ZĂVOIANU, A.-C., LUGHOFER, E., KOPPELSTÄTTER, W., WEIDENHOLZER, G., AMRHEIN, W. and KLEMENT, E.P. 2015. Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems. *Knowledge-based systems* [online], 87, pages 47-60. Available from: https://doi.org/10.1016/j.knosys.2015.05.029

Performance comparison of generational and steady-state asynchronous multi-objective evolutionary algorithms for computationally-intensive problems.

ZĂVOIANU, A.-C., LUGHOFER, E., KOPPELSTÄTTER, W., WEIDENHOLZER, G., AMRHEIN, W. and KLEMENT, E.P.

2015

Performance Comparison of Generational and Steady-State Asynchronous Multi-Objective Evolutionary Algorithms for Computationally-Intensive Problems

Alexandru-Ciprian Zăvoianu [a,c,*] Edwin Lughofer [a] Werner Koppelstätter [c]
Günther Weidenholzer [c] Wolfgang Amrhein [b,c] Erich Peter Klement [a,c]

[a] *Department of Knowledge-based Mathematical Systems/Fuzzy Logic Laboratory Linz-Hagenberg, Johannes Kepler University of Linz, Austria*

[b] *Institute for Electrical Drives and Power Electronics, Johannes Kepler University of Linz, Austria*

[c] *LCM, Linz Center of Mechatronics, Linz, Austria*

Abstract

In the last two decades, multi-objective evolutionary algorithms (MOEAs) have become ever more used in scientific and industrial decision support and decision making contexts the require an a posteriori articulation of preference. The present work is focused on a comparative analysis of the performance of two master-slave parallelization (MSP) methods, the canonical *generational* scheme and the *steady-state asynchronous* scheme. Both can be used to improve the convergence speed of multi-objective evolutionary algorithms that must use computationally-intensive fitness evaluation functions. Both previous and present experiments show that a correct choice for one or the other parallelization method can lead to substantial improvements with regard to the overall duration of the optimization process. Our main aim is to provide practitioners of MOEAs with a simple but effective method of deciding which MSP option is better given the particularities of the concrete optimization process. This in turn, would give the decision maker more time for articulating preferences (i.e., more flexibility). Our analysis is performed based on 15 well-known MOOP benchmark problems and two simulation-based industrial optimization process from the field of electrical drive design. For the first industrial MOOP, when comparing with a preliminary study, applying the steady-state asynchronous MSP enables us to achieve an overall speedup (in terms of total wall-clock computation time) of $\approx 25\%$. For the second industrial MOOP, applying the steady-state MSP produces an improvement of $\approx 12\%$. We focus our study on two of the best known and most widely used MOEAs: the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and the Strength Pareto Evolutionary Algorithm (SPEA2).

Key words: evolutionary computation, multi-objective optimization, performance analysis, master-slave parallelization, steady-state evolution, electrical drive design

[*] Corresponding author, e-mail: ciprian.zavoianu@jku.at, telephone: +43 732 2468 4140, fax: +43 732 2468 4142

1. Introduction and State of the Art

1.1. *Motivation*

Many real-world optimization problems usually arise in decision making contexts that involve several conflicting objectives (e.g. cost vs. quality, risk vs. return on investment) that should be simultaneously optimized. Problems falling within this class are referred to as *multi-objective optimization problems* (MOOPs in short). Generally, such problems do not have a single solution and "solving" them requires finding a set of non-dominated solutions called the *Pareto-optimal set* (in short, PS). Each solution (candidate) from this set is better than any other solution from the set with regard to at least one of the optimization objectives (i.e., no solution from this set is fully dominated by another solution). For many MOOPs, the Pareto-optimal set is unknown and/or infinite. Therefore, in most application domains, decision makers refer to the *Pareto non-dominated set* (in short, PN) which contains an arbitrarily fixed number of solutions that are able to provide a good approximation of the PS. The objective space representation of a Pareto non-dominated set is called the *Pareto front*.

In real-life scenarios, solving a MOOP is actually divided into two distinct stages:

– *the search stage* where the goal is to find solution candidates that are able to optimize (minimize) all the objectives of the MOOP (i.e., discover PNs that are as close as possible to the PS of the MOOP);

– *the decision making / articulation of preferences stage* where the goal is to determine the exact solution candidate(s) that incorporate(s) the best trade-offs in the decision maker's opinion;

The focus of this work lies exclusively with improving the general converge time of methods (evolutionary algorithms) that are commonly used to solve the *search stage* and find high-quality PNs. The motivation for this (and the general concept of a posteriori articulation of preference) is that once a clear and broad picture of all the existing trade-offs in the MOOP is available (i.e., a good Pareto front has been discovered), the decision maker has a lot of flexibility to tune the more subjective multi-criteria decision making part in order to select a very limited number of Pareto non-dominated solutions (sometimes just one) that will be constructed/implemented/applied in the real-life process modeled by the given MOOP.

Multi-objective evolutionary algorithms (MOEAs) have proven to be one of the most successful soft computing techniques for solving MOOPs [1] [2] [3]. This is because they are able to produce complete PNs over single runs. Like most stochastic methods, MOEAs are approximate methods that cannot guarantee finding the optimal solution set of the MOOP (i.e., the PS and the *true Pareto front* associated with it), but these algorithms are fairly robust and can find high quality non-dominated solution sets in reasonable time.

The main drawback of using MOEAs in practical applications is the fact that, in order to discover good solution sets, they usually require a large number of solutions to be evaluated during the optimization run. The issue can become particularly problematic for optimization problems that require very computationally-intensive fitness evaluation functions in order to compute objective or constraint values (e.g., consider physics-orientated simulation methods such as finite element methods, or the usage of software emulators in engineering design). In these cases, optimization runs can last for several days, as shown in [4] where MOEAs are used for the optimization of combustion in a diesel engine, or in [5] where MOEAs are applied for optimizing design parameters of electrical drives.

A very simple idea that displays immediate benefits in reducing the runtime of time intensive optimization runs is the parallelization and/or distribution of the MOEA run over a computer cluster or grid environment.

There are several paradigms (architectural and/or conceptual models) of parallelizing a MOEA: master-slave, island, diffusion, hierarchical and hybrid models (please see Chapter 8 of [3] for an overview).

By far, the most straight forward and easiest to implement parallelization method for evolutionary algorithms is the **master-slave parallelization** (MSP) model: fitness evaluations are distributed between several slave nodes (computational units), while all the evolutionary operations (selection, crossover, mutation, etc.) are performed on a master node (computational unit). The MSP is suitable both for a generational approach, as well as for an asynchronous parallelization approach similar to the steady-state selection scheme described in [6]. The question of which of these two simple parallelization schemes is better, is an age old problem in the field of evolutionary computation. Very recently, Scott and De Jong started to analyze the problem more thoroughly [7]. In the present study we aim to offer some interesting insights into this matter from the general point of view of multi-objective evolutionary algorithms and from the particular perspective of trying to choose the best parallelization method when trying to optimize design parameters in electrical drive engineering. The findings of our analysis show that for two industrial problems, an overall computation time speedup of about 12% and 25% can be achieved by simply using an asynchronous MSP instead of a classical generational MSP (like the one applied in [5]). This improvement of the overall duration of the optimization seems to have no negative impact on the quality of the final solutions (i.e., of the PNs) the MOEAs are able to discover.

1.2. *Basic Concepts*

When considering a computational process parallelized / distributed on a general master-slave architecture, one should distinguish among two types of tasks:

(i) **remote tasks** - very *time-intensive computations* that are performed on the slave nodes;

(ii) **sequential tasks** - include all the computations that must be performed on the master-node in order to *create*, *dispatch* and *retrieve* remote computation tasks;

It should be noted that in the case of most MOEAs, the *"create"* part of the sequential tasks includes applying typical genetic operators like (parent) selection, crossover, mutation, and survival for selection. Apart from these, in real-world master-slave parallelization setups for MOEAs, the duration of the sequential tasks is also affected by the fact that lengthy pre-evaluation steps must be performed locally (on the master node) for each generated individual, before dispatching the individual for remote fitness evaluation on the slave nodes. These pre-evaluation steps must be performed on the master node because of security concerns, software licensing issues, network configuration settings, etc. Whenever the average duration of the sequential tasks carried out on the master node is significant with regard to the average duration of the fitness evaluation tasks (i.e., the system displays a low **parallelization ratio**), the speed-up that can be achieved by employing a parallel / distributed architecture is affected (q.v. Amdahl's law).

Apart from the parallelization ratio, another aspect that must be considered refers to the **heterogeneity of the time-wise distributions of the** remote (i.e., **fitness evaluation** in the case of MOEAs) and sequential tasks. Although literature that focuses on the effects of fitness function time-wise heterogeneity on the MSP choice for MOEAs is scarce, a study by Yagoubi et al. from 2011 [4] indicates that, for MOOPs that display a heterogeneous (non-constant) time-wise fitness distribution, the steady state asynchronous parallelization is somewhat better in terms of convergence (Pareto quality and global run-time) than the generational approach. In their 2008 work [8], Durillo et al. also show evidence that applying a (synchronous) steady state approach when performing a MOEA run can bring improvements in terms of Pareto quality. The present research builds on these earlier findings, considerably extends and generalizes preliminary concepts and results reported in [9], and tries to *outline the main reasons that might influence the average performance* of the two MSP methods in the context of MOEAs.

Our main intention is to provide an analytical framework to help practitioners in this field to decide what is the most efficient parallelization option based on the particularities (achievable parallelization ratio, MOEA choice, MOEA parameterization, MOOP characteristics, etc.) of their concrete optimization scenarios. The comparison is focused on the very practical aspect of achievable Pareto quality / run-time performance. In other words, given a MOOP to solve, a MOEA to use and two very simple and fast-to-implement (master-slave) parallelization options, we want to know *which parallelization method is more likely to deliver the highest quality solution set in a pre-defined global run-time interval?*

In the next paragraphs we provide a compact description of the two MSP methods we consider in the present research and we explain why the parallelization choice can seriously impact the MOEA search (performance) behavior. Together with a motivation for our dual analysis of parallelization performance, in Section 2.3 we provide an outline of the rest of this work.

2. Methodology

2.1. *The Considered Multi-Objective Evolutionary Algorithms*

For the purpose of this work we used two of the most well known ("classic") MOEAs: the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [10] and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [11]. At a high level of abstraction, NSGA-II and SPEA2 are different MOOP orientated implementations of the same concept: the $(\mu + \lambda)$ evolutionary strategy – where μ denotes the parent population size and λ the offspring population size. The two main features of both algorithms are

(i) a highly elitist approach that aims to store the best individuals found during the run;
(ii) a two-tier selection for survival function that uses a primary Pareto non-dominance metric and a secondary solution density estimation metric.

More recent MOEAs also explore concepts like differential evolution (DE) and objective space decomposition [12]. The trend to explore the very good search performance exhibited by differential evolution operators [13] was started with algorithms (like DEMO [14] and GDE3 [15]) that simply replaced the SBX and polynomial mutation operators used by NSGA-II and SPEA2 with various DE variants. Later, DE was also used to improve evolutionary approaches (see MOEA/D [16]) that aim to solve MOOPs by performing a decomposition of the original problem into several single objective optimization problems (e.g., as proposed in the Normal Boundary Intersection Method [17]) that are to be solved simultaneously. Recently, by applying a coevolutionary strategy, DECMO2 [18] has been shown to efficiently combine multiple MOEA design characteristics (e.g., Pareto-based selection for survival, differential evolution, decomposition strategies) in order to generally deliver a fast convergence behavior on a wide set of benchmark MOOPs.

Although on several (artificial and industrial) MOOPs, the previously mentioned four algorithms have been shown to display a better convergence speed and / or final Pareto quality performance than NSGA-II and SPEA2, the same general $(\mu + \lambda)$ evolutionary strategy can also be used to describe these more recent approaches. Therefore, the analysis that we propose can easily be applied to these algorithms as well. The particular choice of the MOEAs used for the tests reported in Section 4 has been made based on the fact that NSGA-II and SPEA2 exhibit a generally good optimization performance for real-life problems and,

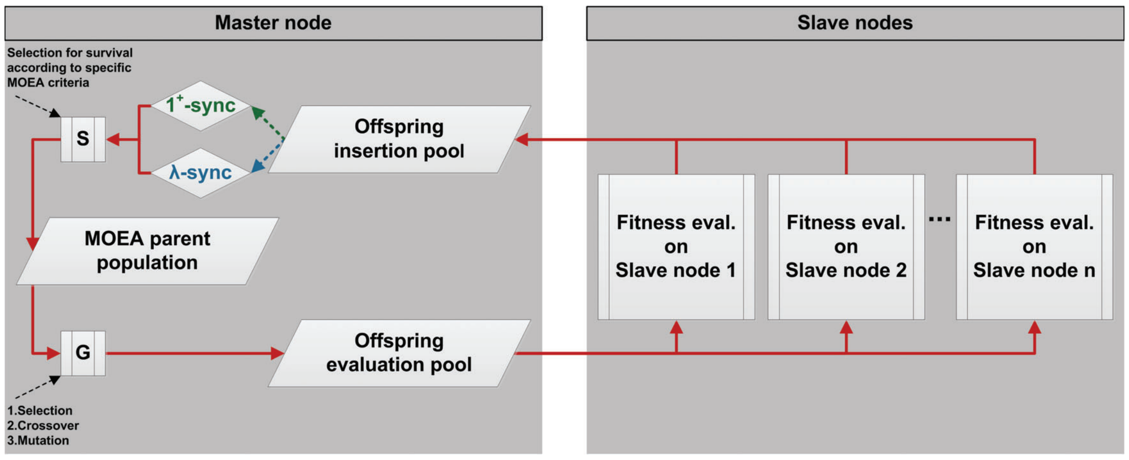
Master node
Slave nodes
Selection for survival according to specific MOEA criteria
S
1⁺-sync
λ-sync
Offspring insertion pool
MOEA parent population
G
Offspring evaluation pool
1.Selection
2.Crossover
3.Mutation
Fitness eval. on Slave node 1
Fitness eval. on Slave node 2
Fitness eval. on Slave node n

Fig. 1. Diagram of the GEN-MSPS (the λ-sync block) and SSA-MSPS (the 1^{+}-sync block) computation cycles

according to citation counts, they are still widely used methods. Most likely, these two approaches are the best known multi-objective evolutionary algorithms.

2.2. *Master-Slave Parallelization Schemes*

The diagram in Figure 1 provides a general explanation of the computation cycles generated by both master-slave parallelization schemes (MSPS) when applying them on a generic $(\mu + \lambda)$ MOEA.

Using as reference the computational components and steps presented in Figure 1, we shall now proceed to describe in more detail the computational cycles induced by the two master-slave parallelization methods. Before starting, we mention that regardless of the chosen MSPS (and the associated computation cycle), the $(\mu + \lambda)$ MOEA can be initialized using an empty parent population by randomly creating λ individuals and inserting them into the *offspring evaluation pool*.

2.2.1. *The Generational Master-Slave Parallelization Scheme (**GEN-MSPS**)*

In this case, the computation cycle is regulated by a λ-*synchronization step*. This step occurs before the integration of the individuals from the *offspring insertion pool* into the *MOEA population*. The master node must block until all the λ individuals of the current offspring population have been evaluated on the slave nodes. After this requirement is satisfied, the specific $(\mu+\lambda)$- selection for survival operation (i.e., S) is used in order to update the *MOEA population* on the master node.

Afterwards, all the λ offspring of the next generation are created sequentially on the master node using the specific genetic operators (i.e., G) and inserted into the *offspring evaluation pool*. Each slave node asynchronously selects an individual from this pool, evaluates it and afterwards places the results in the *offspring insertion pool*. The above described procedure is repeated until the optimization stopping criterion is met.

From an algorithmic point of view, this computation cycle is identical to a sequential $(\mu + \lambda)$ implementation.

2.2.2. *The Steady-State Asynchronous Master-Slave Parallelization Scheme (**SSA-MSPS**)*

For the steady-state asynchronous parallelization, the computation cycle is regulated by a 1^+-*synchronization step* that also occurs before the selection for survival (i.e., S) operation.

The slave nodes function in the same way as in the generational parallelization scheme. The master node is again controlled by a simple loop. While the stopping criterion is not met, the master node first checks if there are any evaluated offspring in the *offspring insertion pool* and, if such individuals exist, it collects them and updates (via the selection for survival genetic operation) the *MOEA parent population*. This is the main difference to GEN-MSPS, which in each cycle has to wait for the evaluation of *all* offspring before new individuals can be generated. Secondly, using the specific genetic operators (i.e., G), the master node creates as many new offspring as it has previously collected and immediately inserts them into the *offspring evaluation pool*.

The above computation cycle resembles classical steady-state selection as, at a given time, usually, only one evaluated offspring is collected and, if *"fit enough"*, it is inserted into the *MOEA parent population* and another offspring is generated. It is important to notice that the SSA-parallelization scheme drastically changes the algorithmic behavior of the given MOEA to that of an asynchronous $(\mu + 1^+)$ evolutionary strategy.

2.3. *Outline and Observations*

Figure 2 provides a sketch / didactic example of how individuals are processed by the two parallelization methods. The more flexible synchronization step of SSA-MSPS enables this method to evaluate more individuals per time interval than GEN-MSPS (e.g., in Figure 2 SSA-MSPS could evaluate 4 more offspring in the remaining time interval). We shall investigate this **quantitative aspect** in Section 3. The downside of using SSA-MSPS is that, intuitively, the same lack of generational synchronization is expected to make SSA-MSPS achieve worse results in terms of attained Pareto front quality after evaluating a fixed number of individuals. This **qualitative aspect** is investigated on 15 artificial MOOPs in Section 4. The last part of Section 4 contains an interpretation of the interplay between the quantitative and qualitative observations when considering two very computationally-intensive optimization scenarios from the field of electrical drive design. Section 5 contains the conclusions and an outlook on open research lines.

In order to better explain the logic behind the quantitative and the qualitative analyses, let us assume that in a given (wall-clock) time interval T a MOEA that is parallelized with GEN-MSPS can compute X_{GEN} individuals and reach a PN of quality Q_{GEN}. In this case:

– the quantitative analysis tries to determine (the factors that influence) X_{SSA} – the number of individuals that can be computed in T by applying SSA-MSPS instead of GEN-MSPS;

– the qualitative analysis tries to determine X_{req} – the number of individuals that must be computed when using SSA-MSPS in order to reach a PN of quality Q_{GEN}.

If $X_{req} < X_{SSA}$ it clearly follows that SSA-MSPS is a better parallelization option (when the duration of the optimization is limited to T). Otherwise, GEN-MSPS should be preferred. Understanding and modeling the interplay between X_{req} and X_{SSA} over the entire optimization run can provide a big help to practitioners who want to make a good MSP choice. In the case of single-objective evolutionary algorithms, a very interesting approach regarding this matter can be found in [7].

Generally, we consider that performing the parallelization performance analysis from this dual point of view is very useful because:

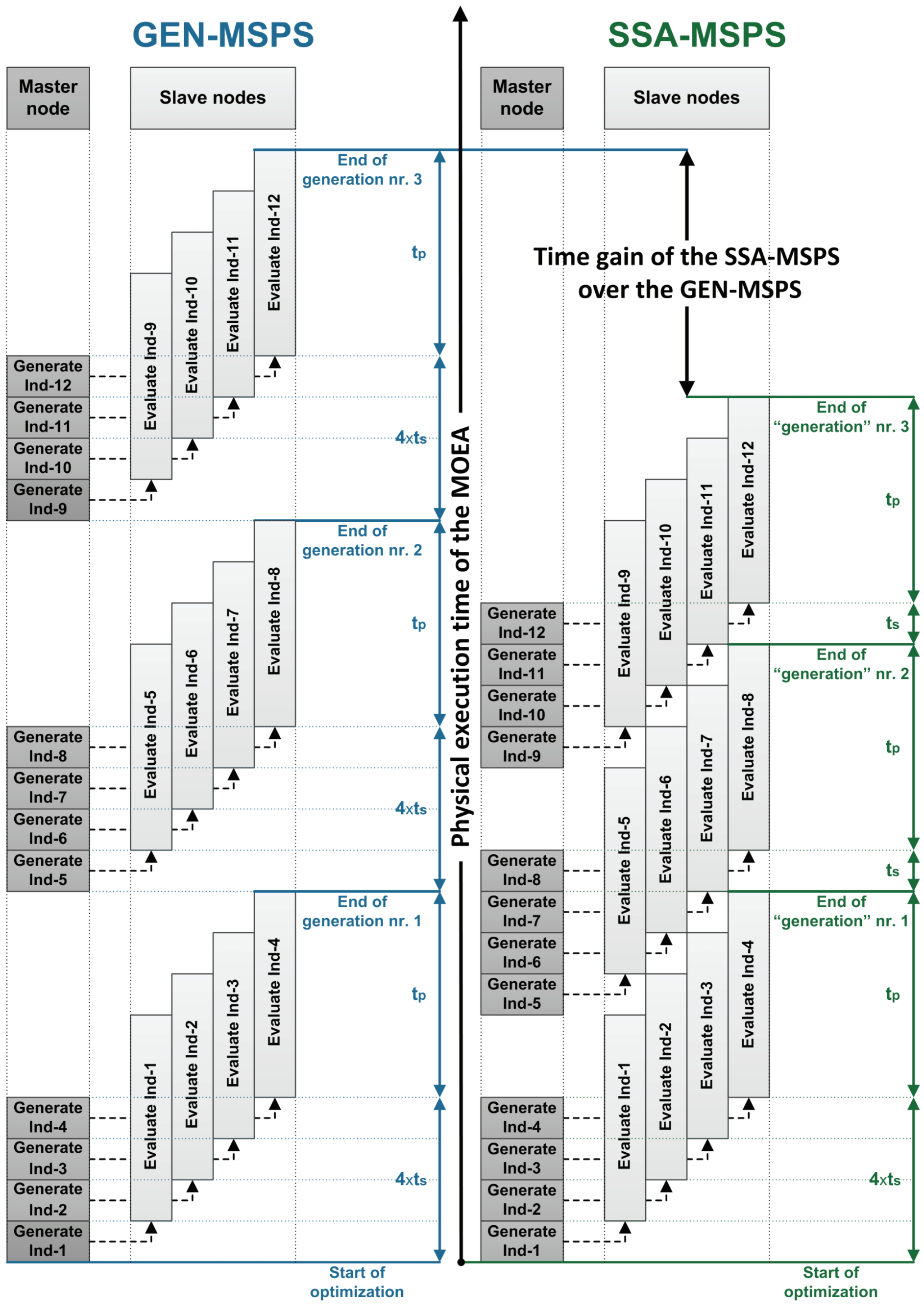

Fig. 2. The comparative computation steps of GEN-MSPS and SSA-MSPS for 3 generations of size 4 in a distributed computing environment with one master node and 4 slave nodes

(i) the quantitative aspects tend to be related more to the physical and software constraints of the available parallel/distributed computing architecture (i.e., they are more or less fixed or hard to change for a practitioner);

(ii) the qualitative aspects are mainly dependent on the chosen MOEA, the chosen algorithm parameterization, and the complexity of the actual MOOP to be solved.

As such, the qualitative aspects exhibit a higher variability as two of the factors that directly influence them (i.e., type of MOEA and MOEA parameterization) can be selected freely by the practitioner.

Furthermore, the quantitative and qualitative analyses have the major role of abstracting the dependency of the overall parallelization performance on the underlying hardware performance as both analyses only consider the number of evaluated individuals. This abstraction should make results obtained in different parallelization environments more comparable and thus meaningful.

3. Examining the Quantitative Performance

3.1. *The Basic Model*

As mentioned in the previous section, it is obvious that, given a fixed hardware architecture and identical MOEA settings, SSA-MSPS is able to compute a fixed number of individuals faster than GEN-MSPS (please see Figure 2 for a sketch). In other words, when using the SSA-MSPS, one is likely to create and evaluate more individuals in the same time interval. In this section, we attempt to quantify this improvement in number of computable individuals and to evaluate how the interplay between the duration of the remote fitness evaluation tasks and the duration of the local sequential tasks affects it.

The theoretical model consists of a $(\mu + \lambda)$ MOEA that is parallelized / distributed over a computing environment with more than λ slave nodes (i.e., we assume that the available number of slave nodes is not the bottleneck of the setup). We mark with $t_p > 0$ the duration (in time units) of distributing and performing the fitness evaluation of any individual on any slave node (i.e., the duration of the remote computation tasks). We also mark with $t_s > 0$ the cumulative duration of the sequential computation tasks (i.e., genetic operations + possible pre-evaluation tasks) that are performed on the master node in order to create one individual. For the time being, we assume that t_s and t_p are constant (i.e., we have a homogeneous time-wise distribution of both the fitness evaluation and the individual creation functions). The *parallelization ratio* is defined as:

$$r = \lceil \frac{t_p}{t_s} \rceil \tag{1}$$

Under the above mentioned restrictions, when considering the GEN-MSPS approach, it is quite straightforward that it needs not use more than $r + 1$ slave nodes simultaneously as the first slave will finish its fitness evaluation by the time individual number $r + 2$ is generated on the master node . The reasoning in this section is made under the restriction $r + 1 \geq \lambda$ and that there are more that λ slave nodes available for performing the remote fitness evaluations.

Assuming that other miscellaneous computation times are negligible with regard to (or integrated in) t_s and t_p, the total time required to compute any generation of λ individuals using the GEN-MSPS is $(\lambda \times t_s) + t_p$. In case of the SSA-MSPS, the time required to compute the first λ individuals is also $(\lambda \times t_s) + t_p$, but the time required to compute any of the next batches of λ individuals is $(t_s + t_p)$, as sketched in Figure 2. Therefore, when wishing to compute N generations, the overall computation time is *a)* $(\lambda \times t_s + t_p) \times N$ in the case of GEN-MSPS; *b)* $(\lambda \times t_s + t_p) + (t_s + t_p) \times (N - 1)$ in the case of SSA-MSPS. After equalizing these computation times and performing the necessary calculations, we have that in the time interval required by the GEN-parallelization to compute N generations of λ individuals, the SSA-parallelization can compute $\Delta_{struct}\%$ more individuals, where Δ_{struct} is given by:

$$\Delta_{struct} = \frac{(N-1) \times (\lambda - 1) \times t_s}{N \times (t_s + t_p)} \times 100 \tag{2}$$

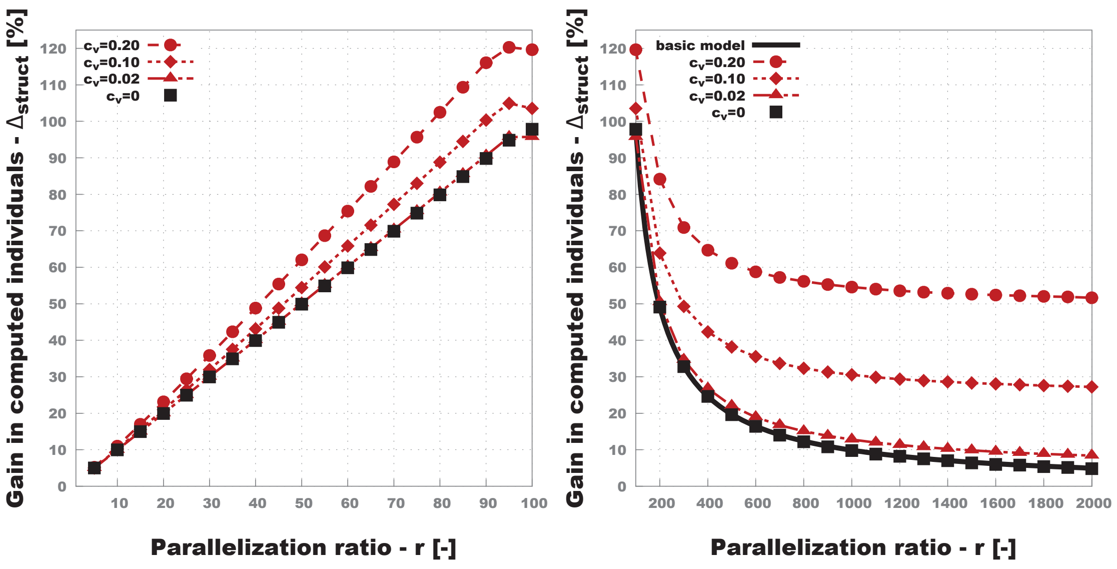

Fig. 3. Δ_{struct} plots for different parallelization ratios and different degrees of variance (i.e. c_v) in the time-wise distribution of the fitness evaluation function

We shall refer to this measure as the **structural improvement** that SSA-MSPS has over GEN-MSPS in terms of computed individuals per given time interval.

It is important to note that while Δ_{struct} does depend on the number of generations to be computed, the dominant factors that influence Δ_{struct} are the ratio between t_s and t_p (i.e., the parallelization ratio r) and the population size (i.e., λ). When dealing with real life, very time-intensive MOOPs, a rather small choice of λ (i.e. 50 to 250) is usually the norm.

When fixing $\lambda = 100$, $N = 500$, and $t_s = 1$, by varying the value of t_p, we can compute the dependency of Δ_{struct} on r. The corresponding plot is presented in Figure 3: the left plot - basic model curve. Unsurprisingly, it shows that *the quantitative improvement* that SSA-MSPS brings *decreases exponentially with the parallelization ratio.*

Although valuable in establishing a baseline for the comparison between GEN-MSPS and SSA-MSPA, the above described comparison has one severe limitation: it is strongly influenced by the idealistic assumption that the duration of the fitness evaluation tasks is constant. As such, in Section 3.2 we proceed to address this issue in order to improve our quantitative performance model.

3.2. *The Effect of Variance on the Quantitative Performance*

At first, we performed tests in order to validate the theoretical model proposed in the previous section. Using a homogeneous (i.e. constant / variance free) time-wise fitness distribution, we simulated the time required by GEN-MSPS and SSA-MSPS runs when considering various values of the parallelization ratio (1) for the same settings ($\lambda = 100$, $N = 500$, and $t_s = 1$) used in Section 3.1. The obtained results (Figure 3 - the $c_v = 0$ data points for $r > 100$) confirm the Δ_{struct} behavior indicated by the theoretical model from (2). Furthermore, the simulation also allowed us to easily estimate Δ_{struct} for values of r smaller than λ (i.e., the left plot). In this case, Δ_{struct} displays a linear behavior that is directly proportional to r.

Next, we wanted to quantify the influence of having ever larger degrees of heterogeneity (i.e. variance) in the time-wise distribution of the fitness evaluation function. Therefore, in the next series of tests, the

Table 1
The observed variance-specific lower thresholds of Δ_{struct}

c_v [-]	Lower threshold for Δ_{struct} [%]	Δ_{struct} for $r = 10^6$ [%]
0.20	48.9700	49.1040
0.10	24.4500	24.5300
0.05	12.1800	12.2140
0.02	4.8140	4.8300

fitness evaluation of each individual took t_p milliseconds where $t_p \sim \mathcal{N}(m, \sigma)$. By fixing $t_s = 1$, we have that $r \sim m$ and, when scaling up m we also modified σ in order to keep the coefficient of variation, $c_v = \frac{\sigma}{m}$, constant at preset values. Using this simple technique we were able to effectively control both the amount of variation in the time-wise distribution of the fitness function and the parallelization ratio. The maximum amount of variance that we could consider under the normal distribution assumption was given by $c_v = 0.2$ - a higher value would result in sampling negative values (which makes no sense for a duration). Because of the induced stochasticity, for each value of r we performed 100 tests and we report averaged results.

The plot in Figure 3 shows how Δ_{struct} behaves for four different variance levels (i.e., c_v). The curves clearly indicate that the exponential decrease of Δ_{struct} is dampened by increased levels of variance. Further experiments have also shown that for $(r > \lambda)$, when having variance in the time-wise fitness distribution function, after reaching a lower threshold, the value of Δ_{struct} tends to stabilize. We have run simulations up to $r = 10^6$ with a step size of 500 and, in Table 1, we report the lower thresholds of Δ_{struct} for different variance levels. We mention that, in the absence of variance, for $r > 49500$, $\Delta_{struct} = 0.0\%$ because, although SSA-MSPS computes the required 50000 individuals faster than GEN-MSPS, no extra individual can be computed when using SSA-MSPS in the remaining time interval.

In conclusion, the theoretical model given by Equation (2) gives an accurate lower limit for Δ_{struct} but the value of Δ_{struct} for a given parallelization ratio r is significantly higher when having variance in the time-wise distribution of the fitness function. Furthermore, in the presence of variance, Δ_{struct} is lower bounded by variance-specific thresholds that display a remarkable stability even at very high values of r.

4. Examining the Qualitative Performance - Empirical Results

4.1. *Evaluation Framework Setup*

The qualitative performance of the two considered master-slave parallelization schemes is harder to quantify as it depends on the concrete MOOP to be solved (i.e., the complexity of the fitness landscape), on the used MOEA and on the parameterization of the algorithm. In the following paragraphs we describe the details of the performance evaluation framework we propose in order to estimate the qualitative performance.

4.1.1. *Test Problems*

We have chosen for benchmarking purposes 15 well known artificial test problems from multi-objective literature. Our choice of artificial problems is self-evident as it is very helpful to know the ground truth (i.e., the PS of the MOOP and its associated *true Pareto front*) in order to compare between parallelization performances. The problems have been specially selected to propose different degrees of difficulty and, subsequently, different convergence behaviors for the two MOEA algorithms that we experiment with. The 15 MOOPs we consider in this study are:

- **DTLZ1**, **DTLZ3**, and **DTLZ7** from the problem set proposed in [19]. These problems feature 7, 12 and 22 variables and 3 objectives.
- **KSW10 -** a classic multi-objective optimization problem that is based on Kursawe's function [20]. The problem has 10 variables and 2 objectives.
- **LZ09-F1**, **LZ09-F2**, **LZ09-F3**, **LZ09-F4**, **LZ09-F5**, **LZ09-F8**, and **LZ09-F9** from the LZ09 problem set [16], a set which is known to be particularly difficult for classic MOEAs like NSGA-II and SPEA2. LZ09-F8 has 10 variables while the other problems have 30. All the selected problems from this set feature 2 objectives.
- **WFG1** and **WFG7** from the problem set proposed in [21]. Both problems have 6 variables and 2 objectives.
- **ZDT3** and **ZDT6** from the problem set described in [22]. Both problems feature 10 variables and two objectives.

The computation of the fitness values for all 15 problems is very fast on any modern processor. In order to make the MOEAs exhibit the desired test behavior, the fitness computation times were artificially increased using the method described in Section 3.2.

4.1.2. *MOEA Parameterization*

Our GEN and SSA implementations of both NSGA-II and SPEA2 are based on the ones provided by the jMetal package [23]. As the main goal is to analyze the comparative performance of two different parallelization models for the same algorithm, we used standard (literature recommended) parameterization options for the tested algorithms.

For all the performed tests, the parent population (i.e., archive in the case of SPEA2) size was set at 100 individuals and we used an offspring population size of 100. In the case of SSA-MSPS the term "generation" is used to denote a batch of 100 individuals. We used the standard genetic operators recommended for NSGA-II and SPEA2: binary tournament selection, simulated binary crossover [24] and polynomial mutation. These operators were parameterized using standard values: 0.9 for the crossover probability, 20 for the crossover distribution index, $1/D$ for the mutation probability (where D is the number of variables of the MOOP to solve) and 20 for the mutation distribution index.

Our comparisons are made using runs of 500 generations which means we perform 50000 fitness evaluations / run. This is because after 500 generations we reach generally good solutions for almost all the considered test problems and because time constraints are very important in many real-world industrial optimization scenarios and practitioners rarely have the time to run an optimization even for a few hundred generations.

Furthermore, we are particularly interested in studying the early and middle-stage convergence behavior of MOEAs when applying GEN-MSPS or SSA-MSPS. This is because in a real-life optimization scenario, that is time-constrained, a practitioner is likely to stop the optimization process as soon as he/she notices that small improvements come come at an ever increasing computational cost (i.e. the MOEA enters the late-stage of convergence). Nevertheless, if one particular parallelization method is constantly outperforming the other in the late-stage of convergence, given the descriptions from Section 2.2, one can easily change during the optimization to the best performing parallelization option by simply switching between the λ and 1^+ synchronization steps.

Given the stochastic nature of MOEAs, for each comparative test that was performed, we made 100

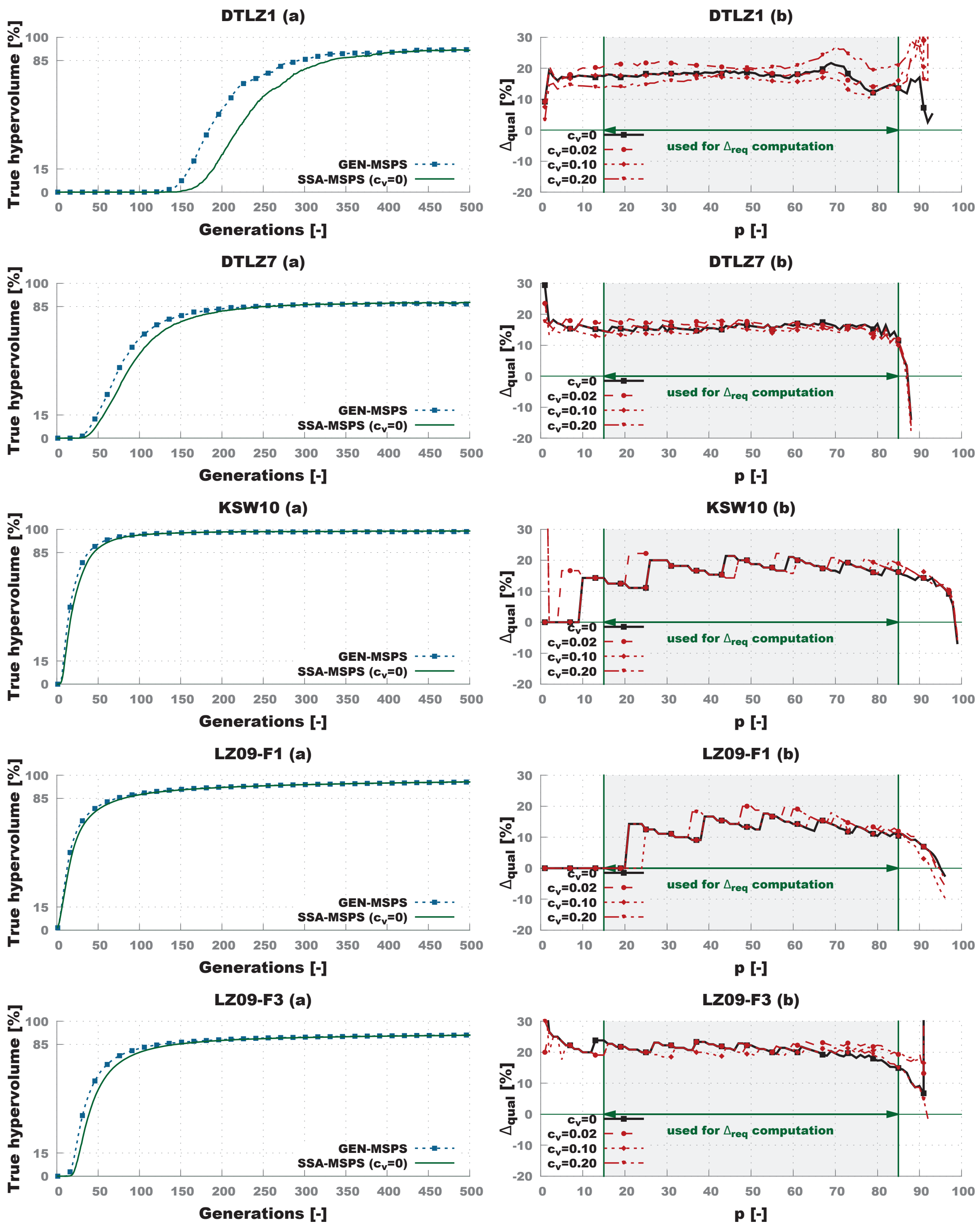

Fig. 4. NSGA-II qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

repeats of each experiment (i.e., MOOP-MOEA run) and we always report the averaged results.

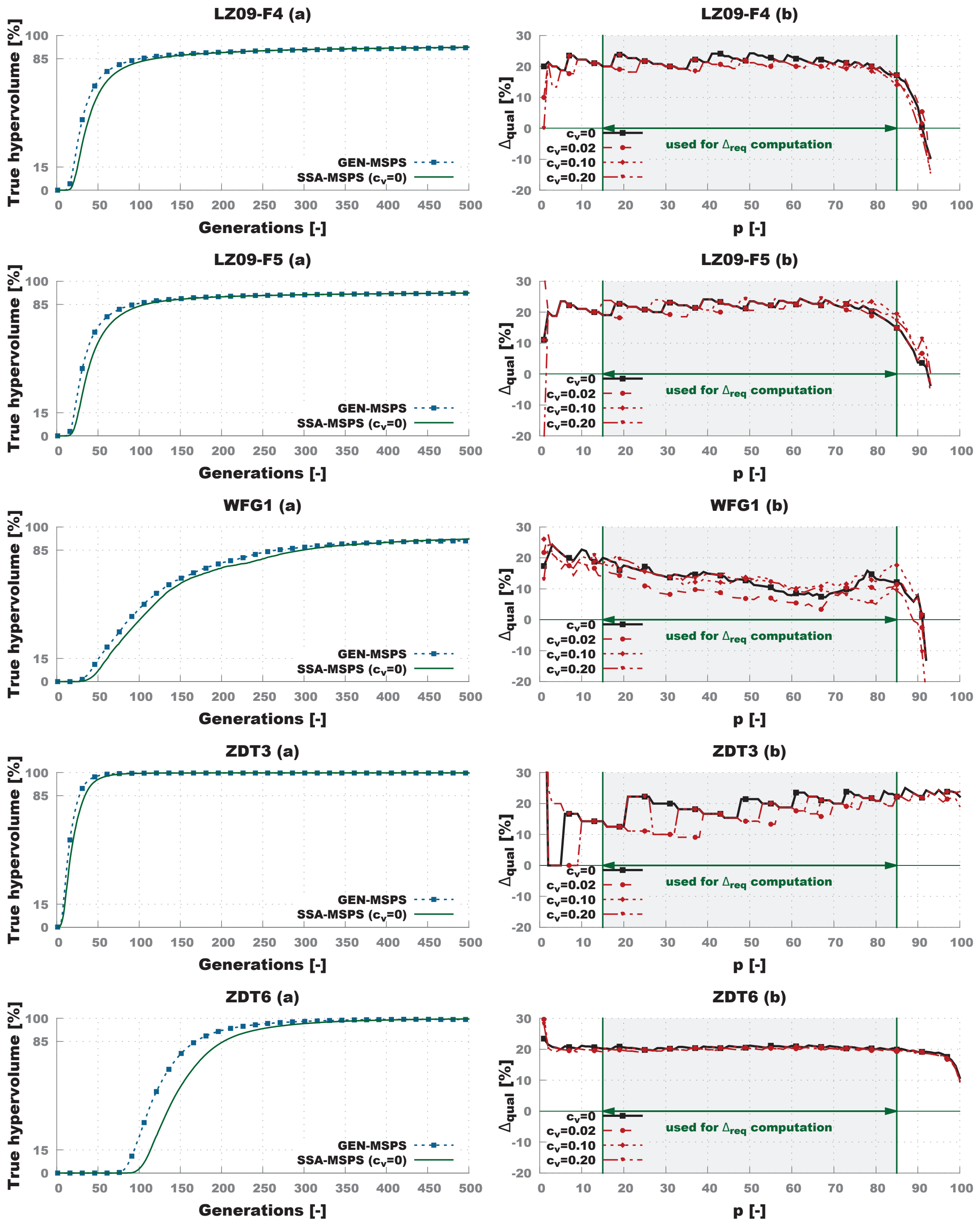

Fig. 5. NSGA-II qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

4.1.3. *Quality Indicators*

For a given solution set S, the hypervolume associated with this solution set, $\mathcal{H}(S)$, has the advantage that it is the only Pareto front quality estimation metric for which there is a theoretical proof [25] of a monotonic behavior. This means that the maximization of the hypervolume constitutes the necessary and

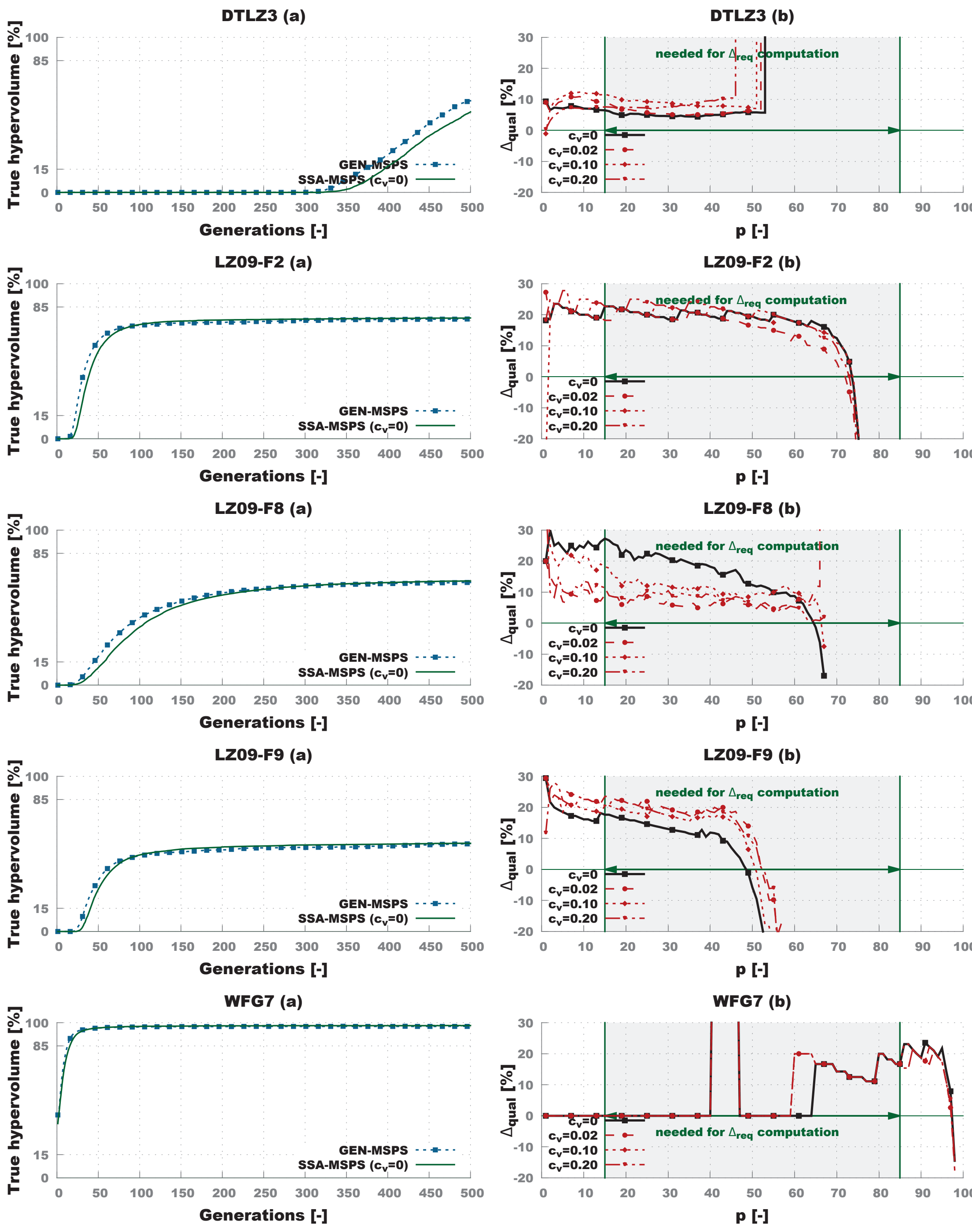

Fig. 6. NSGA-II qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

sufficient condition for the set of solutions to be *"maximally diverse Pareto optimal solutions of a discrete, multi-objective, optimization problem"* [25]. In light of this, for any optimization problem, the *true Pareto front* has the highest achievable hypervolume value.

In our case, for a given MOOP, the monotonic property of the hypervolume metric makes it ideal for assessing the relative quality of an arbitrary solution set S_a. Let us mark with S_{true} the *true Pareto front* of our MOOP. As we deal with artificial problems where S_{true} is known, we can present the quality of the given solution set as a percentage obtained by reporting the hypervolume measure of this solution set to the hypervolume value associated with the true Pareto front of the given MOOP:

$$qual(S_a) = \frac{\mathcal{H}(S_a)}{\mathcal{H}(S_{true})} \times 100 \tag{3}$$

Expressing the quality of a solution set as a true hypervolume percentage also enables us to define more accurately what we mean by the syntagms early, middle and late-stage of convergence. For the purpose of this research, based on the performance of NSGA-II and SPEA2 on the 15 artificial MOOPs and in light of the motivations presented in the last paragraph of Section 4.1.2, we arbitrarily define a MOEA as being in the early stage of convergence if $qual(\mu_c) \leq 15$ where μ_c denotes the current parent population of the MOEA. If $qual(\mu_c) \in (15, 85]$ we consider the algorithm to be in the middle-stage of convergence, while $qual(\mu_c) > 85$ is associated with a late-stage of convergence.

In order to perform our qualitative analysis, we proceed to introduce a new measure based on the relative hypervolume. Consider we wish to solve our MOOP with a certain MOEA. Let: *a)* $p = \overline{1, 100}$ be an integer value; *b)* $C_{GEN}(p)$ be the minimal number of individuals that must be computed when using GEN-MSPS in order to reach a solution set S_1 with the property that $qual(S_1) \geq p$; *c)* $C_{SSA}(p)$ be the minimal number of individuals that must be computed when using SSA-MSPS in order to reach a solution set S_2 with the property that $qual(S_2) \geq p$. For our MOOP-MOEA combination, we define the *SSA qualitative deficit* at the true hypervolume percentage p as:

$$\Delta_{qual}(p) = \left(\frac{C_{SSA}(p)}{C_{GEN}(p)} - 1 \right) \times 100 \tag{4}$$

This $\mathcal{H}$-derived measure is designed to show the relative difference in the minimum number of individuals that must be computed when using the two parallelization schemes in order to reach a solution set S_p with the property that $qual(S_p) \geq p$.

4.2. *Basic Qualitative Performance Tests*

In the first series of performed tests, using a constant fitness distribution (i.e. $c_v = 0$), we measured the quality of the MOEA parent population in terms of hypervolume after the completion of each GEN-MSPS generation / batch of 100 individuals in the case of SSA-MSPS. These results (obtained with NSGA-II) are presented in the left subplots [marked with (a)] from Figure 4, Figure 5, and Figure 6. We consider that a more useful perspective for presenting the same comparative convergence behavior information can be constructed by plotting the Δ_{qual} values of the MOEA parent population as shown (the $c_v = 0$ lines) in the subplots marked with (b) from the previously mentioned figures.

The first important observation is that for 10 of the 15 test problems, more precisely for those presented in Figure 4 and Figure 5, the NSGA-II displays a good convergence behavior as it bypasses the initial and middle stages of convergence and is able to reach the final stage of convergence (i.e., $p > 85$, where p denotes a relative true hypervolme percentage). On the 5 MOOPs presented in Figure 6, the convergence behavior is different:

- for DTLZ3 - the MOEA is only able to reach p values of ≈ 40 after 50000 fitness evaluations (continuing the run would eventually enable it to reach a late stage of convergence);

- for LZ09-F1, LZ09-F8, and LZ09-F9 - the MOEA seems unable (with the given settings) to reach p values > 85 and the charts indicate premature convergence;
- for WFG7 - the MOEA is able to reach the late stage of convergence, but random initializations of the initial population already display relative p values of ≈ 40;

For the remaining of this work we shall refer to the 10 MOOPs from Figure 4 and Figure 5 as the **successfully solved problems** and to the 5 problems from Figure 6 as the **special case problems**.

The results of these initial tests confirm some of the findings from [26], in the sense that, the GEN-MSPS is able to achieve a higher quality Pareto front than SSA-MSPS after the same number of evolved individuals in the early and middle stages of convergence for all the 10 successfully solved problems. This observation also holds for 4 of the special case problems. In the case of LZ09-F8, the graphs indicate that SSA-MSPS has a better performance when wanting to reach p-values > 30.

Furthermore, when abstracting the behavioral shifts and numeric artifacts that characterize the early and late stages of convergence, we notice that Δ_{qual} values are quite constant (within a 10% range) for each successfully solved test problem. When also considering the special case problems, we can state that, **in general, Δ_{qual} values do not display a trend that increases with p.**

Considering the somewhat constant behavior of Δ_{qual} for the successfully solved MOOPs, we can construct an additional metric. Therefore, by averaging the individual Δ_{qual} values associated with the middle stage of convergence (i.e., $p = \overline{16, 85}$), we obtain the **average required SSA improvement** for a MOOP-MOEA combination:

$$\Delta_{req} = \frac{1}{70} \sum_{p=16}^{85} \Delta_{qual}(p) \tag{5}$$

The new Δ_{req} metric is important as it is a rough indicator of how many more individuals/time interval SSA-MSPS must compute in order to mach the results that would be produced by GEN-MSPS in the given time frame.

4.3. *The Effect of Variance on the Qualitative Performance*

The second series of tests that we have performed in order to gain more insight into the qualitative performance of the two parallelization schemes is again related to the influence of having variance in the time-wise fitness distribution function.

The results obtained using NSGA-II are also presented in the subplots marked with (b) from Figure 4, Figure 5, and 6. A quick look over all 15 plots reveals that the effect of variance is not so important in the case of the qualitative performance.

The values of Δ_{req} for the successfully solved problems for both NSGA-II and SPEA2 are shown in Table 2. Given the formulation of Δ_{req} from 5, the metric can not be computed for the special case MOOPs. We mention that when using SPEA2, WFG1 becomes a special case problem and, as such, computing Δ_{req} values for the SPEA2-WFG1 combination does not make sense. Nevertheless, the data from Table 2 clearly indicates that, in the case of the qualitative performance, variance in the time-wise distribution of the fitness evaluation function has a negligible effect as:

- Δ_{req} is not directly proportional to the amount of variance and for 7 out of the 19 MOOP-MOEA combinations, the highest average Δ_{req} value corresponds to the experiment with zero variance (i.e., $c_v = 0$);

Table 2
Averaged values of the Δ_{req} metric over 50 runs for different levels of variance in the time-wise distribution of the fitness evaluation function. For each MOOP-MOEA combination, the highest value is highlighted and marked with "^{S}" if the difference between it and the lowest Δ_{req} value of the combination is statistically significant.

	Δ_{req} for **NSGA-II** [%]				Δ_{req} for **SPEA2** [%]			
Problem	$c_v = 0$	$c_v = 0.02$	$c_v = 0.10$	$c_v = 0.20$	$c_v = 0$	$c_v = 0.02$	$c_v = 0.10$	$c_v = 0.20$
DTLZ1	17.61	18.94	16.28	19.30	10.17	8.20	9.48	11.40
DTLZ7	15.61	16.54^S	14.32	15.25	15.41	13.12	12.90	15.96^S
KSW10	17.22	18.35	17.33	17.19	18.95^S	15.94	13.84	14.77
LZ09-F1	12.36	13.81	11.80	14.03	13.25	13.02	12.63	11.93
LZ09-F3	20.45	21.67	20.35	20.82	17.42	17.11	17.54	14.19
LZ09-F4	21.54	20.24	20.24	19.84	13.59	14.42	14.47	14.38
LZ09-F5	21.73	20.62	22.20	22.37	16.92	15.54	16.51	16.44
WFG1	12.58	8.77	13.27	12.14	–	–	–	–
ZDT3	19.92^S	15.29	16.67	17.99	14.33^S	14.07	13.11	9.90
ZDT6	20.53	19.98	20.25	19.96	20.11	19.78	21.19^S	20.23

– in 13 out of 19 cases, the observed average changes induced on Δ_{req} by having some level of variance are not statistically significant. More precisely, we checked if the difference between the highest and the lowest Δ_{req} values for any MOOP-MOEA combination is statistically significant given a one-sided Mann-Whitney-Wilcoxon test with a considered significance level of 0.05.

All these observations create a stark contrast when comparing with the powerful effect that variance has on the quantitative performance (Section 3.2) and provide a solid indicator that **SSA-MSPS should be favored in the presence of significant variance in the time-wise distribution of the fitness function**.

Across all 15 test problems and regardless of the induced variance, our results indicate that **SSA-MSPS performs quite similar to (i.e., requires the same number of individuals to be computed as) GEN-MSPS in order to reach (good) solutions towards the end of the runs**. The average qualitative performance of SSA-MSPS is even marginally better (i.e., $\Delta_{qual} < 0$) for several problems (notably: DTLZ7, LZ09-F4, LZ09-F2, LZ09-F9, and WFG7). In particular, for the 10 successfully solved MOOPs, this means that it makes no difference (from the number of individuals that must be computed) which parallelization methods is applied when the MOEA enters the late stage of convergence (i.e., $p > 85$).

This identical qualitative performance exhibited towards the end of the runs and the generally stable behavior of the qualitative deficit exhibited by SSA-MSPS throughout the runs allows for the following reasoning regarding the comparative performance of GEN-MSPS and SSA-MSPS: *if, for a given optimization scenario, the quantitative improvement of SSA-MSPS (Δ_{struct}) can overcompensate the qualitative deficit of SSA-MSPS (Δ_{req}), we can say that, on average, SSA-MSPS is the better parallelization choice.* This is because when ($\Delta_{struct} > \Delta_{req}$) we have good reasons to believe that, in any middle-stage (and implicitly late-stage) convergence time interval, SSA-MSPS can produce Pareto fronts that are not worse than those that might have been obtained by using GEN-MSPS for the same time interval. In other words, the percentage of extra individuals that *can* be evaluated when using SSA-MSPS (i.e., Δ_{struct}) is larger than the percentage of extra individuals that *must* be evaluated (i.e., Δ_{req}) in order to reach Pareto non-dominated sets of similar quality.

4.4. *Analysis of Two Industrial Optimization Scenarios*

We have applied the previously described qualitative and quantitative analyses on two real-life multi-objective optimization scenario from the field of electrical drive design. In both cases, the motor topology features an interior rotor with embedded magnets.

The first industrial optimization problem (**IndMOOP1**) contains six variables that denote various geometric dimensions relevant to the rather simple topology illustrated in Figure 7. The goal of IndMOOP1 is to simultaneously optimize four (conflicting) objectives and thus obtain motor designs that display: (a) a high efficiency, (b) a low manufacturing price, low peak-to-peak motor torque values both at (c) load operation as well as for (d) no current excitation.

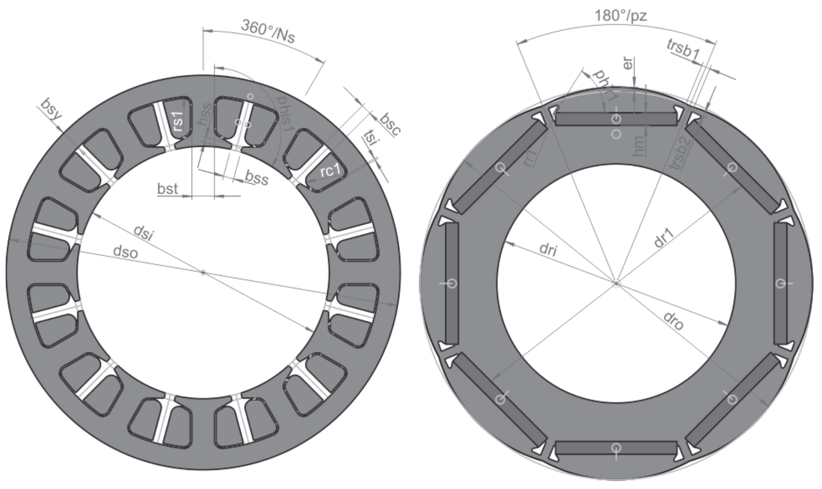

Fig. 7. Cross-sections with the geometric dimensions (variables) that are considered in the case of IndMOOP1 - an optimization problem based on an interior rotor topology with embedded magnets

The second industrial optimization problem (**IndMOOP2**) has 22 real-valued variables that must be configured in order to optimize four unconstrained objectives that regard efficiency and production prices. The performance of the entire assembly is always evaluated (i.e., simulated) at 3000 rotations per minute. Because of the dimension of the search space and the complexity of the underlying topology, IndMOOP2 is a very hard multi-objective optimization problem.

Estimating the quality (i.e., fitness) of a design actually means predicting the operational behavior of the electrical drive under certain conditions and, because of the non-linear behavior of the materials involved, such a prediction is usually based on performing (a series of) time intensive finite element simulations. Because of this, the entire optimization process is distributed over a computer cluster managed using HTCondor$^{\text{TM}}$[27].

For example, in the case of IndMOOP1, based on 10^5 samples, the average duration of the fitness evaluations tasks (i.e., t_p) is 391.94 seconds and the average duration of the sequential computation tasks (i.e., t_s) is 21.48 seconds. This leads to a rather small parallelization ratio of only 18.25. The reason for the rather high value and rather strange distribution (Figure 8 - left plot) of t_s lays with software and licensing restrictions: the software program that transforms the design parameter vector into usable 2D meshes for the FE

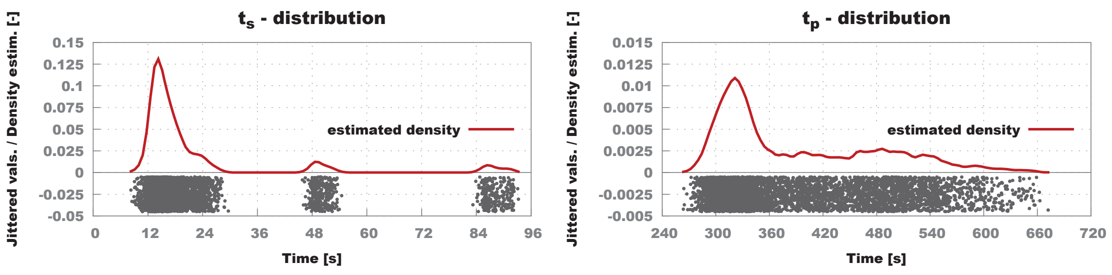

Fig. 8. Kernel density estimations of the time-wise distributions of the sequential (t_s) and of the remote (t_p) computation tasks for the analyzed industrial optimization scenario.

simulations is not licensed to run on the slave nodes. Also, after generating 10 to 15 consecutive meshes, the program needs to be restarted and this is a lengthy operation ($\approx$ 30 or 60 seconds) that directly translates into high t_s values. The distribution of t_p (Figure 8 - right plot) is also quite heterogeneous ($c_v = 0.23$) as the cluster computers used to perform the FE simulations have different (hardware) processing performances. After integrating all this data into the simulation framework described in Section 3.2, the results indicate that, on average, we should expect a Δ_{struct} of 19.47%.

In the case of IndMOOP2, initial measurements also indicate that we should expect a Δ_{struct} value of $\approx$ 20.00%. This should not come as a surprise, since we use the same parallelization environment in both cases.

In order to evaluate the qualitative and confirm our previous predictions of the qualitative performance, we performed:

– 15 SPEA2 optimization runs with each parallelization method in the case of IndMOOP1;
– 3 SPEA2 optimization runs with each parallelization method in the case of IndMOOP2.

Each time, the MOEA was parameterized using the same settings described in Section 4.1.2. As we do not know the true Pareto front of our industrial MOOPs, we have estimated the best attainable solution sets (i.e., S_{true} from (3)) for the problems using historic optimization results (from over 50 optimization runs for IndMOOP1 and 13 runs for IndMOOP2) and expert knowledge.

For IndMOOP1, over the 30 real-life runs, we measured a Δ_{struct} of 20.13% that is quite close to the prediction (19.47%). The qualitative performance plots are presented in top plots from Figure 9. Surprisingly, they show that after evaluating the same number of individuals, SSA-MSPS is able to produce better Pareto fronts than GEN-MSPS. After performing the averaging operation, $\Delta_{req} = -5.71\%$. The actual global run-times of the 30 real-life optimization runs for IndMOOP1 confirm the computed Δ_{struct} and Δ_{req} values as the SSA-MSPS runs are, on average, faster by $\approx$ 25%.

In the case of the IndMOOP2 runs, the obtained Δ_{qual} values are not constant but they do exhibit a clear downward trend as p increases. For example, the bottom right plot from Figure 9 shows that if one would want to obtain a Pareto non-dominated front that covers roughly 20% of the objective space dominated by the "true" solution of IndMOOP2 (i.e., $p = 20$), one would need to compute $\approx$ 50% more individuals when using SSA-MSPS. For PNs that cover $>$ 80% of the objective space dominated by the PS, one would only need to compute $\approx$ 12% more individuals when using SSA-MSPS.

As we mentioned earlier that our parallelization framework enables SSA-MSPS to compute $\approx$ 20% more individuals per time interval (i.e., $\Delta_{struct} \approx 20\%$), the reasoning is that, for this scenario:

– GEN-MSPS obtains better results for very time limited optimizations in which one can only compute a limited number of individuals (generations);

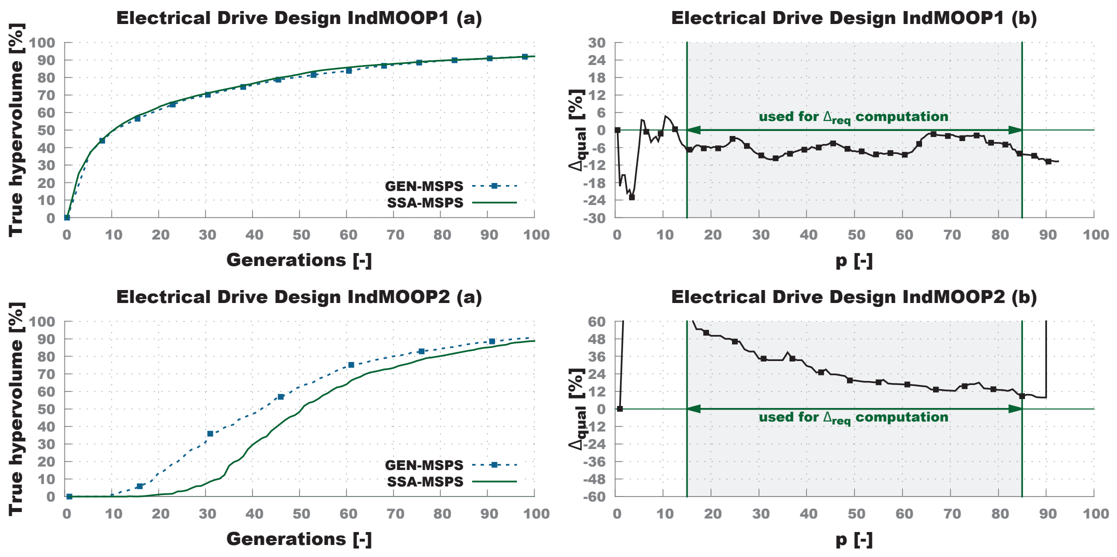

Fig. 9. SPEA2 qualitative performance plots for two electrical drive design MOOPs: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results.

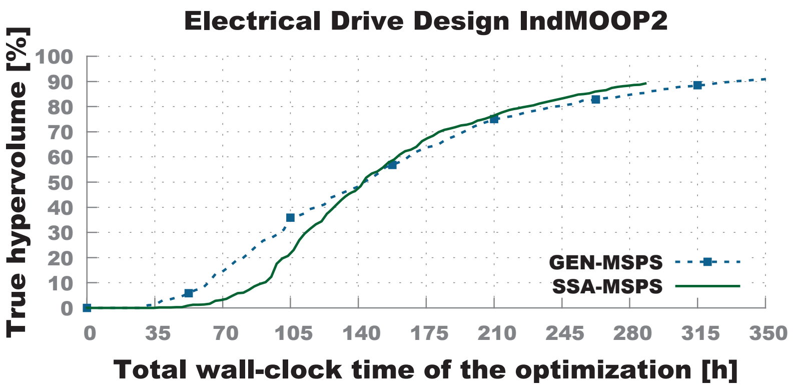

Fig. 10. Average wall-clock performance of SPEA2 with GEN-MSPS and SSA-MSPS on IndMOOP1.

– SSA-MSPS obtains better results when the optimizations are allowed to run for longer periods (and implicitly for $p > 50$).

In order to better explain this, in Figure 10, we display the average performance obtained by the two parallelization schemes when using the wall-clock time as reference. This plot uses exactly the same data as the bottom left subplot from Figure 9, but here the optimization time is used as reference (instead of the number of computed generations). Figure 10 indicates that for runs that take less than 140 hours GEN-MSPS is the better option, while for runs that exceed this threshold, SSA-MSPS should be used. When comparing with the generational runs, the SSA runs are able to reach high quality PNs (i.e., $p > 90$) 12% faster.

It is obvious that the current optimization setup can be improved a lot by solving the issues related to the mesh generation software. Performing this task on the slave nodes would eliminate the major bottleneck of the current software architecture and would result in parallelization ratios of more than 150. Nevertheless, even (more) in this case, the observed high heterogeneity of the time-wise distribution of the fitness evaluation function and the comparative qualitative performance (i.e., negative Δ_{req} for IndMOOP1 and a decreasing Δ_{qual} trend for IndMOOP2) are very strong indicators that SSA-MSPS should be preferred for these optimization scenarios.

Applying the proposed quantitative and qualitative analysis of parallelization performance in real world optimization scenarios is not a complicated process. Nevertheless, the posteriori character of both Δ_{struct} and Δ_{req} means that a few initial, short, mock-up test runs are required in order to estimate the concrete t_s and t_p distributions and get some insight on the problem specific qualitative behavior. The (empirically observed) non-increasing rend of Δ_{req} means that it can be estimated using short runs (25-35 generations). This profiling phase is more than worthwhile when one has to solve multiple MOOPs that fall within the same class (e.g., for the above cases, electrical drive optimization problems with the same topology and with fairly similar design parameter vectors) or perform a lot of optimization runs for the same MOOP. Initial tests show that for the same MOOP class, Δ_{req} is expected to be quite robust while Δ_{struct} can be generally estimated quickly when knowing the t_s and t_p distributions.

5. Conclusions and Future Work

As the use of multi-objective evolutionary algorithms (MOEAs) in complicated industrial decision making scenarios is ever more popular and their is a strong push to improve the speed of these heuristic optimization methods, in the present study, we investigated two master-slave parallelization methods (generational and steady state asynchronous) for MOEAs and tried to discover what are the decisive factors that can make one method outperform the other. We directly applied our proposed performance analysis method in two time-constrained optimization scenarios that also require very computationally-intensive fitness evaluation functions.

In order to achieve our declared goal, we performed a comparative quantitative and qualitative analysis of the behavior of the two parallelization methods when applying them to speed up NSGA-II and SPEA2 optimization runs. The results indicate that 1) *the parallelization ratio* and especially 2) *the level of variance in the time-wise distribution of the fitness evaluation function* are the key factors that influence the relative performance of the two methods. The presence of variance is a key indicator, as a rather heterogeneous time-wise distribution of the fitness function can make the steady state asynchronous parallelization method (SSA-MSPS) considerably outperform its generational counterpart (GEN-MSPS).

In the future, we plan to profile using Δ_{req} more problems, explicitly MOOPs that concern different motor topologies from the field of electrical drive engineering. We also plan to extend the present study by also analyzing more modern MOEAs like the ones mentioned in Section 2.1. Last but not least, we want to investigate if a strategy based on switching between the λ and 1^+ synchronization steps during various stages of the optimization run is generally more successful. This is because, our current results on artificial and real-life test problems (e.g., see the Δ_{qual} plots from Figure 4, Figure 5, and Figure 6) show that GEN-MSPS is always much better than SSA-MSPS in the beginning of the optimization runs (e.g., for $p < 10$).

Acknowledgements

This work was conducted within LCM GmbH as a part of the COMET K2 program of the Austrian government. The COMET K2 projects at LCM are kindly supported by the Austrian and Upper Austrian governments and the participating scientific partners. The authors thank all involved partners for their support.

References

[1] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons, 2001.

[2] M. Gen, R. Cheng, Genetic Algorithms & Engineering Optimization, John Wiley & Sons, 2000.

[3] C. Coello, G. Lamont, D. Van Veldhuisen, Evolutionary Algorithms for Solving Multi-Objective Problems, Genetic and Evolutionary Computation Series, Springer, 2007.

[4] M. Yagoubi, L. Thobois, M. Schoenauert, Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs, in: IEEE Congress on Evolutionary Computation (CEC 2011), 2011, pp. 21–28.

[5] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, S. Silber, W. Amrhein, E. P. Klement, Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives, Engineering Applications of Artificial Intelligence 26 (8) (2013) 1781–1794.

[6] D. E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp. 69–93.

[7] E. O. Scott, K. A. De Jong, Understanding simple asynchronous evolutionary algorithms, in: Foundations of Genetic Algorithms XIII, FOGA '15, Aberystwyth, Wales, UK, January 17-20, 2015, 2015.

[8] J. Durillo, A. Nebro, F. Luna, E. Alba, A study of master-slave approaches to parallelize NSGA-II, in: IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), 2008, pp. 1–8.

[9] A.-C. Zăvoianu, E. Lughofer, W. Koppelstätter, G. Weidenholzer, W. Amrhein, E. P. Klement, On the performance of master-slave parallelization methods for multi-objective evolutionary algorithms, in: L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, J. M. Zurada (Eds.), Artificial Intelligence and Soft Computing, Vol. 7895 of Lecture Notes in Artificial Intelligence, Springer Berlin Heidelberg, 2013, pp. 122–134.

[10] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[11] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization, in: Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.

[12] Q. Zhang, H. Li, MOEA/D: A multi-objective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731.

[13] K. Price, R. Storn, J. Lampinen, Differential evolution, Springer, 1997.

[14] T. Robič, B. Filipič, DEMO: Differential evolution for multiobjective optimization, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005), Springer, Springer Berlin / Heidelberg, 2005, pp. 520–533.

[15] S. Kukkonen, J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in: IEEE Congress on Evolutionary Computation (CEC 2005), IEEE Press, 2005, pp. 443–450.

[16] H. Li, Q. Zhang, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 284–302.

[17] I. Das, J. E. Dennis, Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems, SIAM J. Optim. 8 (3) (1998) 631–657.

[18] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, W. Amrhein, E. P. Klement, DECMO2: a robust hybrid and adaptive multi-objective evolutionary algorithm, Soft Computing [available online], DOI: 10.1007/s00500-014-1308-7 (2014) (2014) 1–19.

[19] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable multi-objective optimization test problems, in: IEEE Congress on Evolutionary Computation (CEC 2002), IEEE Press, 2002, pp. 825–830.

[20] F. Kursawe, A variant of evolution strategies for vector optimization, in: Workshop on Parallel Problem Solving from Nature (PPSN I), Vol. 496 of Lecture Notes in Computer Science, Springer, 1991, pp. 193–197.

[21] S. Huband, L. Barone, L. While, P. Hingston, A scalable multi-objective test problem toolkit, in: Evolutionary Multi-Criterion Optimization (EMO 2005)., Vol. 3410 of Lecture Notes in Computer Science, 2005.

[22] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, Evolutionary Computation 8 (2) (2000) 173–195.

[23] J. J. Durillo, A. J. Nebro, JMETAL: A Java framework for multi-objective optimization, Advances in Engineering Software 42 (2011) 760–771.

[24] K. Deb, R. B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (1995) 115–148.

[25] M. Fleischer, The measure of Pareto optima. applications to multi-objective metaheuristics, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Springer, 2003, pp. 519–533.

[26] J. Durillo, A. Nebro, F. Luna, E. Alba, On the effect of the steady-state selection scheme in multi-objective genetic algorithms, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2009), Springer, 2009, pp. 183–197.

[27] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the condor experience., Concurrency - Practice and Experience 17 (2-4) (2005) 323–356.