EMPIRICAL CONVERGENCE ANALYSIS OF GENETIC ALGORITHM FOR SOLVING UNIT COMMITMENT PROBLEM

Domen Butala
Financial Mathematics, Faculty of Mathematics and Physics
University of Ljubljana, Slovenia
domen.butala@yahoo.com

Dejan Velušček
Department of Mathematics, Faculty of Mathematics and Physics
University of Ljubljana, Slovenia
dejan.veluscek@fmf.uni-lj.si

Gregor Papa
Computer Systems Department
Jožef Stefan Institute, Ljubljana, Slovenia
and
Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
gregor.papa@ijs.si

Abstract This paper presents an implementation and empirical convergence analysis results of genetic algorithm for solving unit commitment problem in a power market. Various parameter settings are presented including an algorithm with a sequence of parameters, also called a variable-structure genetic algorithm. Implemented algorithm successfully solves both small and large scale problems and shows how much more efficient variable-structure genetic algorithm is in practice.

Keywords: Convergence analysis, Genetic algorithm, Empirical results, Unit commitment problem.

1. Introduction

In a power market total production has to meet the demand, net exports and system reserves over a given period of time, subject to the start-up and shut-down times of the generating units. The objective is to minimize the total costs of production while satisfying the start-up and shut-down time constraints. In reality power generating units are not only minimizing costs but also maximizing their profit.

The solution of a unit commitment is a complex optimization problem. It is one of the most widely studied problems in Electrical Engineering. A number of different techniques have been proposed to solve it as Mixed Integer Programming (MIP) [4] or an alternative Mixed Integer Linear Programming (MILP) [1]. Also Lagrangian Relaxation (LR) [4], Benders Decomposition [10], Evolutionary Programming (EP) [7] and Dynamic Programming (DP) are used.

Power generating units can reschedule their commitments (decisions) over and over again. They can do a reschedule when they believe that some parameters have changed enough to affect their decisions. This means that power plants can sell the energy in a futures market, later buy it back and finally commit their schedules on a day-ahead (or often called spot) power auction. If the market price of a certain product in futures market is higher than the expected spot price, power plants with lower marginal prices decide to sell (part of) the energy in advance. When this is not the case, they decide to wait and sell the energy on a day-ahead power auction.

In this paper we will focus not only on the implementation part of the algorithm but also on the convergence analysis. With empirical analysis we will briefly try to justify the relevance of theoretical convergence analysis demonstrated in the section 2.

2. Convergence Analysis

In this section we present three important theorems from [5] and [2]. The first theorem tells us something about an upper bound on the convergence speed and the last two tell us under which conditions we can expect a genetic algorithm (GA) to converge. Based on the presented theorems we do an empirical analysis and try to evaluate theoretical results in practice for a problem of unit commitment using the GA in a power market.

2.1 An Upper Bound on the Convergence Speed

Next theorem tells us an upper bound for convergence of the GA.

Theorem 1 *[5] Let the size of population of the GA be $n \geq 1$, coding length $l > 1$, mutation probability $0 < p_m \leq \frac{1}{2}$ and let $\{\vec{X}_t, t \geq 0\}$ be the Markov chain population, $\pi^{(t)}$ distribution of t^{th} generation of $\vec{X}_t$ and π be the stationary distribution. Then it holds*

$$||\pi^{(k)} - \pi|| \leq (1 - (2p_m)^{nl})^k.$$

Theorem 1 identifies us next relationships.

- Bigger than the mutation probability, faster the convergence.
- Bigger than the population size and coding length, slower the convergence.

But on the other hand it is well known that algorithms with parameters set like this affect negatively on a long term convergence of the GAs. This was shown by studies [3], [8], [12] but also by many others. In section 2.2 let us take a closer look at two theorems which tells us that in case of a very big population with "small enough" mutation probability the GA converges in probability to a global optimum.

2.2 Convergence of Homogenous Algorithm

Theorem 2 *[2] Let $a, b, c > 0$ be constants and i intensity perturbations of algorithm. If it holds*

$$m > \frac{an + c(n-1)\Delta^{\otimes}}{\min(a, b/2, c\delta)}, \tag{1}$$

then

$$\forall x \in S^N : \qquad \lim_{i\to\infty} \lim_{t\to\infty} P([X_t^i] \subset f^* | X_0^i = x) = 1.$$

Theorem 2 tells us that we can, with a big enough population, solve an optimization problem. Additionally we can completely arbitrarily choose the parameters a, b and c. If we choose very big c, then we rule out the importance of δ because it holds $\min(a, b/2) \leq c\delta$. If it is $\Delta^{\otimes}$ a constant and N becomes very big, we can always successfully tackle with a problem. $\Delta^{\otimes}$ represents the difficulty of adopting new and better solutions. Condition 1 is quite rough regarding the population size limit m. It is important that perturbation mechanism lets the process visit the whole space, even though random perturbations could be very small. That is why the role of crossover is not crucial (algorithm without crossover corresponds to the case of $b = \infty$).

2.3 Convergence of Inhomogeneous Algorithm

In practice we will not wait for Markov chain to reach the equilibrium state before we would lower the intensity of perturbations. That is why we want to lower the intensity gradually depending on time. At the same time we want to maintain the properties of a limit law. Let us, from now on, assume that i and t are increasing simultaneously. So, let the i be increasing function of time, where it holds $\lim_{t\to\infty} i(t) = \infty$.

That is how Markov chain becomes inhomogeneous and transition probabilities become time dependent. From now on let us define $X_t = X_t^i$. We can define convergence power of the increasing sequence $i(t)$ with λ, for which holds that

$$i(1)^{-\theta} + i(2)^{-\theta} + \cdots + i(t)^{-\theta} + \ldots$$

converges for $\theta > \lambda$ and diverges for $\theta < \lambda$ where $\lambda \geq 0 \in \mathbb{R}$

With $T_1, \ldots, T_t, \ldots$ we set times of successful visits of the chain $\{X_t\}$ in S, e.g. $T_t = \inf\{k : k > T_{t-1}, X_k \in S\}$. Let us focus on behavior of the chain $\{X_{T_t}\}$.

Theorem 3 *[2]*

1 *That chain $\{X_{T_t}\}$ would reach f^* after finite number of steps*

$$\forall x \in S \qquad P(\exists U, \forall u \geq U, [X_{T_u}] \subset f^* | X^{(0)} = x) = 1,$$

convergence power of sequence $i(u)$ has to be positive constant from the set of real numbers; $\theta_1, \theta_2 > 0 \in \mathbb{R}$ have to exists to

$$\sum_{u\geq 0} i(u)^{-\theta_1} = \infty \text{ and } \sum_{u\geq 0} i(u)^{-\theta_2} < \infty.$$

2 *"If convergence power λ of the sequence $i(t)$ and the population size m suffice the inequalities*

$$an + c(n-1)\Delta^{\otimes} < \lambda < \min(a, b/2, c\delta)m,$$

then with probability 1 chain $\{X_{T_t}\}$ reaches f^* after finite number of steps.

3 *Let exist the constant $t > 1 \in \mathbb{R}$ that for $\forall r \in \mathbb{N}$ sequences $i(\lfloor tu \rfloor + r)$ and $i(u)$ are logarithmic equivalent. If convergence power λ of the sequence $i(u)$ and population size m suffice the inequalities*

$$an + c(n-1)\Delta^{\otimes} < \min(a, b/2, c\delta)m < \lambda,$$

then

$$\forall x \in S \qquad \lim_{t\to\infty} P([X_t] \subset f^* | X_0 = x) = 1.$$

2.4 Combination of Both Approaches

To be able to successfully use the results of both approaches we have to slightly correct an algorithm. Let us define a sequence of parameters $\{(n_t, p_m(t)), t \geq\}$ that it holds $n_t < n_{t+1}$ and $p_m(t) > p_m(t+1)$.

Thus, we run the GA with settings $(n_1, p_m(1))$ first. Let us define a solution close to the optimal with $\vec{X}_1(\infty)$. When it is reached, with predefined scheme, we rename the population $\vec{X}_1(\infty)$ to $\vec{X}_2(0)$ and use it as an initial population of the GA with settings $(n_2, p_m(2))$. Using this approach we are increasing the population size and decreasing the mutation probability thus increasing the algorithm's efficiency.

We can call such the GA a variable-structure GA [5]. We should note that the convergence is not a natural property of the GA but it is followed by elitism property in a selection operator [6].

3. Implementation and Model Description

In this section we will denote a mathematical formulation of the problem. Based on this we will describe an algorithm for the optimization problem.

3.1 Problem formulation

$$\min_{x_{i,t}^{type}} \Big\{ \sum_{t=1}^{T} \sum_{i=1}^{n} (mp_{i,t} x_{i,t}^{type} + \max\{s_{i,t} - s_{i,t-1}, 0\} sc_i) \Big\}$$

$$\sum_{i=1}^{n} x_{i,t}^{type} \geq PDP_t(price),\ \forall t \tag{2}$$

$$s_{i,t} = \left\{ \begin{array}{ll} 1, & \text{"if } x_{i,t}^{type} > 0, \\ 0, & \text{otherwise.} \end{array} \right\},\ \forall t,\ i \tag{3}$$

$$st_{i,t} = (-1)^{1-s_{i,t}} \sum 1_{\left\{ \substack{I=[t-a,t+b] \wedge a,b \geq 0: \\ s_{i,t}=s_{i,\bar{t}}\, \forall \bar{t} \in I \wedge s_{i,t-a-1}=s_{i,t+b+1}=1-s_{i,t}} \right\}} \tag{4}$$

$$st_{i,t} \geq tup_i \vee st_{i,t} \leq -tdown_i,\ \forall t,\ i \tag{5}$$

$$x_{i,t} = xmax_{i,t} \tag{6}$$

Where $t = 1, \dots, T$ is an observed time interval, $i = 1, \dots, n$ is an unit index and *type* is a unit type. Furthermore, $x_{i,t}^{type} \in \mathbb{R}$ is the production of unit i of type *type* in time t, $xmin_{i,t}$ is the technical minimum of unit i in time t, $xmax_{i,t}$ is the installed capacity of unit i in time t, $s_{i,t}$ is a status of unit i in time t, $st_{i,t}$ represents a number of working or non-working hours of unit i in time interval "near" t, tup_i and $tdown_i$ is the minimal time interval of unit i in which unit cannot change its status decision due to the technical limitations, PDP_t is a price dependent production, $mp_{i,t}$ a marginal price of unit i in time t and sc_i are the starting costs of unit i.

Goal. A goal of the optimization is to find a solution with minimal costs to the system. Cost objective function describes costs of the system in a way that it calculates sum of product of production and marginal costs and total start-up costs on the whole time interval.

We have to meet the constraints, especially important are the following two. The first one (2) ensures that needed levels of thermal production are met and the second one (5) ensures that units are not violating their technical constraints.

At this stage it is important to note that the thermal production is dependent on the price and cross border commercial flows and vice-versa. Because both variables are dependent on neighboring markets, determination of both is a very complex problem. Also note that a more representative criterion function of a power generating unit is of a form $ax^2 + bx + c$ but in this case we have to find or know the parameters a, b, c. This is not a content of the article, so we will not discuss this problem.

3.2 Optimization

Let us write the pseudocode ofthe GA for our problem. This will serve as a starting point for later discussion.

Algorithm 1 Genetic Algorithm

```
1: $t = 0$
2: $P(t)$ = SetInitialPopulation($P$)
3: Evaluate($P(t)$)
4: while not EndingCondition() do
5:     $t+ = 1$
6:     $P(t)$ = Selection($P(t-1)$)
7:     $P(t)$ = Crossover($P(t)$)
8:     $P(t)$ = Mutation($P(t)$)
9:     Evaluate($P(t)$)
10: end while
```

Initial Solution. For the initial solution we implemented an algorithm named priority list for a problem of unit commitment. Algorithm is deterministic, for this reason it returns the same value for the same inputs every time. By using this algorithm we get an initial population consisting of only one solution, therefore we have to do a trick to get n different solutions. To differentiate between n initial solutions, we replicate the one as n-times and then mutate all except one.

Mutation. The idea of a mutation operator is very simple. With predefined probability, an individual from a population is chosen, on which a random change is made. But before we do this, we have to make sure that the change gives a feasible solution.

The operator mutation is built in a way that technical constraints of mutated units are satisfied first; e.g. minimal up or down times. After this step we have to check feasibility of constraint (2). If the solution is feasible we accept it otherwise we save it in the dictionary of rejected mutations. At the end of the mutation we loop through the dictionary and check for feasible solutions. We accept all that are feasible. After the step we empty the dictionary.

Crossover. A crossover operator is the most important operator in the GA. The goal is to successfully combine two different solutions to get two better offspring. This operator is the most complex in the implemented algorithm. In the operator mutation we have already realized that some minor corrections have to be done not to reject too many (infeasible) solutions. The operator crossover has to apply similar manipulations.

One-Point Crossover. Let us first present the one-point crossover operator. Later we will also present the multi-point crossover which is a partial generalization of this one.

The idea is very simple. First, we randomly determine pairs of populations for the crossover and then we randomly determine a crossover point t which is the same for all units in a pair. Then we cross same the units within a pair of solutions. If needed we have to "correct" an infeasible solution to generate a feasible one.

A very important property of the operator is that it crosses only the same chromosomes (units) under different solutions. For example, in case of crossing two different units within one solution two units with different parameters (minimal up or down time, different installed capacities, maintenance schedules, ...) would be switched. This has to be treated differently, e.g. with another operator which would already be a joint operator of mutation and crossover.

At determination of a crossover point t we have to check if the place is acceptable for all units in the population. If needed we have to shift the time of crossover for a few time periods to left or to right for each unit. That is how we determine a vector of crossover places $T = (t_1, \ldots, t_n)$ for a pair of solutions. Because we did one additional operation, we avoided the violations of minimum up or down times. Now we just have to check the feasibility under constraint (2). If a solution is feasible and better, we accept it. Otherwise we correct it with probability p and reject it with probability $1 - p$. If we decide to correct it, we have to turn additional units on to meet the constraint (2).

Multi-Point Crossover. One-point crossover is a special case of multi-point crossover. Therefore we can use our knowledge about the one-point crossover.

The main difference between both is already described by its name. Using multi-point crossover m we split the coding length to $m+1$ randomly determined intervals. But crossover operator basically stays the same. Instead of only two intervals $[0,t),[t,T]$ we have $m+1$ intervals $[0,t_1),[t_1,t_2),\ldots,[t_m,T)$. Doing so we have to provide a feasible solution for the each interval.

The idea is to construct a feasible solution with the dealing of two adjacent intervals at the time and basically translating a multi-point crossover to a one-point crossover. So, we cross the $[0,t_1),[t_1,t_2)$ first, then the $[t_1,t_2),[t_2,t_3)$ until we finally get to the $[t_{m-1},t_m),[t_m,T)$. With translation to a one-point crossover we get all needed instruments to deal with the problem.

Selection. The selection operator is the simplest operator in the implemented algorithm. There is no need to create or find new and better offspring but only to "shake" the population and randomly permute n of the solutions where better solutions have higher probability to be chosen as offspring.

As we have seen in the section 2.4, one of the most important properties of selection operator is the property of elitism. Therefore we have chosen to implement it. Later we will see the algorithm's behavior without taking into account the elitism property.

4. Results

In this section we present results of the implemented algorithm of the actual data. To present and study results as unbiased as possible we compare them on the same data set but with different GA settings. The goal is to successfully distinguish between the worst and best solutions. The following parameters

- crossover and mutation probability,
- crossover operator,
- selection operator

are chosen based on the theoretical analysis which we try to confirm with empirical results.

4.1 Assumptions

First let us present the assumptions. The most important condition is to meet the demand or in our case needs of the thermal production. Let us call it a price dependent production. We observe two different time periods; length of the first is 72 hours and length of the second is 168 hours. In the first we analyze different settings of mutation and crossover probability, one-point crossover and selection. In the second we compare both, one-point and multi-point, crossover operators under the same settings. The number of power plants which appear in the optimization for specific problem is equal to N. Therefore the size of our matrices is equal to $72 \times N$ and $168 \times N$.

Each setting is run independently 30 times due to the stochastic properties of the algorithm. Only one would not suffice in providing unbiased results.

4.2 Empirical Analysis

Let us analyze impact of the mutation, crossover and selection of the convergence speed. First we observe a time interval of a length 72 hours and for crossover operator we use only the one-point crossover (OPC) to compare results under different parameters but the same conditions. In this optimization 145 power plants are appearing.

Later we compare one-point and multi-point crossover (MPC) operators. In this case we observe a time interval of a length 168 hours. The reason for this is that the results of the algorithm could be otherwise biased due to the time constraints in the optimization. In this optimization 167 power plants are appearing.

Comparison of various settings on one point crossover. In this subsection we analyze different settings of the algorithm and compare the results. Let us present settings N_i for all i in the table below. We will show 6 different settings which have been obtained by empirical experiments. Using the selected parameters we are able to show the impact on the GA's performance under different domains and, the most important, various conditions.

Table 1. Parameter settings.

Parameters / Settings	N_1	N_2	N_3	N_4	N_5	N_6
Iterations	10,000	10,000	10,000	10,000	10,000	10,000
Population size	30	30	30	60	30	60
Elitism	4	4	4	4	0	4
Crossover	OPC	OPC	OPC	OPC	OPC	OPC
Crossover Probability	50%	75%	50%	50%	50%	vary
Mutation Probability						
- population	25%	25%	40%	25%	25%	vary
- individual	20%	20%	25%	20%	20%	vary

The mutation probability is actually lower than it seems. We have to take a product of both mutation probabilities. In case of N_1 the mutation probability is equal to 10% (which is equal to 40% times 25%). Note that we have used 50% probability to correct infeasible solutions in the crossover operator.

All setting except the setting N_6 are constant over time. For the setting N_6 we have constructed a sequence of parameter settings to show how a sequence of settings affect the performance under different domains. The idea is to analyze the algorithm's performance which explore the space of feasible solutions more aggressive at the beginning and less later.

Thus let us define the sequence under different domains which we obtained by the empirical results. In the interval $[0, 1000)$ iterations we have set the crossover probability to 75% and mutation probability pair to (45%, 25%). In the next interval $[1000, 2000)$ we have decreased the mutation probability to (25%, 20%). In the interval $[2000, 5000)$ we have decreased crossover probability to 50%. And for the last interval $[5000, 10000)$ we have decreased mutation probability to (25%, 10%).

For a higher transparency let us present only the average values of a criterion function of 30 independent runs.

The most obvious thing we see is that the algorithm with the setting N_5 does not converge at all. This confirms that the convergence of the GA is a consequence of the elitism property.

Let us now compare the crossover's probability effect. Thus we limit ourselves to compare the results with parameter settings N_1 and N_2. We see that the algorithm with the parameter setting N_2 is around 20% more efficient at the beginning but after roughly 200 iterations we cannot significantly distinct between the efficiency of both. Due to the

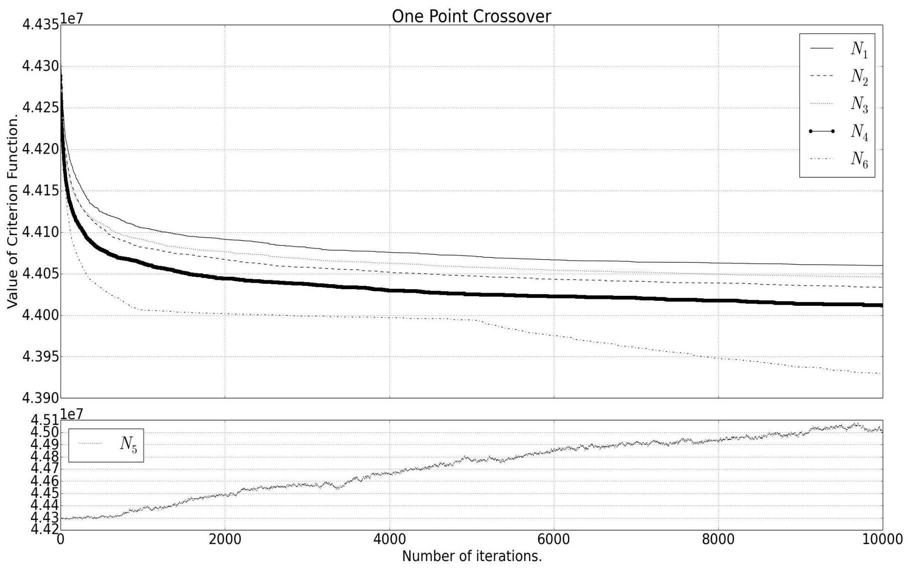

Figure 1. One point crossover vairous settings.

crossover's probability the algorithm with the parameter setting N_2 is around 35% slower than the algorithm with the parameter setting N_1.

If we only compare the mutation probability effect, we limit ourselves to parameter settings N_1 and N_3. Similar to the crossover probability effect, algorithms with higher probability are more efficient at the beginning, but after few iterations we see that the algorithm with the lower mutation probability becomes more efficient. For the first roughly 200 iterations the algorithm with the setting N_3 is around 20% more efficient than the algorithm with the setting N_1, but after that the algorithm N_1 becomes more efficient for about around 10% on average.

Algorithm with the parameter setting N_4 is one of the most efficient at the beginning. In comparison to the N_2 or N_3 it is more efficient for about 5%. After few iterations the efficiency becomes comparable to algorithms with parameter settings N_1, N_2 and N_3. But due to double the population size the algorithm with this setting needs on average twice as much time as with other settings for one iteration.

Based on these results we were able to construct a sequence of parameter settings. We clearly see that the algorithm with this sequence has the best result compared to other settings. Speed of the algorithm with setting N_6 is comparable to the algorithm with setting N_4, but because of the overall efficiency it is clearly significantly better. In addition to this, we need only around 1,000 iterations to get at least as good a result as the algorithm with any other setting at 10,000th iteration.

One-Point versus Multi-Point Crossover. Let us now analyze and compare both crossover operators. We compare operators on the same data set with the same parameter settings. In this case we do the optimization on a time interval of 168 hours.

Table 2. Parameter settings.

Parameters / Settings	N_7	N_8
Iterations	10,000	10,000
Population size	30	30
Elitism	4	4
Crossover	MPC	OPC
Crossover Probability	50%	50%
Mutation Probability		
- population	25%	25%
- individual	20%	20%

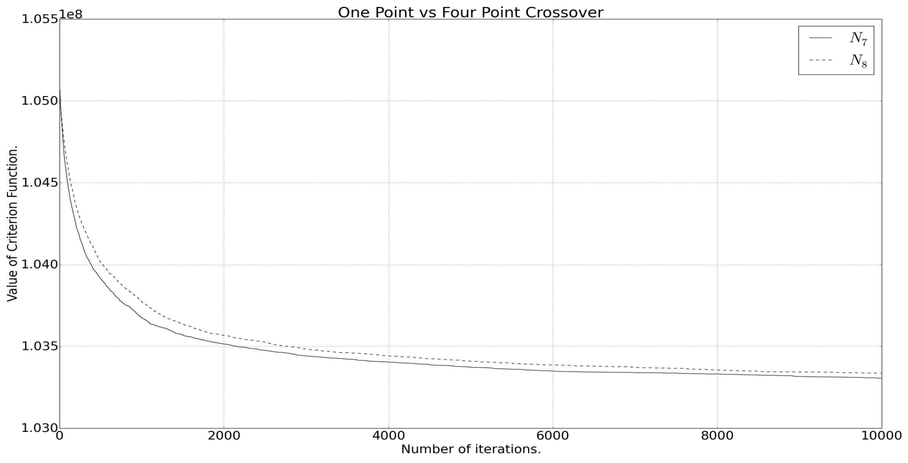

Figure 2. Multi point vs one point crossover.

We see a similar pattern than when comparing the algorithm with higher and lower crossover probabilities. At the beginning the algorithm with multi-point crossover is significantly (20%) more efficient, but later one-point crossover gets an advantage of 10% on average. This is due to the fact that closer we get to a sub-optimal solution, harder it is to find a better solution and therefore smaller steps should be taken to find it.

We see that the difference between best solutions of the algorithm with selected parameters is minimal, but the range between the best and worst solution of each setting at the 10,000th iteration is for 37% smaller in the N_7 case. This could tell us that with this setting we are able to find a better solution with a higher probability but it also could tell us that it is more likely to stay in the local optima. The algorithm with the setting N_7 maintains the advantage though due to the more efficient starting iterations and the fact that it is on average slower only for about 15-20%.

Thus if we would like to optimize a time interval of which length is appropriate for multi-point crossover, it is good to combine and apply both when constructing a sequence of parameter settings. First we should use multi-point crossover and later one-point crossover.

5. Conclusion

We have seen that the parameter settings of the algorithm can strongly affect on the performance properties and the efficiency. Empirical results have shown theoretical analysis as valid for the unit commitment problem solved by the GA. Regardless of that, detailed analysis of the algorithm settings on real data should be done for each of the goals to get the best results possible. This can be a time consuming, however worth the effort due to the performance and efficiency gains. Further projects in the future could be an implementation of a self-adaptive GA [11]. That is how we would be able to find a good enough (or even optimal) parameter settings. On the other hand one has to be aware of the difficulty of finding better feasible solutions. This means that we will, sooner or later, reach the performance limit.

In this article we have successfully analyzed and compared different algorithm settings and were able to differentiate between them. We did not focus on finding a global optimum on a chosen data set. We have also shown successfully that a variable-structure GA can drastically improve the efficiency of the algorithm.

References

[1] M. Carrion and J. Arroyo. A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE T. Power Syst.*, 21(3):1371–1378, 2006.

[2] R. Cerf. Asymptotic Convergence of Genetic Algorithms. *Adv. Appl. Probab.*, 30(2):521–550, 1998.

[3] K. A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* The University of Michigan, Ann Arbor, 1975.

[4] A. Fragioni, C. Gentile, and F. Lacalandra. Tighter Approximated MILP Formulations for Unit Commitment Problems. *IEEE T. Power Syst.*, 24(1):105–113, 2009.

[5] Y. Gao. An Upper Bound on the Convergence Rates of Canonical Genetic Algorithms. *Complexity International*, Vol. 5, 1998.

[6] M. Iosifescu. *Finite Markov Processes and Their Applications.* Wiley, Chichester, 1980.

[7] K. A. Juste, H. Kita, and E. Tanaka. An evolutionary programming solution to the unit commitment problem. *IEEE T. Power Syst.*, 14(4):1452–1459, 1999.

[8] Y. Leung, Y. Gao, and Z. B. Xu. Degree of population diversity: A perspective on premature convergence in genetic algorithms and its Markov chain analysis. *IEEE T. Neural Networ.*, 8(5), 1165–1176, 1996.

[9] J. A. López, J. L. Ceciliano-Meza, I. Guillén, and R. N. Gómez. A Heuristic algorithm to solve the unit commitment problem for real-life large-scale power systems. *International Journal of Electrical Power & Energy Systems*, 49:287-295, 2013.

[10] T. Niknam, A. Khodaei, and F. Fallahi. A new decomposition approach for the thermal unit commitment problem. *Appl. Energy*, 86(9):1667–1674, 2009.

[11] G. Papa. Parameter-less algorithm for evolutionary-based optimization. *Comput. Optim. Appl.*, 56 (1):209–229, 2013.

[12] G. G. Robertson. Population size in classifier systems. In *Proc. 5th International Conference on Machine Learning*, pages 142–152, 1988.

[13] W. Snyder, H. Powell, and J. Rayburn. *Dynamic programming approach to unit commitment. IEEE T. Power Syst.*, 2(2):465–473, 1987.