A fine-grained message passing MOEA/D

Bilel Derbel, Arnaud Liefooghe, Gauvain Marquet, El-Ghazali Talbi

A Fine-Grained Message Passing MOEA/D

Bilel Derbel, Arnaud Liefooghe, Gauvain Marquet and El-Ghazali Talbi
Université Lille 1, CRIStAL (UMR CNRS 9189) — Inria Lille–Nord Europe, France
Email: firstname.lastname@univ-lille1.fr

***Abstract*—We propose the first large-scale message passing distributed scheme for parallelizing the computational flow of MOEA/D, a popular decomposition-based evolutionary multi-objective optimization algorithm. We show how synchronicity and workload granularity can impact both quality and computing time, in an extremely fine-grained configuration where each individual in the MOEA/D population is mapped to a single distributed processing unit. More specifically, we deploy our distributed protocol using a large-scale environment of 128 computing cores and conduct a throughout analysis using a broad range of bi-objective combinatorial ρMNK-landscapes. Besides being able to show significant speed-ups while maintaining competitive search quality, our experimental results provide insights into the behavior of the proposed scheme in terms of quality/speed-up trade-offs; thus pushing a step towards the achievement of effective and efficient parallel decomposition-based approaches for large-scale multi-objective optimization.**

I. Introduction

Multi-objective optimization problems (MOPs) refer to the situation where multiple objectives are to be optimized in a simultaneous manner. Many-often, the objectives to be optimized are conflicting, and solving a given MOP turns out to seek a whole set of solutions instead of just a single one. The set of optimal solutions is called the Pareto set, which is the set of all non-dominated solutions, i.e., solutions for which there exist no other solution providing a better trade-off in all objectives. It is well understood that computing the Pareto set is a difficult task which is in addition intractable for several MOPs. In this context, evolutionary multi-objective optimization (EMO) algorithms have been proved extremely helpful in computing a high-quality approximation of the Pareto set which portrays good and representative balance among conflicting objectives. One can find different classes and paradigms in EMO algorithms, ranging from Pareto-based methods, indicator-based methods, and aggregation-based methods. In this paper, we are interested in the last group of methods and particularly on the design of new advantageous approaches dedicated to large-scale and distributed computing environments. In this respect, the motivation of this paper is to be understood from two perspectives.

On the one hand, aggregation-based EMO methods have attracted a lot of attention during the last few years. In particular, the so-called MOEA/D (multi-objective evolutionary algorithm based on decomposition) framework [1] is being actively developed by the community due to its simplicity, but also due to its high search ability, computational efficiency, and adaptability to many problems. In MOEA/D, a MOP is reformulated as a set of single-objective sub-problems by means of scalarizing functions configured with different weight vectors in the objective space. As such, MOEA/D solves the so-obtained single objective sub-problems in a cooperative manner using evolutionary mating selection, replacement and variation operators. On the other hand, when abstracting away from the optimization-specific operations, one can view MOEA/D as a divide-and-conquer algorithm which might expose intrinsic parallelism since the original MOP is explicitly decomposed in order to be solved. Such a claim would allow one to tackle large-scale problems and to take much benefits from the increased availability and computing power of nowadays large-scale distributed platforms such as clusters and grids. Generally speaking, designing parallel and distributed EMO algorithms has a well established foundation and a long-term practical usefulness in solving difficult MOPs. Nevertheless, investigating to what extent the MOEA/D framework can be redesigned to run efficiently in large-scale compute environments is surprisingly not yet fully addressed although one can find some recent attempts in this direction [2], [3], [4]. We attribute this to the following two challenging questions:

- How to maintain the search ability of the MOEA/D framework when attempting to break the dependencies in the computational flow of its original implementation? In fact, MOEA/D is inherently sequential and it needs to be accurately adapted in order not only to fit the distributed setting, but also to guarantee as much approximation quality as possible.
- How to deal with the fine-grained parallelism that is likely to be encountered when effectively deploying the so-obtained variants at a large distributed scale? In fact, fine-grained workload can drastically prevent high performance when scaling up computing resources, especially in the scenario where communication cost is non-negligible compared to objectives' evaluation cost of the target MOP.

The ultimate challenge standing behind the previous questions is to derive novel parallel MOEA/D algorithms presenting a good balance between approximation quality and speed-up in the largest scales where fine-grained computations are unavoidable. The aim of this paper is to address this challenge and to shed more light into decomposition-based EMO best practices in the context of large-scale distributed environments. More precisely, our contributions can be summarized as follows:

- We propose a new fine-grained message-passing scheme to distribute the MOEA/D computations. Our scheme can be configured so as to run in synchronous, as well as in asynchronous manner, and has an additional parameter allowing to control workload granularity; thus eliciting different trade-offs between parallel efficiency and approximation quality.
- We implement our distributed scheme using the MPI message-passing library and deploy the so-obtained

distributed protocols in a real experimental set-up on top of a cluster of 128 computing cores. To our best knowledge, this is the first time that such a computing environment is considered. As a case study, we consider an extensive set of combinatorial bi-objective ρMNK-landscapes, ranging from small- to large-size instances, as well as instances with different characteristics in terms tractability and difficulty.

- We provide a throughout evaluation of our distributed scheme and its implemented variants. Overall, we are able to experimentally validate the accuracy of our approach. In particular, we obtain significant speed-ups while maintaining very competitive approximation quality even in the most fine-grained scenarios; thus pushing parallelism in MOEA/D to its limits. More importantly, our experimental study provides the first comprehensive results on the benefits and the limits of parallelizing the MOEA/D computational flow.

The rest of this paper is organized as follows. In Section II, we recall some basic definitions and provide a related work overview. In Section III, we describe our distributed scheme and discuss its message passing implementation. In Section IV, we provide a throughout experimental study on bi-objective ρMNK-landscapes. In Section V, we conclude the paper.

II. Background and related work

A. Multi-objective Optimization

A *multi-objective optimization problem* (MOP) can be defined by a set of M objective functions $f = (f_1, f_2, \ldots, f_M)$, and a set X of feasible solutions in the *decision space*. In the combinatorial case, X is a discrete set. Let $Z = f(X) \subseteq \mathbb{R}^M$ be the set of feasible outcome vectors in the *objective space*. To each solution $x \in X$ is assigned an objective vector $z \in Z$, on the basis of the vector function $f : X \to Z$. In a maximization context, an objective vector $z \in Z$ is dominated by an objective vector $z' \in Z$ ($z \prec z'$) iff $\forall m \in \{1, 2, \ldots, M\}$, $z_m \leq z'_m$ and $\exists m \in \{1, 2, \ldots, M\}$ such that $z_m < z'_m$. A solution $x \in X$ is dominated by a solution $x' \in X$ ($x \prec x'$) iff $f(x) \prec f(x')$. A solution $x^\star \in X$ is termed *Pareto optimal* (or *non-dominated*), if there does not exist any other solution $x \in X$ such that $x^\star \prec x$. The set of all Pareto optimal solutions is the *Pareto set*. Its mapping in the objective space is the *Pareto front*. In the combinatorial case, the size of the Pareto set is typically exponential in the size of the problem instance, and deciding if a solution belongs to the Pareto set may be NP-complete. Therefore, our goal is to identify a good *Pareto set approximation*. For this purpose, evolutionary multi-objective optimization (EMO) constitute a popular, effective and efficient alternative. In this paper, we are interested in decomposition-based methods, and especially on the MOEA/D framework.

B. The MOEA/D framework

Decomposition-based EMO algorithms [1], [5], [6] seek a good-performing solution in multiple regions of the Pareto front by *decomposing* the original MOP into a number of *scalarized* single-objective *sub-problems*. Different scalarizing functions have been proposed so-far [7]. In this paper, we use the weighted Chebyshev (g^{te}) function, to be minimized:

$$g^{te}(x, \lambda) = \max_{i \in \{1,\ldots,m\}} \lambda_i \cdot \left| z_i^\star - f_i(x) \right| ,$$

where $x \in X$, $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a positive weighting coefficient vector such that $\lambda_i \geqslant 0$ for all i, and $z^\star = (z_1^\star, \ldots, z_m^\star)$ is a reference point.

In the MOEA/D framework, sub-problems are optimized in a dependent manner by defining a *neighborhood relation* between sub-problems. In fact, solving a single sub-problem is likely to benefit from the current solutions found for the corresponding neighboring sub-problems. In more details, let $(\lambda^1, \ldots, \lambda^\mu)$ be a set of μ uniformly distributed weighting coefficient vectors defining μ sub-problems. MOEA/D maintains a population $P = (x^1, \ldots, x^\mu)$, each individual corresponding to a good-quality solution for one sub-problem. For each sub-problem $i \in \{1, \ldots, \mu\}$, a set of neighbors $\mathcal{B}(i)$ is defined by considering the T closest weight vectors. To evolve the population, sub-problems are optimized iteratively. At a given iteration corresponding to one sub-problem i, two solutions are selected at random from $\mathcal{B}(i)$, and an offspring solution x is created by means of variation operators (mutation and crossover). A problem-specific repair or improvement heuristic is potentially applied on solution x to produce x'. Then, for every sub-problem $j \in \mathcal{B}(i)$, if x' improves over j's current solution x^j then x' replaces it, i.e., if $g^{te}(x', \lambda^j) \leq g^{te}(x^j, \lambda^j)$ then set $x^j = x'$. The algorithm loops over sub-problems until a stopping condition is satisfied.

One can find other more specific variations of MOEA/D which may differ in the way scalarizing functions are defined, or in the way the neighbors are computed. We do not address these variations and we rather focus on the parallelization of the main MOEA/D components. In fact, the design of MOEA/D is inherently sequential since sub-problems are considered iteratively in a dependent manner. These dependencies are one of the main issues one has to deal with in a parallel setting. In the following, we review existing related solutions.

C. Parallelism in MOEA/D

Although one can find an extensive literature on the benefits of parallel and distributed computing with respect to EMO algorithms in general [8], we can report relatively few recent work related to the MOEA/D framework. The first study is conducted in [2] where the authors investigated the intuitive idea that non-overlapping sub-problems, *i.e.* sub-problems having disjoint T-neighbors, could be processed in parallel. A thread-based implementation, specific to a shared memory system with multi-core processors, was derived. The population is partitioned so that each thread handles a separate partition. Since the neighbors of sub-problems at the borders of partitions overlap, some elements of a thread could be concurrently modified. To avoid race conditions, critical sessions are identified and implemented accordingly. Using eight-core machine and continuous MOP benchmarks, it is shown that speed-ups can be obtained at the price of significantly deteriorating the approximation quality compared to the sequential MOEA/D. Actually, above 4 partitions for a population size of 600, the approximation quality starts to drop substantially, which indicates that the proposed approach cannot scale properly.

The idea of partitioning the MOEA/D population was extended in [3], and the partitions were made disjoint to avoid the overlapping issues. The observation that disjoint partitions may require different computational efforts was investigated. Advanced mechanisms controlling the size of partitions dynamically or evenly distributing the number of function evaluations were studied. Experimental results with 8 partitions for a population size of 300 were shown. Besides the limitations inherent to such a configuration, no parallel implementation was actually provided. In fact, the focus there-in was on studying the impact of the proposed models on quality, and not on parallel issues.

More recently, a parallel variant of MOEA/D based on the island model, and called PaDe, was investigated in [4]. In such a model, every island evolves a sub-population of individuals with respect to given sub-problems. Selected individuals are then sent to other islands during a migration phase. The parallel efficiency of the so-designed model is demonstrated with a 8-core shared-memory machine on a thread-basis implementation where a thread can handle a subset of islands. However, two major issues are left open. While PaDe is experimented with a self-adaptive differential evolution (DE) operator, different from the standard DE operator used to run MOEA/D, it is still not able to output similar approximation results. More importantly, a thread-based implementation is suitable for a shared memory system, and it is known that scaling up and adapting it for a distributed setting while maintaining its accuracy can suffer several shortcomings; because concurrent shared-memory read/write operations are no more possible, and distributed communication is typically many orders of magnitude more costly, which can be prohibitive.

Tightly related to the island model, novel generational sequential MOEA/D variants were described in [9]. The idea developed there-in is to rethink MOEA/D selection and replacement mechanisms by evolving the whole individuals simultaneously at once instead of iteratively one after the other. In their conclusion, the authors pointed out that a generational approach opens the door to incorporating parallelism in MOEA/D. However, moving to a distributed setting is not obvious and requires further careful investigations.

D. Discussion and Contribution Positioning

Despite the skillful design and the valuable efforts involved in the mentioned approaches, understating how the quality of MOEA/D is affected by parallelism, and what speed-ups can be attained when facing fine-grained parallelism at large distributed scales, is not yet fully accomplished. In this respect, our work departs from the previous studies in several aspects; but it also retains insightful lessons learnt from them. As in [2], [3], the idea of handling overlapping neighbors is a key point for scalability and high-quality approximation. As in [9], we adopt a generational approach to maintain good approximation quality. As in [4], we get inspired by the island model for the purpose of parallel efficiency. However, our approach is not explicitly based on the concept of islands and it is finely optimized to face fine-grained parallelism and to handle concurrency in the MOEA/D computational flow. As a by-product, the so-obtained scheme is proved very competitive when effectively implemented using message-passing under harsh configurations. This is a distinctive aspect of our work which gives room for further research in this line.

III. A FINE GRAINED DISTRIBUTED SCHEME

Our MOEA/D distributed scheme, called MP-MOEA/D, and its high-level implementation using the message passing paradigm are summarized in Algorithm 1 and the procedures of Algorithms 2 and 3. Before going into the details, it is important to remark that the template of Algorithm 1 is to be executed independently in parallel by *every* processing unit (all variables are local and not shared in any way). In fact, we assume that each processing unit (PU) p_i, $i \in \{0, \ldots, \mu\}$, is handling *one single sub-problem* with respect to weight vector λ^i, obtained by decomposition as in the standard MOEA/D framework. In other words, the population is partitioned into disjoint single-individual sub-populations evolved separately by every single PU[1]. In addition, the T neighbors of PU p_i are defined consequently as those holding the T neighboring sub-problems corresponding to weight vectors λ^j, $j \in \mathcal{B}(i)$. Having this in mind, the main responsibility of a PU is to identify the best solution with respect to its sub-problem while cooperating with its neighbors. For this purpose, the computations performed by PU p_i within the MP-MOEA/D scheme can be divided in two stages: The first stage (lines 3 to 14) is performed locally without any communication, whereas the second stage requires distributed communication with neighbors (line 16) as expanded in Algorithms 2 and 3.

A. Stage #1: Local Computations

The goal of PU p_i is two-fold: (i) to identify an improving solution for its own sub-problem, and (ii) to check whether an improving solution is found w.r.t. neighboring sub-problems. For this purpose, p_i maintains a representative *copy* of the solution of each neighbor. For clarity, we delay the mechanism to distributively maintain these copies to later.

Using its own solution and the copies with respect to its neighbors, a PU p_i performs the same selection and variation mechanisms as in the conventional MOEA/D, with essentially three main modification: (i) p_i's actual solution x^i is always selected for reproduction (line 7), (ii) the number of iterations during which new offspring solutions are generated is controlled by a parameter $t_{\max}$ (line 5), and (iii) since the actual remote solutions of neighbors are not known nor directly accessible, PU p_i simply checks whether any newly generated offspring does improve any of the local copy maintained for every neighbor (lines 13 to 14). The idea here is that, if the local copies are sufficiently up-to-date, then the protocol has the ability to distributively generate a 'good' offspring and to correctly detect any improvement on behalf of neighboring PUs. The improving offspring solutions are momentarily saved for the next stage.

B. Stage #2: Distributed Update

After the local computation stage, the second stage involving distributed communications and encapsulated within procedure DISTRIBUTED_UPDATE (line 16) is activated. Its goal is to distributively update the local states of PUs. In fact, two main situations can occur at every PU p_i after executing the first stage: (i) an improving offspring solution for its own sub-problem has been identified, (ii) an improving offspring solutions for one (or more) neighboring sub-problem(s) has

[1]Notice that this is a harsh assumption which allows us to fairly study the scalability of our scheme with very fine-grained computations.

Algorithm 1: MP-MOEA/D: High level code to be executed by *every* processing unit (PU) p_i, $i \in \{0, \ldots, \mu\}$

Input: $\mathcal{B}(i)$: neighboring sub-problems;
λ^j for every $j \in \mathcal{B}(i)$: neighbors' weight vectors;

INITIALIZE$\left(\cup_{j \in \mathcal{B}(i)} x^j, z^\star\right)$; flag $\leftarrow 0$;
while STOPPING CONDITION **do**
  // Stage #1: Local computations
  for $j \in \mathcal{B}(i) \setminus \{i\}$ **do** $y^j \leftarrow x^j$;
  ;
  Repeat $t_{\max}$ **times:**
    // Mating selection and variation
    $\ell \leftarrow rand(\mathcal{B}(i) \setminus \{i\})$;
    $y \leftarrow$ CROSSOVER_MUTATION_REPAIR(x^i, x^ℓ);
    for $m \in \{1, \ldots, M\}$ **do**
      if $z^\star_m < f_m(y)$ **then** $z^\star_m \leftarrow f_m(y)$;
      ;
    // Local Replacement
    if $g(y, \lambda^i) < g(x^i, \lambda^i)$ **then**
      $x^i \leftarrow y$; flag $\leftarrow 1$;
    // Check for neighbors' improvements
    for $j \in \mathcal{B}(i) \setminus \{i\}$ **do**
      if $g^{te}(y, \lambda^j) < g^{te}(y^j, \lambda^j)$ **then** $y^j \leftarrow y$;
      ;
  // Stage #2: Distributed update
  DISTRIBUTED_UPDATE ();

Algorithm 2: DISTRIBUTED_UPDATE: *Synchronous* message-passing implementation

// Notify neighbors
for $j \in \mathcal{B}(i) \setminus \{i\}$ **do**
  if *flag* $= 1$ **then** *Msg[0]* $\leftarrow x^i$;
  ;
  if $x^j \neq y^j$ **then** *Msg[1]* $\leftarrow y^j$;
  ;
  Send (Non-blocking) *Msg* **to** p_j;
// Wait for neighbors' notifications
flag $\leftarrow 0$;
for $j \in \mathcal{B}(i) \setminus \{i\}$ **do**
  Wait until reception (Blocking) of *Msg* **from** p_j;
  // Update neighbor's copy
  $v \leftarrow$ *Msg*[0];
  if $v \neq \varnothing$ **then** $x^j \leftarrow v$;
  ;
  // Update local solution
  $x \leftarrow$ *Msg*[1];
  if $x \neq \varnothing$ ***and*** $g^{te}(x, \lambda^i) < g^{te}(x^i, \lambda^i)$ **then**
    $x^i \leftarrow x$; flag $\leftarrow 1$;
  // Update local reference point
  for $m \in \{1, \ldots, M\}$ **do**
    if $z^\star_m < v_m$ **then** $z^\star_m \leftarrow v_m$;
    ;
    if $z^\star_m < x_m$ **then** $z^\star_m \leftarrow x_m$;
    ;

Algorithm 3: DISTRIBUTED_UPDATE: *Asynchronous* message-passing implementation

// Differences with Algo.2 are highlighted
for $j \in \mathcal{B}(i) \setminus \{i\}$ **do**
  if $x^j \neq z^j$ ***or*** *flag* $= 1$ **then**
    if *flag* $= 1$ **then**
      Msg[0] $\leftarrow i$; *Msg[1]* $\leftarrow x^i$;
    if $x^j \neq y^j$ **then** *Msg[2]* $\leftarrow y^j$;
    ;
    Send (Non-blocking) *Msg* **to** p_j;
// Check for neighbors' notifications
flag $\leftarrow 0$;
while *There is a pending message Msg* **do**
  $j \leftarrow$ *Msg*[0] ;
  $v \leftarrow$ *Msg*[1];
  if $v \neq \varnothing$ **then** $x^j \leftarrow v$;
  ;
  $x \leftarrow$ *Msg*[2];
  if $x \neq \varnothing$ ***and*** $g^{te}(x, \lambda^i) < g^{te}(x^i, \lambda^i)$ **then**
    $x^i \leftarrow x$; flag $\leftarrow 1$;
  for $m \in \{1, \ldots, M\}$ **do**
    if $z^\star_m < v_m$ **then** $z^\star_m \leftarrow v_m$;
    ;
    if $z^\star_m < x_m$ **then** $z^\star_m \leftarrow x_m$;
    ;

been identified. In the first case, p_i has to notify its neighbors so that they can update their local copies with p_i's new current solution. In the second case, p_i has to notify the corresponding neighbors so that they can update their own solutions with a new improving offspring. Symmetrically, p_i has to check whether these situations occur at one (or more) neighbor(s) before resuming a new stage of local computations.

The implementation of the above described mechanism is to be handled with special care since this is where fine-grained parallelism can prevent scalability. In fact, the fastest the states of PUs are updated with fresh information from neighbors, the better should be the approximation quality. On the other side, synchronizing the PU states distributively more often implies more communication cost, that might dominate the cost of local computations. Notice that this is precisely the reason why we introduced the parameter $t_{\max}$ in the previous stage, which offers the possibility of controlling the relative cost of local computations by fixing the communication frequency. Moreover, we experiment both synchronous (Algo. 2) and asynchronous (Algo. 3) message passing implementations where all distributed update operations are aggregated into a single message in order to reduce the number of messages transmitted over the network.

In the *synchronous* case, every PU sends a message with accurate information to its neighbors, and then blocks waiting for their respective messages to be received. This synchronous implementation provides the guarantee that all PUs will be updated with fresh information before resuming a new round of local computations. However, in order to avoid deadlocks, we need that every PU sends an acknowledgement (empty message) to its neighbors even if no change in the PU local state was observed. This has the drawback of introducing idle times where PUs are actually waiting for non-useful messages. The difficulty here is that a PU has no way to know whether a remote neighbor actually received a useful information.

The *asynchronous* implementation is technically very similar, but it removes the need to systematically send a message for neighbors and to wait for their messages. In fact, a PU updates a neighbor by sending non-empty messages containing

useful fresh information only when needed. Symmetrically, a PU needs to simply check, without blocking, whether some messages are pending. Although this asynchronous mechanism removes waiting times and synchronization costs, it has a major limitation compared to the synchronous version. In fact, depending on the relative cost of communications compared to local computations, a PU may miss to update itself with fresh information. Hence, the first stage can eventually be resumed for several rounds with outdated information, which can constitute a sever penalty in terms of approximation quality.

C. Discussion

We notice that once a PU p_i receives an offspring from a neighbor, this does not mean that the offspring is necessarily improving the actual solution of p_i since the sender PU has only a copy of p_i's solution which may be outdated at the time the new offspring was computed. In addition, if a PU p_i updates its solution because it received an improving offspring, then PU p_i needs to inform its neighbors. This is handled using the $flag$ variable. Another important point is the update of the reference point ($z^\star$) required by the Chebyshev function. In our implementation, each PU relies only on the information received from neighbors. To terminate the description of our scheme, we also notice that every process needs to initialize the local copies of its neighbors. This is performed before starting the main optimization loop by making every PU generating an initial solution for its sub-problem, and then sending it to its neighbors.

All along the previous description, each PU was assumed to optimize a single sub-problem. It should be clear that the scenario where population size is much larger than the number of available PUs can also be captured by simply partitioning the population accordingly and modifying our protocols to handle the local copies of neighboring sub-problems being at the frontier of each partition. It should also be clear that such a scenario would be much more favorable and less challenging. In fact, our preliminary experiments showed that the resulting workload is less fine-grained and maintaining the local state at each PU's partition is less critical. We however choose to focus on the most harsh scenario where every PU is handling one single solution in order to push our scheme to its limits and to fairly appreciate its strengths and weaknesses.

IV. Experimental Study

A. Experimental Setup

1) ρMNK-Landscapes: To analyze the behavior of our approach, we consider combinatorial bi-objective ρMNK-landscapes with a broad range of instances with different structures and sizes. The family of ρMNK-landscapes constitutes a problem-independent model used for constructing multi-objective multi-modal landscapes with objective correlation [10]. A multi-objective ρMNK-landscape aims at maximizing an objective function vector $f : \{0,1\}^N \rightarrow [0,1]^M$. Solutions are binary strings of size N. The parameter K defines the number of variables that influence a particular position from the bit-string (the epistatic interactions). By increasing the number of variable interactions k from 0 to $(N-1)$, landscapes can be gradually tuned from smooth to rugged. The objective correlation parameter ρ defines the degree of conflict between the objectives. The positive (resp. negative) data correlation allows to decrease (resp. increases) the degree of conflict between the objective function values. This has an impact on the cardinality of the Pareto front [10]. We investigate 50 random ρMNK-landscapes with $\rho \in \{-0.8, -0.4, 0.0, 0.4, 0.8\}$, $M = 2$, $N \in \{128, 256, 512, 1024, 2048\}$ and $K \in \{4, 8\}$.

2) Parameter setting: We deploy our scheme on top of the Grid5000 French grid [11]. We use a number of $\mu = 128$ cores (using a cluster of 46 times 2 CPUs Intel@2.83GHz, 4 cores/CPU), which also corresponds to the population size used in all our experiments. Our synchronous and asynchronous implementations are based on the C++ MPI library. The conventional sequential MOEA/D [1] is also considered as a baseline algorithm. All approaches use an independent bit-flip mutation with a rate $1/N$ and one-point crossover with probability 0.9. The initial population is generated randomly. The stopping condition is set to $\mu \cdot 10^4$ evaluation function calls. Based on preliminary experiments, we investigate a neighborhood size $T \in \{3, 5, 7\}$ and $t_{\max} \in \{1, 2, 3, 4, 5, 6\}$. Overall, we tested 1 950 different configurations, each one executed 30 times. Due to space limitations, we only highlight a subset of settings allowing us to state our findings.

B. Impact of Synchronicity

1) Approximation Quality: We follow the performance assessment design proposed in [12] using the hypervolume difference and the multiplicative epsilon indicators [13] which are both to be minimized. The hypervolume difference indicator (I_H^-) gives the difference between the portion of the objective space that is dominated by a Pareto set approximation and some reference set. The reference point is set to the worst value obtained over all approximations, and the reference set is the best-found approximation over all tested configurations. The epsilon indicator ($I_\varepsilon^\times$) gives the minimum multiplicative factor by which the approximation set has to be translated in the objective space in order to weakly dominate the reference set.

In Table I, we show a representative subset of results assessing the approximation quality of MP-MOEA/D when compared against the conventional MOEA/D. The sign "▲" (resp. "▽", "≈") indicates that the algorithm in the column outperforms (resp. is outperformed by, equivalent to) MOEA/D significantly using a Wilcoxon signed ranked test with a p-value of 0.05. We can see that *synchronous* MP-MOEA/D is the best performing algorithm. It is able to significantly improve upon the standard MOEA/D w.r.t. both the epsilon and hypervolume indicators in 25 out of 50 instances, while never being worst. The asynchronous implementation provides less impressive results since it improves upon MOEA/D on 7 instances, while being worst in also 7 instances (w.r.t. both indicators). It is also clear that the difference between MP-MOEA/D and MOEA/D is more substantial when considering higher dimensions, no matter whether the synchronous or the asynchronous implementation is considered.

Remember that the only difference between the synchronous and the asynchronous implementation is that in the former PUs have the guarantee to receive fresh information about neighbors states after every local computation stage, whereas in the latter, PUs do not spend time waiting for this information. Hence, these first observations show that synchronicity in the distributed update stage is a critical point

TABLE I. APPROXIMATION QUALITY OF MOEA/D *vs.* MP-MOEA/D FOR $T = 5$ AND $t_{\max} = 1$. A LOWER INDICATOR-VALUE INDICATES A BETTER PERFORMANCE. THE AVERAGE VALUE AND THE STANDARD DEVIATION (IN BRACES, WITH A MULTIPLICATIVE FACTOR OF 10^2) ARE REPORTED.

N	K	ρ	Hypervolume Difference (I_H^-)							Epsilon Indicator ($I_\varepsilon^\times$)						
			MOEA/D	MP-MOEA/D						MOEA/D	MP-MOEA/D					
				Synchronous			Asynchronous				Synchronous			Asynchronous		
128	4	−0.8	0.013	(0.25)	0.013	(0.19)	≈	0.015	(0.23)	≈	1.054	(1.15)	1.046	(1.02)	▲	1.055	(1.33)	≈
128	4	−0.4	0.017	(0.36)	0.017	(0.29)	≈	0.021	(0.37)	▽	1.054	(1.00)	1.049	(0.67)	≈	1.060	(1.07)	▽
128	4	0.0	0.020	(0.30)	0.017	(0.29)	▲	0.018	(0.22)	≈	1.073	(1.43)	1.059	(0.85)	▲	1.068	(1.13)	≈
128	4	0.4	0.011	(0.24)	0.010	(0.27)	≈	0.010	(0.22)	≈	1.051	(1.09)	1.046	(1.26)	≈	1.047	(0.79)	≈
128	4	0.8	0.005	(0.15)	0.004	(0.12)	▲	0.005	(0.12)	≈	1.047	(1.38)	1.035	(0.95)	▲	1.043	(1.07)	≈
128	8	−0.8	0.016	(0.27)	0.016	(0.28)	≈	0.019	(0.23)	▽	1.070	(1.24)	1.058	(1.41)	▲	1.071	(1.43)	≈
128	8	−0.4	0.020	(0.24)	0.018	(0.34)	≈	0.023	(0.29)	▽	1.064	(0.86)	1.057	(1.06)	▲	1.069	(0.88)	▽
128	8	0.0	0.021	(0.41)	0.022	(0.32)	≈	0.022	(0.36)	≈	1.073	(1.55)	1.077	(1.35)	≈	1.078	(1.19)	≈
128	8	0.4	0.016	(0.31)	0.015	(0.31)	≈	0.015	(0.18)	≈	1.068	(1.25)	1.064	(1.30)	≈	1.065	(0.94)	≈
128	8	0.8	0.007	(0.21)	0.007	(0.14)	≈	0.006	(0.12)	≈	1.052	(1.66)	1.057	(1.19)	≈	1.047	(1.05)	≈
256	4	−0.8	0.010	(0.18)	0.011	(0.22)	≈	0.013	(0.16)	▽	1.055	(1.11)	1.041	(0.96)	▲	1.055	(1.18)	≈
256	4	−0.4	0.013	(0.17)	0.012	(0.20)	▲	0.014	(0.24)	≈	1.059	(1.05)	1.043	(0.71)	▲	1.050	(0.87)	▲
256	4	0.0	0.011	(0.19)	0.010	(0.22)	▲	0.012	(0.17)	≈	1.050	(0.79)	1.041	(0.77)	▲	1.049	(0.81)	≈
256	4	0.4	0.008	(0.16)	0.007	(0.18)	≈	0.008	(0.13)	≈	1.045	(1.07)	1.039	(0.72)	▲	1.044	(0.68)	≈
256	4	0.8	0.004	(0.10)	0.003	(0.10)	≈	0.004	(0.08)	≈	1.037	(1.06)	1.033	(0.96)	≈	1.038	(0.92)	≈
256	8	−0.8	0.010	(0.22)	0.010	(0.18)	≈	0.013	(0.16)	▽	1.053	(1.26)	1.042	(0.90)	▲	1.056	(0.95)	≈
256	8	−0.4	0.013	(0.20)	0.013	(0.23)	≈	0.014	(0.24)	≈	1.054	(0.69)	1.044	(0.77)	▲	1.049	(0.85)	▲
256	8	0.0	0.028	(0.65)	0.024	(0.53)	▲	0.027	(0.57)	≈	1.285	(1.88)	1.274	(1.41)	▲	1.277	(1.61)	≈
256	8	0.4	0.008	(0.21)	0.009	(0.17)	≈	0.009	(0.21)	▽	1.042	(1.11)	1.045	(0.75)	≈	1.049	(0.87)	▽
256	8	0.8	0.004	(0.12)	0.004	(0.10)	≈	0.004	(0.12)	≈	1.039	(1.23)	1.038	(1.06)	≈	1.041	(1.18)	≈
512	4	−0.8	0.008	(0.14)	0.008	(0.12)	≈	0.010	(0.14)	▽	1.053	(1.01)	1.029	(0.43)	▲	1.043	(1.25)	▲
512	4	−0.4	0.011	(0.15)	0.009	(0.19)	▲	0.012	(0.16)	≈	1.056	(0.73)	1.038	(0.64)	▲	1.045	(0.53)	▲
512	4	0.0	0.009	(0.15)	0.007	(0.16)	▲	0.009	(0.14)	≈	1.048	(0.78)	1.034	(0.65)	▲	1.043	(0.46)	▲
512	4	0.4	0.007	(0.13)	0.006	(0.16)	▲	0.007	(0.10)	≈	1.045	(0.59)	1.036	(0.81)	▲	1.040	(0.46)	▲
512	4	0.8	0.003	(0.07)	0.003	(0.09)	≈	0.003	(0.06)	≈	1.031	(0.62)	1.028	(0.84)	≈	1.032	(0.67)	≈
512	8	−0.8	0.008	(0.14)	0.008	(0.13)	≈	0.010	(0.10)	▽	1.056	(0.98)	1.031	(0.46)	▲	1.045	(0.72)	▲
512	8	−0.4	0.010	(0.15)	0.008	(0.16)	▲	0.011	(0.13)	≈	1.056	(0.78)	1.035	(0.54)	▲	1.041	(0.43)	▲
512	8	0.0	0.010	(0.14)	0.009	(0.16)	≈	0.011	(0.13)	▽	1.050	(0.75)	1.042	(0.81)	▲	1.047	(0.51)	≈
512	8	0.4	0.006	(0.13)	0.005	(0.13)	▲	0.007	(0.13)	▽	1.037	(0.75)	1.033	(0.68)	▲	1.042	(0.73)	▽
512	8	0.8	0.003	(0.09)	0.003	(0.09)	≈	0.003	(0.07)	≈	1.033	(0.96)	1.030	(0.86)	≈	1.036	(0.82)	≈
1024	4	−0.8	0.007	(0.09)	0.006	(0.07)	▲	0.008	(0.11)	▽	1.063	(0.70)	1.028	(0.34)	▲	1.039	(0.57)	▲
1024	4	−0.4	0.009	(0.13)	0.007	(0.10)	▲	0.008	(0.09)	▲	1.061	(0.40)	1.039	(0.65)	▲	1.038	(0.60)	▲
1024	4	0.0	0.008	(0.15)	0.007	(0.11)	▲	0.008	(0.09)	≈	1.049	(0.65)	1.041	(0.63)	▲	1.040	(0.44)	▲
1024	4	0.4	0.005	(0.11)	0.004	(0.11)	▲	0.005	(0.08)	≈	1.036	(0.52)	1.029	(0.52)	▲	1.033	(0.48)	▲
1024	4	0.8	0.003	(0.08)	0.002	(0.08)	▲	0.003	(0.05)	≈	1.023	(0.69)	1.021	(0.70)	≈	1.029	(0.66)	▽
1024	8	−0.8	0.007	(0.09)	0.007	(0.06)	▲	0.008	(0.08)	▽	1.058	(0.67)	1.029	(0.33)	▲	1.040	(0.83)	▲
1024	8	−0.4	0.010	(0.10)	0.009	(0.09)	▲	0.009	(0.11)	▲	1.061	(0.60)	1.051	(0.92)	▲	1.043	(0.49)	▲
1024	8	0.0	0.008	(0.11)	0.006	(0.12)	▲	0.008	(0.09)	▲	1.053	(0.65)	1.042	(0.90)	▲	1.041	(0.33)	▲
1024	8	0.4	0.006	(0.10)	0.005	(0.10)	▲	0.006	(0.08)	≈	1.036	(0.63)	1.035	(0.71)	≈	1.039	(0.45)	▽
1024	8	0.8	0.002	(0.07)	0.002	(0.06)	▲	0.003	(0.06)	▽	1.020	(0.55)	1.017	(0.54)	▲	1.029	(0.64)	▽
2048	4	−0.8	0.007	(0.08)	0.005	(0.05)	▲	0.007	(0.05)	▽	1.074	(0.79)	1.036	(0.43)	▲	1.039	(0.45)	▲
2048	4	−0.4	0.009	(0.09)	0.007	(0.10)	▲	0.008	(0.07)	▲	1.075	(0.72)	1.057	(1.06)	▲	1.043	(0.68)	▲
2048	4	0.0	0.009	(0.10)	0.006	(0.07)	▲	0.007	(0.05)	▲	1.057	(0.46)	1.049	(1.05)	▲	1.047	(0.46)	▲
2048	4	0.4	0.006	(0.08)	0.003	(0.08)	▲	0.005	(0.06)	▲	1.035	(0.45)	1.031	(0.76)	▲	1.035	(0.48)	≈
2048	4	0.8	0.003	(0.09)	0.002	(0.06)	▲	0.004	(0.07)	▽	1.020	(0.50)	1.018	(0.49)	≈	1.026	(0.47)	▽
2048	8	−0.8	0.007	(0.06)	0.005	(0.05)	▲	0.007	(0.05)	≈	1.070	(0.54)	1.033	(0.37)	▲	1.039	(0.44)	▲
2048	8	−0.4	0.008	(0.04)	0.007	(0.06)	▲	0.007	(0.07)	▲	1.072	(0.34)	1.053	(0.99)	▲	1.037	(0.43)	▲
2048	8	0.0	0.008	(0.08)	0.006	(0.09)	▲	0.006	(0.07)	▲	1.060	(0.38)	1.047	(0.80)	▲	1.039	(0.40)	▲
2048	8	0.4	0.005	(0.07)	0.004	(0.07)	▲	0.005	(0.05)	▲	1.036	(0.43)	1.036	(0.64)	≈	1.036	(0.45)	≈
2048	8	0.8	0.002	(0.06)	0.002	(0.06)	≈	0.003	(0.05)	▽	1.016	(0.34)	1.018	(0.65)	≈	1.028	(0.42)	▽

in terms of approximation quality. However, as analyzed below, synchronicity comes with a running time penalty.

2) Running Time and Parallel Efficiency: In Fig. 1, we report some observations rendering the interrelation between parallel running time and approximation quality. On the left column of Fig. 1, we can appreciate the acceleration obtained by our implementations where we also vary the neighborhood size. The acceleration is measured as the ratio between the sequential running time of MOEA/D and the parallel running time of MP-MOEA/D. First, we can see that the *asynchronous* MP-MOEA/D is substantially faster. Second, we observe that the acceleration depends on the neighborhood size and also on the instance dimension. This is better illustrated in Fig. 2 where acceleration is analyzed as a function of the problem size. In fact, the evaluation cost increases with the problem dimension, thus making the relative cost of communication lower, and then leading to more significant accelerations. Moreover, larger neighborhoods imply a lager amount of communication, and thus less parallel efficiency. Notice however that even for the smallest instances of size $N = 128$ and the largest neighborhood, asynchronous MP-MOEA/D is still able to obtain significant accelerations while being very competitive in terms of approximation quality. This can be seen in the two last columns of Fig. 1 where the approximation quality of the synchronous and asynchronous implementations compared to sequential MOEA/D is well maintained.

3) Discussion: From the previous subset of results, we can conclude that there is a non-trivial trade-off to attain w.r.t. approximation quality and speed-up when parallelizing MOEA/D; which is well captured by our synchronous and asynchronous MP-MOEA/D. Actually, the previous analysis holds for $t_{\max} = 1$, which means that PUs distributively update their respective states immediately after performing one single evolutionary step. Like for synchronicity, controlling the number of steps a PU is authorized to perform in parallel before attempting to distributively update its neighbors has a deep impact on the type of trade-offs our MP-MOEA/D scheme is able to provide. This is analyzed in the rest of the paper.

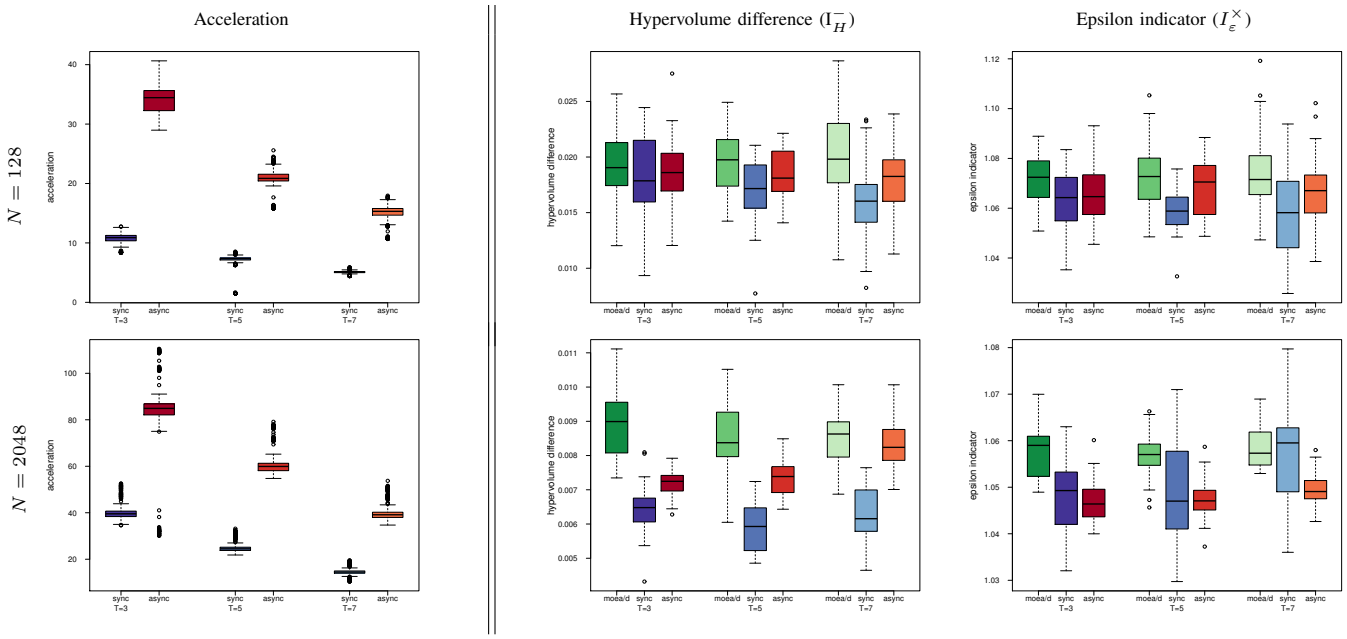

Fig. 1. Acceleration (Left side) *vs.* Approximation quality (Right side) for ρMNK-landscapes with $\rho = 0.0$ and $K = 4$; $t_{\max} = 1$.

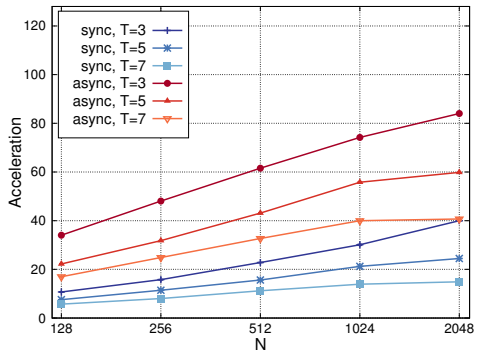

Fig. 2. Acceleration as a function of the problem size N for ρMNK-landscapes with $\rho = 0.0$ and $K = 4$; $t_{\max} = 1$.

C. Impact of Workload Granularity

We start by examining the acceleration we are able to obtain as a function of parameter $t_{\max}$. This is illustrated in Fig. 3 where one can additionally appreciate the relative impact of synchronicity and neighborhood size. We can see that larger $t_{\max}$-values lead to better speed-ups, which can clearly be attributed to the fact that workload granularity increases as a function of $t_{\max}$, thus reducing the relative cost of distributed communications. However, one may wonder the impact on quality. In order to better analyze the so-obtained trade-offs, we consider approximation quality (indicator's value) and acceleration as the *couple of goals* one would like to achieve simultaneously. This is illustrated in Fig. 4 where each point represents the acceleration (x-axis) and the approximation quality (y-axis) corresponding to one value of $t_{\max}$ (reported as a label in the figure), and where the horizontal line with thick contour represents the average approximation quality obtained by the sequential MOEA/D with a 95% confidence interval. Roughly speaking, and since the quality indicators are to be minimized, the points being below this line indicate that MP-MOEA/D is competitive compared to MOEA/D. Similarly, since acceleration is to be maximized (with an ideal value at 128), points being farther on the right-side of each subfigure indicate better parallel efficiency.

Several insightful observations can be extracted from the figure. Firstly, there is a general tendency that approximation quality drops with higher $t_{\max}$-values, whereas acceleration becomes better. The interesting observation is that there exists values for which quality is very competitive to sequential MOEA/D with substantial improvements in acceleration. In particular, when considering small-size instances, for which relatively high speed-ups are more difficult to achieve since the evaluation function cost becomes relatively low compared to communication cost, we observe that the impact of $t_{\max}$ on quality is less pronounced. Hence, larger $t_{\max}$-values have the overall effect of significantly improving acceleration without a substantial drop in terms of speed-up. Notice for example that speed-ups of up to 70 can be obtained with a quality being similar to MOEA/D for $N = 128$ whereas a speed-up of only 10 to 20 is obtained with $t_{\max} = 1$. Secondly, the obtained trade-offs are surprisingly dependent on the correlation between the objective functions, especially for large-size instances. In fact, the more correlated the objective functions, the harder it is to obtain higher acceleration without a significant drop in performance. We attribute this to the fact that for such an objective correlation, an improving offspring solution found at some PU with respect to a given sub-problem, is more likely to dominate neighboring solutions, and thus to subsequently produce new improving offspring that speed-up the convergence of the optimization of neighboring sub-problems. Hence, communicating improving offspring solutions immediately, as they are discovered at every PU, is more critical for approximation quality. This is less likely for anti-correlated instances for which the size of the Pareto front is larger, and where diversity can balance this side-effect. Notice also that the more conflicting the objectives, and the larger the instances, the more the asynchronous dominates the synchronous MP-MOEA/D in both approximation quality and acceleration. For example, for such configurations, asynchronous MP-MOEA/D is able to attain a near linear acceleration of about 100 while

being as good as MOEA/D in terms of approximation quality.

V. Conclusions and Perspectives

The study conducted in this paper provides new insights into the benefits of using parallel and distributed computations within the MOEA/D framework. Besides being able to provide high-quality approximation and acceleration, the proposed MP-MOEA/D distributed scheme and its message passing implementations allow us to better harness the key ingredients for a successful distributed deployment of decomposition-based approaches, and to elicit in a comprehensive manner the different trade-offs attainable at a large-scale distributed setting. The extensive experiments conducted on top of a real distributed testbed and the throughout analysis, show that MP-MOEA/D is able to counteract the fine-grained parallelism exposed by the original sequential MOEA/D framework. Our results are in fact obtained under the very harsh assumption that the number of available PUs are equal to the population size. It is our opinion that relaxing this assumption would hopefully lead to better trade-offs; however, it would be worthwhile conducting a new experimental study to complement our results when varying the number of sub-problems handled at each PU.

In addition, a natural, but challenging, open question is whether there exist other distributed strategies allowing to enhance acceleration while providing similar approximation quality. We believe in fact that there are still some opportunities to extend and to improve our MP-MOEA/D and its implementations in order to address this issue An interesting idea would be to continuously self-adjust the $t_{\max}$-value (instead of fixing it), as a function of the offspring improvement observed both locally at every processing units and remotely when performing distributed updates from neighboring processes.

References

[1] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE TEC*, vol. 11, no. 6, pp. 712–731, 2007.

[2] A. J. Nebro and J. J. Durillo, "A study of the parallelization of the multi-objective metaheuristic MOEA/D," in *International Conference on Learning and Intelligent Optimization (LION 4)*, 2010, pp. 303–317.

[3] J. Durillo, Q. Zhang, A. Nebro, and E. Alba, "Distribution of computational effort in parallel MOEA/D," in *International Conference on Learning and Intelligent Optimization (LION 5)*, 2011, pp. 488–502.

[4] A. Mambrini and D. Izzo, "PaDe: A parallel algorithm based on the MOEA/D framework and the island model," in *Parallel Problem Solving from Nature (PPSN XIII)*, 2014, pp. 711–720.

[5] I. Giagkiozis, R. C. Purshouse, and P. J. Fleming, "Generalized Decomposition," in *EMO*, 2013, pp. 428–442.

[6] E. J. Hughes, "Multiple Single Objective Pareto Sampling," in *CEC*, 2003, pp. 2678–2684.

[7] K. Miettinen, *Nonlinear Multiobjective Optimization*. Boston, MA, USA: Kluwer, 1999.

[8] D. A. Van Veldhuizen, J. B. Zydallis, and G. B. Lamont, "Considerations in engineering parallel multiobjective evolutionary algorithms." *IEEE TEC*, vol. 7, no. 2, pp. 144–173, 2003.

[9] G. Marquet, B. Derbel, A. Liefooghe, and E.-G. Talbi, "Shake them all! rethinking selection and replacement in MOEA/D," in *Parallel Problem Solving from Nature PPSN XIII*, 2014, pp. 641–651.

[10] S. Verel, A. Liefooghe, L. Jourdan, and C. Dhaenens, "On the structure of multiobjective combinatorial search space: MNK-landscapes with correlated objectives," *EJOR*, vol. 227, no. 2, pp. 331–342, 2013.

[11] Grid'5000 French national gird, "https://www.grid5000.fr/." [Online]. Available: https://www.grid5000.fr/

[12] J. Knowles, L. Thiele, and E. Zitzler, "A tutorial on the performance assessment of stochastic multiobjective optimizers," ETH Zurich, TIK Report 214, 2006.

[13] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca, "Performance Assessment of Multiobjective Optimizers: An Analysis and Review," *IEEE TEC*, vol. 7, no. 2, pp. 117–132, 2003.

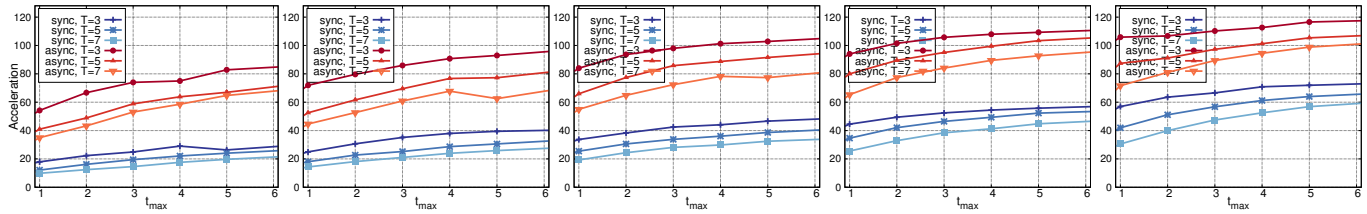

Fig. 3. Acceleration as a function of $t_{\max}$ for ρMNK-landscapes with $\rho = 0.0$, $K = 4$ and $N \in \{128, 256, 512, 1024, 2048\}$ resp., from left to right.

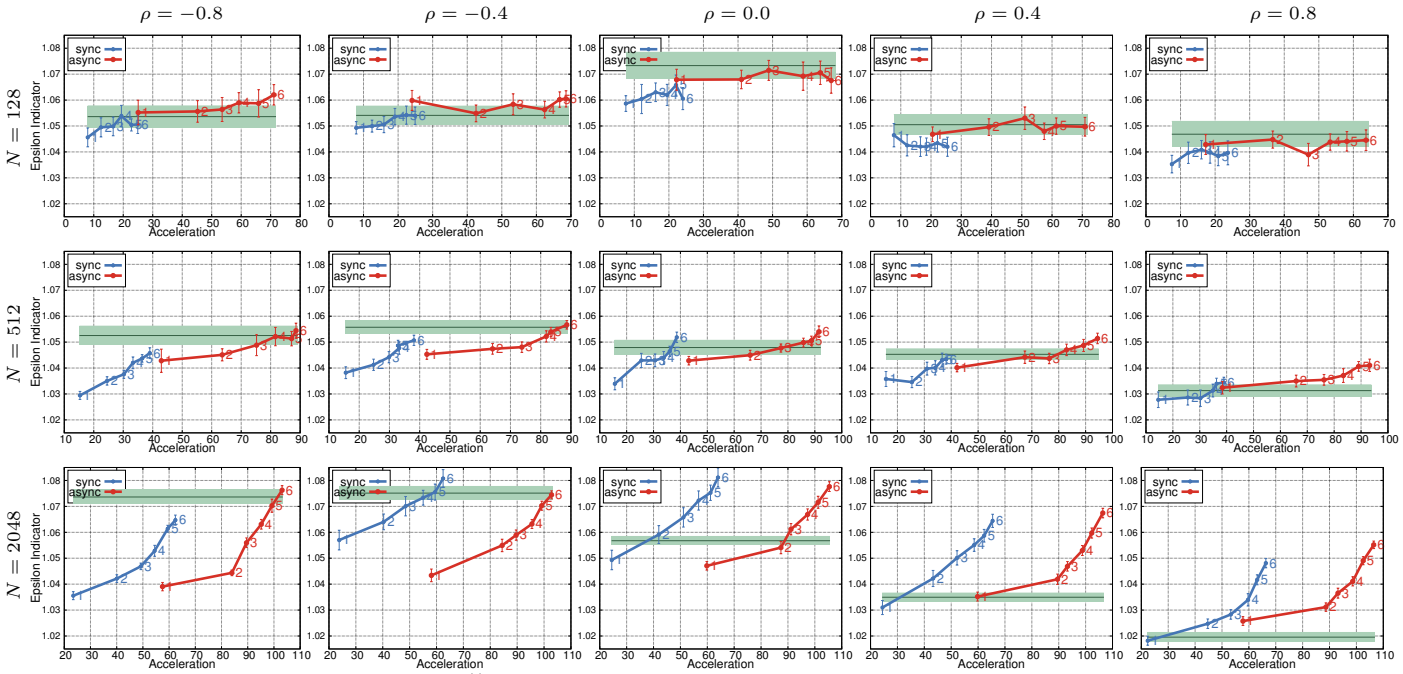

Fig. 4. Acceleration *vs.* Quality (Epsilon indicator $I_\varepsilon^\times$) as a function of $t_{\max}$ for ρMNK-landscapes with $K = 4$; the neighborhood size is $T = 5$.