# Distributed Localized Bi-objective Search

Bilel Derbel, Jérémie Humeau, Arnaud Liefooghe, Sébastien Verel

# Distributed Localized Bi-objective Search

Bilel Derbel[a,b], Jérémie Humeau[c], Arnaud Liefooghe[a,b,*], Sébastien Verel[d]

[a] *Université Lille 1, LIFL – CNRS, 59655 Villeneuve d'Ascq cedex, France*
[b] *Inria Lille-Nord Europe, 59650 Villeneuve d'Ascq, France*
[c] *École des Mines de Douai, département IA, 59508 Douai, France*
[d] *Université du Littoral Côte d'Opale, LISIC, 62228 Calais, France*

**Abstract**

We propose a new distributed heuristic for approximating the Pareto set of bi-objective optimization problems. Our approach is at the crossroads of parallel cooperative computation, objective space decomposition, and adaptive search. Given a number of computing nodes, we self-coordinate them locally, in order to cooperatively search different regions of the Pareto front. This offers a trade-off between a fully independent approach, where each node would operate independently of the others, and a fully centralized approach, where a global knowledge of the entire population is required at every step. More specifically, the population of solutions is structured and mapped into computing nodes. As local information, every node uses only the positions of its neighbors in the objective space and evolves its local solution based on what we term a '*localized fitness function*'. This has the effect of making the distributed search evolve, over all nodes, to a high quality approximation set, with minimum communications. We deploy our distributed algorithm using a computer cluster of hundreds of cores and study its properties and performance on $\rho$MNK-landscapes. Through extensive large-scale experiments, our approach is shown to be very effective in terms of approximation quality, computational time and scalability.

*Keywords:* Multiple objective programming, Combinatorial optimization, Parallel and distributed computing, Evolutionary computation

*Corresponding author
*Email addresses:* bilel.derbel@lifl.fr (Bilel Derbel), jeremie.humeau@mines-douai.fr (Jérémie Humeau), arnaud.liefooghe@univ-lille1.fr (Arnaud Liefooghe), verel@lisic.univ-littoral.fr (Sébastien Verel)

## 1. Introduction

### *1.1. Context and Motivation*

Many real-life problems arising in a wide range of application fields can be modeled as multi-objective optimization problems. One of the most challenging issues in multi-objective optimization is to identify the set of Pareto optimal solutions, *i.e.*, solutions providing the best compromises between the objectives. It is well understood that computing such a set is a difficult task. Designing efficient heuristic algorithms for multi-objective optimization requires one to tackle the classical issues arising in the single-objective case (*e.g.*, intensification *vs.* diversification), but also and more importantly, to find a set of solutions having good properties in terms of trade-off distribution in the objective space.

When dealing with such sophisticated problems, it is with no surprise that most existing approaches are costly in terms of computational complexity. A natural idea is to subdivide the problem being solved into subtasks which can be processed in parallel. This is a very intuitive idea when dealing with computing intensive applications, not only in the optimization field but in computer science in general. Besides, with the increasing popularity of high performance (*e.g.*, clusters), massively parallel (*e.g.*, multi-cores, GPUs), and large-scale distributed platforms (*e.g.*, grids, clouds), it is more and more common to distribute the computations among available resources taking much benefit of the induced huge computational power. Many parallel/distributed models and algorithms have been designed for specific optimization contexts. This witnesses the hardness of the tackled problems and the complexity of related algorithmic issues. Multi-objective optimization does not stand for an exception, since the multi-objective nature of the problem being solved induces additional computing intensive tasks.

One can find an extensive literature on designing parallel/distributed multi-objective solving methods (Van Veldhuizen et al., 2003; Coello Coello et al., 2007; Talbi et al., 2008; Bui et al., 2009). Most existing approaches are designed in a top-down manner, starting with a centralized algorithm requiring a *global* information about the search state; and then trying to adapt its components to the distributed/parallel computing environment. This design process usually requires to tackle parallel-computing issues which are challenging to solve efficiently and/or may impact the performance of the original sequential optimization algorithm. In contrast, *locality* in distributed computing is a well-known general paradigm that states that global informa-

tion is not always necessary to solve a given problem and local information is often sufficient (see *e.g.*, Peleg (2000)). Therefore, adopting a *localized* approach when tackling a given problem can allow one to derive novel algorithms which are by essence parallel and designed in a bottom-up manner. Those algorithms are more likely to allow distributed resources to coordinate their actions/decisions locally, and to take full benefit of the available computational power.

### *1.2. Contribution Overview*

In this paper, we describe a new simple and effective generic scheme dedicated to bi-objective heuristic search in distributed/parallel environments. Our approach is inherently *local*, meaning that it is thought to be independent of any global knowledge. Consequently, its deployment on a large-scale distributed environment does not raise parallel-specific issues.

Generally speaking, each computing node contains a candidate solution and is able to search in a region of the search space in coordination with other neighboring nodes. The sub-region where a node operates is delimited implicitly in an adaptive way based on the relative position of its cooperating neighbors in the objective space. The way local cooperation is designed, as well as its induced optimization process, are the heart of our approach. In our study, we propose novel *localized cooperative strategies* inspired by the classical weighted-sum scalarizing function (Ehrgott, 2005) and hypervolume-based approaches (Zitzler and Thiele, 1999), without requiring any global knowledge about the search state. The designed rules allow distributed nodes to self-coordinate their decisions adaptively and in an autonomous way while communicating a minimal amount of information; thus being effective when deployed on a real and large-scale distributed environment. To evaluate the performance of our approach, we conduct extensive experiments involving more than two hundred computing cores, and using $\rho$MNK-landscapes (Verel et al., 2013) as a benchmark. As baseline algorithms, we consider both a pure parallel strategy and an inherently sequential approach. Our experimental results show that our localized approach is extremely competitive in terms of approximation quality; while being able to achieve near-linear speed-ups in terms of computational complexity. Besides, we provide a comprehensive analysis of our approach highlighting its properties and dynamics.

*1.3. Outline*

In Section 2, we review existing works related to multi-objective optimization, especially those dealing with parallel and distributed issues. In Section 3, we describe the distributed localized bi-objective search approach proposed in the paper, and give a generic fully distributed scheme which can be instantiated in several ways. In Section 4, we provide the experimental setup of our analysis. In Section 5, we present numerical results and we discuss the properties of our approach. Finally, we conclude the paper in Section 6 and we discuss some open research issues.

## 2. Background on Multi-objective Optimization

In the following, we first introduce the basics of multi-objective optimization and then we position our work with respect to the literature.

*2.1. Definitions*

A *multi-objective optimization problem* can be defined by an objective function vector $f = (f_1, f_2, \ldots, f_M)$ with $M \geqslant 2$, and a set $\mathcal{X}$ of feasible solutions in the *solution space*. In the combinatorial case, $\mathcal{X}$ is a discrete set. Let $\mathcal{Z} = f(\mathcal{X}) \subseteq \mathbb{R}^M$ be the set of feasible outcome vectors in the *objective space*. To each solution $x \in \mathcal{X}$ is then assigned exactly one objective vector $z \in \mathcal{Z}$, on the basis of the function vector $f : \mathcal{X} \to \mathcal{Z}$ with $z = f(x)$. In a maximization context, an objective vector $z \in \mathcal{Z}$ is dominated by an objective vector $z' \in \mathcal{Z}$, denoted by $z \prec z'$, iff $\forall m \in \{1, 2, \ldots, M\}$, $z_m \leqslant z'_m$ and $\exists m \in \{1, 2, \ldots, M\}$ such that $z_m < z'_m$. By extension, a solution $x \in \mathcal{X}$ is dominated by a solution $x' \in \mathcal{X}$, denoted by $x \prec x'$, iff $f(x) \prec f(x')$. A solution $x^\star \in \mathcal{X}$ is said to be *Pareto optimal* (or *efficient*, *non-dominated*), if there does not exist any other solution $x \in \mathcal{X}$ such that $x^\star \prec x$. The set of all Pareto optimal solutions is called the *Pareto set* (or the *efficient set*). Its mapping in the objective space is called the *Pareto front*. One of the most challenging task in multi-objective optimization is to identify a complete Pareto set of minimal size, *i.e.* one Pareto optimal solution for each point from the Pareto front.

However, in the combinatorial case, generating a complete Pareto set is often infeasible for two main reasons (Ehrgott, 2005): (*i*) the number of Pareto optimal solutions is typically exponential in the size of the problem instance, and (*ii*) deciding if a feasible solution belongs to the Pareto set may be NP-complete. Therefore, the overall goal is often to identify a good

*Pareto set approximation*. To this end, heuristics in general, and evolutionary algorithms in particular, have received a growing interest since the late eighties (Deb, 2001; Coello Coello et al., 2007).

### *2.2. Literature Overview*

A large body of literature exists concerning parallel multi-objective algorithms. Two interdependent issues are usually addressed: (*i*) how to decrease the computational complexity of a specific multi-objective algorithms and (*ii*) how to make parallel processes cooperate to improve the quality of the Pareto set approximation, see *e.g.*, Zhu and Leung (2002); Jozefowiez et al. (2002); Deb et al. (2003); Coello Coello and Sierra (2004); Melab et al. (2006); Tan et al. (2006); Coello Coello et al. (2007); Mostaghim et al. (2007); Hiroyasu et al. (2007); Durillo et al. (2008); Talbi et al. (2008); Figueira et al. (2010); Mostaghim (2010). Often, parallel and cooperative techniques implicitly come with the idea of *decomposing* the search into many sub-problems so that a diversified set of solutions, in terms of Pareto front quality, can be obtained. The main challenge is on defining efficient strategies to either divide the search space or the objective space.

For instance, the population induced by a particle swarm multi-objective algorithm is divided by Mostaghim et al. (2007) into sub-swarms which are then coordinated through a master-slave approach by injecting the so-called subswarm-guides in each sub-population. The diffusion model (Van Veldhuizen et al., 2003) and the island model (Tomassini, 2005) have also been extensively adopted to design distributed cooperative methods. In the so-called cone separation technique (Branke et al., 2004), the objective space is divided into regions distributed over some islands. Each island explores the same search space. When a solution is outside its corresponding objective space region, it is migrated to neighboring islands. This idea is refined by Streichert et al. (2005) using a clustering approach. Bui et al. (2009) propose a distributed framework where a number of adaptive spheres spanning the search space and controlled by an evolutionary algorithm is studied. In Zhu and Leung (2002), a model where fully connected islands exchange information about their explored regions is considered. A strength Pareto evolutionary algorithm (Zitzler and Thiele, 1999) is then adopted to form the backbone for each island, but it is additionally equipped with a so-called adjusting instructive phenotype/genotype distance measure, computed according to the information exchanged with all other islands. In Zhang and

Li (2007), the authors described a decomposition-based approach, the so-called MOEA/D, which associates with each single solution a *fixed* scalar single-objective function called a sub-problem. Given a solution and its corresponding sub-problem, a new offspring is created using the genotypes of solutions corresponding to neighboring sub-problems. This process is then repeated iteratively for each sub-problem which makes it inherently sequential. Some recent attempts exist in order to adaptively define the sub-problem parameters in the sequential setting (Qi et al., 2014). Parallel extensions and models for MOEA/D are described by Nebro and Durillo (2010) and Durillo et al. (2011) for shared memory systems. The so-obtained approximation quality is however shown to deteriorate significantly for more than 8 parallel processes.

To the best of our knowledge, existing parallel and cooperative algorithms usually treat a multi-objective optimization process in a global manner and do not fully explore other more *local* alternatives when thinking the role of cooperation.

### *2.3. Positioning*

The approach proposed in this paper can be viewed as a parallel and cooperative method, since solutions in our approach shall both evolve in parallel while cooperating locally. However, the information exchanged between neighboring nodes does not involve any migration of solutions as it is the case in most island-based approaches. It can also be viewed as a decomposition-oriented strategy since it implicitly induces a partition of the global search in many sub-search processes, focusing on different regions of the objective space. The search process is however dynamically distributed over the objective space without relying on any global information, *e.g.*, elite solutions, external population, global fitness measure, etc. In other words, we do *not* explicitly partition the search space through cooperating entities (islands, processes, etc), *nor* we explicitly partition the objective space among parallel entities. We simply evolve solutions in an adaptive manner based on localized fitness functions, which are instantiated dynamically. Unlike previous centralized/sequential adaptive approaches, we focus on distributing the search among cooperating computing entities, while relying on a strictly local information learned from neighbors.

## 3. A Distributed Localized Bi-objective Search Approach

Let us consider that we are given a set of $n$ computing nodes scattered over a network. Our idea is to distribute a population of $\mu$ solutions among the $n$ computing nodes with the aim of $(i)$ evolving them towards a good Pareto set approximation, and $(ii)$ naturally fitting the distributed nature of the computing environment to significantly gain in terms of execution time. For this purpose, we structure the population of solutions following a *line* where every solution, except those being at the two extremes of the line, have exactly two distinct neighbors. According to this logical line structure, we design *local* rules based on the relative positions of neighboring solutions in the objective space. These rules are based on the definition of *localized fitness functions* allowing current solutions to be replaced by new candidate solutions cooperatively; and to evolve distributively while exploring diversified regions of the objective space. The localized fitness function, denoted by $\mathcal{LF}$, is the key ingredient of our approach.

In the following, we start describing our approach in the scenario where each solution is mapped to a single computing node. This specific scenario shall allow us to better illustrate the locality of our distributed approach and its parallel nature in a more comprehensive way. Later, we shall show how we can extend to other scenarios where an arbitrary number of computing nodes is considered.

### *3.1. Algorithm Overview*

In the following sections, we consider that each solution is assigned to one computed node such that $n = \mu$. In this case, and since we structure the population according to a line, we can also view computing nodes as organized in a logical communication line graph $L_n = (v^1, v^2, \cdots, v^n)$, *i.e.*, node $v^1$ (resp. $v^n$) holding solution $x^1$ can communicate with neighbor $v^2$ (resp. $v^{n-1}$) and any other node $v^i$, with $i \in \{2, \cdots, n-1\}$, holds solution $x^i$ and can communicate with neighbors $v^{i-1}$ and $v^{i+1}$, holding respectively solutions $x^{i-1}$ and $x^{i+1}$. In the following, we interchangeably use the terms *node* and *solution* to describe both the evolution and the communication mechanisms involved in our approach.

The proposed distributed localized bi-objective search (DLBS) algorithm is illustrated in the high-level pseudo-code of Algorithm 1. Distributively in parallel, every computing node in the line graph $L_n$ operates in local rounds until a stopping condition is satisfied. At each communication round, a

**Algorithm 1:** DLBS – Pseudo-code for *every* node $v^i \in L_n$

```
1  x^i ← initial solution corresponding to node v^i;
2  repeat
3      /* communicate positions */
4      z^i ← (z_1^i, z_2^i) the position of solution x^i in the bi-objective space, z^i = f(x^i);
5      Send z^i to neighboring nodes;
6      Z^i ← receive neighboring positions;
7      /* variation */
8      S^i ← New_Solutions(x^i);
9      /* selection for replacement */
10     x^i ← Select(S^i, LF^{Z^i});
11 until STOPPING_CONDITION;
```

node simply exchanges the current position of its incumbent solution in the objective space with its neighbors, *i.e.*, every node $v^i \in L_n$ sends the position $f(x^i) = z^i = (z_1^i, z_2^i)$ of its current solution $x^i$ to its neighbors and receives the positions $Z^i = (z^{i-1}, z^{i+1})$ sent symmetrically by its neighbors.

After each local communication round, a node $v^i$ evolves its current solution $x^i$ in the following way. First, it generates a bunch of new solutions $S^i$ based on the current solution $x^i$, (function New_Solutions, Line 8). This function is to be understood as any component that, given a solution, is able to generate a set of candidate solutions $S^i$ by means of a problem specific variation operator. Among the candidate set $S^i$, a new solution is selected to replace the current one (function Select, Line 10), and so on, concurrently for all nodes.

The line graph connecting solutions can then be viewed as a line linking some points in the objective space. The goal is to push the line a little bit more towards the Pareto front at each round by replacing current solutions with new ones. The selection for replacement is made on the basis of a scalar value computed by means of a *localized fitness function*, denoted by $\mathcal{LF}$. Notice that function $\mathcal{LF}$ is itself parametrized by the pair $Z^i$, referring to the positions communicated by neighboring nodes. We emphasize the fact that function $\mathcal{LF}$ does not use any other kind of information but the position of neighboring solutions; thus making it very local in nature. In the following paragraphs, we describe in detail how the selection for replacement instruction (Line 10) is instantiated in the proposed DLBS algorithm.

*3.2. Selection for Replacement*

We start by describing the local rules for both nodes $v^1$ and $v^n$, holding the extreme solutions $x^1$ and $x^n$, which play a particular role in our distributed algorithm. In fact, extreme nodes $v^1$ and $v^n$ shall guide the search through the extreme points of the Pareto front, following the lexicographic order implied by the objective functions. For node $v^1$, we consider that a solution $x$ is lexicographically better than or equal to a solution $x'$, if $f_1(x) > f_1(x')$ or if $f_1(x) = f_1(x')$ and $f_2(x) > f_2(x')$. Symmetrically, for node $v^n$, a solution $x$ is lexicographically better than or equal to a solution $x'$, if $f_2(x) > f_2(x')$ or if $f_2(x) = f_2(x')$ and $f_1(x) > f_1(x')$. Using respectively these lexicographical orders, the local selection used by nodes $v^1$ and $v^n$ to replace their current solutions is then fully defined. Notice that each lexicographic optimal solution is a Pareto optimal solution of the initial multi-objective problem, mapping to an extreme point of the Pareto front (Ehrgott, 2005).

The local strategy applied by nodes $v^1$ and $v^n$ is independent of their respective neighbors $v^2$ and $v^{n-1}$. This is essentially due to the fact that we want the extreme nodes to push the line graph as much as possible to the extreme regions of the Pareto front. For other nodes $v^i$, $i \in \{2, \cdots, n-1\}$, the selection for replacement is based on a localized fitness function $\mathcal{LF}$ that depends on neighbors' positions. At each step, the candidate solution with *the best* $\mathcal{LF}$-value is selected for the next round. In the next paragraphs, we define and discuss the localized fitness functions designed for DLBS.

*3.3. Localized Fitness Functions*

Two localized fitness functions, to be used within the DLBS algorithm, are proposed below. They are based on two different strategies for aggregating the objective function values.

*3.3.1. Orthogonal-Directed Localized Fitness Function*

Our first localized fitness function, denoted by $\mathcal{LF}_{OD}$, is based on a classical scalarizing approach from multi-objective optimization, namely a weighted-sum aggregation. At each node $v^i$, $i \in \{2, \cdots, n-1\}$, let $Z^i$ be the pair of neighboring positions for node $v^i$. More specifically, $(z_1^{i-1}, z_2^{i-1})$ (resp. $(z_1^{i+1}, z_2^{i+1})$) refers to position $z^{i-1}$ (resp. $z^{i+1}$) communicated by neighbor $v^{i-1}$ (resp. $v^{i+1}$). Without loss of generality, we assume that $z_1^{i-1} \leqslant z_1^{i+1}$, otherwise node $v^i$ simply interchanges the coordinate of its neighbors in the following equations. Given a candidate solution $x$ taken from the candidate

set $S^i$ generated at node $v^i$, $x$ is scored according to the following function.

$$\mathcal{LF}_{OD}^{Z^i}(x) = w_1 \cdot f_1(x) + w_2 \cdot f_2(x) \tag{1}$$

where

$$w_1 = z_2^{i-1} - z_2^{i+1} \quad , \quad w_2 = z_1^{i+1} - z_1^{i-1}$$

Notation $OD$ stands for *Orthogonal Direction*. This is inspired by the dichotomic scheme proposed by Aneja and Nair (1979). In such approach, weighting coefficient vectors are determined according to the position of solutions found in previous iterations. However, in our approach we use the *current* neighboring positions at each node to evolve the corresponding solution at runtime. The weighting coefficient vector $w = (w_1, w_2)$ is then calculated distributively at each node as the orthogonal of the segment defined by $z^{i-1}$ and $z^{i+1}$, as illustrated in Figure 1. This localized fitness function defines the search direction of the distributed algorithm concurrently at each node of the line graph. It is important to remark that the computed weighting coefficient vectors can change from one round to another. It may also happen that a weighting coefficient has a negative value, which should not be necessarily perceived as a drawback, since it could help to explore diversified regions. Notice moreover that a number of Pareto optimal solutions, known as *unsupported solutions*, are not optimal for any definition of the weighting coefficients (Ehrgott, 2005). Our distributed strategy using an orthogonal-directed localized fitness function $\mathcal{LF}_{OD}$ will be denoted by DLBS$_{OD}$ in the remainder of the paper.

*3.3.2. A Hybrid Hypervolume-based Localized Fitness Function*

The second variant of our localized fitness function, denoted by $\mathcal{LF}_H$, is based on the hypervolume indicator (Zitzler and Thiele, 1999; Zitzler et al., 2003). Many efficient evolutionary multi-objective optimization algorithms are based on optimizing the hypervolume value of the output set, see *e.g.* Beume et al. (2007); Bader and Zitzler (2011). Given $M$ objective functions, the hypervolume indicator value of a set $A$ of mutually non-dominated objective vectors can be defined as follows.

$$I_H(A) = \Lambda \left( \bigcup_{z \in A} [z_1, z_1^{ref}] \times \cdots \times [z_M, z_M^{ref}] \right) \tag{2}$$

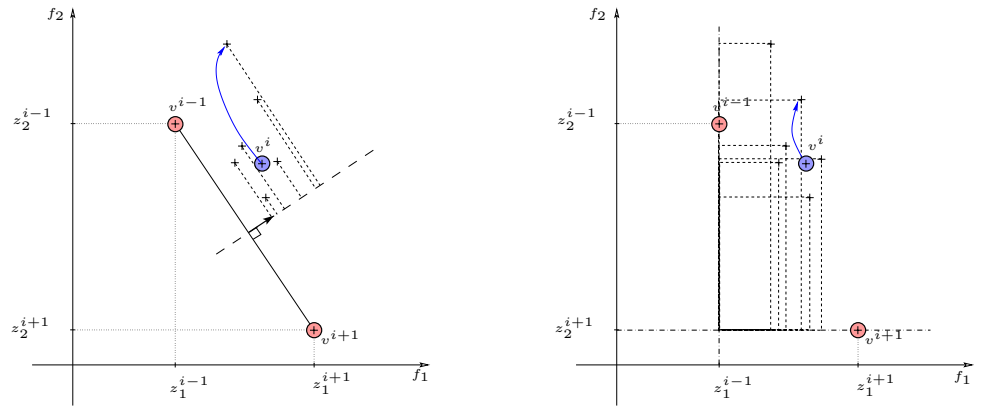


Figure 1: Illustration of the selection for replacement using to the localized fitness function $\mathcal{LF}_{OD}$ (left) and $\mathcal{LF}_H$ (right). All solutions $i \in \{2, \cdots n-1\}$ concurrently adopt the same strategy with respect to their relative neighbors. The crosses without circle are the candidate solutions $S^i$. The arrow shows the selected candidate solutions that replace the current one $v^i$.

with $z^{ref} \in \mathcal{Z}$ a reference point and $\Lambda(\cdot)$ the Lebesgue measure. The *hypervolume contribution* of a point $z \in \mathcal{Z}$ with respect to a non-dominated set $A$ is then given as follows (Beume et al., 2007).

$$\Delta_H(z, A) = I_H(A) - I_H(A \setminus \{z\}) \tag{3}$$

Dominated points do not contribute to the hypervolume. In the two-objective case, if we assume that the elements of the non-dominated set $A$ are sorted in the increasing order with respect to $f_1$-values, the hypervolume contribution can be reduced as follows.

$$\Delta_H(z^i, A) = (z_1^i - z_1^{i-1}) \cdot (z_2^i - z_2^{i+1}) \tag{4}$$

In our distributed approach, a node does not have a global view of the current population of solutions being processed in parallel by other nodes. The only information a node $v^i$ can use is the position of its two neighboring solutions in objective space, *i.e.* $Z^i$. Without loss of generality, let us assume that $z_1^{i-1} \leqslant z_1^{i+1}$. Our second hybrid hypervolume-based localized fitness function is defined as follows.

$$\mathcal{LF}_H^{Z^i}(x) = \begin{cases} (f_1(x) - z_1^{i-1}) \cdot (f_2(x) - z_2^{i+1}) & \text{if } f_1(x) \geqslant z_1^{i-1} \text{ and } f_2(x) \geqslant z_2^{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

This is illustrated in Figure 1. Notice that $\mathcal{LF}_H$ is though to be the local adaptation/version of Eq. (4). In particular, it intuitively states that, by selecting those candidate solutions maximizing the local hypervolume contribution at each node, the global hypervolume of the new set of solutions is likely to be better than the previous one. However, it may happen that all solutions generated in the candidate set have a $\mathcal{LF}_H$-value of 0, *e.g.* when they are all dominated by at least one neighboring position. Therefore, in this special case where no solution has a positive hypervolume contribution, we use the orthogonal-directed localized fitness function in order to avoid a random selection and make the current solutions evolving closer to the Pareto front. When using the hybrid hypervolume-based localized fitness function $\mathcal{LF}_H$, our approach is denoted by DLBS$_H$.

*3.4. General Population Mapping*

In the previous paragraphs, the number of computing nodes $n$ is assumed to be equal to the population size $\mu$, *i.e.*, $n = \mu$. However, in order to achieve a better Pareto set approximation, one might want to use a population size which is substantially larger than the number of computing nodes available in practice. In this case, we argue that restrictions on the number of available computing resources cannot prevent the implementation and the deployment of the DLBS approach for a large population size.

For the scenario where $n < \mu$, we shall simply increase the number of solutions evolving *at every* computing node. For simplicity, let us assume that $\mu$ is a multiple of $n$. As done previously, the population is then structured following a line graph $L_\mu$ and every solution in the line graph is evolved following the previously defined localized rules. However, the line graph is now split into $n$ contiguous sub-lines of length $\mu/n$. In other words, we distribute the population evenly among available computing resources by assigning a unique sub-line to every single computing node. Every node $v^j \in \{1, \cdots, n\}$ is then responsible for evolving the whole path of solutions $L_n^j = (x^{\frac{(j-1)\cdot\mu}{n}+1}, \cdots, x^{\frac{j\cdot\mu}{n}})$ according to the same localized rules. It is easy to see that no communication is required for any solution inside a sub-line $L_n^j$; since the position of solutions inside $L_n^j$ are available at the same computing node $v^j$. Communication is only required in order to exchange the positions of solutions being at the boundaries of the sub-line, *i.e.* solutions $x^{\frac{(j-1)\cdot\mu}{n}+1}$ and $x^{\frac{j\cdot\mu}{n}}$ for computing node $v^j$. Notice that in the case where $\mu \bmod n \neq 0$, it is also easy to manage the size of the sub-lines to be the same up to a difference of one.

In the remainder of the paper, we use the term *granularity* to refer to the number of computing nodes with respect to the population size in DLBS. The lowest granularity is for the configuration where $n = \mu$, *i.e.*, one solution *per* node as depicted in the pseudo-code of Algorithm 1; and the highest one is for $n = 1$, *i.e.*, all solutions are assigned to a single computing node. Different granularities in these two extreme ranges will be investigated in order to evaluate the performance of DLBS from a purely parallel perspective. For clarity, notation DLBS$(n, \mu)$ shall refer to a configuration with $n$ computing nodes and a population of size $\mu$. It is important to remark that for a given population size and a given stopping condition, the granularity induced by the number of computing nodes $n$ does not have any impact on the quality of the obtained Pareto set approximation.

## 4. Experimental Setup

This section summarizes the experimental setting allowing us to analyze the proposed approach on the bi-objective $\rho$MNK-landscapes, with a broad range of problems with different structures and sizes.

### *4.1. $\rho$MNK-landscapes*

In the single-objective case, the family of NK-landscapes is a problem-independent model used for constructing multimodal landscapes (Kauffman, 1993). Feasible solutions are represented as binary strings of size $N$, *i.e.* the solution space is $\mathcal{X} = \{0, 1\}^N$. Parameter $N$ refers to the problem dimension (*i.e.* the bit-string length), and $K$ to the number of variables that influence a particular position from the bit-string (*i.e.* the epistatic interactions). In single-objective NK-landscapes, the objective function $f : \{0, 1\}^N \rightarrow [0, 1)$ is defined as follows.

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} c_i(x_i, x_{i_1}, \cdots, x_{i_K}) \tag{6}$$

where $c_i : \{0, 1\}^{K+1} \rightarrow [0, 1)$ defines the component function associated with variable $i \in \{1, \cdots, N\}$, and where $K < N$. By increasing the number of variable interactions $K$ from 0 to $(N - 1)$, NK-landscapes can be gradually tuned from smooth to rugged. In this work, we set the position of these interactions uniformly at random.

In multi-objective NK-landscapes (Aguirre and Tanaka, 2007), component values are defined randomly and independently for every objective so that it results in a set of $M$ independent objective functions. More recently, multi-objective NK-landscapes with correlated objective functions have been proposed (Verel et al., 2013). Component values now follow a multivariate uniform law of dimension $M$, defined by a correlation matrix. We here consider the same correlation between all pairs of objective functions, given by a correlation coefficient $\rho > \frac{-1}{M-1}$. The same epistasis degree $K_m = K$ is used for all $m \in \{1, \cdots, M\}$. For more details on $\rho$MNK-landscapes, the reader is referred to Verel et al. (2013).

*4.2. Competing Algorithms*

We recall that our distributed strategies are denoted by DLBS$_{OD}$ and DLBS$_H$ when an orthogonal-directed localized fitness function $\mathcal{LF}_{OD}$, or respectively, a hybrid hypervolume-based localized fitness function $\mathcal{LF}_H$, is used. To evaluate the relative approximation quality of our algorithms, we compare them against a pure parallel approach, denoted by PIWS, and a pure sequential one, denoted by HEMO. They are sketched below.

- PIWS (Parallel Independent Weights Search) is a weighted-sum scalarized approach, where weighting coefficient vectors are uniformly defined *a priori*, and do not change during the search process. For each node $v^i$, $i \in \{1, \cdots, n\}$, the weighting coefficient vector is defined as follows.

$$w_1^i = \frac{n-i}{n-1} \quad and \quad w_2^i = 1 - w_1^i \tag{7}$$

  PIWS consists in running multiple rounds of parallel independent heuristic search algorithms following different fixed weighting coefficient vectors. Compared to our localized strategies, no information is communicated between nodes when running PIWS. This allows us to appreciate the impact of our localized strategies on approximation quality and also the impact of distributed communications on running time.

- HEMO is a *sequential* and *global* hypervolume-based evolutionary multi-objective optimization algorithm (HEMO). It is based on dominance-depth ranking, and on the contributing hypervolume at the second-level sorting criterion. In other words, the second-level sorting criterion of NSGA-II (Deb et al., 2002), *i.e.* the crowding distance, is replaced by

the contributing hypervolume, as in the hypervolume-based localized fitness function, but here used in a more global way. The resulting algorithm can also be seen as a $(\mu + \lambda)$ variant of SMS-EMOA (Beume et al., 2007), with a one-shot replacement strategy. Notice that we have implemented HEMO using the fast $\mathcal{O}(\mu \log \mu)$ dominance-depth ranking procedure (Deb, 2001). HEMO allows us to appreciate how efficient our local strategies are compared with a global strategy having a full global knowledge of the search state, *i.e.* the whole current population.

*4.3. Parameter Setting*

We remind that $N$ refers to the problem dimension of $\rho$MNK-landscapes. The number of computing nodes is denoted by $n$. The population size is denoted $\mu$. The initial population is generated as random binary strings. At every round/iteration of DLBS or PIWS, we generate a set of $\lambda$ offspring *per* solution using an independent bit-flip mutation operator, where each bit is mutated at random with a probability $1/N$. In other words, a $(1 + \lambda)-$evolutionary algorithm iteration with stochastic bit-flip mutation is performed for each single solution in the population. In the reported results, we shall consider the case where $\lambda$ is set to $N$. For each solution in the population, we perform $N$ iterations. The total number of evaluations for DLBS and PIWS is thus $\mu \times \lambda \times N = \mu \times N^2$

For the competing HEMO algorithm, the population size $\mu$ and the number of offspring $\lambda$ are set to same values than DLBS and PIWS, *i.e.*, $\lambda = N$. The HEMO variation operator is also based on bit-flip mutation only; *i.e.*, no recombination operator is used. For comparison purposes, the stopping condition of HEMO is given in terms of a maximum number of evaluations which is chosen to be same than the other algorithms, *i.e.*, $\mu \times N$ generations and hence $\mu \times N^2$ evaluations in total.

All algorithms have been implemented with the help of the Paradiseo software framework (Liefooghe et al., 2011; Humeau et al., 2013), available at the following URL: `http://paradiseo.gforge.inria.fr/`. The distributed implementation and the communication between nodes have been done using the standard MPI library. In our parallel implementation, every two MPI processes exchanging the solution positions are implicitly synchronized using standard MPI send and receive blocking primitives. However, no barrier is used to synchronize the whole MPI processes. In this way, our implementation is very faithful to the semi-synchronous pseudo-code given in Algorithm 1. The experiments have been conducted on a cluster of 70 computing

nodes inter-connected in a distributed computing environment running under CentOS 5.2, with a total number of 600 cores, 6 TeraFlops, and 1872 GB RAM. The following nodes have been used during our experiments: up to 30 computing nodes with two quad-core Opterons Shangai processors (2.5 GHz, 16 GB RAM), and up to 22 computing nodes with two quad-core Intel Xeon L5520 processors (2.26 GHz, 24 GB RAM).

In the following, we conduct an experimental study on the influence of the problem dimension ($N$), the non-linearity ($K$), and the objective correlation ($\rho$) for bi-objective $\rho$MNK-landscapes ($M = 2$) on the performance of the algorithms under study in the paper. In particular, we investigate the following parameters: $N \in \{128, 256, 512, 2048\}$, $K \in \{4, 8\}$ and $\rho \in \{-0.7, 0.0, +0.7\}$. One instance, generated at random, is considered *per* parameter setting. The corresponding $\rho$MNK-landscape instances can be found at the following URL: `http://mocobench.sourceforge.net/`.

For the DLBS algorithm, we shall consider several configurations by varying the population size $\mu \in \{8, 16, 32, 64, 128, 256\}$. If not stated explicitly, the finest granularity is considered when deploying DLBS; which corresponds to the the situation where $n$ is set to be equal to $\mu$ (*i.e.*, one single solution *per* computing node). Nevertheless, we shall also study DLBS under different granularities to experimentally investigate the issues discussed in Section 3.4. More specifically, for a fixed population size $\mu$, results are reported for $n \in \{1, 8, 16, 32, 64, 128\}$. For each tuple of parameter setting and algorithm variant, 30 independent executions are performed.

## 5. Experimental Results and Analysis

Due to the parallel nature of DLBS, one should examine simultaneously approximation quality and running time in order to fully appreciate its performance with respect to other competing algorithms. The running time of DLBS depends on the granularity chosen when effectively deploying it on a computational environment (see Section 3.4). For our first set of experiments, the number of nodes $n$ is set to the population size $\mu$, corresponding to the finest possible granularity. We start discussing approximation quality and then we relate it to running time.

### *5.1. Approximation Quality*

A set of 30 runs *per* instance is performed for each algorithm. In order to evaluate the quality of the approximations found for every considered in-

stance, we follow the performance assessment protocol proposed by Knowles et al. (2006). Given a $\rho$MNK-landscape instance, we compute a reference set $Z_N^\star$ containing the non-dominated points of all the Pareto front approximations we obtained during all our experiments. To measure the quality of a Pareto front approximation $A$ in comparison to $Z_N^\star$, we use both the hypervolume difference indicator ($\mathrm{I}_H^-$) and the multiplicative epsilon indicator ($\mathrm{I}_\epsilon^1$) (Zitzler et al., 2003). The $\mathrm{I}_H^-$ indicator gives the portion of the objective space that is dominated by $Z_N^\star$ and not by $A$. The reference point is set to the worst objective value on every dimension of the objective space obtained in all approximation sets found during our experiments. The $\mathrm{I}_\epsilon^1$-indicator gives the minimum multiplicative factor by which an approximation $A$ has to be translated in the objective space in order to dominate the reference set $Z_N^\star$. Note that both $\mathrm{I}_H^-$- and $\mathrm{I}_\epsilon^1$-values are to be minimized. The experimental results report the descriptive statistics on the indicator values, together with a Wilcoxon signed-rank statistical test with a $p$-value of 0.05. This procedure has been achieved using R as well as the performance assessment tools provided in PISA (Bleuler et al., 2003; Knowles et al., 2006). Table 1 gives the rank of the different competing algorithms for different configurations. The lower the rank, the better the algorithm.

According to both indicators, for all instances, the hypervolume-based localized scalar strategy $\textsc{Dlbs}_H$ never outperforms the orthogonal-directed one $\textsc{Dlbs}_{OD}$; which indicates that $\mathcal{LF}_{OD}$ is a better localized fitness function to select locally the next solution when compared to $\mathcal{LF}_H$. Although the hypervolume indicator, when used by global algorithms, can outperform algorithms using weighted-sum, the local information induced by the hypervolume at each node in $\textsc{Dlbs}_H$ turns to be less efficient to guide the search process globally compared to orthogonal weighted-sum directions.

When comparing Dlbs to the Piws approach, we can first see that Dlbs performs substantially better with respect to both indicators ($\mathrm{I}_H^-$ and $\mathrm{I}_\epsilon^1$) for all instances. This is obviously attributed to the local information exchanged in our cooperative strategies; which is to contrast with Piws where search directions are fixed statically. In other words, the adaptive search directions used in Dlbs outperform the directions of Piws, that are fixed prior to the search process. This result advocates the usefulness of adaptive search directions, that enable to fit different shapes of the Pareto front.

Comparing the approximation quality of $\textsc{Dlbs}_{OD}$ with Hemo, we find that Dlbs performs better than Hemo for instances with conflicting objectives, *i.e.* when $\rho < 0$. With respect to the hypervolume indicator, Hemo per-

Table 1: Comparison of the different algorithms with respect to the hypervolume difference indicator ($\mathrm{I}_H^-$) and to the multiplicative unary epsilon indicator ($I_\epsilon^1$). The first value stands for the number of algorithms that statistically outperform the one under consideration. The number in brackets stands for the average indicator-value (lower is better). The population size is $\mu = 128$.

| $\rho$ | N | K | DLBS$_{OD}$ ($n = 128, \mu = 128$) | | DLBS$_H$ ($n = 128, \mu = 128$) | | PIWS | | HEMO | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $\mathrm{I}_H^-$ ($\times 10^{-2}$) | | | | |
| −0.7 | 128 | 4 | **0** | (1.846) | **0** | (1.919) | 2 | (2.365) | 3 | (3.667) |
| −0.7 | 128 | 8 | **0** | (1.914) | **0** | (1.984) | 2 | (2.275) | 2 | (2.375) |
| −0.7 | 256 | 4 | **0** | (1.529) | 1 | (1.618) | 2 | (1.779) | 3 | (4.190) |
| −0.7 | 256 | 8 | **0** | (1.580) | 1 | (1.680) | 2 | (1.771) | 3 | (2.906) |
| −0.7 | 512 | 4 | **0** | (0.985) | 1 | (1.107) | 2 | (1.253) | 3 | (3.352) |
| −0.7 | 512 | 8 | **0** | (1.248) | 1 | (1.318) | 2 | (1.461) | 3 | (2.836) |
| 0.0 | 128 | 4 | 1 | (1.778) | 1 | (1.876) | 3 | (2.491) | **0** | (1.406) |
| 0.0 | 128 | 8 | 1 | (1.677) | 2 | (1.821) | 3 | (2.178) | **0** | (1.043) |
| 0.0 | 256 | 4 | **0** | (1.272) | 2 | (1.390) | 3 | (1.613) | **0** | (1.284) |
| 0.0 | 256 | 8 | 1 | (1.219) | 2 | (1.349) | 3 | (1.582) | **0** | (0.667) |
| 0.0 | 512 | 4 | **0** | (1.038) | 1 | (1.115) | 3 | (1.339) | **0** | (1.068) |
| 0.0 | 512 | 8 | 1 | (1.107) | 2 | (1.214) | 3 | (1.379) | **0** | (0.822) |
| +0.7 | 128 | 4 | 1 | (1.518) | 2 | (1.651) | 3 | (2.277) | **0** | (1.255) |
| +0.7 | 128 | 8 | **0** | (0.629) | 2 | (0.743) | 3 | (0.968) | **0** | (0.567) |
| +0.7 | 256 | 4 | 1 | (0.618) | 2 | (0.695) | 3 | (0.804) | **0** | (0.378) |
| +0.7 | 256 | 8 | 1 | (0.526) | 2 | (0.609) | 3 | (0.721) | **0** | (0.329) |
| +0.7 | 512 | 4 | 1 | (0.521) | 2 | (0.571) | 3 | (0.647) | **0** | (0.252) |
| +0.7 | 512 | 8 | 1 | (0.556) | 2 | (0.623) | 3 | (0.673) | **0** | (0.316) |
| | | | | | | $I_\epsilon^1$ | | | | |
| −0.7 | 128 | 4 | **0** | (1.061) | **0** | (1.062) | 2 | (1.076) | 3 | (1.260) |
| −0.7 | 128 | 8 | **0** | (1.063) | **0** | (1.065) | 2 | (1.076) | 3 | (1.165) |
| −0.7 | 256 | 4 | **0** | (1.061) | 1 | (1.065) | **0** | (1.060) | 3 | (1.312) |
| −0.7 | 256 | 8 | **0** | (1.058) | 1 | (1.063) | 1 | (1.064) | 3 | (1.231) |
| −0.7 | 512 | 4 | **0** | (1.040) | 1 | (1.044) | 2 | (1.054) | 3 | (1.260) |
| −0.7 | 512 | 8 | **0** | (1.048) | **0** | (1.048) | 2 | (1.059) | 3 | (1.216) |
| 0.0 | 128 | 4 | **0** | (1.052) | 1 | (1.056) | 3 | (1.080) | **0** | (1.052) |
| 0.0 | 128 | 8 | 1 | (1.058) | 1 | (1.061) | 3 | (1.077) | **0** | (1.039) |
| 0.0 | 256 | 4 | **0** | (1.048) | 1 | (1.052) | 2 | (1.065) | 3 | (1.071) |
| 0.0 | 256 | 8 | 1 | (1.049) | 2 | (1.051) | 3 | (1.070) | **0** | (1.027) |
| 0.0 | 512 | 4 | **0** | (1.045) | **0** | (1.046) | 2 | (1.061) | 2 | (1.063) |
| 0.0 | 512 | 8 | **0** | (1.049) | **0** | (1.049) | 3 | (1.065) | **0** | (1.048) |
| +0.7 | 128 | 4 | **0** | (1.040) | 2 | (1.043) | 3 | (1.065) | **0** | (1.036) |
| +0.7 | 128 | 8 | **0** | (1.038) | 2 | (1.042) | 3 | (1.061) | **0** | (1.036) |
| +0.7 | 256 | 4 | 1 | (1.039) | 2 | (1.041) | 3 | (1.056) | **0** | (1.025) |
| +0.7 | 256 | 8 | 1 | (1.036) | 2 | (1.039) | 3 | (1.055) | **0** | (1.024) |
| +0.7 | 512 | 4 | 1 | (1.037) | 2 | (1.039) | 3 | (1.051) | **0** | (1.018) |
| +0.7 | 512 | 8 | 1 | (1.036) | 2 | (1.041) | 3 | (1.049) | **0** | (1.022) |

forms significantly better than DLBS$_{OD}$ on 9 over the 18 instances, whereas DLBS$_{OD}$ performs better than HEMO on 6 instances. The local information used in DLBS seems to be more valuable when the objectives are in conflict. In this case, the search directions computed locally enable to explore more independent and diverse regions of the objective space. On the contrary, when the objective correlation is positive, there are more interactions between the sub-problems induced by the search directions. Thus, diversification seems to play a less important role, and a global information allowing to take into account the interactions between the population is more useful. At this point of the discussion, we can make two important observations to better understand how the *very local* decisions made by DLBS can be effectively competitive with respect to the *global* step-by-step decisions made by the *sequential* HEMO algorithm.

Firstly, when analyzing in more details the Pareto set approximations achieved by DLBS compared to HEMO, we remark that DLBS is able to find more diversified solutions spanning a wider range of the Pareto front. This is illustrated in Figure 2, showing the empirical attainment function of DLBS *vs.* HEMO for six illustrative instances with different problem dimensions and objective correlations. Note that similar observations can be made with other instances. Empirical attainment functions (López-Ibáñez et al., 2010) provide the probability, estimated from several executions, that an arbitrary objective vector is dominated by, or equivalent to, a solution obtained by a single run of the algorithm. The difference between the empirical attainment functions for two different algorithms enables to identify the regions of the objective space where one algorithm performs better than another. We can see that for the class of instances with conflicting objectives, where the Pareto front is likely to be more stretched in the objective space, the local strategy induced by DLBS is able to find more points at the extreme regions of the Pareto front. In the box-plots of Figure 3, we additionally show the distribution of the achieved hypervolume indicator values for the same set of instances. We can see that the gap between DLBS and HEMO is relatively small for the instances with a high objective correlation. Notice the relatively high tails of the hypervolume indicator distribution obtained with HEMO.

Secondly, the previous discussion holds when the different approaches are experimented using the same fixed number of function evaluations; but with no considerations to execution time. This is one crucial issue in DLBS due to its parallel nature. Actually, it turns out that the running time of HEMO is dramatically worse than DLBS. Computing complexity is an important issue

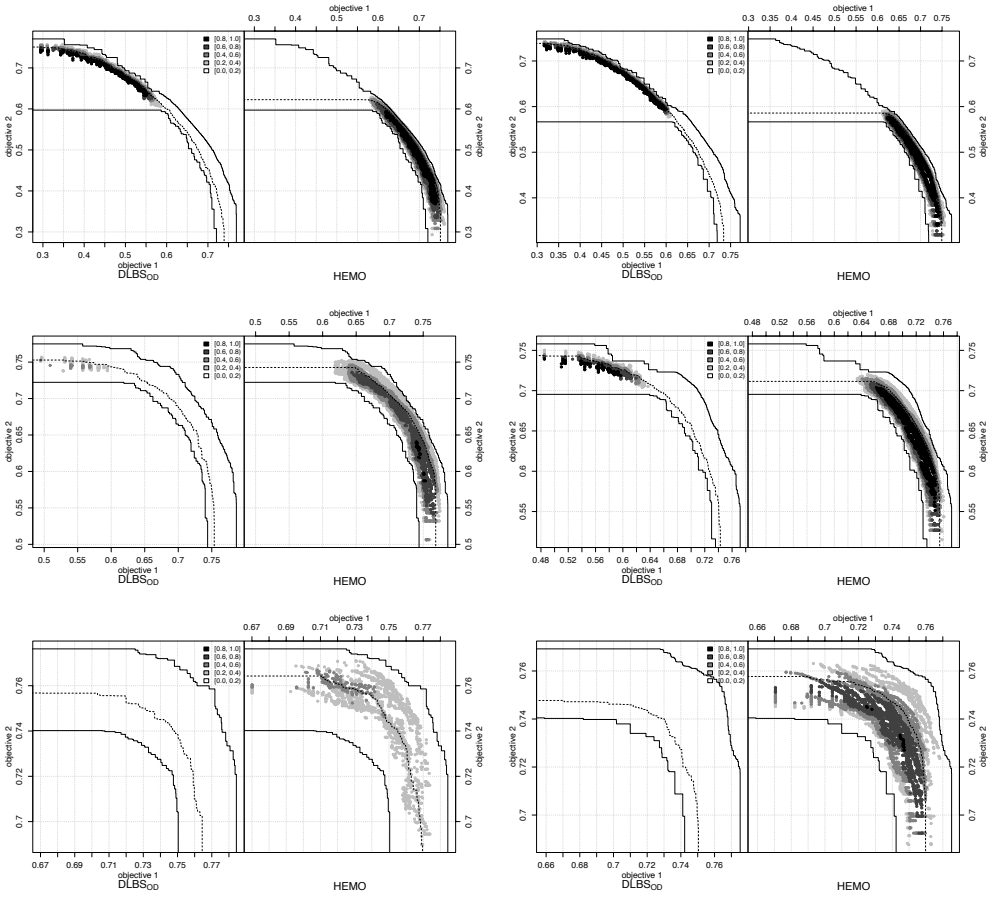


Figure 2: Comparison of DLBS$_{OD}$ and HEMO with respect to the empirical attainment function. The population size is $\mu = 128$. The problem size is $N = 128$ (left) and $N = 256$ (right), the variable interaction is $K = 4$ and the objective correlation is $\rho = -0.7$ (top), $\rho = 0.0$ (middle) and $\rho = +0.7$ (bottom).

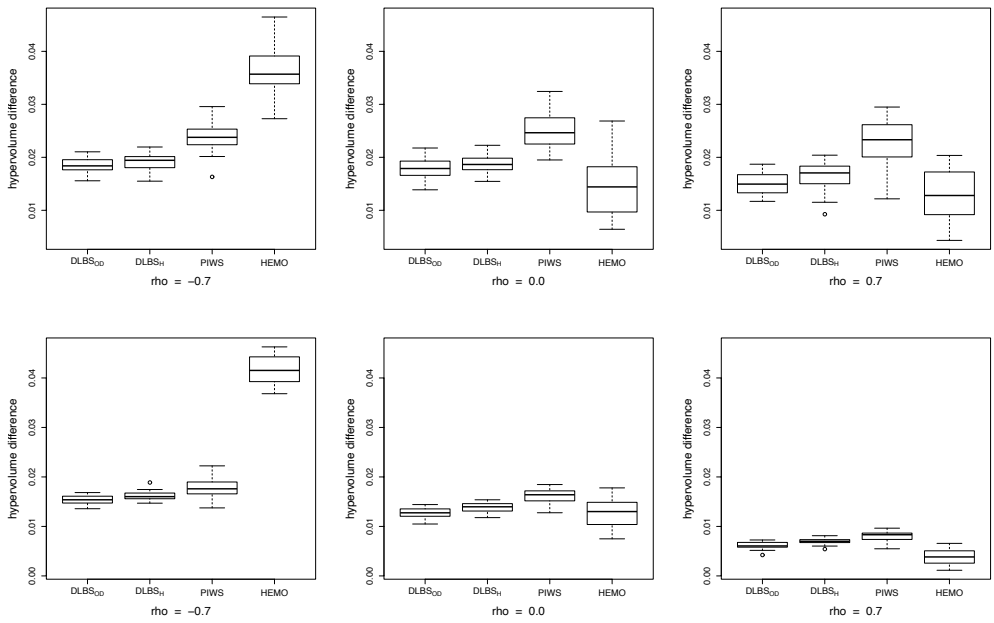


Figure 3: Comparison of DLBS$_{OD}$ and HEMO with respect to the hypervolume difference indicator ($I_H^-$) values (lower is better). The population size is $\mu = 128$. The problem size is $N = 128$ (top) and $N = 256$ (bottom), the variable interaction is $K = 4$ and the objective correlation is $\rho = -0.7$ (left), $\rho = 0.0$ (center) and $\rho = +0.7$ (right).

which is analyzed in more details in the next section.

### *5.2. Running Time*

In Figure 4, we show the relative execution time of our competing algorithms as a function of the population size $\mu$. Since HEMO is inherently a sequential algorithm, we also experiment the 'sequential' variant of DLBS by fixing $n = 1$. This means that DLBS is executed on a single computing node (*i.e.*, no parallelism is involved). Two main observations can be extracted from Figure 4. Depending on the size of the considered instance, the execution time of DLBS is many magnitudes lower than HEMO, even for $n = 1$. This is with no surprise, since in contrast to HEMO, the DLBS approach does not need sophisticated global operation like non-dominated sorting and ranking. In particular, this suggests that, by allowing DLBS to consume slightly more evaluations, inducing a very marginal increase in execution time, the

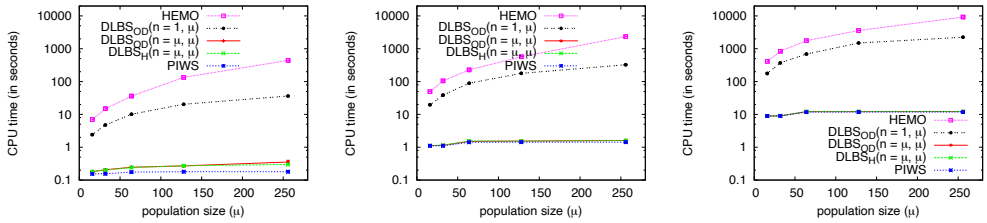


Figure 4: Influence of the population size $\mu \in \{16, 32, 64, 128, 256\}$ on the average CPU time required by the different approaches for $K = 4$ and $\rho = 0.0$ (from left to right $N = 128$, $N = 256$, and $N = 512$). Notice the log-scale.

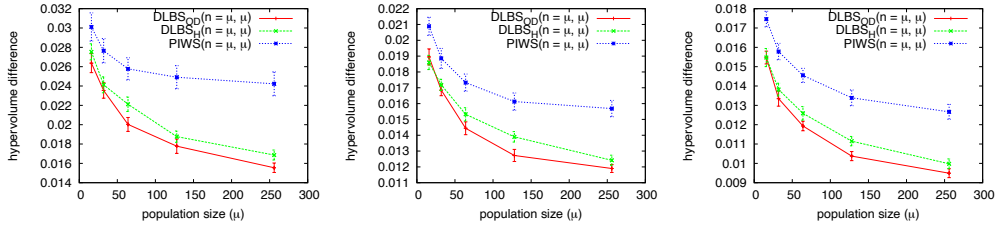


Figure 5: Influence of the population size $\mu \in \{16, 32, 64, 128, 256\}$ on the hypervolume difference indicator ($I_H^-$) values (lower is better) for $K = 4$ and $\rho = 0.0$ (from left to right $N = 128$, $N = 256$, and $N = 512$).

approximation found by DLBS can be substantially improved with respect to HEMO. We also notice that the execution time of DLBS increases very marginally with respect to the population size $\mu$, compared to PIWS. This shows that the local communications and the semi-synchronized nature of DLBS do not have a significant impact on the parallel execution time, even in the fine-grained granularity of $n = \mu$.

In Figure 5, we push the latter discussion further by studying the approximation quality obtained by DLBS$_{OD}$, DLBS$_H$, and PIWS as a function of the population size $\mu$, for three instances of different sizes and in the finest grained scenario of $n = \mu$. We can see that the approximation quality, in terms of hypervolume, increases with the population size. Although the increase in quality slows down with the population size, these results show that DLBS can handle increasing population size while providing better approximation quality and a very small increase in parallel execution time.

*5.3. Parallel Efficiency and Computational Speed-up*

In the previous sections, we were mostly concerned with the analysis of the approximation quality of DLBS and its relation to execution time. However, from a parallel efficiency perspective, DLBS exhibits interesting intrinsic properties that we shall study following three complementary axis: (*i*) the impact of the fitness function evaluation time (Figure 6), (*ii*) the acceleration ratio with respect to a sequential execution (Figure 7), and (*iii*) the speed-up obtained with different granularities (Figure 8).

In Figure 6, we first report the parallel efficiency obtained with DLBS for increasing problem sizes $N \in \{128, 256, 2048\}$ and for $n = \mu = 128$. The reported values refer to the average ratio of computing time over execution time (including communication). This reflects the proportion of time spent by a node in processing the optimization problem and the proportion of time that a node pays when communicating using message exchange. We observe that the parallel efficiency increases sharply from 64% for $N = 128$ to up to 96% for $N = 2048$. This behavior relates directly to the time it takes for a node to evaluate a solution. In fact, as $N$ increases for the $\rho$MNK-landscapes we are considering, the time needed to evaluate a solution increases linearly. In contrast, since only solution coordinates are communicated in DLBS, the amount of information exchanged by two neighboring nodes is independent of the problem size and stays constant. As a consequence, DLBS cannot suffer any performance drop and its parallel efficiency reaches relatively high trade-offs. This interesting property is to contrast with classical parallel evaluation model, see *e.g.* (Talbi et al., 2008), where the whole solution genotypes have to be periodically distributed over computing nodes. Therefore, DLBS can be highly accurate for real-world applications where the fitness evaluation function is usually very time-consuming.

In Figure 7, we show the acceleration ratio obtained when running DLBS for two different problem sizes $N \in \{128, 2048\}$ with respect to the population size $\mu > 1$, and using $n = \mu$ computing nodes, compared to the DLBS version where only a single computing node is used. More specifically, in order to analyze the performance of DLBS in the extreme case of the lowest granularity ($n = \mu$) with respect to the case where *no* parallelism is available at all (highest granularity of $n = 1$), we report the ratio of the execution time of DLBS($n = \mu$,$\mu$) over the execution time of DLBS($n = 1$,$\mu$). Here, it is important to remark that the so-experimented DLBS($\mu$,$\mu$) and DLBS(1,$\mu$) algorithms are exactly the same from a solution quality perspective; only the execution time is different. As one can see in Figure 7, the acceleration

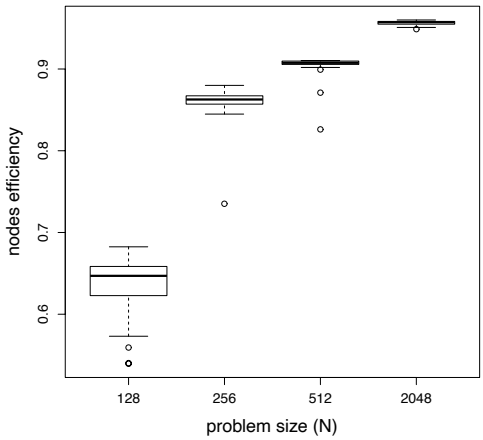


Figure 6: Impact of the fitness function evaluation time: Computing efficiency (average ratio of computing time over execution time) with respect to the problem size $N$. Boxplots are for $\mu = n = 128$, $\lambda = N$, and a total number of $\lambda \times N$ function evaluations *per* computing node (*i.e.* $N$ parallel generations *per* node).

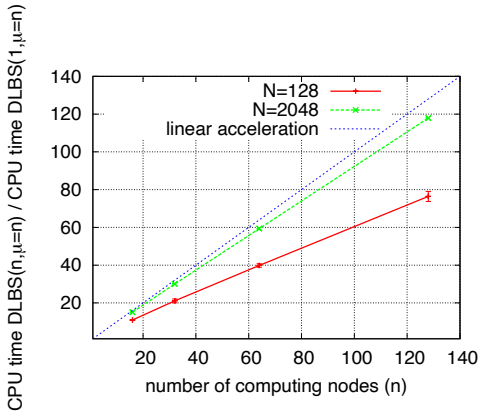


Figure 7: Average acceleration ratio of DLBS($n = \mu$,$\mu$) compared to DLBS(1,$\mu$) with respect to the population size $\mu$. Results are for $N \in \{128, 2048\}$, $\lambda = N$, and a total number of $\lambda \times N$ function evaluations *per* computing node.

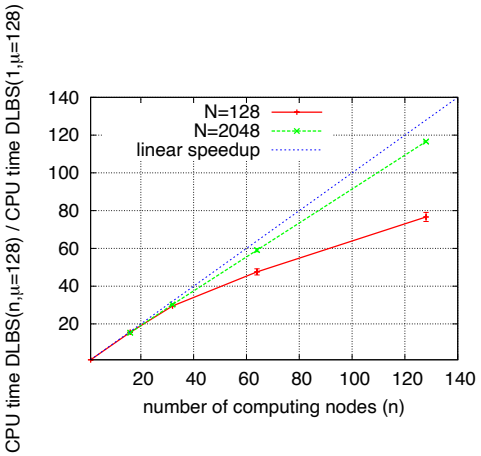


Figure 8: Impact of granularity: Average speedup of DLBS for a fixed population size $\mu = 128$ with respect to the number $n$ of available computing nodes. Results are for $N \in \{128, 2048\}$, $\lambda = N$, and a total number of $\lambda \times N$ function evaluations *per* computing node.

ratio is linear in the population size, independently of the problem size $N$. Interestingly, the slope of the acceleration (0.58 for $N = 128$ and 0.92 for $N = 2048$) roughly corresponds to the parallel efficiency depicted before in Figure 6. From this set of experiments, we can say that the fine-grained parallelization strategy of DLBS (*i.e.*, $n = \mu$) scales efficiently with increasing population sizes. Moreover, the more the problem-dependent fitness function is time consuming, the more DLBS is able the attain high acceleration ratios, which is in accordance with the results of Figure 6, *e.g.*, from 76 up to 118 acceleration using 128 computing nodes.

To study the performance of DLBS with a variable granularity, we conduct a new set of experiments where the population size is now fixed to $\mu = 128$. We then deploy DLBS with a variable number $n$ of computing nodes ranging in $\{8, 16, 32, 64, 128\}$. In this scenario, the $\mu = 128$ solutions are distributed evenly over the $n$ nodes, as discussed previously in Section 3.4. In Figure 8, we report the obtained speed-ups; that is the execution time of the sequential DLBS(1,128) divided by the execution time of the parallel DLBS($n$,128). We can see that the scalability of DLBS depends on the time needed for fitness evaluation which is again in accordance with the results of Figure 6. Overall, we can conclude that for a fixed population size, DLBS is able to scale efficiently with the number of available nodes. A near-linear speedup is obtained for the largest instance, even with the configuration with the highest communication cost ($n = \mu = 128$). In practice, deploying DLBS with a large number of computing nodes (or a large population size) can thus be guaranteed to be very efficient independently of the chosen granularity and without extra-design efforts.

### *5.4. Algorithm Dynamics and Convergence Analysis*

We conclude our analysis by providing some insights into the dynamics and the behavior of our distributed strategy.

In Figure 9, we provide qualitative observations to illustrate how the population is cooperatively evolving closer towards a better Pareto front approximation. For instance in Figure 9 (bottom-left) — showing the average distance of solution objectives to the origin — we see that, as distributed computations are going on, it becomes more and more difficult to push solutions further. This is obviously attributed to the fact that the more solutions are far away from the origin, the more difficult it is for the mutation operator to produce improving solutions. The interesting observation is that,

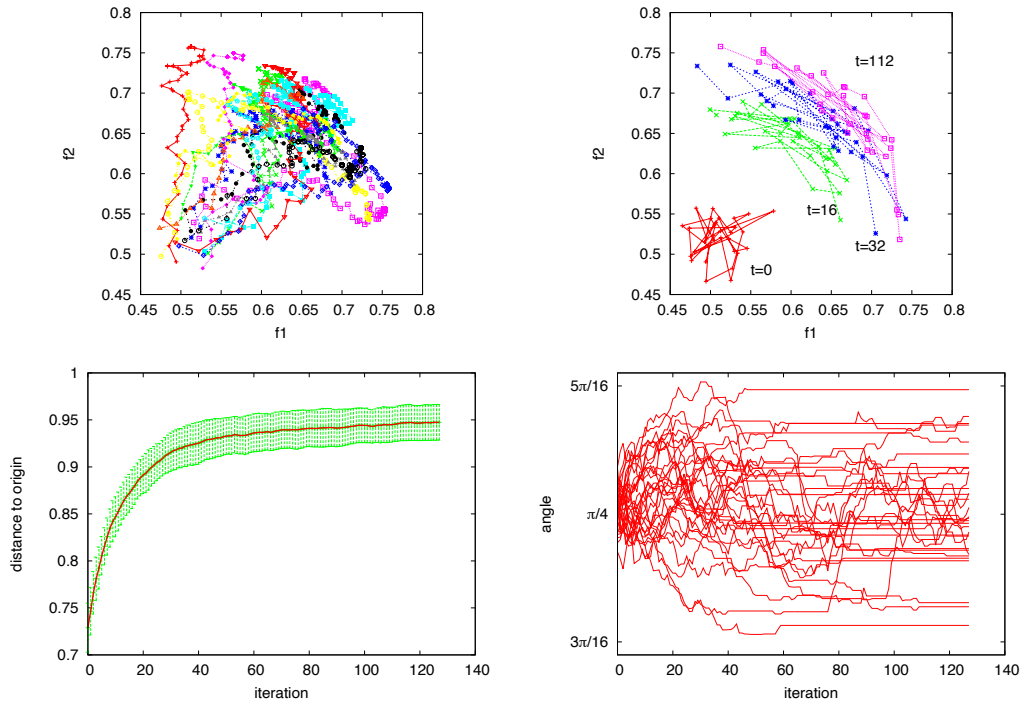


Figure 9: Dynamics of $\text{DLBS}_{OD}(n = 128, \mu = 128)$ for $K = 4$, $\rho = 0.0$, $N = 128$ and $n = \mu = 128$. Top-left: Evolution of the nodes trajectory. Top-right: Evolution of the neighborhood graph. Bottom-left: Evolution of the average distance (and standard deviation) between node positions and the origin in the objective space. Bottom-right: Evolution of node angles (the first objective being the reference). For the sake of clarity, we did not plot all 128 points and restrict ourselves to a comprehensive subset of solutions.

whenever it becomes difficult to get closer to the Pareto front, some solutions start zigzagging right and left in the objective space. As one can see in the trajectories depicted in Figure 9 (top-left), this has the effect of making nodes traveling parallel to the front instead of going straightly towards it. In Figure 9 (top-right), one can further see that the line graph defining the neighborhood in the objective space is not planar, meaning that the line is not automatically disentangled. Actually, this is due to the fact that it is difficult to distributively maintain the solutions sorted. This behavior may also be influenced by the stochastic nature of the mutation operator and to the difficulty of finding dominating solutions as nodes are becoming closer

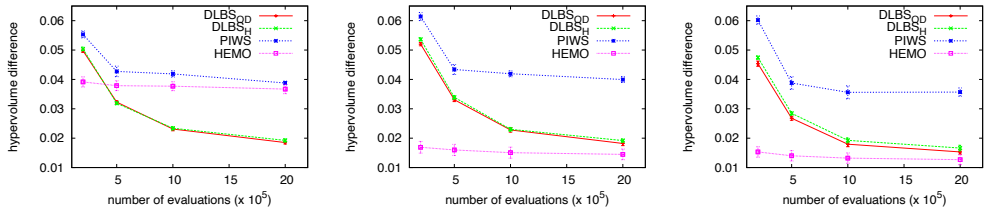


Figure 10: Convergence plot for the hypervolume difference indicator ($\mathrm{I}_H^-$) values (lower is better). The population is $\mu = 128$. The problem size is $N = 128$, the variable interaction is $K = 4$ and the objective correlation is $\rho = -0.7$ (left), $\rho = 0.0$ (center), and $\rho = +0.7$ (right).

to the Pareto front. Notice however the nice distribution of nodes in the objective space. Although the line graph is not planar, Figure 9 (bottom-right) reveals that the distribution of node angles is rather uniform. This means that the distributed strategy succeeds in guiding nodes to different and diverse regions of the objective space. Moreover, one can see that some solutions stay stable in the sense that their angles are not moving, whereas some others are moving smartly around some fixed values.

Figure 10 complements the above observations by reporting the evolution of the hypervolume difference indicator value achieved by DLBS and PIWS during the execution. We can clearly see that, independently of the type of objective correlation, PIWS is quickly stuck with a relatively bad approximation set while DLBS is able to continue improving the hypervolume indicator. This shows that the local decisions made in DLBS do allow solutions to continue evolving dynamically through a better approximation set.

## 6. Conclusion

### *6.1. Discussion*

In this paper, we presented and experimented a new cooperative distributed heuristic search approach for identifying a Pareto set approximation for bi-objective optimization problems. In the proposed approach, the region of the Pareto front where every solution in the population operates is adaptively defined according to other solutions' positions in the objective space. A line graph connecting solutions is assumed, so that this region evolves

during the search process. Two localized fitness functions have been proposed. One is based on a weighted-sum aggregation, while the other consists in improving the hypervolume in-between the neighboring node positions. As a consequence, only a minimum local information is exchanged between solutions.

Our distributed algorithmic scheme has been successfully implemented and experimented throughly using up to 256 computing nodes and $\rho$MNK-landscapes of different structures and sizes. First of all, our experiments confirmed that the algorithm dynamics behave as expected, the localized strategies on each node being able to improve the overall quality of the Pareto set approximation. On the one hand, when compared against a fully independent parallel approach, the information communicated between nodes lead to a very marginal overhead in terms of computational time, whereas clear improvements were shown in terms of approximation quality. On the other hand, even with a very basic single solution-based randomized hill-climbing algorithm performed on every node, competitive results were obtained against a fully centralized sequential evolutionary multi-objective optimization algorithm. We also studied the parallel properties of our scheme by deploying it in different computing configurations and with different granularities. Overall, we find that our approach is highly efficient; in particular by reaching near-linear acceleration and parallel speed-up.

### *6.2. Future Works*

Although our approach can potentially be generalized to more than two objective functions by introducing a multidimensional grid instead of the line graph, and by adapting the localized fitness functions accordingly, it is still an open question to know how it would perform against state-of-the-art parallel and sequential evolutionary multi-objective optimization algorithms for problems with three objective functions and more. Furthermore, extending the experimental analysis conducted in the paper to other multi-objective optimization problems would allow a better understanding of the pros and cons of the proposed algorithmic scheme.

We believe that many other extensions of our approach are also possible and would provide further gains in performance. One of them consists in designing advanced strategies for generating the candidate set and selecting the 'best' performing candidate solution. For instance, one can imagine other localized strategies based on, *e.g.* some localized fitness function variants, some adaptive strategies for selecting a new current solution, or some

recombination operators for generating the mating pool. This can then be plugged within our scheme as far as the decisions are made on the basis of the local information exchanged with neighboring nodes. Notice however that using recombination operators requires to communicate incumbent solutions, and not only node positions. This would result in an increase of communication cost, especially when dealing with heavy solution representations. Another extension would be to design a *collaborative and decentralized* archiving strategy, at each computing node, in order to avoid loosing non-dominated solutions as the distributed search is going on in parallel. In fact, our distributed scheme is oblivious, meaning that we do not remember the set of previously visited solutions. We believe that in case a node is trapped in a local optima or it is dominated, the information learned during the collaborative search process can highly serve for diversification purposes and would enable to reach better regions of the objective space. In the same spirit, our approach does not guarantee that the line graph is fully planar; this is because of the very local view that each solution owns on the population. A possible extension would be to try to sort the solutions from the population at *every* iteration, and to experiment the gain one may obtain by doing so. Deploying this idea is obviously not straightforward without loss in execution time, since nodes would have to communicate more information. At last, compared to our semi-synchronized distributed implementation of DLBS, one may ask whether thinking about a fully asynchronous variant can enable: (*i*) to adapt the local computations according to the power of the possibly heterogeneous parallel computing units, and (*ii*) to distribute the load evenly for those solutions being in relatively more difficult regions of the Pareto front, *e.g.* when some sub-problems require more computing effort.

*Acknowledgements.* The authors would like to gratefully acknowledge the anonymous referees for their valuable feedback that highly contributed to improve the quality of the paper.

## References

Aguirre, H. E., Tanaka, K., 2007. Working principles, behavior, and performance of MOEAs on MNK-landscapes. European Journal of Operational Research 181 (3), 1670–1690.

Aneja, Y. P., Nair, K. P. K., 1979. Bicriteria transportation problem. Management Science 25 (1), 73–78.

Bader, J., Zitzler, E., 2011. HypE: An algorithm for fast hypervolume-based many-objective optimization. Evolutionary Computation 19 (1), 45–76.

Beume, N., Naujoks, B., Emmerich, M., 2007. SMS-EMOA: Multiobjective selection based on dominated hypervolume. European Journal of Operational Research 181 (3), 1653–1669.

Bleuler, S., Laumanns, M., Thiele, L., Zitzler, E., 2003. PISA — A platform and programming language independent interface for search algorithms. In: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003). Vol. 2632 of Lecture Notes in Computer Science. Springer, Faro, Portugal, pp. 494–508.

Branke, J., Schmeck, H., Deb, K., Reddy, M., 2004. Parallelizing multi-objective evolutionary algorithms: Cone separation. In: IEEE Congress on Evolutionary Computation (CEC 2004). Portland, USA, pp. 1952–1957.

Bui, L. T., Abbass, H. A., Essam, D., 2009. Local models — An approach to distributed multi-objective optimization. Computational Optimization and Applications 42 (1), 105–139.

Coello Coello, C. A., Lamont, G. B., Van Veldhuizen, D. A., 2007. Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd Edition. Springer, New York, USA.

Coello Coello, C. A., Sierra, M. R., 2004. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In: Mexican International Conference on Artificial Intelligence (MICAI 2004). Vol. 2972 of Lecture Notes in Computer Science. Springer, Mexico City, pp. 688–697.

Deb, K., 2001. Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester, UK.

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

Deb, K., Zope, P., Jain, A., 2003. Distributed computing of Pareto-optimal solutions with evolutionary algorithms. In: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003). Vol. 2632 of Lecture Notes in Computer Science. Springer, Faro, Portugal, pp. 534–549.

Durillo, J., Zhang, Q., Nebro, A., Alba, E., 2011. Distribution of computational effort in parallel MOEA/D. In: International Conference on Learning and Intelligent Optimization (LION 5). Vol. 6683 of Lecture Notes in Computer Science. Springer, Rome, Italy, pp. 488–502.

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

Ehrgott, M., 2005. Multicriteria optimization, 2nd Edition. Springer, Berlin, Germany.

Figueira, J. R., Liefooghe, A., Talbi, E.-G., Wierzbicki, A. P., 2010. A parallel multiple reference point approach for multi-objective optimization. European Journal of Operational Research 205 (2), 390–400.

Hiroyasu, T., Yoshii, K., Miki, M., 2007. Discussion of parallel model of multi-objective genetic algorithms on heterogeneous computational resources. In: Genetic and Evolutionary Computation Conference (GECCO 2007). ACM, London, UK, pp. 904–904.

Humeau, J., Liefooghe, A., Talbi, E.-G., Verel, S., 2013. ParadisEO-MO: From fitness landscape analysis to efficient local search algorithms. Journal of Heuristics 19 (6), 881–915.

Jozefowiez, N., Semet, F., Talbi, E.-G., 2002. Parallel and hybrid models for multi-objective optimization: Application to the vehicle routing problem. In: International Conference on Parallel Problem Solving from Nature (PPSN VII). Vol. 2439 of Lecture Notes in Computer Science. Springer, Granada, Spain, pp. 271–280.

Kauffman, S. A., 1993. The Origins of Order. Oxford University Press, New York, USA.

Knowles, J., Thiele, L., Zitzler, E., 2006. A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland.

Liefooghe, A., Jourdan, L., Talbi, E.-G., 2011. A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO. European Journal of Operational Research 209 (2), 104–112.

López-Ibáñez, M., Paquete, L., Stützle, T., 2010. Exploratory analysis of stochastic local search algorithms in biobjective optimization. In: Experimental Methods for the Analysis of Optimization Algorithms. Springer, Ch. 9, pp. 209–222.

Melab, N., Talbi, E.-G., Cahon, S., 2006. On parallel evolutionary algorithms on the computational grid. In: Parallel Evolutionary Computations. Vol. 22 of Studies in Computational Intelligence. Springer, pp. 117–132.

Mostaghim, S., 2010. Parallel multi-objective optimization using self-organized heterogeneous resources. In: Parallel and Distributed Computational Intelligence. Vol. 269 of Studies in Computational Intelligence. Springer, pp. 165–179.

Mostaghim, S., Branke, J., Schmeck, H., 2007. Multi-objective particle swarm optimization on computer grids. In: Genetic and Evolutionary Computation Conference (GECCO 2007). ACM, London, UK, pp. 869–875.

Nebro, A. J., Durillo, J. J., 2010. A study of the parallelization of the multi-objective metaheuristic MOEA/D. In: International Conference on Learning and Intelligent Optimization (LION 4). Vol. 6073 of Lecture Notes in Computer Science. Springer, Venice, Italy, pp. 303–317.

Peleg, D., 2000. Distributed computing: A locality-sensitive approach. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Qi, Y., Ma, X., Liu, F., Jiao, L., Sun, J., Wu, J., 2014. MOEA/D with adaptive weight adjustment. Evolutionary Computation 22 (2), 231–264.

Streichert, F., Ulmer, H., Zell, A., 2005. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005). Vol. 3410 of Lecture Notes in Computer Science. Springer, Guanajuato, Mexico, pp. 92–107.

Talbi, E.-G., Mostaghim, S., Okabe, T., Ishibuchi, H., Rudolph, G., Coello Coello, C. A., 2008. Parallel approaches for multiobjective optimization. In: Multiobjective Optimization – Interactive and Evolutionary Approaches. Vol. 5252 of Lecture Notes in Computer Science. Springer, pp. 349–372.

Tan, K., Yang, Y. J., Goh, C., 2006. A distributed cooperative coevolutionary algorithm for multiobjective optimization. IEEE Transactions on Evolutionary Computation 10 (5), 527–549.

Tomassini, M., 2005. Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time. Natural Computing Series. Springer, Berlin, Germany.

Van Veldhuizen, D. A., Zydallis, J. B., Lamont, G. B., 2003. Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation 7 (2), 144–173.

Verel, S., Liefooghe, A., Jourdan, L., Dhaenens, C., 2013. On the structure of multiobjective combinatorial search space: MNK-landscapes with correlated objectives. European Journal of Operational Research 227 (2), 331–342.

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

Zhu, Z.-Y., Leung, K.-S., 2002. Asynchronous self-adjustable island genetic algorithm for multi-objective optimization problems. In: IEEE World on Congress on Computational Intelligence (WCCI 2002). Honolulu, USA, pp. 837–842.

Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation 3 (4), 257–271.

Zitzler, E., Thiele, L., Laumanns, M., Foneseca, C. M., Grunert da Fonseca, V., 2003. Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation 7 (2), 117–132.