot width	32 mm
l_m	length of magnets	7 mm
α_m	Magnet width / pole pitch ($\frac{w_m}{\tau_p}$)	0.9701
$d_{r,yoke}$	Thickness of rotor yoke	41 mm
J	Current density	3.11 A/mm^2
n_{base}	Number of base windings	4
	Machine parameters:	
l	Length	0.686 m
η	Efficiency	97 %
pf	Power factor	0.85
	Current loading	26682 A/m
w_{Cu}	Copper weight	738 kg
w_{magn}	Magnet weight	374 kg
w_{Fe}	Iron weight	6318 kg

C. Computational efficiency

One key parameter for comparing different optimization algorithms is computational efficiency, e.g. the amount of resources that is needed to perform each optimization. Time is chosen as a measurement on resource usage in this paper. Different configurations of each optimization method is compared and the algorithms are compared to each other. The effect of introducing parallel computing solution on multi-core machines is investigated.

Computation times for the PSO algorithm presented in Figure 26 indicates that the PSO as implemented for this paper needs from 67 minutes to 18.5 hours with populations that span from

25 to 400 particles. 12 processor cores are used in Figure 26. The number of iterations does not seem to have any significant weight on the computation time.

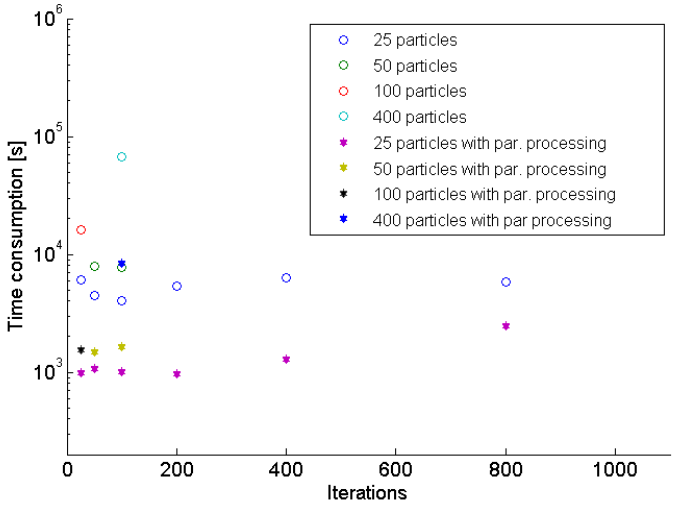

Fig. 26. Time consumption for Particle Swarm Optimization

Computation times recorded from running the GA are in the range from 4 minutes to 12.2 hours with populations that spans from 25 to 2500 individuals. Genetic Algorithms have been measured to be 16.75 times faster than Particle Swarm Optimization when both algorithms operated with a population of 25 individuals. GA needed 34 % less computation time with 2500 individuals than what PSO needed to optimize with 400 particles.

TABLE XI.
AVERAGE COMPUTATION TIME STANDARD PSO

25 particles, 25 iterations	779 s
25 particles, 200 iterations	800 s
100 particles, 25 iterations	2026 s

Parallel processing with 12 cores was tested on different configurations of PSO and presented in Table XI. Mean run time for 25 particles was reduced to around one seventh of the run time for the single-core processor. When optimizing with 100 particles, the average time needed for the optimization was reduced to one eight of the single-core case. This corresponds to a 3.9 hour reduction in computation time.

TABLE XII.
AVERAGE COMPUTATION TIME OF PSO VARIATIONS

Standard PSO	800 s
Improved PSO	1052 s
Centroid PSO	1323 s
Hybrid PSO	1231 s

Data from optimizing with different versions of Particle Swarm Optimization presented in Table XII indicate that the standard version is faster than the other three that was tested. It seems to be 24 % faster than the improved version, 40 % faster than Centroid PSO and 35 % faster than the hybrid version.

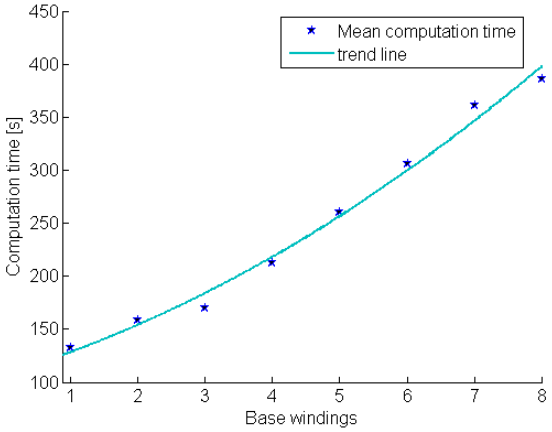

Fig. 27. Mean computation time for hybrid GA with different number of base windings

TABLE XIII.
AVERAGE COMPUTATION TIME OF GA VARIATIONS

Standard GA, 25 individuals	79 s
Standard GA, 250 individuals	408 s
Hybrid GA, 25 individuals	213 s

Running the optimization on 12 cores with GA yielded, according to Table XIII, a computation time that is three times faster than with one processor core for problems with 25 individuals. Parallel processing with 250 individuals on 12 cores was 6.3 times faster than with one core. Hybrid GA increased the computation time with 134 seconds for problems with 25 individuals on the same 12 core computer.

Figure 27 indicates that computation time for the GA is proportional to the number of base windings. Optimizing with few base windings seems to be faster than optimization with more base windings. Table XIII and Table XII presents a 17 minute difference in run time for the hybrid GA versus hybrid PSO, with the hybrid GA being 5.8 times faster.

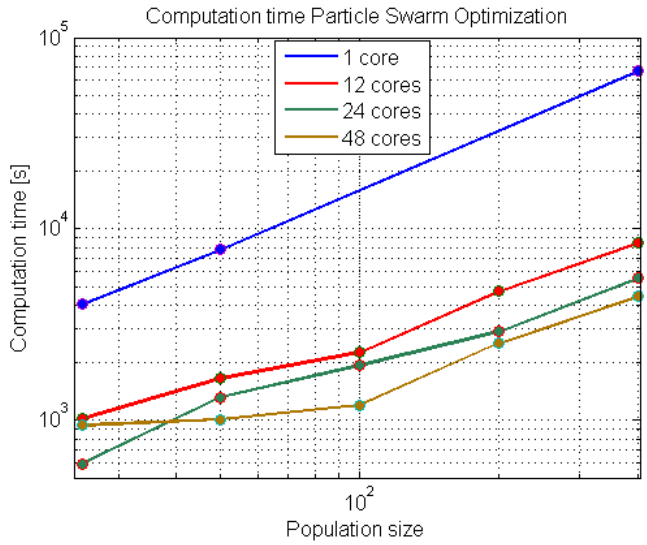

Fig. 28. Parallel processing with varying number of cores

Parallel processing seems to be very useful in reducing the time it takes to run stochastic optimization algorithms. Figure 28 presents the benefit that can be gained by using 12, 24 and 48 processor cores instead of one for PSO.

The largest case that has been run with PSO is 400 particles and 100 iterations. Moving from 12 to 24 cores reduced the time consumption 34 %. Doubling the number of processor cores to 48 gave an additional 19.8 % reduction. Running on 48 cores was 15 times faster than running on one. The smaller 25 particle population problem was made 6.8 times faster with 24 parallels which amounts to a reduction from 67 minutes to 10. The run time for the larger case went from 18.5 hours to 74 minutes.

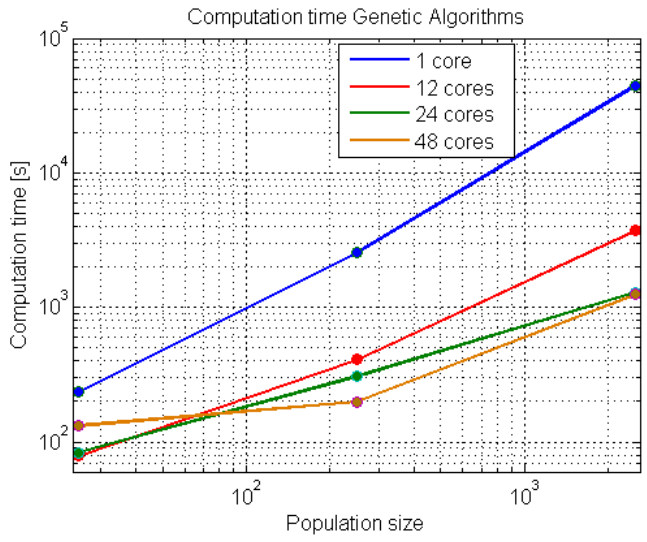

Fig. 29. Parallel processing with varying number of cores

The results from optimizing with GA on multi-core machines are displayed in Figure 29. Large problem formulations may become up to 35 times faster with 48 cores. This was the case when a population of 2500 individuals was optimized. Computing this case 35 times faster reduced the computation time from more than 12 hours to 21 minutes.

Figures 28 and 29 indicates that 48 cores are slower than 24 cores for PSO populations with less than 40 particles, and slower than both 24 and 12 cores for GA populations smaller than 70 individuals.

VI. Discussion of Results

A. Particle Swarm Optimization versus Genetic Algorithms

Both PSO and GA are able to find the optimal region of the design space. They managed to find the same optimal solution with the aid of gradient optimization, see Tables VII and VIII. A standard population size of 25 was selected for comparing the algorithms.

PSO with 25 particles and 200 iterations gave an average generator cost of 215 460 € over 100 optimization runs. The corresponding GA configuration with 25 individuals, a crossover fraction of 40 % and three base windings gave a mean cost of 174 330 €. This means that GA is able to find a machine design that is 19.1 % cheaper on average for a population size of 25. PSO results had a variance of $5.78 \cdot 10^8$ € ^{2} while the variance of the GA optimization was $2.78 \cdot 10^9$ € ^{2}. This is a 380 % increase from PSO to GA.

PSO with 25 particles and 200 iterations gives a mean value of the cost that is 0.3 % higher than for the case with 25 iterations. 200 iterations instead of 25 iterations gave a 1.15 % reduction in the variance of the results.

Two modifications to the PSO was tested, but they only resulted in higher mean results and more variance. Mean values for the modified PSO algorithms were 0.4 % and 3.45 % higher than for the standard version. Variance for the optimization results increased with 355 % and 143 % respectively. Based on these results, standard PSO seems to be a better optimization algorithm for models of electrical machines.

Increasing the population size for the standard version of PSO from 25 to 100 particles while keeping the number of iterations at 25 yielded better results. Mean generator cost was lowered 11.9 % to 189 860 € and variance went down 18.2 % to $4.73 \cdot 10^8$ € ^{2}. This indicates that larger population sizes gives better convergence towards the global minimum.

Anne-Siri Borander has used a modified version of the PSO from [3] in her Master's thesis [38]. PSO was used to optimize the height and width of permanent magnets in an electrical machine using Finite Element Analysis (FEA) in order to maximize the torque of the machine.

Results presented by Anne-Siri Borander [38] indicates that more particles may be more efficient than more iterations in order to increase convergence. These findings reinforce the

results already discussed in this section and contradicts the initial findings made in [3] where data indicated that increasing the number of iterations were more efficient than increasing the population size.

Population size was increased to 250 for the GA while the other parameters remained unchanged. Average generator cost went down 18.2 % to 142 610 € as a result of this. Variance decreased 41 % to $1.64 \cdot 10^9$ € ^{2}.

Stochastic optimization algorithms like PSO does not generally guarantee convergence to the global optimum [23], [39]. The problem of premature convergence, that the population converges too fast to a suboptimum, is a known challenge for the GA as well [27].

All data from standard PSO and GA points to premature convergence, as neither method is capable of reaching the same results as gradient optimization managed to do in [17].

B. Hybrid optimization methods

Convergence of gradient optimization techniques are generally good [14], while convergence towards local optima is one of the main challenges of these methods. Global search with PSO and GA are used as a first stage of the hybrid optimization that has been tested in this paper. The Interior Point Algorithm in MATLAB was added as a final stage in the optimization process on both PSO and GA.

Changing GA into a hybrid version with gradient end-optimization, made the mean output value from the optimization 31.2 % lower. At the same time the variance was reduced to one 3400th of the initial value from the standard GA. These results indicates that hybrid GA may be a good and reliable optimization tool for problems involving electrical machine design.

Hybrid GA managed to reach an average generator cost of 117 450 €. This was 5.2 % less than the mean value of 123 860 € that was reached by the hybrid PSO. Variance with hybrid GA was 70.6 times smaller than the variance of hybrid PSO. At the end of the gradient step of the hybrid optimization, the variance of the PSO and GA are down to $1.55 \cdot 10^8$ € ^{2} and $2.19 \cdot 10^6$ € ^{2} respectively. Hybrid GA gives better convergence towards the optimum solution with a variance that is several orders of magnitude lower than the hybrid PSO. It seems that hybrid GA is a better optimization algorithm than hybrid PSO for machine design problems.

The inclusion of a gradient step to the PSO reduced the mean value of the generator cost with 42.5 % and the variance with 73.2 %. Hybrid PSO has a convergence that seems to make it a viable method for design of electrical machines. However, in comparison to the results from hybrid GA, the mean resulting value of the objective function and the variance are higher. This makes the hybrid GA a more suitable method.

GA, PSO and hybrid versions of both have been used to optimize an induction machine in [10]. In [10], the objective function was reduced 19.6 % using a hybrid GA and 23.6 % using a hybrid PSO. These results stand in contrast to what has been discussed in this paper. One reason why the PSO found design proposals that were worse than the proposals GA found may be that GA is implemented by MATLAB by professional programmers, while PSO has been implemented by the author.

The significant improvements in both convergence and accuracy that has been reached, by combining stochastic techniques with gradient methods, are in line with other findings from studies done on benchmark cases for both GA [13] and PSO [14].

Integer variable optimization was allowed for the GA. This was done by allowing the number of base windings and the number of poles in the generator model to be optimized. Expanding the number of independent variables to 10 increased the feasible region and reduced the average value of the objective function to 101 660 €. This is down 42.7 % from the initial value from the standard GA. Allowing the number of poles to be optimized as an independent variable made no visible change to the harmonic spectrum.

C. Parallel processing solutions

Parallel processing of PSO and GA has reduced the computation time from hours to minutes. Optimizing on computers with up to 48 processor cores has made it possible to perform a quantitative analysis that would otherwise have been infeasible due to the high demand for computational resources.

Time consumption was reduced from 12 hours to 21 minutes for a run of GA with 2500 individuals using 48 processor cores. Using the 48 core computer on a PSO problem with 400 particles and 100 iterations reduced the computation time from 18.5 hours to 74 minutes compared to running it on a single-core computer. Similar results were obtained in [38] where computation time was reduced from 13 hours to less than one hour using 25 processor cores.

Series of 100 optimizations on each parameter configuration was run on a 12 core computer. Standard PSO with 25 particles and 200 iterations used on average 800 seconds per run. GA without gradient optimization used 79 seconds on average for populations of 25 individuals.

Hybrid PSO and hybrid GA used 1231 seconds and 213 seconds respectively. This implies that hybrid GA is 5.8 times faster, on average, than hybrid PSO for the given configurations.

Another promising parallel processing tool has been tested by Anne-Siri Borander in [38]. The method called Techila Distributed Computing uses idle computer processors connected in a cloud solution to process the optimization algorithm [37]. This seems to be a good solution in cases where large multi-core computers are not available.

Optimizing the FEA problems in [38] using PSO yielded computation times that was up to 25 times faster with Techila

than on single-processor machines. Reductions in run time were in line with what was observed using the 48 core machine. Techila seems to be a good solution for cases with many particles and few iterations.

Parallel processing tools have proven efficient at reducing the need for computational resources for performing stochastic optimization. An obvious next step to the refinement of the process of designing electrical machines is to combine FEA and algebraic models of electrical machines to gain more accurate results. Expanding the number of independent variables to include winding layout as presented in [29] and [30] should be explored in combination with parallel processing in order to make it feasible. Techila Distributed Computing and other cloud solutions should be tested more extensively.

VII. Conclusions

Tidal energy extraction has the potential to become an important source of renewable electricity production. For tidal energy to become competitive with traditional sources of electric energy the cost per kWh must be reduced. One way of achieving this is to minimize the cost of the generator.

Genetic Algorithms (GA) have proven to be more useful than Particle Swarm Optimization (PSO) in design of electrical machines. A permanent magnet synchronous generator was optimized on cost. GA managed, on average, to find design proposals that yielded 19.1 % cheaper machines than PSO with populations of 25 individuals. Both PSO and GA performed better with increased population sizes.

Stochastic algorithms like PSO and GA are in general good at finding the optimal region of the design space. They are, however, poor instruments for finding the optimal point on their own. Hybrid optimization, where gradient methods are included as a final step in the optimization should be used to overcome the phenomenon of premature convergence in stochastic algorithms.

Hybrid GA managed to converge to an average cost of the generator that is 5.2 % lower than what was reached by the hybrid PSO. Optimization results show a variance that is 98.6 % lower for hybrid GA than it is for hybrid PSO. Moving from a pure GA optimization to the hybrid version resulted in a 31.2 % reduction in the average cost and a variance that was 99.97 % lower. Including the gradient step in PSO lowered the mean generator cost 42.5 %. Variance became 73.2 % smaller.

Parallel computing on multi-core computers has proven very useful for optimization and should be utilized as much as possible. Parallel processing features are able to reduce the computation time of each optimization up to 97 % for large problems. The run time for GA with 2500 individuals was reduced 35 times from 12 hours to 21 minutes when changing from a single processor computer to a computer with 48 processor cores. The run time for PSO with 400 particles and 100 iterations went from 18.5 hours to 74 minutes, a 93 % reduction.

VIII. Further work

The next step of the process of designing electrical machines is to combine Finite Element Analysis (FEA) and algebraic models of electrical machines to gain more accurate results. More exact loss calculations should be made by integrating FEA-models of the generator. This could be done using programs like COMSOL Multiphysics.

One possible point of departure for any one wishing to continue this work could be to combine the work done in this paper with the work presented in the master's thesis of Anne-Siri Borander [38].

Expanding the number of independent variables to include winding layout as presented in [29] should be explored in combination with parallel processing in order to make it feasible. Including the stator winding layout in the optimization process could produce better design proposals.

Acknowledgment

I would like to thank my supervisors Professor Robert Nilssen and PhD candidate Astrid Røkke at the Department of Electric Power Engineering for help and guidance during the work on this thesis. I am grateful that Astrid Røkke allowed me to use her generator model for optimization.

Special thanks goes to my friends at the office; Anne-Siri, Idun, Martin and Thomas for giving me motivation and a lot of fun during the work on this thesis. Creative discussions at the office helped improve my thesis.

REFERENCES

[1] S.E. Ben Elghali and M.E.H. Benbouzid and J.F. Charpentier, "Marine Tidal Current Electric Power Generation Technology: State of the Art and Current Status," in *Electric Machines Drives Conference, 2007. IEMDC '07. IEEE International*, vol. 2, May 2007, pp. 1407–1412.

[2] A. Røkke and R. Nilssen, "Marine Current Turbines and Generator preference. A technology review," *Renewable Energy and Power Quality Journal (RE&PQJ)*, no. 11, Mar 2013.

[3] E. L. Engevik, "Comparative study of stochastic optimization techniques for electrical machine design," *Department of Electric Power Engineering, Norwegian University of Science and Technology*, dec 2013.

[4] O. Andersen, "Optimum design of electrical machines," *Doctoral dissertation, Chalmers University of Technology - Gothenburg*, 1969.

[5] A. Fanni, A. Manunza, M. Marchesi, and F. Pilo, "Tabu Search metaheuristics for global optimization of electromagnetic problems," *Magnetics, IEEE Transactions on*, vol. 34, no. 5, pp. 2960–2963, Sep 1998.

[6] D. Fodorean and L. Idoumghar and A. N'Diaye and D. Bouquain and A. Miraoui, "Simulated annealing algorithm for the optimisation of an electrical machine," *Electric Power Applications, IET*, vol. 6, no. 9, pp. 735–742, Nov 2012.

[7] R. Wrobel and P. Mellor, "The use of a genetic algorithm in the design optimisation. of a brushless dc permanent magnet machine rotor," *(PEMD 2004). Second International Conference on Power Electronics, Machines and Drives, 2004*, vol. 2, pp. 823 – 827, mar. / apr. 2004.

[8] A.D. Lilla and M.A. Khan and P. Barendse, "Comparison of Differential Evolution and Genetic Algorithm in the design of permanent magnet Generators," in *Industrial Technology (ICIT), 2013 IEEE International Conference on*, Feb 2013, pp. 266–271.

[9] M. van der Geest, H. Polinder, J. Ferreira, and D. Zeilstra, "Optimization and comparison of electrical machines using particle swarm optimization," *ICEM. 2012 XXth International Conference on Electrical Machines*, pp. 1380 – 1386, Sept 2012.

[10] S. Łacheciński and M. Dems, "Optimization of big power low voltage induction motor using hybrid optimization algorithm," in *Proceedings of the 2008 International Conference on Electrical Machines ICEM*, 2008.

[11] Y. Ahn and J. Park and C.-G. Lee and J. Kim and S. Jung, "Novel Memetic Algorithm implemented With GA (Genetic Algorithm) and MADS (Mesh Adaptive Direct Search) for Optimal Design of Electromagnetic System," *Magnetics, IEEE Transactions on*, vol. 46, no. 6, pp. 1982–1985, Jun 2010.

[12] V. Kelner, F. Capitanescu, O. Lonard, and L. Wehenkel, "A hybrid optimization technique coupling an evolutionary and a local search algorithm," *Journal of Computational and Applied Mathematics*, vol. 215, no. 2, pp. 448 – 456, 2008, proceedings of the Third International Conference on Advanced Computational Methods in Engineering (ACOMEN 2005). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042706007552

[13] I. Oh and J. Lee and B. Moon, "Hybrid genetic algorithms for feature selection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 11, pp. 1424–1437, 2004.

[14] M. M. Noel, "A new gradient based particle swarm optimization algorithm for accurate computation of global minimum," *Applied Soft Computing*, vol. 12, no. 1, pp. 353 – 359, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494611003206

[15] V. Plevris and M. Papadrakakis, "A hybrid particle swarmgradient algorithm for global structural optimization," *Computer-Aided Civil and Infrastructure Engineering*, vol. 26, no. 1, pp. 48–68, 2011.

[16] F. Hillier and G. Lieberman, *Introduction to Operations Research*, 9th ed. McGraw-Hill, 2010.

[17] A. Røkke, "Gradient based optimization of permanent magnet generator design," *Department of Electric Power Engineering, Norwegian University of Science and Technology*, 2014.

[18] R. Eberhart and J. Kennedy, "A New Optimizer Using Particle Swarm Theory," *MHS '95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39 – 43, 1995.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," IEEE International Conference on Neural Networks, 1995. Proceedings. vol. 4, pp. 1942 - 1948, 1995.

[20] J. Robinson and Y. Rahmat-Samii, "Particle swarm optimization in electromagnetics," *IEEE Transactions on Antennas and Propagation*, vol. 52, no. 2, pp. 397 – 407, feb. 2004.

[21] M. Balaji and V. Kamaraj, "Particle swarm optimization approach for optimal design of switched reluctance machine," *American Journal of Applied Sciences*, vol. 8, no. 4, pp. 374–381, apr. 2011.

[22] C. Mei, G. Liu, and X. Xiao, "Improved particle swarm optimization algorithm and its global convergence analysis," in *Control and Decision Conference (CCDC), 2010 Chinese*, May 2010, pp. 1662–1667.

[23] H. Y. H. Y. Luo and Z. Li, "Convergence analysis of particle swarm optimizer and its improved algorithm based on velocity differential evolution," *Intell. Neuroscience*, vol. 2013, pp. 4:4–4:4, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1155/2013/384125

[24] J. He and H. Guo, "A modified particle swarm optimization algorithm," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 11, no. 10, pp. 6209–6215, 2013. [Online]. Available: http://iaesjournal.com/online/index.php/TELKOMNIKA/article/view/2947

[25] S. Liang, S. Song, L. Kong, and J. Cheng, "An improved particle swarm optimization algorithm and its convergence analysis," in *Computer Modeling and Simulation, 2010. ICCMS '10. Second International Conference on*, vol. 2, Jan 2010, pp. 138–141.

[26] E. Mezura-Montes and J. Flores-Mendoza, "Improved particle swarm optimization in constrained numerical search spaces," in *Nature-Inspired Algorithms for Optimisation*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2009, vol. 193, pp. 299–332. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-00267-0_11

[27] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Springer-Verlag, 1996.

[28] B. Miller and D. Goldberg, "Genetic Algorithms, Tournament Selection, and the Effects of Noise," *Complex Systems*, 1995.

[29] T. Nordaunet, "Reducing sub-harmonic spatial fields in concentrated winding machines using genetic algorithms," Master's thesis, Department of Electric Power Engineering, Norwegian University of Science and Technology, 2013.

[30] D. Delgado, "Automated ac winding design," Master's thesis, The University of Manchester - School of Electrical and Electroni Engineering, 2009.

[31] J. Pirhönen, T. Jokinen, and V. Hrabocová, *Design of Rotating Electrical Machines*, 1st ed. John Wiley & Sons, 2008.

[32] D. Hanselmann, *Brushless Permanent-Magnet Motor Design*. McGraw-Hill, 1994.

[33] MathWorks. (2013) MATLAB - Global Optimization Toolbox - User's Guide - R2013b. http://www.mathworks.com/help/pdf_doc/gads/gads_tb.pdf. MathWorks. [Online]. Available: http://www.mathworks.com/help/pdf_doc/gads/gads_tb.pdf

[34] ——, "MATLAB - Optimization Toolbox - User's Guide - R1013b," http://www.mathworks.com/help/pdf_doc/optim/optim_tb.pdf, nov 2013.

[35] E. Rasmussen. (2002) Parallel processing. Computer Sciences. Retrieved June 05, 2014 from Encyclopedia.com.

[36] MathWorks, "MATLAB - Parallel Computing Toolbox - User's Guide - R2013b," http://www.mathworks.com/help/pdf_doc/distcomp/distcomp.pdf, des 2013.

[37] J. Floan, "Techila Distributed Computing Solution - Norwegian University of Science and Technology - Presentation," https://www.hpc.ntnu.no/display/hpc/Techila+Distributed+Computing+Solution, nov. 2013.

[38] A.-S. Borander, "Parallel processing of optimization algorithms," Master's thesis, Department of Electric Power Engineering, Norwegian University of Science and Technology, 2014.

[39] D. ping Tian, "A Review of Convergence Analysis of Particle Swarm Optimization," *International Journal of Grid and Distributed Computing*, vol. 6, no. 6, pp. 117–128, 2013.

APPENDIX A
MODIFICATIONS TO THE PARTICLE SWARM OPTIMIZATION ALGORITHM

A. Improved PSO

This section presents the details of the modification to Particle Swarm Optimization called Improved PSO in the paper *Improved Particle Swarm Optimization in Constrained Numerical Search Spaces* [26].

TABLE XIV.
IMPROVED-PSO PARAMETER VALUES

Δt	1
c_1	2.7
c_2^0	2.5
w	1
k	0.729

The original velocity and position calculations are replaced with Eqs. 13 and 14.

$$v_n = k \cdot [v_n + c_1 \cdot r() \cdot (x_n^{pb} - x_n) + c_2 \cdot r() \cdot (x_n^{gb} - x_n)] \quad (13)$$

$$x_n = x_n + v_n \quad (14)$$

Factors k and c_2 are varied based on the iteration counter as listed in Eqs. 16 and 17.

$$y = \frac{i}{i_{max}} \quad (15)$$

$$k^{t+1} = k \cdot y^4 \quad (16)$$

$$c_2^{t+1} = c_2 \cdot y^4 \quad (17)$$

For each iteration i a fraction p given by Eq. 18, of the population will use the initial values of k and c_2. The rest of the population will use the dynamic values given by Eqs. 16 and 17.

$$p = k + \frac{\sin(4\pi y)}{10.3} \quad (18)$$

B. Centroid PSO

Another modification to the PSO that has been tested is what is called Particle Swarm Optimization with Centroid (CPSO) [25]. This version extends the traditional PSO algorithm by introducing the geometric centre, or centroid, of the swarm population as one of the points that each particle is drawn towards.

TABLE XV.
CENTROID-PSO PARAMETER VALUES

Δt	1
c_1	1.4
c_2	1.4
c_3	1.4
w	$\in [0.4, 0.9]$
α	0.5

Eqs. 19 and 20 gives the expressions for the centroid of the swarm and for the centroid of personal best locations of the swarm respectively.

$$x_n^c = \frac{1}{M} \sum_{i=1}^{M} x_{in} \quad (19)$$

$$x_n^{pbc} = \frac{1}{M} \sum_{i=1}^{M} x_{in}^{pb} \quad (20)$$

The modified velocity function is given in Eq. 21. Position is calculated as for the IPSO.

$$\begin{aligned} v_n = w \cdot v_n + c_1 \cdot r() \cdot (x_n^{pb} - x_n) + c_2 \cdot r() \cdot (x_n^{gb} - x_n) \\ + c_3 \cdot rand \cdot (\alpha \cdot x_n^c + (1 - \alpha) \cdot x_n^{pbc} - x_n) \end{aligned} \quad (21)$$

APPENDIX B
SETUP OF GENETIC ALGORITHMS IN MATLAB

In this section, a detailed description is given for how to use the Genetic Algorithms in MATLAB.

Boundaries for independent variables are defined as vectors LB and UB for upper and lower bounds. These are included in the settings for GA optimization using the **'PopInitRange'** command. LB and UB must be included in the function call to the GA as presented at the bottom of the code presented in Figure 30.

Population size and number of generations are set using the **'Generations'** and **'PopulationSize'** command in the code. Each set of parent individuals is selected through tournament selection. This procedure was chosen based on the results presented in [3].

Constraints are included in the optimization process through the call to 'iTool_con' which returns the constraint vector from the machine model called iTool. iTool is developed by Astrid Røkke, PhD candidate at the Department of Electric Power Engineering, and is therefore not included in the appendices of this paper.

```
% Performing the optimisation using the built-in
    function ga in matlab
options = gaoptimset('FitnessScalingFcn',{
    @fitscalingrank }, ...
    'PopInitRange',[LB ; UB],...% Decides the range
        of variables for the initial population
    'StallGenL',10,...
    'Generations',30, ... %Sets the number of
        generations
    'PopulationSize',25,...% Sets the population
        size
    'EliteCount',2, ...    %Number of individes that
         goes directly to next generation with out
        crossover and mutation
    'CrossoverFraction',0.4,...   % Sets the
        percentage of next generation that is
        created by crossover, e.g. 40%
    'UseParallel','always',...    %Enables parallel
        computing
    'MigrationInterval',10,...
    'MigrationFraction',0.1,...
    'MigrationDirection','both', ...
    'PopulationType','doubleVector',...
    'SelectionFcn', {  @selectiontournament [] });
        ... %Selects the selection function

 %Setup for use of the function ga for continuous
     variables
[X fval exitflag output population scores] = ga(
    @objective_fun,nVar,[],[],[],[],LB,UB,
    @constraint_fun,options);

%Setup for use of function ga for integer variables
intcons = [9,11]; % Example for when variable 9 and
    11 are integers

[X fval exitflag output population scores] = ga(
    @objective_fun,nVar,[],[],[],[],LB,UB,
    @constraint_fun,intcons,options);
```

Fig. 30. Setup of Genetic Algorithms in MATLAB

The crossover fraction and the number of elite individuals are set by **'EliteCount'** and **'CrossoverFraction'** in Figure 30. Setting **'UseParallel'** to **'always'** enables the use of built-in parallel processing features in the Genetic Algorithms.

Genetic Algorithms as implemented in the Global Optimization toolbox [33] allows for integer variables in the problem configuration. Integer GA uses a special creation-, crossover- and mutation function [33]. It tries to minimize a penalty function instead of the fitness function. In addition it uses a special binary tournament selection procedure.

The implication of this is that the line specifying the selection function, named **'SelectionFcn'**, must be removed. A vector **intcons** must be included in the call to the GA function as presented in figure 30.

APPENDIX C
MATLAB SCRIPTS

A. *Setup of PSO in Matlab*

```
function object = PSO_setup

%%Defining upper and lower boundaries and startpoint
     for all variables
% Variables:
% x(1) Outer diameter of active material[m]
% x(2) stator yoke thickness [m]
% x(3) slot depth [m]
% x(4) slot width [m]
% x(5) magnet length (in magnetisation direction)[m]
% x(6) magnet width/pole pitch
% x(7) rotor yoke thickness [m]
% x(8) current density [A/mm^2]
% x(9) number of base windings

UB_1 = 3.5;
LB_1 = 3;
UB_2 = 60e-3*10;
LB_2 = 15e-3*10;
UB_3 = 75e-3*10;
LB_3 = 20e-3*10;
UB_4 = 50e-3*10;
LB_4 = 15e-3*10;
UB_5 = 30e-3*10;
LB_5 = 2e-3*10;
UB_6 = 0.98;
LB_6 = 0.8;
UB_7 = 60e-3*10;
LB_7 = 10e-3*10;
UB_8 = 4;
LB_8 = 1.5;
UB_9 = 13;
LB_9 = 1;
LB_10 = 0.9;
UB_10 = 1.0;

efficiency=0.97;
popSize = 25;
imax = 200;

LB = [LB_1 LB_2 LB_3 LB_4 LB_5 LB_6 LB_7 LB_8 LB_9
    efficiency];
UB = [UB_1 UB_2 UB_3 UB_4 UB_5 UB_6 UB_7 UB_8 UB_9
    efficiency];


%Call to standard version of PSO:
[xGb, gb ] = PSOfunction(popSize,imax, LB, UB);
%Call to improved version of PSO:
[xGb, gb ] = IPSOfunction(popSize,imax, LB, UB);
%Call to centroid version of PSO:
[xGb, gb ] = CPSOfunction(popSize,imax, LB, UB);

%Setup of gradient step when hybrid PSO is used:
 LB = [LB_1 LB_2 LB_3 LB_4 LB_5 LB_6 LB_7 LB_8 xGb
     (1,9) efficiency];
 UB = [UB_1 UB_2 UB_3 UB_4 UB_5 UB_6 UB_7 UB_8 xGb
     (1,9) efficiency];
      options = optimset('Algorithm','interior-point
          ','MaxFunEvals',1500,'TolCon',1e-10,'TolX'
          ,1e-12,'ObjectiveLimit',0); % run interior
          -point algorithm
      [X,fval,exitflag,output] = fmincon(@iTool_fun,
          xGb,[],[],[],[],LB,UB,@iTool_con,options);

object = fval;

end
```

B. Particle Swarm Optimization Algorithm

```
function [xGb,gb] = PSOfunction(popSize,imax,LB,UB)

    c1 = 2; %Governs how much a particel is pulled
            %towards the personal best position
    c2 = 2; %Governs how much a particel is pulled
            %towards the global best position
    dt = 0.01; %Governs how position is updated
    fitness = zeros(popSize,1);

    %The call to initialPop creates the
    %initial population.
    [x,values] = initialPop(popSize,LB,UB);
    count = zeros(popSize,1);
    fitness = values;

    %If all particles have feasible solutions
    %initially, then all future recordings
    %of personal best positions will be
    %feasible solutions.

    pb = fitness;
    xPb = x;
    v = zeros(popSize,length(LB));

    gb = fitness(1,1);
    xGb = x(1,:);


    for m = 2 : popSize
        if ((fitness(m,1) < gb) && (fitness(m,1) >
             0))
            gb = fitness(m,1);
            xGb = x(m,:);

        end
    end


    i = 1;
    while i < imax
       w = inertialWeight(i,imax);
       for j = 1 : popSize
           y = x(j,:);
           [f,c,ceq,ms]=iTool(y);
           count(j,1) = 0;
             for k = 1:length(c)
                if (c(k) <= 0)
                    count(j,1) = count(j,1) +1;
                end
             end

              if (count == length(c))
                fitness(i,1) = f;
              end
     %The following section compares the current
     %poition to the personal best of the
     %particle and the global best of
     %the swarm.
           if fitness(j,1) < pb(j,1)
               pb(j,1) = fitness(j,1);
               xPb(j,:) = x(j,:);
           end
           if fitness(j,1) < gb
               gb = fitness(j,1);
               xGb = x(j,:);

           end
       end

       %A particle that reaches the outer boundary
       %in one direction will have its velocity
       %component in that given direction
       %set to zero.
            for  k = 1 : popSize
                 v(k,:) = w*v(k,:) + c1*rand*(xPb(k
                      ,:) - x(k,:)) +  c2*rand*(xGb -
                       x(k,:));
                 x(k,:) = x(k,:) + dt*v(k,:);
                 for m = 1 : length(LB)
                     if (x(k,m) < LB(m))
                         x(k,m) = LB(m);
                         v(k,m) =  0;
                     elseif (x(k,m) > UB(m))
                         x(k,m) = UB(m);
                         v(k,m) = 0;
                     elseif (m == 9)
                         temp = x(k,m);
                         x(k,m) = round(temp);
                     end
                 end
            end
       i = i + 1;
    end
end
```

C. Improved Particle Swarm Optimization Algorithm

```
function [xGb,gb] = IPSOfunction(popSize,imax,LB,UB)

%CPSOfunction.m is a modification to PSOfunction
%based on what is presented in Appendix A.
    c1 = 2.7;
    c20 = 2.5;
    kon0 = 0.729;
    fitness = zeros(popSize,1);

    [x, values] = initialPop(popSize,LB,UB);
    count = zeros(popSize,1);
    fitness = values;

    pb = fitness;
    xPb = x;
    v = zeros(popSize,length(LB));

    gb = fitness(1,1);
    xGb = x(1,:);

    for m = 2 : popSize
        if ((fitness(m,1) < gb) && (fitness(m,1) >
             0))
            gb = fitness(m,1);
            xGb = x(m,:);

        end
    end

    i = 1;
    while i < imax
       for j = 1 : popSize
           y = x(j,:);
           [f,c,ceq,ms]=iTool(y);
           count(j,1) = 0;
             for k = 1:length(c)
                if (c(k) <= 0)
```

```
                    count(j,1) = count(j,1) +1;
                end
            end

             if (count == length(c))
               fitness(i,1) = f;
             end

          if fitness(j,1) < pb(j,1)
              pb(j,1) = fitness(j,1);
              xPb(j,:) = x(j,:);
          end
          if fitness(j,1) < gb
              gb = fitness(j,1);
              xGb = x(j,:);

          end
       end

y = i/imax;
kon = kon0*(y^4);
c2 = c20*(y^4);
p = kon0 + ((sin(4*pi*y))/10.3);

 for  k = 1 : popSize
     d = rand;
                if(d < p)
                 v(k,:) = kon0*(v(k,:) + c1*rand*(
                     xPb(k,:) - x(k,:)) +  c20*rand
                     *(xGb - x(k,:)));
                 x(k,:) = x(k,:) + v(k,:);
                 for m = 1 : length(LB)
                     if (x(k,m) < LB(m))
                         x(k,m) = LB(m);
                         v(k,m) =  0;
                     elseif (x(k,m) > UB(m))
                         x(k,m) = UB(m);
                         v(k,m) = 0;
                     elseif (m == 9)
                         temp = x(k,m);
                         x(k,m) = round(temp);
                     end
                 end
                else
                    v(k,:) = kon*(v(k,:) + c1*rand*(
                        xPb(k,:) - x(k,:)) +  c2*
                        rand*(xGb - x(k,:)));
                 x(k,:) = x(k,:) + v(k,:);
                 for m = 1 : length(LB)
                     if (x(k,m) < LB(m))
                         x(k,m) = LB(m);
                         v(k,m) =  0;
                     elseif (x(k,m) > UB(m))
                         x(k,m) = UB(m);
                         v(k,m) = 0;
                     elseif (m == 9)
                         temp = x(k,m);
                         x(k,m) = round(temp);
                     end
                 end
                end
 end
       i = i + 1;
    end
end
```

D. Particle Swarm Optimization with Centroid Algorithm

```
function [xGb,gb] = CPSOfunction(popSize,imax,LB,UB)

%CPSOfunction.m is a modification to PSOfunction
%based on what is presented in Appendix A.
    c1 = 1.4;
    c2 = 1.4;
    c3 = 1.4;
    alpha = 0.5;
    fitness = zeros(popSize,1);

    [x, values] = initialPop(popSize,LB,UB);
    count = zeros(popSize,1);
    fitness = values;

    pb = fitness;
    xPb = x;
    v = zeros(popSize,length(LB));

    gb = fitness(1,1);
    xGb = x(1,:);

    for m = 2 : popSize
        if ((fitness(m,1) < gb) && (fitness(m,1) >
            0))
            gb = fitness(m,1);
            xGb = x(m,:);

        end
    end

    i = 1;
    while i < imax
        xc = zeros(1,length(LB));
        xpc = zeros(1,length(LB));

        for d = 1 : length(LB)
            for f = 1 : popSize
                xc(1,d) = xc(1,d) + x(f,d);
                xpc(1,d) = xpc(1,d) + xPb(f,d);
            end
            xc(1,d) = xc(1,d)./popSize;
            xpc(1,d) = xpc(1,d)./popSize;
        end

       w = inertialWeight(i,imax);
       for j = 1 : popSize
           y = x(j,:);
           [f,c,ceq,ms]=iTool(y);
           count(j,1) = 0;
             for k = 1:length(c)
                if (c(k) <= 0)
                    count(j,1) = count(j,1) +1;
                end
             end

              if (count == length(c))
                fitness(i,1) = f;
              end

           if fitness(j,1) < pb(j,1)
               pb(j,1) = fitness(j,1);
               xPb(j,:) = x(j,:);
           end
           if fitness(j,1) < gb
               gb = fitness(j,1);
               xGb = x(j,:);

           end
       end
```

```
 for  k = 1 : popSize

                v(k,:) = w*v(k,:) + c1*rand*(xPb(k
                    ,:) - x(k,:)) +  c2*rand*(xGb -
                     x(k,:)) +c3*rand*(alpha*xc +
                    (1-alpha)*xpc - x(k,:));
                x(k,:) = x(k,:) + v(k,:);
                for m = 1 : length(LB)
                    if (x(k,m) < LB(m))
                        x(k,m) = LB(m);
                        v(k,m) =  0;
                    elseif (x(k,m) > UB(m))
                        x(k,m) = UB(m);
                        v(k,m) = 0;
                    elseif (m == 9)
                        temp = x(k,m);
                        x(k,m) = round(temp);
                    end
                end

 end
       i = i + 1;
    end
end
```

```
            ctemp = 0;
            for k = 1:length(c)
                if (c(k) <= 0)
                    ctemp = ctemp + 1;
                end
            end
            count = ctemp;

            if (count == length(c))
                values_temp(1,1,i) = f;
                x_temp(:,:,i) = b(:,:,i);
            end
        end


    end



     for m = 1:popSize
         x_sum(m,:) = x_temp(:,:,m);
         values(m,1) = values_temp(1,1,m);
     end

end
```

E. Initialization of PSO

```
function [x_sum, values] = initialPop(popSize,LB,UB)

    %parpool('local')

    values_temp = zeros(1,1,popSize);
    values = zeros(popSize,1);
    x_temp = zeros(1,length(LB),popSize);
    x_sum = zeros(popSize,length(LB));
    b = zeros(1,length(LB),popSize);


    parfor i = 1 : popSize


         count = 0;
         while(count < 16) %Number of constraints =
             16 at this point.
            v = zeros(1,length(LB));
        for j = 2 : (length(LB)+1)
            if (j-1) == (9)
                v(j-1) = randi([LB(j-1),UB(j-1)],1);
            elseif j == 10
                v(j-1) = LB(j-1);
            else
               v(j-1) = rand*UB(j-1);
                if (LB(j-1) == UB(j-1))
                    v(j-1) = LB(j-1);
                elseif (v(j-1) < LB(j-1))
                    v(j-1) = LB(j-1);
                end
            end
        end

        b(:,:,i) = v;


%     count = 0;
         y = b(:,:,i);
            [f,c,ceq,ms]=iTool(y);
```

F. Calculation of inertial weight

```
function w = inertialWeight(i,imax)
    wmax = 0.9;
    wmin = 0.4;
    w = wmax - ((wmax-wmin)/imax)*i;
end
```