DOI: 10.48011/sbse.v1i1.2499

A Parallel Mix Self-Adaptive Genetic Algorithm for Solving the Dynamic Economic Dispatch Problem

Egidio de Carvalho Ribeiro Júnior [*]
Omar Andres Carmona Cortes [**]
Osvaldo Ronald Saavedra [***]

[*] *Programa de Pós-Graduação em Engenharia da Computação e Sistemas (PECS), Universidade Estadual do Maranhão (UEMA), MA, (e-mail: egidio.carvalho@gmail.com).*
[**] *Departamento de Computação (DComp), Instituto Federal do Maranhão (IFMA), MA, (e-mail: omar@ifma.edu.br)*
[***] *Instituto de Energia Elétrica (IEE), Universidade Federal do Maranhão (UFMA), MA, (e-mail: o.saavedra@ieee.org)*

Abstract: The purpose of this paper is to propose a parallel genetic algorithm that has adaptive and self-adaptive characteristics at the same time for solving the Dynamic Economic Dispatch (DED) problem that is a challenging problem to solve. The algorithm selects the proper operators (using adaptive features) and probabilities (using the self-adaptive code) that produce the most fittable individuals. Regarding operations, the choice is made between four different types of crossover: simple, arithmetical, non-uniform arithmetical, and linear. Concerning mutation, we used four types of mutations (uniform, non-uniform, creep, and enhanced apso). The choice is made scholastically, which is uniform at the beginning of the algorithm, being adapted as the AG executes. The crossover and mutation probabilities are coded into the genes, transforming this part of the algorithm into self-adaptive. The multicore version was coded using OpenMP. An ANOVA test, along with a Tukey test, proved that the mixed self-adaptive algorithm works better than both: a random algorithm, which chooses operators randomly, and a combination of operators set previously in the DED optimization. Regarding the performance of the parallel approach, results have shown that a speedup of up to 3.19 can be reached with no loss in the quality of solutions.

Keywords: Genetic Algorithms; Adaptive; Self-adaptive; Dynamic Economic Dispatch; Parallel.

1. INTRODUCTION

The Economic Dispatch (ED) (Pereira-Neto et al., 2005) (Barros et al., 2013) can be considered as one of the essential tools in the operation of power systems. In its purest form, the input-output characteristics or cost function of generators can be approximated by a quadratic function, making it possible to formulate the ED as an optimization problem. Thus, the primary purpose of ED is to minimize the total energy production costs, while various generator constraints are satisfied. However, generator output curves have a high degree of non-linearities and discontinuities due to the effect of ”valve points”.

In the current context, natural gas thermal units play an important role in helping to mitigate rapid generation variations caused by the large insertion of wind farms in the electricity grid. This makes the economic dispatch problem a revisited and current problem. Since the operating costs of different generating units differ significantly, it is a challenging problem to schedule the best mix of generation from several units to attend a particular load demand at minimum cost. This problem is called Economic Dispatch Problem, which is a static optimization problem. Even though this scheduling might be beneficial for a particular hour of the day, it might not be for different hours. To tackling the different demand scenarios imposed by twenty-four hours of a day, we can transform the classical Economic Dispatch into the Dynamic Economic Dispatch (DED), in which each hour of the day may demand a different power consumption.

The DED problem is a much more challenging problem than classical EDP because, as previously stated, it has to satisfy different demands during the day. In this context, classical optimization methods are inefficient mainly because they can deal neither with constraints nor with many variables. In this context, evolutionary approaches such as Genetic Algorithms (GA) represent an attractive form of solving this kind of optimization problem. A GA implements a search method to find out a solution for a particular problem. To do so, a set of parameters, such as crossover and mutation operators, crossover and mutation probabilities, population size, the number of genes, the number of individuals, and selection method, need to be set before the execution starts. These parameters are directly responsible for the exploration and exploitation capabilities of the algorithm. Thus, setting them is a challenge because different problems might require different parameter combinations, demanding consequently more executions and computational time.

In order to automatically choose those parameters, two types of algorithms are common: adaptive and self-adaptive algorithms. In adaptive algorithms, there is some form of feedback from the search that serves as inputs to a mechanism used to determine the change to the strategy parameter (Eiben et al., 2006). In self-adaptive algorithms, parameters are encoded into the chromosomes and undergo mutation and recombination along with all genes. There is a third kind of adaptive algorithm called deterministic. In which all parameters are updated using a deterministic rule that usually is an equation. Some efforts have been made to either adapt or self-adapt genetic algorithms. For example, Lu et al. (2015), Xiao et al. (2015), and Miranda et al. (2015) developed adaptive or self-adaptive genetic algorithms; however, these works are not parallel. On the other hand, Pessini (2003) and da Silva et al. (2013) studied multi-thread genetic algorithms; nonetheless, they are neither adaptive nor self-adaptive. The first attempt to implement a multi-thread adaptive algorithm was Carvalho et al. (2017), in which some multimodal benchmarks functions are solved.

Regardless of which kind of GA has been developed, it is clear that adaptive techniques are adequate to the task of choosing operators. On the other hand, it would be challenging to select which operator to use based on crossover operations such as arithmetical or linear crossover because they are naturally real-coded operations. In opposition, selecting the best probabilities (crossover and mutation) seems suitable for self-adaptive algorithms regardless of the operator being used during the execution of the GA. In this context, our solution aims to mix both kinds of selectors: adaptive and self-adaptive, giving more flexibility to GAs to solve different kinds of problems efficiently. Moreover, these algorithms are parallelized using OpenMP in order to improve the performance of our proposal. At the best of our knowledge, there are no other works that mix adaptive and self-adaptive GA in a parallel environment.

This work is divided as follows: Section 2 presents the DED problem. Section 3 introduces the adaptive and self-adaptive algorithm, including how the algorithm has been parallelized. Section 4 shows how the experiment has been set up, including the configuration of the DED system and the results of the experiment. Finally, Section 5 presents the conclusions of this work and future directions.

2. DYNAMIC ECONOMIC DISPATCH

In the Dynamic Economic Dispatch problem, we aim to optimize the power dispatch in a power plant. In other words, we have to attend different demands of power during a day minimizing the cost of doing it. Computationally, we calculate the necessary power in the available generator units according to the expected demand during a period. Furthermore, the generation must obey all restrictions of the system. Hence, the ultimate goal is to minimize the cost of power generation, satisfying all the following constraints: (i) real power balance limit; (ii) real power generation limit; and (iii) generation unit ramp rate limits. Besides the constraints, there are more challenges to face in the DED problem. Whenever the demand changes, the power generation must change as well. This causes a hitch because there are costs associated with changing the levels of production in the power generation. This behavior becomes the problem impossible to solve by a gradient-based method; thus, the problem is entirely suitable to be tackled by evolutionary algorithms. According to Kumar and Alwarsamy (2011), DED is a dynamic problem, due to the dynamic nature of the power system and the considerable variation of load demands.

Regarding the cost, Equation 1 calculates the power generation cost, in which F is the power generation cost within the considered period, T is the number of time intervals, N represents the quantity of generation units, and $F_{it}(P_{it})$ is the cost according to the real power P_{it} in a time interval t. The function uses 24 intervals of 1 hour each.

$$\min F = \sum_{t=1}^{T}\sum_{i=1}^{N} F_{it}(P_{it}) \tag{1}$$

Considering the valve effects, the function that computes the cost of the generation unit i can be expressed as a sum of a quadratic and sinusoidal function as shown in equation 2, in which a_i, b_i, and c_i are cost coefficients, e_i and fi are constants of the valve effect of the generation unit i, and P_i is the power output of this unit expressed in MW.

$$F_{it}(P_{it}) = a_i + b_i P_{it} + c_i P_{it}^2 + |e_i \sin f_i(P_{imin} - P_{it})| \tag{2}$$

Concerning constraints, Equation 3 expresses the power balance constraint, where P_{Dt} represents the total power demanded in a period t, P_{Lt} is the power loss during transmission in the same period, both are in MW. The second constraint is the capacity of generation of each unit, as shown by Equation 4, in which P_{imin} is the lower bound and P_{imax} is the upper bound of the generation unit i, respectively. Equations 5 and 6 describe the ramp limit constraints of a unit i, where UR_i represents the increasing limit of a generation unit i, and DR_i is the decreasing limit. Furthermore, P_{Dt} must obey the loss constraint presented in Equation 7, in which B is a matrix of loss coefficients.

$$\sum_{i=1}^{N} P_{it} - P_{Dt} - P_{Lt} = 0, \text{ in which } t = 1, 2...T \tag{3}$$

$$P_{imin} \le P_{it} \le P_{imax}, \text{in which } t = 1, 2...T, i = 1, 2...N \tag{4}$$

$$P_{it} - P_{it-1} \le UR_i, \text{ in which } i = 1, 2...N \tag{5}$$

$$P_{it-1} - P_{it} \le DR_i, \text{ in which } i = 1, 2...N \tag{6}$$

$$P_{Dt} = \sum_{i=1}^{n}\sum_{j=1}^{n} P_i B_{ij} P_j + \sum_{i=1}^{n} B_{0i} P_i + B_{00} \tag{7}$$

3. THE MIX SELF-ADAPTIVE GA (MSGA)

Genetic Algorithms are search techniques able to discover the best candidate solution among a massive number of possible solutions. They were inspired by Darwin's natural selection and use concepts like inheritance, crossover, and mutation. Details of the canonical GA for numerical optimization and its genetic operators can be seen in the work of Cortes and da Silva (2019).

In our approach, the first thing we have to take into account is how the algorithm chooses the best operators and taxes for crossover and mutation. Firstly, the MSGA must choose which genetic operators will be used in the first iteration. The chosen is made by using a uniform distribution in all operators, *i.e.*, the same probability for each crossover and mutation operator (because we are using four crossovers and four mutations operators the probability of choosing each one is 25%). The following operators can be chose: (i) **Crossover** - Simple, Arithmetical, Non-Uniform Arithmetical, and Linear; (ii) **Mutation** - Uniform, Non-uniform, Creep and Enhanced APSO (EAPSO)

After setting the genetic operators, the algorithm verifies whether an enhancement was produced in the best solution; if so, the probabilities of that kind of crossover and mutation will be increased according to Equation 8, in which t is the current iteration and T_{max} is the maximum iteration count. As we can see, the increment is smaller at the beginning of the algorithm and increases as the algorithm evolves. The purpose of this equation is to avoid a premature convergence to a specific combination of crossover and mutation operators.

$$step = \frac{t}{T_{max}} \tag{8}$$

As mentioned previously, our approach is also self-adaptive, *i.e.*, the probabilities of crossover (P_c) and mutation (P_m) are embedded into the genes. Thus, when a crossover occurs, the probabilities for generating sons will depend on parents' probabilities. Therefore, there is a tendency to repeat the taxes along with the population because those more adapted individuals tend to generate more offspring. The Algorithm 3 shows the pseudo-code of the MSGA.

```
Pop ← InitPopulation();
fit ← Eval(Pop)
for (i←1 to #Generations) do
    for (p←1 to #PopSize) do
        NewPop ← tournamentSelection(Pop);
        Cross ← decideCrossover(NewPop);
        Mut ← decideMutation(NewPop);
        $P_c$ ← decideCrossoverProb(step);
        $P_m$ ← decideMutationProb(step);
        applyCrossover(NewPop, Cross, $P_c$);
        applyMutation(NewPop, Mut, $P_m$);
    end for
    Pop ← NewPop;
    fit ← fitNewPop;
end for
```

Figure 1. MSGA Pseudo-Code

As we can observe, the MSGA is very similar to a canonical GA excepting the choosing of the genetic operators and the application of P_c and P_m. It is important to notice that: (i) the selection of individuals is made by using tournament selection; (ii) we use elitism, keeping the best solution in the population; (iii) the probabilities of selecting an operator are updated using Equation 8; (iv) when a crossover or mutation is applied, the decision of performing the crossover (P_c), or the mutation (P_m) is based on the mean of their parents. For example, if the algorithm selects $indiv_1$ and $indiv_2$ for performing a simple crossover, the decision of either doing it or not is based on a new P_c, which is computing by $P_c^{offspring} = (P_c^{indiv1} + P_c^{indiv2})/2$. In the same step the probability of mutation is updated by $P_m = P_m^{offspring} = (P_m^{indiv1} + P_m^{indiv2})/2$; and (v) when the time of performing the mutation has come, the operation is performed on each individual using the P_m computed previously. Next, we show details on how the MSGA was parallelized using OpenMP.

3.1 The Parallel MSGA

As mentioned previously, we parallelized the MSGA using OpenMP. Its use starts by adding the library $< omp.h >$ in a program that can be C, C++, or Fortran. In our case, we implemented the adaptive algorithm using C language. The second step is to choose which part of the code will run in parallel. To do so, the directive is *#pragma omp parallel for*. Parallel directives say to the compiler to distribute all operations inside a block. In this case, we are using *parallel for*; therefore, the compiler will try to distribute all instruction inside the loop between threads. The number of threads can be set by using the instruction *set OMP_NUM_THREADS = number_of_threads* in the shell, using the instruction *omp_set_num_threads(number_of_threads)* in the code, or using a parameter *num_threads(number_of_threads)* in the *#pragma omp parallel for*. Figure 3.1 shows where the directive was placed to achieve a better speedup factor.

```
Same initialization process of the MSGA
for (i←1 to #Generations) do
    OMP Parallel For Directive
    for (p←1 to #PopSize) do
        Same operations of the MSGA
    end for
    OMP Barrier
    Pop ← NewPop;
    fit ← fitNewPop;
end for
```

Figure 2. Parallel MSGA Pesudo-Code

Note that directives were placed before dealing with the population and after computing it (barrier). The first directive informs the compiler to distribute the creation of the new population between the available threads. The second one notifies the compiler to proceed the execution only after all threads have processed the entire population. The parallel mode is the master-slave, in which each thread receives part of the task to be completed. On the one hand, this form of parallelism can generate poor speedup if functions are not computationally intense. On the other hand, placing the directive as we have done guarantees that the quality of solutions will not be affected as occurs when the directive is located before the iteration loop.

3.2 Dealing with Constraints

There are some ways of dealing with constraints. The most radical is the death penalty, in which any violation in constrains leads to the elimination of the individual. The drawback of this approach is to obtain a low number of individuals if the problem has many constraints. A common way of tackling constraint is the static penalty, in which a penalty is added to the objective function for any violations in the constrains. Mathematically, the static penalty is shown in Equation 9, in which x is a solution (an individual), $p(x)$ is a penalty function, and F is the feasible search space.

$$fitness = \begin{cases} f(x), & \text{If } x \in F \\ f(x) + p(x), & \text{If } x \notin F \end{cases} \tag{9}$$

In our approach, $p(x)$ is the sum of all values that exceed or are inferior to the constraints.

4. EXPERIMENTS

4.1 Setup

The algorithm was executed in a Core i7 2.8GHZ computer, 8Gb of Ram, Windows 7 - 64 bits, and a hard drive of 1TB. The processor has 4 physical cores and handles up to 2 simultaneous threads per core. Functions were executed for 600 and 2000 generations, using 50 trials, allowing parametric tests such as ANOVA and Tukey test. The H_0 hypothesis says the averages are the same, *i.e.*, the difference between algorithm is not meaningful. The acceptance region in the F table for the H_0 hypothesis is within the range [-2.605, 2.605]. The comparison was done between the MSGA (MOD1), an approach where the operators are randomly chosen (MOD2), and every possible combination between the crossover and mutation operators. Table 1 refers to all abbreviations used in this work. The probability of crossover and mutation were set to 75 and 5 percent, a dimension of 30 genes, and respectively with a population size of 25. Two, four and eight threads were used in the parallel version.

The DED problem solved in this work is shown in Table 2 according to the definitions introduced in Section 2. Table 3 presents all required demands along 24h of the day. The system data was obtained in Alsumait et al. (2010).

4.2 Results

The Dynamic Economic Dispatch (DED) Problem was tested with 600 and 2000 iterations and Table 4 shows the best, the worst, the average fitness, and the standard deviation for every set. As we can notice, the best costs that attends all the demands were reached by our adaptive algorithm. The value of F is 2262.298 in the ANOVA test, consequently we reject H_0. Table 5 shows the Tukey test, in which we can notice that our approach definitively presents the best results.

Table 1. Comparision Sets

Abb.	Operators
MOD1	Adaptive operators and selfadaptive taxes
MOD2	Random Crossover and Mutation operators
MOD3	Simple Crossover and Uniform Mutation
MOD4	Simple Crossover and Non-Uniform Mutation
MOD5	Simple Crossover and Creep Mutation
MOD6	Simple Crossover and Eapso Mutation
MOD7	Arithmetical Crossover and Uniform Mutation
MOD8	Arithmetical Crossover and Non-Uniform Mutation
MOD9	Arithmetical Crossover and Creep Mutation
MOD10	Arithmetical Crossover and Eapso Mutation
MOD11	Non-Uniform Arithmetical Crossover and Uniform Mutation
MOD12	Non-Uniform Arithmetical Crossover and Non-Uniform Mutation
MOD13	Non-Uniform Arithmetical Crossover and Creep Mutation
MOD14	Non-Uniform Arithmetical Crossover and Eapso Mutation
MOD15	Linear Crossover and Uniform Mutation
MOD16	Linear Crossover and Non-Uniform Mutation
MOD17	Linear Crossover and Creep Mutation
MOD18	Linear Crossover and Eapso Mutation

Table 2. Five Unit System

	Unit 1	Unit 2	Unit 3	Unit 4	Unit 5
a_i	25	60	100	120	40
b_i	2	1.8	2.1	2	1.8
c_i	0.0080	0.0030	0.0012	0.0010	0.0015
e_i	100	140	160	180	200
f_i	0.042	0.040	0.038	0.037	0.035
P_i^{min}	10	20	30	40	50
P_i^{max}	75	125	175	250	300
UR_i	30	30	40	50	50
DR_i	30	30	40	50	50

Table 3. Load demand for the five-unit system

Hour	Load (MW)	Hour	Load (MW)	Hour	Load (MW)
1	410	9	690	17	558
2	435	10	704	18	608
3	475	11	720	19	654
4	530	12	740	20	704
5	558	13	704	21	680
6	608	14	690	22	605
7	626	15	654	23	527
8	654	16	580	24	463

This experiment was repeated with 2000 generations. The results in favor of our approach improved even more as shown in Table 6. The value of F is 3545.329 in the ANOVA test, so we reject H_0. Table 7 suggests that our approach is the best one for solving the DED problem considering our experiment environment.

The efficiency of the results presented by our adaptive GA is compared against six other approaches for solving the DED problem: an adaptive PSO and original PSO (Niknam and Golestaneh, 2012), an enhanced DE (Balamurugan and Subramanian, 2007), a Hybrid Harmony Search (HS) mixing HS and DE (Chakraborty et al., 2012), using Maclaurin series and Lagrange methods (Hemamalini and Simon, 2010), Artificial Bee Colony (Hemamalini and Sishaj, 2011), Hybrid DE with Sequential Quadratic

Table 4. DED function results with 600 generations

MOD	Best	Worst	Average	St. Dev.
MOD1	**43033.396**	**44145.354**	**43508.565**	**267.858**
MOD2	44739.187	46777.094	45666.455	447.813
MOD3	44947.823	46792.516	45955.839	365.487
MOD4	45254.125	47083.895	46265.956	406.646
MOD5	46985.017	49807.796	48293.546	708.029
MOD6	50507.091	53266.839	51747.328	586.930
MOD7	45782.635	47732.233	46700.023	469.022
MOD8	46064.331	47669.766	46788.025	453.361
MOD9	47973.116	50197.533	49246.654	522.190
MOD10	50793.328	52741.426	51943.302	504.483
MOD11	46774.862	48913.740	48085.977	413.210
MOD12	48076.383	49987.787	48950.502	475.592
MOD13	51691.106	54283.460	52862.064	551.949
MOD14	54838.061	56867.487	55780.407	467.042
MOD15	44345.968	45870.094	45021.371	287.278
MOD16	44724.122	45898.430	45292.000	325.259
MOD17	45719.850	47819.367	46787.815	429.319
MOD18	48545.007	51191.313	49787.084	601.141

Table 5. Tukey test for DED function with 600 generations

MODs	Avg. Differ.	DMS	Conclusion
1 e 2	**-2157.890**		The difference is meaninful.
1 e 3	**-2447.275**		The difference is meaninful.
1 e 4	**-2757.391**		The difference is meaninful.
1 e 5	**-4784.981**		The difference is meaninful.
1 e 6	**-8238.764**		The difference is meaninful.
1 e 7	**-3191.458**		The difference is meaninful.
1 e 8	**-3279.460**		The difference is meaninful.
1 e 9	**-5738.090**		The difference is meaninful.
1 e 10	**-8434.737**	242.708	The difference is meaninful.
1 e 11	**-4577.412**		The difference is meaninful.
1 e 12	**-5441.937**		The difference is meaninful.
1 e 13	**-9353.499**		The difference is meaninful.
1 e 14	**-12271.842**		The difference is meaninful.
1 e 15	**-1512.806**		The difference is meaninful.
1 e 16	**-1783.435**		The difference is meaninful.
1 e 17	**-3279.251**		The difference is meaninful.
1 e 18	**-6278.519**		The difference is meaninful.

Table 6. DED function results with 2000 generations

MOD	Best	Worst	Average	St. Dev.
MOD1	**42767.573**	**43245.864**	**42906.757**	**104.277**
MOD2	43019.313	44237.415	43559.208	337.673
MOD3	43234.213	45109.266	43834.154	345.072
MOD4	43130.079	45376.238	44141.758	464.060
MOD5	45450.687	47597.517	46491.233	579.420
MOD6	50210.717	52830.333	51689.640	591.785
MOD7	43470.334	44614.110	44075.960	237.773
MOD8	43300.878	44620.560	44005.227	281.200
MOD9	44943.132	46985.304	46115.605	467.918
MOD10	50328.768	53227.408	51894.550	494.512
MOD11	43854.935	45843.169	44744.837	453.699
MOD12	44377.233	46303.106	45229.291	448.603
MOD13	47559.731	50055.640	49005.567	606.409
MOD14	54689.069	56883.415	55852.117	501.519
MOD15	42981.929	43933.902	43300.202	219.038
MOD16	42842.028	43843.120	43336.296	251.725
MOD17	43324.205	45164.966	44130.372	429.116
MOD18	47704.689	49868.205	48777.485	620.822

Table 7. Anova results for DED function with 2000 generations

MODs	Avg. Differ.	DMS	Conclusion
1 e 2	**-652.451**		The difference is meaninful.
1 e 3	**-927.398**		The difference is meaninful.
1 e 4	**-1235.001**		The difference is meaninful.
1 e 5	**-3584.477**		The difference is meaninful.
1 e 6	**-8782.884**		The difference is meaninful.
1 e 7	**-1169.203**		The difference is meaninful.
1 e 8	**-1098.470**		The difference is meaninful.
1 e 9	**-3208.849**		The difference is meaninful.
1 e 10	**-8987.793**	224.8	The difference is meaninful.
1 e 11	**-1838.080**		The difference is meaninful.
1 e 12	**-2322.534**		The difference is meaninful.
1 e 13	**-6098.810**		The difference is meaninful.
1 e 14	**-12945.360**		The difference is meaninful.
1 e 15	**-393.445**		The difference is meaninful.
1 e 16	**-429.540**		The difference is meaninful.
1 e 17	**-1223.616**		The difference is meaninful.
1 e 18	**-5870.728**		The difference is meaninful.

Table 8. Results Comparision for DED Function

Approach	Best	Average
Our Adaptive GA	**43033.395**	**43508.564**
Adaptive PSO	43784	43794
Original PSO	45194	46499
Enhanced DE	45800	Not available
Hybrid Harmony Search	43210.95	Not available
MSL	49219.81	Not available
ABC	44046.00	44065.00
DE-SQP	43161	Not available
HPSO	43223.00	43732.00
SPS-DE	43848.511	44345.294

Problem (DE-SQL) (Elaiw et al., 2013), a hybrid PSO (HPSO) (Zhang et al., 2014), and SPS-DE (Zou et al., 2018) using a Differential Evolution Algorithm.

Table 8 presents the comparison. The comparison is in terms of best and average results, even though, some of the related papers did not present the mean. As we can observe, our adaptive GA presented better results than all referred propositions. Therefore, all in all, we consider the results presented by our adaptive GA very promising.

4.3 Speedup

As stated previously, the MSGA was parallelized using OpenMP, which is an API for developing multi-thread applications. Thus, we tested the parallel version using two, four, and eight threads. The speedup is computed by $Sp = \frac{T_s}{T_p}$, in which T_s is the time executing the application using one thread and T_p is the time executing the application in parallel (Alba, 2002). The primary motivation to use this metric is because T_s is the same code of the parallel version but using only one thread.

The efficiency is computed by $Ef = \frac{Sp}{np}$, in which Sp is the speedup, and np is the number of processes or threads. Table 9 present the results regarding the speedup and efficiency.

Concerning efficiency, the best trade-off is using two threads because cores are well used, reaching a speedup of 1.829 that is almost the ideal speedup. As expected in

Table 9. Speedup and efficiency of the MSAG in the optimization of DED

	2 threads	4 threads	8 threads
Speedup	1.829	2.635	3.914
Efficiency	0.915	0.659	0.489

parallel applications, the efficiency tends to decrease as the number of threads increases due to overhead in the communication between threads. However, a speedup of 3.914 represents that the application can run almost four times faster using eight threads, which is significant in an application that demands computationally intensive tasks.

5. CONCLUSIONS

This paper presented a parallel adaptive and self-adaptive genetic algorithm for solving the DED problem. Results have shown that the adaptive algorithm overcomes all combinations of operators for solving the DED problem using 600 and 2000 generations. The differences between MSGA and the combination of operators has been demonstrated meaningful using an ANOVA and a Tukey test. In terms of parallelism, the algorithm reached a speedup of 3.914 with one crucial characteristic: there was no loss in the quality of solutions. This means that more processor units can be added if the users require faster solutions.

Future work includes: a parallel version using C-CUDA in a General Purpose Graphical Processor Unit (GPGPU); extension of the code for solving multiobjective problems, including de the Environmental-Economic Dispatch (EED); and, hybridizing the Adaptive GA with other meta-heuristics, such as Differential Evolution and Particle Swarm Optimization in order to obtain better results.

REFERENCES

Alba, E. (2002). Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82, 7–13.

Alsumait, J., Qasem, M., Sykulski, J., and Al-Othman, A. (2010). An improved pattern search based algorithm to solve the dynamic economic dispatch problem with valve-point effect. *Energy Conversion and Management*, 51(10), 2062 – 2067.

Balamurugan, R. and Subramanian, S. (2007). An improved differential evolution based dynamic economic dispatch with nonsmooth fuel cost function. *Journal of Electrical Systems*, 3(3), 151–161.

Barros, R.S., Cortes, O.A.C., Lopes, R.F., and Silva, J.C.d. (2013). A hybrid algorithm for solving the economic dispatch problem. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, 617–621.

Carvalho, E., Cortes, O.A.C., Costa, J.P., and Vieira, D. (2017). A parallel adaptive genetic algorithm for unconstrained multimodal numerical optimization. In *Simpósio Brasileiro de Automação Inteligente*.

Chakraborty, P., Roy, G.G., Panigrahi, B.K., Bansal, R.C., and Mohapatra, A. (2012). Dynamic economic dispatch using harmony search algorithm with modified differential mutation operator. *Electrical Engineering*, 94(4), 197–205. doi:10.1007/s00202-011-0230-6.

Cortes, O.A.C. and da Silva, J.C. (2019). Unconstrained numerical optimization using real-coded genetic algorithms: a study case using benchmark functions in r from scratch. 11(3), 1–11.

da Silva, F.J.M., de Oliveira, A.C., and de M. S. Veras, R. (2013). Um algoritmo genético paralelo aplicado ao problema de cobertura de conjuntos. *SBSI UFMG*.

Eiben, A., Schut, M., and de Wilde, A. (2006). Boosting genetic algorithms with self-adaptive selection. In *IEEE Congress on Evolutionary Computation*, 1534–1589.

Elaiw, A.M., Xia, X., and Shehata, A.M. (2013). Hybrid de-sqp method for solving combined heat and power dynamic economic dispatch problem. *Mathematical Problems in Engineering*, 1–7.

Hemamalini, S. and Simon, S.P. (2010). Dynamic economic dispatch with valve-point effect using maclaurin series based lagrangian method. *Energy Convers. Manage*, (5), 2212–2219.

Hemamalini, S. and Sishaj, P. (2011). Dynamic economic dispatch using artificial bee colony algorithm for units with valve-point effect. *Electrical Energy Systems*, 21(1), 70–81.

Kumar, C. and Alwarsamy, T. (2011). Dynamic economic dispatch – a review of solution methodologies. *European Journal of Scientific Research*, (4).

Lu, H., Wen, X., Lan, L., An, Y., and Xiao-ping, X.L. (2015). A selfadaptive genetic algorithm to estimate ja model parameters considering minor loops. *Journal of Magnetism and Magnetic Materials*.

Miranda, R.C., Montevechi, J.A.B., and de Pinho, A.F. (2015). Development of an adaptive genetic algorithm for simulation optimization. *Acta Scientiarum: Technology*, 37(3).

Niknam, T. and Golestaneh, F. (2012). Enhanced adaptive particle swarm optimisation algorithm for dynamic economic dispatch of units considering valve-point effects and ramp rates. *IET Generation, Transmission Distribution*, 6(5), 424–435. doi:10.1049/iet-gtd.2011.0219.

Pereira-Neto, A., Unsihuay, C., and Saavedra, O.R. (2005). Efficient evolutionary strategy optimisation procedure to solve the nonconvex economic dispatch problem with generator constraints. *IEE Proceedings - Generation, Transmission and Distribution*, 152(5), 653–660.

Pessini, E.C. (2003). Algoritmos genéticos paralelos - uma implementação distribuída baseada em javaspaces. *Dissertação de Mestrado da Universidade Federal de Santa Catarina. Orientador: J.Mazzucco Jr.*

Xiao, W., Wu, L., Tian, X., and Wang, J. (2015). Applying a new adaptive genetic algorithm to study the layout of drilling equipment in semisubmersible drilling platforms. *Mathematical Problems in Engineering*.

Zhang, Y., Gong, D.W., Geng, N., and Sun, X.Y. (2014). Hybrid bare-bones pso for dynamic economic dispatch with valve-point effects. *Applied Soft Computing*, 18(Supplement C), 248–260.

Zou, D., Li, S., Kong, X., Ouyang, H., and Li, Z. (2018). Solving the dynamic economic dispatch by a memory-based global differential evolution and a repair technique of constraint handling. *Energy*, 147, 59 – 80.