

A research overview and evaluation of performance metrics for self-organization algorithms

Benedikt Eberhardinger, Gerrit Anders, Hella Seebach, Florian Siefert, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Eberhardinger, Benedikt, Gerrit Anders, Hella Seebach, Florian Siefert, and Wolfgang Reif. 2015. "A research overview and evaluation of performance metrics for self-organization algorithms." In 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 21-25 September 2015, Cambridge, MA, USA, edited by Gerrit Anders, Jean Botev, and Markus Esch, 122-27. Piscataway, NJ: IEEE. <https://doi.org/10.1109/sasow.2015.25>.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/>



A Research Overview and Evaluation of Performance Metrics for Self-organization Algorithms

Benedikt Eberhardinger, Gerrit Anders, Hella Seebach, Florian Siefert, and Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, Germany
E-Mail: {eberhardinger, anders, seebach, siefert, reif}@isse.de

Abstract—Self-organization (SO) algorithms are supposed to restructure and reconfigure the system at run-time in order to empower it to fulfill its requirements under uncertain environmental conditions. For this purpose, information about the state of the environment and the system is used in feedback loops to establish a flexible, powerful system. Consequently, the performance of the SO algorithms has a significant effect on the overall performance of the system. Indeed, it is hard to design high-performing SO algorithms, because the environmental conditions the system has to operate in are partially unpredictable at design time. A crucial aid for the development of SO algorithms are tools that enable the evaluation of the algorithms’ performance at design time. These tools could also be used to select the best-fitting algorithm and parametrization for a specific application, among others. We show how existing performance metrics can be applied to SO algorithms by evaluating different partition-based algorithms. Based on these results, we discuss the advantages and limitations of the existing metrics and deduce requirements for performance metrics for SO algorithms.

Keywords—Self-organizing System, Self-organization Algorithm, Performance Metrics, Evaluation, Self-Adaptation

I. PERFORMANCE OF SELF-ORGANIZATION ALGORITHMS

Self-organizing (SO) systems structure or organize themselves without explicit control from outside [1]. In general, SO systems encompass a large number of interacting components in an ever-changing environment. Despite these changing conditions, they are able to operate robustly and flexible in open and interoperable system settings. To achieve this behavior, a SO mechanism has to collect information about the current state of the system, analyze if the requirements are sufficiently met, and, if this is not the case, trigger a SO algorithm that re-establishes the requirements’ satisfaction by reorganizing the system structure. Due to the system’s non-determinism, it is very challenging to analyze its global behavior.

As a SO algorithm plays a central role in the system, its quality and characteristics are crucial for the overall system’s performance. Since SO algorithms have to operate under ever-changing environmental conditions that are partially unpredictable at design time, it is, indeed, far from obvious how to design and implement the best performing SO algorithm for a certain kind of system. Metrics that allow to evaluate the algorithms’ performance at design time are thus an indispensable tool for engineering SO algorithms efficiently.

a) Performance Analysis: Analyzing the performance of algorithms has been a topic of interest in computer science for decades, mostly driven by theoretical analysis that is based on abstraction, theorems, and proofs in order to find an

asymptotic bound on the dominant operation under a worst-case or average-case model [2]. For example, the theoretical analysis of Dijkstra’s algorithm [3]—which is applied to a graph of n vertices and m edges—results in a worst-case bound of $O(m \log n)$ or $O(m + n \log n)$. However, experimental analysis by Cherkassky et al. [4] has shown that the worst-case bound is overly pessimistic since Dijkstra’s algorithm exhibits $O(n + m)$ performance in many real-world situations. This is just one example illustrating that there is not only a need for experimental analysis of algorithms but also for sound metrics. In the context of SO algorithms, this demand is even stronger, as van Dyke Parunak and Brueckner [5] argue, because the concepts of theoretical analysis are stretched to their limits given that the majority of SO systems are formally undecidable.

b) Performance Criteria: McGeoch [2] categorizes metrics for evaluating the performance of classical algorithms into the two performance criteria *solution quality* and *time*. The latter indicates how much time (CPU, real, or logical) the algorithm spent to solve given problems. Although the time could be measured with high precision, it neither guarantees accuracy nor generality [2]; the results depend on the platform used to evaluate the performance. An essential technique to cope with these issues is data analysis [6]. The other aspect of performance, *solution quality*, is of interest when the task at hand is to solve an optimization problem, that is, to find a solution that is optimal with regard to a specific objective function in a set of feasible solutions. In this case, the metric for evaluating the solution quality can correspond to the problem’s objective function.

c) Measuring Performance of SO Algorithms: For our further investigations and as a running example, we use a SO algorithm called *PSOPP* [7] (**P**article **S**warm **O**ptimizer for the **P**artitioning **P**roblem) that represents an algorithm for weak SO¹. *PSOPP* is a particle swarm optimizer that partitions a set of agents representing a (sub)system into pairwise disjoint and non-empty groups. These groups constitute the (sub)systems’ organizational structure. Feasible organizational structures can be described by so-called *partitioning constraints* that restrict the number and the size of these groups. *PSOPP* is an anytime algorithm and a metaheuristic that optimizes the groups’ composition with respect to an objective function. In our running example, *PSOPP* is used to optimize the groups’ composition in each so-called *separate subsystem* of a hierarchically structured system. An example of such a system structure is shown in Figure 1: Here, two separate subsystems

¹The classification of weak SO is used according to the definition of Serugendo et al. [8].

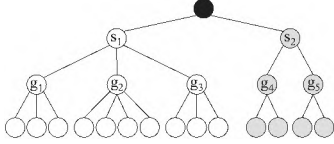


Fig. 1. The graph shows a possible hierarchy formed by a SO algorithm. Within this hierarchy, it is possible that just a subgraph, e.g., the right gray part of the graph, is part of a reorganization. The impact on the working time of the entire system is much smaller if the gray subgraph is reorganized than in case of a reorganization of the left white part. Given that such hierarchical structures are often exploited by SO systems (cf. [9], [7]), this characteristic has to be addressed by metrics for SO algorithms.

s_1 and s_2 are shown with subordinate groups g_1, g_2, g_3 and g_4, g_5 , respectively. Each separate subsystem, such as s_1 or s_2 , uses an instance of *PSOPP* to maintain a suitable partitioning. We call this form of SO *regio-central*.

Clearly, both *solution quality* as well as *time* are performance criteria that are applicable to SO algorithms. As shown later in Equations (1) to (3), it makes sense to estimate a SO algorithm’s performance by comparing the working time of the SO algorithm to the working time of the controlled system. This relation reflects the fact that the performance of the SO algorithm determines the performance of the controlled system. However, in a system consisting of many subsystems that can reorganize themselves independently from each other, it is not obvious how to define the working time of the controlled system when, with respect to a specific time frame, some parts might reorganize their structure and others do not. With regard to Figure 1, for example, the influence on the system’s working time should be the higher the larger the subsystem is that reorganizes its structure and thus does not contribute to the system’s actual task.

The solution quality reflects how well a SO algorithm lives up to its responsibilities, i.e., maintaining a suitable system structure with regard to some objective function. However, the interpretation of the solution quality is quite hard in the setting of SO algorithms because the quality of feasible solutions is likely to change over time due to the ever-changing environment.

d) *Contributions of this Paper:* In our paper, we provide the following contributions:

- (i) a research overview on performance metrics applicable to SO algorithms and their evaluation by means of the SO algorithm *PSOPP* (see Section II)
- (ii) a set of requirements for performance metrics for SO algorithms based on our evaluation of existing performance metrics (see Section III)

II. EVALUATION OF APPLICABLE METRICS FOR MEASURING PERFORMANCE OF SO ALGORITHMS

To put it simply, the performance of an algorithm describes how well or badly it works. SO algorithms work on the structure or organization of the system. Consequently, their performance is defined by how well they structure or organize the system. In the literature, different metrics are defined that concretize “how well” algorithms work by identifying several fine-grained performance criteria.

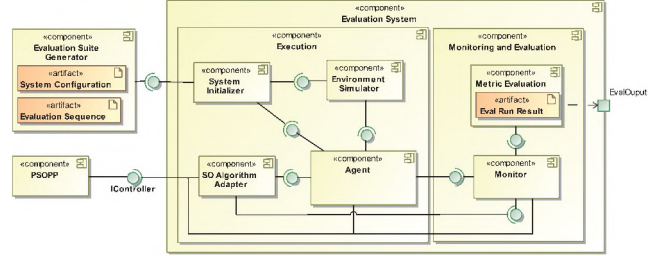


Fig. 2. The UML component diagram shows the essential components of the test bed that is used to measure *PSOPP*’s performance. The test bed consists of an Evaluation Suite Generator, an Execution, as well as a Monitoring and Evaluation component. To investigate the applicability of metrics outlined in Sections II-B to II-C, we plugged *PSOPP* into this test bed.

Our summary of the state of the art, which we provide in Sections II-B to II-C, aims at identifying and discussing different performance metrics with regard to their applicability to SO algorithms. We base our discussion on empirical data we obtained during an evaluation of *PSOPP*’s performance by means of these metrics (cf. Section I). We introduce our test bed in Section II-A, before we outline the state of the art and discuss the results of our evaluation in Sections II-B to II-C.

A. Test Bed for Evaluating SO Performance

To apply and evaluate the metrics introduced in Sections II-B and II-C, we use a test bed² that is shown in its essential components in Figure 2. It is structured into three main components that encompass the generation of *evaluation suites*, the execution of the evaluation suites, and the observation as well as evaluation of the SO algorithm that is plugged into the evaluation system via an interface.

An evaluation suite consists of a System Configuration and a number of Evaluation Sequences. These are automatically generated by the Evaluation Suite Generator. A randomly created *system configuration* describes a basic system setting:

- a set of agents
- an initial state for each agent (for the time being, each state is represented by a real value from the interval $[0, 1]$)
- a number of predefined separate subsystems
- an initial system structure for each separate subsystem

Based on such a configuration, a number of *evaluation sequences* is generated. Each evaluation sequence represents a series of state changes of the agents within the system. The length of a series corresponds to the number of time steps that should be simulated within the Evaluation System. Consequently, each agent will change its state exactly once in each time step. To generate the evaluation sequences, we simulate a probabilistic model of the system and its environment—described as a Markov chain—for the specified number of time steps. Thus, it is possible to gain simulation runs causing likely conditions under which the SO algorithm has to operate. The generated evaluation suites are executed within the Evaluation System where the

²The test bed for evaluating SO algorithms is grounded on the *IsoTeSO* framework introduced in [10].

System Initializer sets up the system as described in the corresponding configuration. After the system is initialized, the evaluation sequences are executed by the Environment Simulator by applying the state changes to the agents. From time to time, a reorganization might be necessary to re-establish a suitable partitioning of the overall system. In our evaluation setting “PSOPP HP”, *PSOPP*’s goal was to create a homogeneous partitioning in each separate subsystem. A homogeneous partitioning is a partitioning in which each partition, i.e., agent group, has a similar average state value. This is accomplished by minimizing the standard deviation of the average state values by creating new or dissolving existing partitions and exchanging agents between them. Such a structure has shown to be rather robust against changing states compared to heterogeneous partitions as they are formed in the evaluation setting “PSOPP k-means” (cf. Section II-B). The Monitor logs the working time of the agents and the time needed for reorganizations in real time. Further, it logs the fitness value achieved by the SO algorithm and forwards the data to the Metric Evaluation component that applies the different metrics and provides their results as output of the Evaluation System.

This test bed is used to assess the capabilities of the metrics discussed in the next section. For comparison, we executed all 100 generated evaluation suites, each comprising 10 evaluation sequences, in three different settings: (1) In the setting “noSO HP”, PSOPP was disabled. (2) In the setting “PSOPP HP”, PSOPP established partitionings according to the homogeneous partitioning objective function described above. (3) In the setting “PSOPP k-means”, PSOPP established heterogeneous partitionings according to the well-known k-means objective function. All evaluation sequences represented 300 time steps and have been performed in a distributed cluster of 12 computers with an Intel Core-i5 CPU and 4GB RAM for about a week. We performed each setting on a predefined system structure consisting of 1, 2, and 5 separate subsystems.

B. Metrics for Adaptation Algorithms

There are several metrics for adaptation (resp. self-adaptation) algorithms in the literature. As is the case with classical algorithms, they can be clustered into *time-oriented metrics* and *solution-quality-oriented metrics*. The research survey of Villegas et al. [11] as well as the criteria for the evaluation of self-* systems of Kaddoum et al. [12]³ are *time-oriented metrics* that reflect the relationship between time for adaptation and working time. The performance metrics of Becker et al. [14], Tarnu and Tiemann [15], and Reinecke et al. [16] address the *solution quality* of the adaptation algorithm. Overall, we selected those metrics that are applicable to SO algorithms, which represent some special form of adaptation. We discuss if the metrics are suitable for measuring the performance of SO algorithms on the basis of our evaluation results.

1) Investigated Metrics:

a) *Time-oriented Metrics*: Kaddoum et al. [12] extend classical performance metrics to metrics for self-adaptive systems by distinguishing *nominal* and *self-* situations* and

³Parts of the criteria for evaluation of adaptive systems have been applied by Cámara et al. [13].

focusing on their relation. One example is the *WAT* metric that is defined as follows:

$$WAT := \frac{\text{working time}}{\text{adaptivity time}} \quad (1)$$

The codomain of the *WAT* is $[0, \infty]$, where the performance of the adaption algorithm is said to increase with the value of *WAT*. The intuition of *WAT* is that adaptation is responsible for keeping the controlled system working with as little disruption as possible by an adaptation algorithm. Further metrics introduced in [12] are also defined as the ratio between adaption and working time, but focus on service-oriented systems, e.g., the response time of a service.

The metrics proposed by Villegas et al. [11] also focus on service-oriented adaptive systems. Proposed information of interest are, for example, monetary execution costs or the reliability of a service according to task completion. Villegas et al. defined the availability (*A*) resp. unavailability (*U*) metrics as follows:

$$A := \frac{MTTF}{MTTF + MTTR} \quad (2)$$

$$U := \frac{MTTR}{MTTF + MTTR} \quad (3)$$

MTTR is the mean time to recover and *MTTF* is the mean time to fail with a codomain of *A* and *U* of $[0, 1]$.⁴ A large value of *A* and a small value of *U* is desired to attest to an algorithm’s good performance. The metrics are based on the concepts of reliability engineering (cf. [17]) and define the performance of an adaptation algorithm over the reliability it yields for the controlled system.

b) *Solution-quality-oriented Metrics*: Taranu and Tiemann [15] use a cost function to evaluate the solution quality of an adaptation algorithm in the context of network scenarios. The function maps a performance value to different situations. Each situation may consist of sub-situations with individual costs. Costs are, for example, defined by the generated network traffic, where as little traffic as possible is desired. With regard to a specific situation *sit*, their performance metric is defined on the basis of the measured costs C_{subsit} and the maximum costs C_{max} of the sub-situations *subsit*:

$$\text{perf}(sit) := 1 - \frac{\sum_{\text{subsit} \in sit} C_{\text{subsit}}}{\sum_{\text{subsit} \in sit} C_{\text{max}}} \quad (4)$$

The metric $\text{perf}(sit)$ yields a normalized⁵ cost value—resp. a solution quality for a situation *sit*—on the basis of the costs of different sub-situations, e.g., different time steps. The resulting performance is within the codomain of $[0, 1]$. The best performance is 1.

Becker et al. [14] derive performance metrics from requirements by measuring the time the requirements are fulfilled and, above that, how well they are fulfilled. This approach is grounded on requirements specified as *RELAXed* requirement (cf. [18]) and a function that maps the satisfaction resp. dissatisfaction of the requirement within a given time interval to a numeric value. Let us consider an example where the requirement *RF* is defined as follows:

⁴Note that $A + U = 1$.

⁵In our paper, the term *normalization* is used in the sense of adjusting values measured on different scales to a notionally common scale of $[0, 1]$.

“The system SHALL keep the rental fee AS CLOSE AS POSSIBLE to 0.”

The function $\Delta(\phi_{RF}, [i, j])$ evaluates the dissatisfaction of the requirement RF in the time interval $[i, j]$, e.g., if there is no rental fee, the value is 0. The corresponding performance metric for a requirement RF is defined as follows:

$$m_{RF} := \begin{cases} 0 & \text{if } \Delta(\phi_{RF}, [i, j]) \geq RF_{max} \\ 1 - \frac{\Delta(\phi_{RF}, [i, j])}{RF_{max}} & \text{else} \end{cases} \quad (5)$$

RF_{max} is the rental fee threshold. For m_{RF} , the codomain is $[0, 1]$ with the optimal performance being 1 since the requirement RF is completely fulfilled. The metric is quite similar to Equation (4) apart from the property that the solution quality is bounded by RF_{max} .

Reinecke et al. [16] propose to measure the performance according to the sum of benefits obtained by the decisions made of the adaptation algorithm. To this end, they use three different sets to remember the time steps $i \in \{1, 2, 3, \dots, N\}$ in which the payoff p_i decreased, did not change, or increased from one time step to another:

$$\begin{aligned} D_{\ominus} &:= \{i = 2, 3, \dots, N \mid p_{i-1} > p_i\} \\ D_{\circ} &:= \{i = 2, 3, \dots, N \mid p_{i-1} = p_i\} \\ D_{\oplus} &:= \{i = 2, 3, \dots, N \mid p_{i-1} < p_i\} \end{aligned}$$

You can think of these sets as sets of negative, neutral, and positive decisions. The performance metric over these decisions is defined by the following formula that reflects the total benefit of adaptation:

$$Ad := \frac{\sum_{i \in D_{\oplus}} \Delta_i + \sum_{i \in D_{\circ}} p_i}{N - 1} \quad (6)$$

Δ_i defines the benefit of a decision as $\Delta_i := \frac{p_i + p_{i-1}}{2}$ and p_i is the benefit in a time step i . Both the function Δ_i and the exclusion of the set D_{\ominus} smooths the payoff function of the system over the considered time period. Since the payoff is within $[0, 1]$, the maximum payoff is 1. Ad is normalized by dividing by $N - 1$.

The last three presented metrics ([14], [16], [15]) have been introduced and applied to different case studies from the field of service-oriented, adaptive systems. The other metrics ([11], [12]) have been evaluated in a quantitative analysis, also with a strong focus on service-oriented, adaptive systems.

2) *Discussion*: All metrics have been used to analyze the *PSOPP* algorithm in different settings. The results of our evaluation are summarized in Table I. In the following, we discuss their significance for SO algorithms.

a) *Time-oriented Metrics*: The metrics *WAT*, *A*, and *U* (see Equations (1) to (3)) rely on the ratio between working time and adaptivity time resp. the mean time to fail and the mean time to recover. All three focus on the impact of the adaption on the working system and reflect the stability as well as the robustness of the organizations established by the SO algorithm. The results of Table I indicate that *PSOPP HP* is able to achieve more robust partitionings than *PSOPP k-means*. This reflects our intuition that heterogeneous partitions, as obtained by homogeneous partitioning, are more robust than homogeneous partitions favored by the k-means fitness function (cf. [7]).

Unfortunately, the locality of SO algorithms is neglected by the three metrics. Thus, a reconfiguration in a small part of the system is rated as an adaption period of the entire system. Although the results in Table I show that the average number of agents participating in a reconfiguration decreases with an increasing number of subsystems, the whole system is rated “in reconfiguration”. This contradicts the reality, which is shown in the number of reorganized separated subsystems per reorganization. Ergo, it is, for instance, hard to reason (based on the metrics) whether *PSOPP HP* performs better in a system with 2 or 5 separate subsystems: While *WAT* states that 2 separate subsystems are better, the *A* and *U* metrics prefer 5 separate subsystems.

Considering only the time-oriented metrics, it is possible that a SO algorithm that causes the system to work inefficiently is rated very good in terms of time if it generates a robust structure. Therefore, we claim that there is a need for combining time-oriented metrics with solution-quality-oriented metrics to rate the overall performance of a SO algorithm. This is also indicated in the evaluation results of *PSOPP k-means* since the very high number of reorganizations results in a very high payoff in terms of solution quality. An open question is how to aggregate the results of different metrics, e.g., is an organizational structure’s robustness better than one that enables the system to work in an efficient manner?

b) *Solution-quality-oriented Metrics*: To rate the performance of a SO algorithm, the optimality of its solution plays a crucial role. In the sense of SO, the optimality depends on the quality of the selected organizational structure from the set of all possible structures with respect to a given fitness function. Such an optimization problem can often be described as a constrained optimization problem. The problem is constrained since we can implement the requirements for valid system structures in the form of constraints [19] that allow to determine the set of valid structures.

Let us consider the metrics in Equations (4) and (5) first since both are quite similar in how they measure the normalized fitness of the SO algorithm over time. Note, however, that Equation (5) bounds the optimum by RF_{max} . Because of their similarity, we only evaluated the *perf(sit)* metric.

Challenges that arise during the evaluation of SO algorithms with the metrics defined in Equations (4) and (5) are mainly caused by the locality of the SO algorithms. This is a major difference to the adaptation algorithm considered by Becker et al. [14] as well as Taranu and Tiemann [15] who regard a central approach of only one adaptation algorithm within the entire system. In case of multiple subsystems, as is the case with our evaluation scenarios, the metrics could be applied to the separate subsystems, but it is not obvious how to calculate the performance for the overall system. We used the average value of the *perf(sit)* metric applied to each separate subsystem as an approximation of the quality of the entire system. However, it is unclear whether this approach reflects the performance adequately. Depending on the intended purpose of the performance measurement, this average value might be weakly informative, e.g., one might want to have the worst rating of all subsystems in case of a risk-averse evaluation.

As shown in Table I, the *perf(sit)* metric obtains high val-

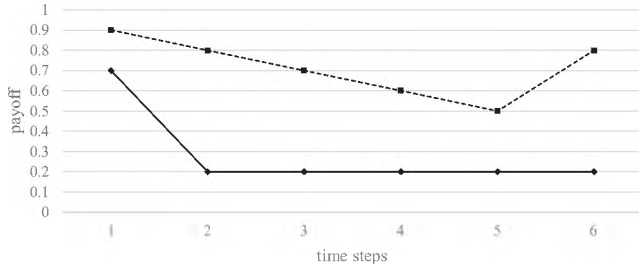


Fig. 3. The figure shows the payoff of two different series of fitness values (*dashed* and *solid*) for six time steps. If we apply the metric Ad defined in Equation (6) to both series, we get $Ad_{solid} = 0.2$ and $Ad_{dashed} = 0.1625$, which is not reasonable since the payoff of the *dashed* series is at every time step clearly above the one of the *solid* series.

ues in the settings *PSOPP HP* and *PSOPP k-means*, indicating that *PSOPP* performs well in choosing organizational structures. Furthermore, it is possible to compare the performance of an organizational structure without SO (see *noSO HP* in Table I), i.e., no restructuring or reorganization of the system at run-time, with a system with SO: The results clearly indicate that SO with the *PSOPP* algorithm is worthwhile since the fitness value of the system is almost twice as high compared to a system without SO (note that *PSOPP*'s quality was rated using the objective function of homogeneous partitioning in case of *noSO HP* as well as *PSOPP HP*).

The Ad metric in Equation (6) intends to smooth the development of the fitness value used by $perf(sit)$. Alas, the metric shows some bad side effects that are illustrated in Figure 3 where a series of fitness values that is obviously worse than another is actually rated better. Due to this unwanted effect of the metric, it is hard to use the value for performance evaluation. This is also reflected in the evaluation results of Table I, where the performance of *noSO HP* is rated ten times better than *PSOPP HP*, which is quite incomprehensible considering that Ad only tries to smooth the development of the fitness value that is used by $perf(sit)$.

C. Metrics for SO Algorithms

Hitherto, there has not been any work focusing on the design of general metrics for measuring the performance of SO algorithms. The work of Kaddoum et al. [12] lists metrics under the umbrella of self-* systems and consequently includes self-organization, but the shown metrics as well as the paper and pencil evaluation are only considering self-adaptive systems. That is why we introduced their metrics already in the previous section as metrics for self-adaption algorithms. Nevertheless, the implementation and design of SO algorithms has brought along specialized evaluations of the developed algorithms during the recent years (cf. [20], [21], [7]).

In general, the evaluation results are used to show the strengths and limitations of the SO algorithms in a specific setting. Sometimes, the results are also used to optimize the parametrization of the developed algorithms. Nevertheless, to our knowledge, there is no general approach that addresses systematic and comparable performance evaluations. We claim that it is necessary to have a better comparability of SO algorithms and a common understanding of performance as

already established in the field of classical as well as adaptation algorithms.

III. REQUIREMENTS FOR PERFORMANCE METRICS FOR SELF-ORGANIZATION ALGORITHMS

For a future systematic uniform approach to evaluate the performance of SO algorithms, we propose a set of requirements for performance metrics suited for SO algorithms as well as for their application within the performance evaluation process. We derived these requirements from the results discussed in Section II and experiences gained during the evaluations of our developed SO algorithms. The requirements are split into requirements for the metrics themselves (Section III-A) and their implementation in a framework for SO algorithm evaluation (Section III-B).

A. Metrics Suited for SO Algorithms

We claim that the following requirements are the most important to be met by metrics to assess the performance of SO algorithms in terms of *time* and *solution quality*:

- Req. 1:** The **locality** of SO algorithms has to be taken into account and the aspects of *time* and *solution quality* have to be evaluated within the existing (over run-time changing) subsystems that are differently affected by the SO algorithm, e.g., one subsystem can be reorganized while another keeps on working. Furthermore, it is important to be able to assess the performance of the entire system based on the performance of the subsystems.
- Req. 2:** Since SO algorithms have control over the system's structure, their performance strongly influences those of the entire system. So the overhead of a reorganization can be worthwhile if it sufficiently improves the behavior of the controlled system. Consequently, a metric has to take the **benefit of the reorganization** into account.
- Req. 3:** The interpretation of a value provided by a metric strongly depends on the current state of the system. In self-organizing systems, the possible values for the solution quality can change over time. For instance, a solution quality of 0.7 would be optimal if possible values were defined by the interval $[0, 0.7]$ but quite bad if they stem from the interval $[0, 200]$; the same applies to the parameter *time*. Consequently, there is a need for **dynamic boundaries for the evaluation**—a requirement resulting from the ever-changing environment of SO algorithms.

B. Application within Performance Evaluation Process

To achieve a systematic approach, we claim that there is a need for a framework for performance evaluation that has to satisfy at least the following requirements:

- Req. 4:** The overall process has to be supported by a **framework for performance evaluation** that is able to systematically evaluate SO algorithms. The framework's components should support the generation of evaluation runs to perform, the simulation itself, and the application of performance metrics.

Setting #Separate Subsystems	noSO HP			PSOPP HP			PSOPP k-means		
	1	2	5	1	2	5	1	2	5
<i>WAT</i>	—	—	—	6.92 (1.35)	3.24 (0.90)	0.97 (0.22)	0.02 (0.01)	0.01 (0.01)	0.01 (0.01)
<i>Working Time</i> [s]	—	—	—	29.78 (37.29)	29.69 (92.45)	29.27 (0.21)	15.85 (1.69)	11.57 (3.84)	7.77 (4.84)
<i>Adaptivity Time</i> [s]	—	—	—	4.59 (1.39)	9.93 (2.89)	31.57 (6.62)	623.411 (76.31)	907.80 (151.51)	1,617.07 (394.80)
<i>A</i>	—	—	—	0.77 (0.02)	0.66 (0.02)	0.44 (0.03)	0.02 (0.01)	0.01 (0.01)	0.01 (0.01)
<i>U</i>	—	—	—	0.22 (0.02)	0.33 (0.02)	0.55 (0.03)	0.97 (0.01)	0.98 (0.01)	0.99 (0.02)
<i>MTTR</i> [s]	—	—	—	3.97 (0.19)	5.41 (1.08)	5.43 (1.29)	4.43 (0.09)	5.03 (0.67)	7.32 (0.85)
<i>MTTF</i> [s]	—	—	—	14.13 (1.82)	10.83 (3.18)	4.51 (1.75)	0.11 (0.04)	0.07 (0.05)	0.04 (0.04)
<i>perf(sit)</i>	0.52 (0.09)	0.96 (0.01)	0.52 (0.07)	0.96 (0.01)	0.57 (0.07)	0.96 (0.01)	0.99 (0.01)	0.99 (0.01)	0.99 (0.01)
<i>Ad</i>	0.14 (0.03)	0.01 (0.01)	0.14 (0.02)	0.01 (0.01)	0.14 (0.02)	0.01 (0.01)	0.37 (0.09)	0.14 (0.03)	0.04 (0.01)
#Reorganized Separate Subsystems	—	—	—	1.15 (0.37)	2.98 (0.91)	10.21 (2.16)	140.47 (16.93)	245.00 (42.52)	497.63 (123.17)
#Reorganized Separate Subsystems per Reorg.	—	—	—	1.00 (0.00)	1.49 (0.50)	1.64 (1.42)	1.00 (0.00)	1.33 (0.47)	2.24 (1.19)
#Reorganized Agents per Reorg.	—	—	—	1000.00 (0.00)	646.78 (385.14)	283.87 (318.04)	1000.00 (0.00)	714.74 (305.04)	479.35 (284.08)

TABLE I. EVALUATION RESULTS FOR THE THREE SETTINGS “NO SO HP”, “PSOPP HP”, AND “PSOPP K-MEANS” WITH DIFFERENT NUMBERS OF SEPARATE SUBSYSTEMS. ALL VALUES ARE AVERAGES OVER 1000 EVALUATION SEQUENCES; VALUES IN PARENTHESIS DENOTE STANDARD DEVIATIONS.

Req. 5: In order to achieve significant results, the evaluation must comprise **simulation runs** that induce an environmental behavior **reflecting likely conditions under which the SO algorithms have to operate**.

In future work, we will address these requirements and integrate a performance evaluation component into our testing framework IsoTeSO, which is introduced in [10].

ACKNOWLEDGMENT

This research is sponsored by the research project *Testing self-organizing, adaptive Systems (TeSOS)* of the *German Research Foundation*.

REFERENCES

- [1] G. Di Marzo Serugendo, N. Foukia, S. Hassas, A. Karageorgos, S. Mostfaoui, O. Rana, M. Ulieru, P. Valckenaers, and C. Van Aart, “Self-organisation: Paradigms and applications,” in *Engineering Self-Organising Systems*, ser. LNCS, D. M. Serugendo et al., Eds. Springer Berlin Heidelberg, 2004, vol. 2977, pp. 1–19.
- [2] C. C. McGeoch, *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.
- [3] E. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [4] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, “Shortest paths algorithms: Theory and experimental evaluation,” *Mathematical Programming*, vol. 73, pp. 129–174, 1993.
- [5] H. V. D. Parunak and S. A. Brueckner, “Software Engineering for Self-organizing Systems,” in *Proc. 12th Int. Wsh. Agent-Oriented Software Engineering (AOSE’2011)*, 2011, pp. 1–22.
- [6] R. Ackoff, “From data to wisdom,” *Journal of Applied Systems Analysis*, vol. 16, no. 1, pp. 3–9, 1989.
- [7] G. Anders, F. Siefert, and W. Reif, “A particle swarm optimizer for solving the set partitioning problem in the presence of partitioning constraints,” in *Proc. 7th Int. Conf. Agents & AI (ICAART)*, 2015.
- [8] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, “Self-organization in multi-agent systems,” *Knowl. Eng. Rev.*, vol. 20, no. 2, pp. 165–189, Jun. 2005.
- [9] J.-P. Steghöfer, P. Behrmann, G. Anders, F. Siefert, and W. Reif, “HiSPADA: Self-Organising Hierarchies for Large-Scale Multi-Agent Systems,” in *ICAS 2013, The 9th Int. Conf. on Autonomic and Autonomous Systems*. Lisbon, Portugal: IARIA, March 2013, pp. 71–76.
- [10] B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, A. Knapp, and W. Reif, “An approach for isolated testing of self-organization algorithms,” in *Software Engineering for Self-Adaptive Systems III*, ser. LNCS, R. de Lemos et al., Eds. Springer, 2015, vol. (submitted).
- [11] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, “A framework for evaluating quality-driven self-adaptive software systems,” in *Proc. 6th Int. Symposium on Software Engineering for Adaptive and Self-managing Systems*. ACM, 2011, pp. 80–89.
- [12] E. Kaddoum, C. Raibulet, J. Georgé, G. Picard, and M. P. Gleizes, “Criteria for the evaluation of self-* systems,” in *2010 ICSE Wsh. on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2010, Cape Town, South Africa, May 3-4, 2010*, R. de Lemos and M. Pezzè, Eds. ACM, 2010, pp. 29–38.
- [13] J. Cámara, P. Correia, R. de Lemos, and M. Vieira, “Empirical resilience evaluation of an architecture-based self-adaptive software system,” in *Proc. 10th Int. ACM Sigsoft Conference on Quality of Software Architectures*. ACM, 2014, pp. 63–72.
- [14] M. Becker, M. Luckey, and S. Becker, “Performance analysis of self-adaptive systems for requirements validation at design-time,” in *9th ACM SigSoft Int. Conf. Quality of Software Architectures (QoSA’13)*. ACM, 2013.
- [15] S. Taranu and J. Tiemann, “On assessing self-adaptive systems,” in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE Int. Conf. on*. IEEE, 2010, pp. 214–219.
- [16] P. Reinecke, K. Wolter, and A. Van Moorsel, “Evaluating the adaptivity of computing systems,” *Performance Evaluation*, vol. 67, no. 8, pp. 676–693, 2010.
- [17] M. R. Lyu, *Handbook of software reliability engineering*. IEEE computer society press CA, 1996, vol. 222.
- [18] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, “Relax: Incorporating uncertainty into the specification of self-adaptive systems,” in *Requirements Engineering Conf. 2009. RE’09. 17th IEEE Int.*. IEEE, 2009, pp. 79–88.
- [19] B. Eberhardinger, J.-P. Steghöfer, F. Nafz, and W. Reif, “Model-driven synthesis of monitoring infrastructure for reliable adaptive multi-agent systems,” in *Proc. 24th IEEE Int. Symp. Software Reliability Engineering (ISSRE’13)*. IEEE, 2013, pp. 21–30.
- [20] M. Al-Zinati and R. Wenkstern, “A self-organizing virtual environment for agent-based simulations,” in *Proc. 2015 Int. Conf. on Autonomous Agents and Multiagent Systems*, ser. AAMAS ’15. Int. Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 1031–1039.
- [21] J. Pitt, D. Busquets, and S. Macbeth, “Distributive justice for self-organised common-pool resource management,” *ACM Trans. on Autonomous and Adaptive Systems (TAAS)*, vol. 9, no. 3, p. 14, 2014.