Parallel Pareto local search revisited – First experimental results on bi-objective UBQP

Jialong Shi, Qingfu Zhang, Bilel Derbel, Arnaud Liefooghe, Jianyong Sun

Parallel Pareto Local Search Revisited

First experimental results on Bi-objective UBQP

Jialong Shi
School of Mathematics and Statistics,
Xi'an Jiaotong University
Xi'an, China
shi.jl@outlook.com

Qingfu Zhang
City University of Hong Kong
Hong Kong
The City University of Hong Kong
Shenzhen Research Institute
Shenzhen, China
qingfu.zhang@cityu.edu.hk

Bilel Derbel
Univ. Lille
Inria Lille – Nord Europe
Lille, France
bilel.derbel@univ-lille1.fr

Arnaud Liefooghe
Univ. Lille
Inria Lille – Nord Europe
Lille, France
arnaud.liefooghe@univ-lille1.fr

Jianyong Sun
School of Mathematics and Statistics,
Xi'an Jiaotong University
Xi'an, China
jy.sun@xjtu.edu.cn

ABSTRACT

Pareto Local Search (PLS) is a simple, yet effective optimization approach dedicated to multi-objective combinatorial optimization. It can however suffer from a high computational cost, especially when the size of the Pareto optimal set is relatively large. Recently, incorporating decomposition in PLS had revealed a high potential, not only in providing high-quality approximation sets, but also in speeding-up the search process. Using the bi-objective Unconstrained Binary Quadratic Programming (bUBQP) problem as an illustrative benchmark, we demonstrate some shortcomings in the resulting decomposition-guided Parallel Pareto Local Search (PPLS), and we propose to revisit the PPLS design accordingly. For instances with a priori unknown Pareto front shape, we show that a simple pre-processing technique to estimate the scale of the Pareto front can help PPLS to better balance the workload. Furthermore, we propose a simple technique to deal with the critically-important scalability issue raised by PPLS when deployed over a large number of computing nodes. Our investigations show that the revisited version of PPLS provides a consistent performance, suggesting that decomposition-guided PPLS can be further generalized in order to improve both parallel efficiency and approximation quality.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; **Random search heuristics**; • **Computing methodologies → Optimization algorithms**; **Discrete space search**; *Massively parallel algorithms*; • **Applied computing → Multi-criterion optimization and decision-making**;

KEYWORDS

Pareto Local Search, Parallel Computation, Combinatorial Optimization

ACM Reference Format:
Jialong Shi, Qingfu Zhang, Bilel Derbel, Arnaud Liefooghe, and Jianyong Sun. 2018. Parallel Pareto Local Search Revisited: First experimental results on Bi-objective UBQP. In *GECCO '18: Genetic and Evolutionary Computation Conference, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205455.3205577

1 INTRODUCTION

Solving a multi-objective optimization problem (MOP) requires to compute a set of solutions representing the possible compromises with respect to the underlying objectives. Identifying such a set, referred to as the *Pareto* optimal set, is a difficult task. The goal is many-often to find a high-quality Pareto set *approximation*, which provides a good balance among conflicting objectives. Evolutionary algorithms and other search heuristics have been proved to be extremely effective. Despite their skillful design, existing multi-objective algorithms can imply computing intensive operations. For instance, it is well known that for large scale MOPs, maintaining a relatively large solution set can be mandatory. The computational time required to run a multi-objective algorithm can also constitute a bottleneck in order to reach a good approximation set. Therefore, incorporating parallelism and distributed computations have attracted much attention from the community, not only to speed-up the search, but also to design novel high-level parallel approaches, providing improved approximations. In this context, the work presented in this paper is at the crossroad of three aspects dealing with the design of effective and efficient multi-objective algorithms, namely *Pareto local search*, *decomposition* and *parallel computing*, as discussed in the following paragraphs.

Pareto Local Search. On the one hand, we are interested in tackling *combinatorial* MOPs for which Local Search (LS) is a key building-block in many combinatorial optimization algorithms. In a single-objective setting, LS is a single solution-based walk in the solution space X, that iteratively improves the current solution by

Algorithm 1: A basic Pareto Local Search (PLS).

```
1 A ← initial set of non-dominated solutions;
2 while not all solutions in A are visited do
      /* Selection                                   */
3     Select a non-visited solution x ∈ A;
      /* Solution neighborhood exploration           */
4     Explore N(x) and mark x as visited;
      /* Acceptance criteria for archiving           */
5     Update A with non-dominated neighbors;
6 return A
```

means of local improving transformations. Those transformations are usually based on a *neighborhood* relation $\mathcal{N} : X \rightarrow 2^X$, which assigns a set of neighboring solutions $\mathcal{N}(x) \subset X$ to any solution $x \in X$. When moving to a multi-objective setting, LS can be extended to deal with a whole *set* of solutions instead of just a single one. This simple idea is the cornerstone of the well-established Pareto Local Search (PLS) algorithm [8], for which a basic variant is depicted in Algorithm 1 for the clarity of the discussion.

Starting with an initial set of non-dominated solutions, PLS maintains an archive of non-dominated solutions. At each iteration, a non-visited solution from the archive is selected, its neighborhood is explored and the archive is updated. PLS can then be viewed as a local search operating at a *set* level and stopping naturally after reaching a Pareto local optimum set [8]. Although the basic PLS illustrated in Algorithm 1 enables to obtain high-quality approximation sets, it is well known that its convergence speed is low and several strategies have been proposed in order to overcome this issue [1–4]. Actually, the three key components of PLS, namely *selection*, *neighborhood exploration* and *acceptance criteria*, and the way they are instantiated when tackling a given problem, were shown to be crucially-important for the anytime performance of PLS [2, 4]. Besides, it is well-understood that the size of the set maintained by PLS can be required to have a relatively big size, which implies further computational issues.

Decomposition and Parallelism. On the other hand, in some recent papers [10, 11], it is shown how to incorporate decomposition into PLS with the aim of bringing parallelism into the scene to enhance search performance. Inspired by the MOEA/D algorithm [12], Shi et al. [10, 11] proposed to use different scalar functions to guide a number of *independent* PLS processes in parallel towards different regions of the objective space. The archives obtained by different processes are merged together into one output archive at the end of the run. In order to guarantee that the sub-archives at every PLS process do not overlap, while avoiding any distributed communications, the objective space is further partitioned into different search regions by defining boundaries. These regions are mapped to the parallel PLS processes, and the corresponding weight vectors are used to guide selection. It is, in particular, shown that this high-level parallelizing approach allows one to speed-up very substantially the computational flow, while maintaining seemingly the same approximation quality than the original PLS. To the best of our knowledge, the so-obtained Parallel Pareto Local Search (PPLS) [10, 11] can in fact be considered as a state-of-the-art high-level parallel design of basic PLS. However, we argue that the mechanism of PPLS still requires further improvement. This is precisely one of the main motivations of this paper.

Contributions. In this paper, we identify two major issues in the initial design of PPLS [10] for which we propose two simple, yet effective, solutions. Firstly, the objective space partition used in PPLS is found to introduce some load-imbalance with respect to the computational efforts underlying every parallel PLS process. This is basically because the mapping of the Pareto set into the objective space, i.e., the Pareto front (PF), may span a region of variable and unknown width. Accordingly, we propose to estimate the extreme points of the PF prior to running the independent PLS processes. This allows us to better adjust the boundary of the objective space to be decomposed over the PLS processes. More importantly, when increasing the number of available computing nodes, PPLS is found to suffer from a scalability issue, both in terms of running time and approximation quality. This is because constraining the search to multiple narrow regions in the objective space makes it more likely for a single independent PLS to be trapped into local optima. Considering that scalability is one of the most desirable properties of parallel algorithms, dealing with such an issue is critically important when it comes to deploy PPLS in a massively parallel environment. Accordingly, and instead of a stringent partition of the objective space, we manage to use overlapping search regions when running the independent parallel PLS processes. This is handled by simply opening the initial boundary vectors used for decomposition by a pre-defined angle in the objective space.

Our experimental study on a number of standard instances of the well-established bi-objective Unconstrained Binary Quadratic Programming (bUBQP) problem provides evidence on the accuracy the proposed PPLS variant to deal with the before-mentioned issues. In this respect, our investigations are to be considered as an attempt to push a step forward the design of scalable and efficient parallel multi-objective optimization algorithms in large-scale distributed environments and for large-scale instances.

Outline. The rest of this paper is organized as follow. For the paper to be self-contained, we recall in Section 2 the initial design of PPLS as described in [10, 11]. In Section 3, we discuss the first issue of PPLS related to load-imbalance when dealing with an unknown Pareto front shape. In Section 4, we discuss the the second issue of PPLS related to the scalability of PPLS. In Section 5, we conclude the paper and highlight some perspectives.

2 PARALLEL PARETO LOCAL SEARCH

In this section we review the design of PPLS [10, 11].

2.1 Definitions

A *multi-objective optimization problem* 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 following , X is a discrete set since our focus is on combinatorial optimization problems. 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 $f : X \rightarrow Z$. In a maximization context, an objective vector $z \in Z$ is dominated by an objective vector $z' \in Z$ (i.e., $z \prec z'$) iff $\forall m \in \{1, 2, \ldots, M\}$,

$z_m \leqslant z'_m$ and $\exists m \in \{1, 2, \ldots, M\}$ such that $z_m < z'_m$. A solution $x \in X$ is dominated by a solution $x' \in X$ (i.e., $x \prec x'$), iff $f(x) \prec f(x')$. Given a set $A \subset X$ and a solution $x \in X$, x is dominated with respect to A (i.e., $x \prec A$), if x is dominated by at least one solution in A. A solution $x^\star \in X$ is termed *Pareto optimal*, iff 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* (PS). Its mapping in the objective space is the *Pareto front* (PF). We also use a scalarizing function allowing to transform the original multi-objective problem a number of *scalarized* single-objective *sub-problems* based on pre-defined *weight vectors*. Different scalarizing functions have been proposed so far in the literature [7]. In this paper, we use the weighted sum (g^{ws}) function, to be maximized:

$$g^{ws}(x, \lambda) = \sum_{i=1}^{M} \lambda_i \cdot f_i(x),$$

where $\lambda = (\lambda_1, \ldots, \lambda_M)$ is a weight vector among the objectives. As a benchmark, we consider the well-established multi-objective Unconstrained Binary Quadratic Programming (mUBQP) problem [5]:

$$\text{maximize}: \; f_k(x) = x^T \, Q_k \, x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij}^k \, x_i \, x_j$$

where $Q_k = [q_{ij}^k]$ is a $n \times n$ matrix for the k^{th} objective function f_k, and x is a vector of n binary $(0-1)$ variables (i.e., $X = \{0,1\}^n$). We focus on the bi-objective UBQP, i.e. $M = 2$, for which PLS is recognized as a state-of-the-art algorithm [6]. Similar to [6, 10, 11], the neighborhood structure $\mathcal{N}$ used in PLS is the 1-bit-flip.

2.2 An Overview of PPLS

PPLS was designed using different alternative selection, neighborhood exploration and acceptance strategies [10]. Without loss of generality, we focus on one specific PPLS variant as depicted in the template of Algorithm 2.

First, the objective space is decomposed evenly into several small regions. In the bi-objective case, and as illustrated in Figure 1, this consists in delimiting the boundaries of contiguous regions using a reference point and a set of two consecutive lines (vectors) passing through the reference point. Then, several *independent* PLS processes are executed *in parallel*, each one operating in exactly one of the so-defined regions. This is encoded in the template of Algorithm 2 by initially providing PPLS with an input set of N constraints Ω. Every process p^i is hence running one independent PLS where a solution cannot be accepted to enter the archive set A_i if it is outside the search regions defined by constraints Ω^i. More precisely, when a process p_i finds a solution whose objective vector maps outside the boundaries of its region Ω^i, it simply ignores it. Nevertheless, this is with the exception of the situation where all solutions in the local archive A_i are outside the search region constraints Ω^i (see Line 10 and Line 19 in Algorithm 2). This exception prevents PPLS from stopping early when PPLS starts from an initial archive A_{init} that is outside Ω^i.

In order to guide the independent parallel processes within their pre-defined regions, basic PLS is also redesigned using a weight vector (see Figure 1), corresponding to the region where a parallel process is expected to operate. A weighted sum is then used to rank the explored solutions and to update every local archive

Algorithm 2: Parallel Pareto Local Search (PPLS) [10, 11]

Input: $P := \{p^1, \ldots, p^N\}$: N parallel processes;
$\Lambda := \{\lambda^1, \ldots, \lambda^N\}$: N target weight vectors;
$\Omega := \{\Omega^1, \ldots, \Omega^N\}$: N constrained search regions;
$g(\cdot, \lambda)$: a scalarizing function to be maximized ;

1 $A_{\text{init}} \leftarrow$ initial set of non-dominated solutions;
2 **For every process** p^i, $i \in \{1, \ldots, N\}$
do independently in parallel:
3 $A_i \leftarrow A_{\text{init}}$;
4 $\forall x \in A_i$, marked as non-visited;
5 $g_i^* \leftarrow \max\{g(y, \lambda^i) \mid y \in A_i\}$;
6 **while** $\exists x \in A_i$ *s.t., x is non-visited* **do**
7 $x \leftarrow \arg\max_{y \in A_i} \{g(y, \lambda^i) \mid y \text{ is non-visited}\}$;
8 SuccessFlag $\leftarrow false$;
9 **for** $x' \in \mathcal{N}(x)$ **do:**
10 **if** $\Omega^i(x') = true$ ***or*** $\forall y \in A_i, \Omega^i(y) = false$ **then**
11 **if** $g(x', \lambda^i) > g_i^*$ **then**
12 $g_i^* \leftarrow g(x', \lambda^i)$;
13 $x' \leftarrow$ (marked as) non-visited;
14 $A_i \leftarrow$ non-dominated-sol$(A_i \cup \{x'\})$;
15 SuccessFlag $\leftarrow true$;
16 **break**;
17 **if** *SuccessFlag* $= false$ **then**
18 **for** $x' \in \mathcal{N}(x)$ **do:**
19 **if** $\Omega^i(x') = true$ ***or*** $\forall y \in A_i, \Omega^i(y) = false$ **then**
20 **if** $x' \not\prec A_i$ **then**
21 $x' \leftarrow$ (marked as) non-visited;
22 $A_i \leftarrow$ non-dominated-sol$(A_i \cup \{x'\})$;
23 $x \leftarrow$ (marked as) visited;
24 **for** $x \in A_i$ **do:** $x \leftarrow$ (marked as) non-visited;
// Re-check the entire neighborhood of every solution in A_i
25 **Redo** the while loop in Line 6, but skip Line 16;
26 $A \leftarrow$ non-dominated-sol$(\cup_{i=1}^{N} A_i)$;
27 **return** A

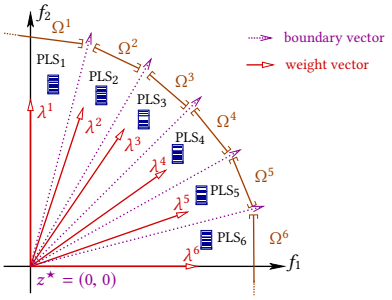
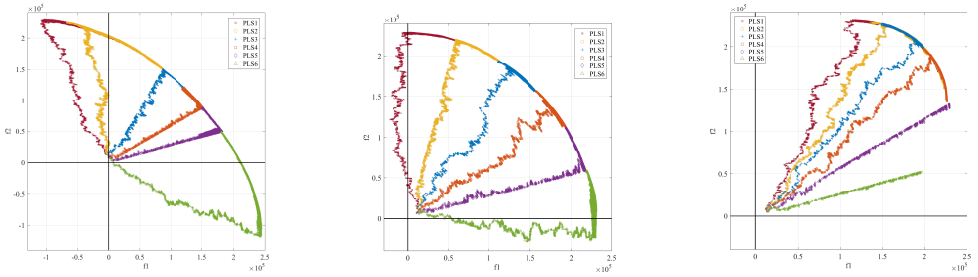


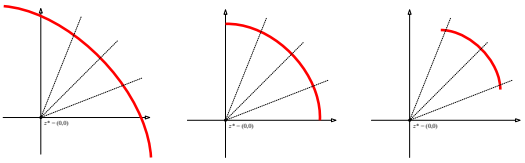
Figure 1: Illustration of search region (i.e. Ω^i) decomposition in PPLS. Six PLS are executed independently in parallel, each one using one weight vector λ^i.

A_i. This is to contrast with basic PLS (see Algorithm 1) which is solely based on the dominance relation. In fact, at the selection step, a basic PLS typically selects a non-visited solution at random



(a) $\rho = -0.5$ (b) $\rho = 0$ (c) $\rho = 0.5$

Figure 2: Visualization of PPLS trajectory on the bUBQP instances with $n = 500$ and different objective correlation ρ.



(a) $\rho \rightarrow -1$ (b) $\rho \approx 0$ (c) $\rho \rightarrow 1$

Figure 3: Different ρ values lead to different scale of the PF

from the archive in order to explore its neighborhood, while in PPLS the non-visited solution that has the highest weight-sum scalarizing function value is selected (Line 7). Moreover, PPLS stops the neighborhood exploration and marks the current solution as visited immediately when it finds a neighboring solution that has an even higher scalarized function value than the highest value from the archive (Lines 11–16). As for the acceptance criteria, the basic PLS typically accepts all the neighboring solutions that are non-dominated by any solution from the archive, while PPLS first only accepts the neighboring solution that has an even higher scalarized function value, then if no such neighboring solution can be found, PPLS switches to accepting solutions that are non-dominated (Line 17–22). Finally, after all solutions in A_i have been visited (end of the while loop), all solutions are marked as non-visited and their neighborhoods are explored again to make sure that the entire neighborhood of every solution in A_i is explored. (Lines 24–25). Notice that although a weighted-sum scalarizing function is used, which at a first sight might seem solely appropriate to reach supported solutions, using the dominance relation still allows PPLS to eventually find non-supported ones.

The previous steps are applied independently by each parallel process. Only after all processes have terminated, the global archive is returned by merging the local archives found by the different processes. Notice that this specific combination of selection, neighborhood exploration and acceptance in PPLS aims at (i) coordinating the parallel processes without requiring any communication, and (ii) reducing drastically the size of the local archive A_i maintained by every parallel process all along the different iterations.

3 ADDRESSING PPLS LOAD IMBALANCE

The first design issue in PPLS is related to the choice of the reference point, used for the definition of the decomposed regions, which can lead to unbalanced parallel efforts as discussed in the following.

Impact of the Reference Point. [10] proposed a diagram called *trajectory tree* to show the search history of a PLS process. Specifically, the trajectory tree shows all the solutions ever accepted by the archive during the entire PLS process. In Figure 2, we show the trajectory trees of three exemplary PPLS runs (with 6 computing nodes) corresponding to three standard bUBQP instances taken from [5]. Using such instances, one can explicitly consider an important feature of multi-objective optimization problems which is the correlation between the objectives. The objective correlation of bUBQP instances described in [5] is controlled by means of a real-valued parameter $\rho \in [-1, 1]$. When $\rho = 0$, the objective values are meant to be uncorrelated. However, as ρ is set to a negative (resp. positive) value, the objective values tend to be more conflicting (resp. correlated). This has an impact on the scale of the PF, which could then span a region of variable size, i.e., ranging from very small (when $\rho \rightarrow 1$) to very large (when $\rho \rightarrow -1$), see Figure 3.

Following the initial design of PPLS as described originally in [10], the sub-regions given as input to our PPLS exemplary runs are obtained by: (i) setting a *pre-fixed* reference point, and (ii) using a uniform distribution of boundary lines passing through this point and decomposing the dominated region of the objective space into $N = 6$ parts (as depicted previously in Figure 1). The critical issue comes precisely from the setting of the reference point value. The chosen value, namely $(0, 0)$, might be a good choice when the objectives are independent and when the PF lies around the boundaries of the two extreme regions (see Figure 2(b) and Figure 3(b)). However, since one cannot have such a strong guarantee in general, this choice of the reference point is problematic. In fact, as can be seen in Figure 2(a) and Figure 3(a), when the objective correlation $\rho \rightarrow -1$, the parallel processes mapping the edges of the objective space will have relatively large PF fragments to approximate, e.g., the processes PLS_1 and PLS_6 in Figure 2(a). This also happens to process PLS_2 in Figure 2(a), because when it reaches the PF, it is far from its pre-defined search region Ω^2. Hence, it has to approximate a large part of the PF until it arrives to its own search region. In fact, remember that when the solutions in the local archive are outside the boundaries, then a PLS process is allowed to search outside its region boundaries. When the objective correlation $\rho \rightarrow 1$, it should also be clear that processes in the center will have relatively large PF fragments to approximate, as can be seen in Figure 2(c) and Figure 3(c). In Figure 2(c), the search regions of PLS_1, PLS_2, PLS_5 and PLS_6 do not contain (or only contain a small part of) the final

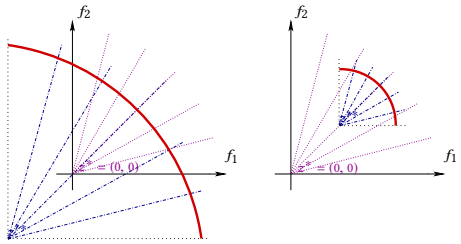


Figure 4: Illustration of the difference between decomposed sub-regions using different reference points.

PF. Hence, if at the beginning the processes do not find solutions in their own search regions, their search will not stop until they have approximated the entire PF. This is exactly what happens for PLS_1 and PLS_2 in Figure 2(c). On the other side, if at the beginning the processes do find solutions inside their regions, they stop very early, e.g. PLS_5 and PLS_6 in Figure 2(c).

Estimating the PF Boundaries. The aforementioned issues can obviously cause a significant difference of the workload among different parallel processes. In the following, we propose a simple technique in order to estimate the boundaries of the PF which then should allow to better define the decomposed sub-regions. We proceed as follows. First, we conduct two single-objective LS processes starting with a randomly-generated solution. The first LS optimizes the first objective f_1 while the second LS deals with the second objective f_2. Let us assume that the two LS processes end with respectively two solutions $x_{LS,1}$ and $x_{LS,2}$. We then use the point $z^* = (f_1(x_{LS,2}), f_2(x_{LS,1}))$ as a reference point to perform objective space decomposition. In other words, we define the boundary lines as traversing the so-obtained point. In Figure 4, we show an illustration on the ideal expected shape of the so-defined sub-regions compared to an ad-hoc setting of the reference point.

Experimental Validation. Compared to the original PPLS [10], two extra LS processes must be executed before the main PPLS could start. This can hence introduce extra-computing time. However, one should keep in mind that the penalty one can pay when missing the true boundaries of the PF in original PPLS could even be more severe. To validate such a claim, we experiment PPLS with and without the so-defined reference point. We consider a range of bi-objective UBQP instances taken from [5], by varying the objective correlation $\rho \in \{0.75, 0.5, 0.25, 0, -0.25, -0.5, -0.75\}$ and the problem size $n \in \{200, 300, 500\}$. A more diverse set of instances possibly coming from other domains is left for future investigations. A PPLS using 6 CPU cores is considered. Notice that a more fine-grained setting raises further issues that we shall consider separately later in the paper. PPLS is then run until its natural stopping condition, i.e., all solutions in the local archives are visited. On each instance/configuration, 20 runs are executed. We evaluate the performance of PPLS as follows. First, we measure the Hypervolume [13] of the output archive to appreciate the approximation set quality. The reference point needed to compute the Hypervolume is set as follow. Let $f_{\max}$ and $f_{\min}$ respectively the objective vectors formed by the maximum and minimum objective values we ever found during all algorithm executions. Then, the reference point used for the Hypervolume computation is set to $f_{\min} - 0.1 \cdot (f_{\max} - f_{\min})$. Second, we measure the overall runtime (including pre-processing the boundaries) required by PPLS to terminate. All algorithms are implemented in GNU C++ and executed on a parallel environment using 2.00GHz 6-core Intel Xeon CPUs. Our experimental results are summarized in Table 1, where we additionally report the ratios of the hypervolume-values and CPU times of the two experimented PPLS versions. The quality ratio is defined as the hypervolume of the PPLS configuration described in this paper over the one used originally in [10]. The acceleration ratio is defined as the CPU time of the original PPLS configuration [10] over our modification.

Results. From Table 1, we can see that, on 14 out of 21 instances, the proposed method is not significantly outperformed by the original method in terms of hypervolume, whereas on 18 out of 21 instances the runtime of the new method is significantly shorter than that of the original method. As indicated by the quality ratio, the hypervolume achieved by the revised PPLS is around 99% of the original one. In contrast, the gap in running time is relatively large as indicated by the acceleration ratio, i.e., in average over all instances, and independently of the objective correlation, the revised PPLS is 4 times faster. We also notice that the largest gaps are obtained for large-size instances with a high objective correlation. This is consistent with our previous discussion arguing that, for such instances, the PLS process at the extremes of the objective space might either (i) stop prematurely before reaching some parts of the PF, or (ii) spend a large amount of time traversing the whole PF since no solutions in their specified sub-regions can be found at the early stages of the search process. This can also explain the slight differences observed for the hypervolume. Finally, we illustrate in Figure 5 the trajectory trees of three exemplary runs of the modified PPLS configuration using the same instances than previously in Figure 2, which provides more visual evidence on the accuracy of the modified search region decomposition.

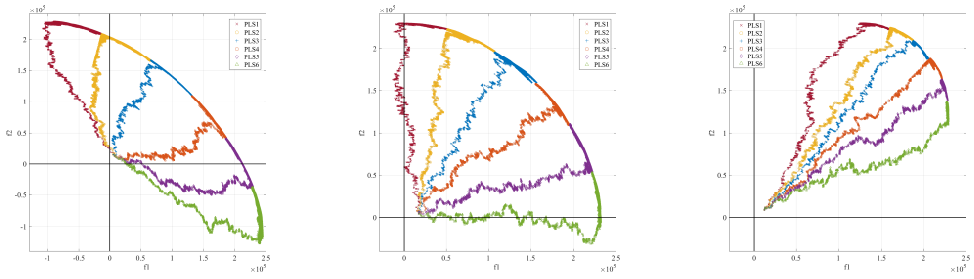
4 ADDRESSING PPLS SCALABILITY

The main desirable feature of PPLS is the ability of speeding-up the search through parallelism while still providing a high-quality approximation. Maintaining such an ability while achieving scalability, with respect to an increasing number of computing resources, constitutes the second issue in the initial design of PPLS.

Scalability of PPLS. Obviously, as the number of processes increases, the decomposed search sub-region of each parallel PLS process becomes narrower. On the one hand, as soon as some solutions satisfying the boundary conditions is archived by a PPLS sub-process, most solutions generated subsequently using the neighborhood structure are likely to be outside the (narrow) boundaries of the corresponding sub-region, even if they are non-dominated. Consequently, this can force the local search to stop prematurely and make the independent process more easily stuck. Hence, the quality of the output approximation set can significantly drop when scaling the number of parallel resources. On the other hand, as the search sub-regions become tighter, the probability that an independent PPLS sub-process quickly finds a solution inside its boundaries decreases. As discussed previously, this implies that more solutions from outside the (narrow) sub-region boundaries

Table 1: Impact of the reference point choice and sub-region decomposition. The sign "+"(resp. "−","≈") indicates that the revised PPLS achieves better (resp. worse, equivalent) results than the original PPLS configuration from [10] based on the Wilcoxon test at the 0.05 significance level.

Instance		Average hypervolume ($\times 10^9$)				Average runtime (s)			
		z^*		quality ratio		z^*		acceleration ratio	
ρ	n	**[10]**: (0,0)	$(f_1(x_{LS,2}), f_2(x_{LS,1}))$			**[10]**: (0,0)	$(f_1(x_{LS,2}), f_2(x_{LS,1}))$		
0.75	200	0.4468	0.4361	0.976	(-)	0.33	0.06	5.5	(+)
	300	0.8354	0.8305	0.994	(-)	0.80	0.19	4.2	(+)
	500	4.8290	4.8158	0.997	(≈)	11.98	0.92	13.02	(+)
0.5	200	1.2968	1.2902	0.995	(-)	0.70	0.09	7.78	(+)
	300	3.7403	3.7216	0.995	(-)	4.43	0.63	7.03	(+)
	500	15.3228	15.2888	0.998	(≈)	48.02	3.98	12.06	(+)
0.25	200	2.8881	2.8959	1.003	(≈)	0.36	0.22	1.63	(+)
	300	7.7298	7.7533	1.003	(+)	1.49	1.02	1.46	(+)
	500	38.8932	39.0615	1.004	(+)	32.64	9.73	3.35	(+)
0.0	200	3.8175	3.8166	0.999	(≈)	0.34	0.36	0.94	(≈)
	300	14.4194	14.4146	0.999	(≈)	2.22	2.18	1.02	(≈)
	500	59.3375	59.3323	0.999	(≈)	25.49	27.14	0.94	(≈)
-0.25	200	4.9220	4.9174	0.999	(-)	1.03	0.66	1.56	(+)
	300	19.2335	19.2255	0.999	(≈)	6.35	3.97	1.6	(+)
	500	87.9575	87.9008	0.999	(≈)	124.70	50.11	2.49	(+)
-0.5	200	7.6433	7.6432	0.999	(≈)	3.05	1.33	2.29	(+)
	300	27.4432	27.4402	0.999	(≈)	12.24	7.53	1.62	(+)
	500	122.3464	122.3105	0.999	(≈)	213.57	80.20	2.66	(+)
-0.75	200	9.3167	9.3122	0.999	(-)	17.00	3.52	4.83	(+)
	300	34.6094	34.5806	0.999	(-)	104.50	29.24	3.57	(+)
	500	166.4474	166.4445	0.999	(≈)	958.59	298.48	3.21	(+)



(a) $\rho = -0.5$ (b) $\rho = 0$ (c) $\rho = 0$

Figure 5: Visualization of the trajectory of the modified PPLS using the same three bUBQP instances than in Figure 2.

are likely to be accepted, even if the weighted sum-based selection is designed to guide the sub-process toward its corresponding sub-region boundaries. As we scale the distributed resources, it is hence more likely that different parallel processes are operating longer in different regions than what is expected initially from the definition of the objective space decomposition. This can be interpreted as follows. The parallel PPLS sub-process are wasting time searching for their target sub-regions. It is worth noticing that, depending on the characteristics of the problem at hand, it is a real challenge to predict at which scales such a scalability issue might occur.

Enhanced Sub-region Decomposition. To deal with the aforementioned issue, we propose a simple modification in the definition of the decomposed sub-regions. Instead of using the boundary lines to define a strict partition of the objective space, we enlarge them by a (small) factor, hence allowing two neighboring regions to overlap. More precisely, each of the initially-defined sub-region is enlarged by a prefixed opening angle θ as shown in Figure 6. The rationale behind this modification is that a small, but non-zero, value of θ should allow each parallel PPLS sub-process both to progress more easily, and to reach more quickly its initial target region in the objective space. Obviously, when $\theta = 0$, we obtain the exact same setting than the original PPLS sub-region decomposition [10].

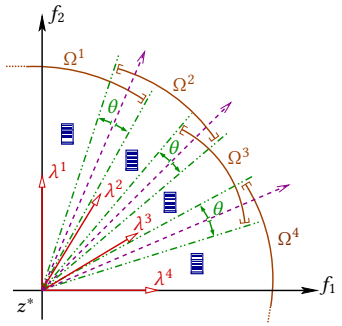


Figure 6: Illustration of the overlapping search subregions using the opening angle θ.

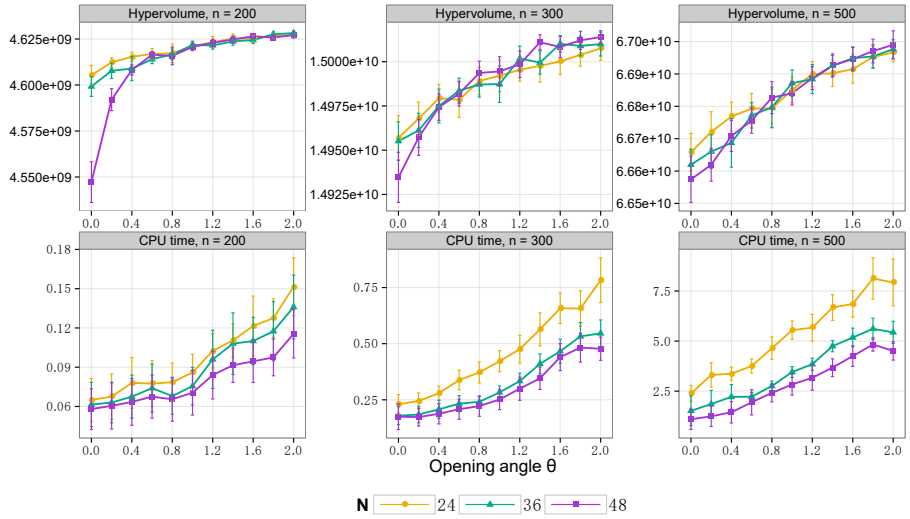


Figure 7: Hypervolume (first row) and CPU runtime (second row) of PPLS using an increasing number $N \in \{24, 36, 48\}$ of parallel process as a function of opening angle θ and for different UBQP instances with $\rho = 0$ and size $n \in \{200, 300, 400\}$ (respectively from left to right).

Experimental Validation. In the following, we study the behavior of PPLS using different (small) θ-values, $\theta \in \{0°, 0.2°, 0.4°, \ldots, 2°\}$ of the opening angle and different (relatively large) numbers of parallel computing nodes $N \in \{24, 36, 48\}$. For this purpose, we consider the same bi-objective UBQP instances with $n \in \{200, 300, 500\}$ using an objective correlation $\rho = 0$. For fairness, the reference point for sub-region decomposition is fixed to $(0, 0)$. Notice in fact that for $\rho = 0$, this was found to be a reasonable choice, and hence our analysis on the impact of the opening angle on scalability is not biased by such a setting. Besides, due to space restriction, we do not consider the other values of ρ since the only difference are with respect to the considered scales. As previously, PPLS is then run 20 times for each configuration, and stops naturally when all solutions in the archive are explored.

Results. In Figure 7, we report the performance of PPLS in terms of hypervolume and CPU time. We clearly see that both performance measures are positively correlated with the value of the opening angle θ. For a *fixed* scale-value N, the larger the value of θ, the better the hypervolume obtained by PPLS, but the larger the time spent by PPLS before terminating. In the setting where θ is *set* to 0, which corresponds to the original configuration of PPLS [10], we can see that, when scaling the number of parallel processes N, the performance in terms of hypervolume is worst, especially for the instance with the smallest size $n = 200$, in which the granularity of the search is likely to be more fine-grained. In turn, the running time decreases slightly. In the setting where θ is *fixed* to a value larger than 0, and the number of processes N is scaled, the difference in the obtained hypervolume is more balanced, but the running time is improved substantially. This means that the runtime of PPLS improves with N increasing, which is interesting from a pure parallel scalability point-of-view, while maintaining approximately the same quality. Thus, we can state that our modification implies that there is a trade-off between the gain in approximation quality and the time required by PPLS to converge, which otherwise would not be possible by simply setting $\theta = 0$.

However, at this stage of the analysis, it is still unclear how good is the enabled trade-off between quality and cost, and if it implies a reasonable scalability behavior of PPLS. In Figure 8, we propose to show the performance of PPLS as a function of quality and time, considered more explicitly as two performance 'objectives' to be maximized and minimized, respectively. We can then clearly see for each fixed scale N, the set of attainable quality/time trade-offs. The first notable observation is that a setting where PPLS is run with a given number of parallel processes N, 'dominates' (respectively, is dominated) by a setting using less (respectively, more) parallel processes. Notice that for an instance size $n = 200$, implying very fine-grained computations, the set of global trade-offs obtained by PPLS do not necessarily improve, but never get worse when scaling N. Looking at the largest instance size $n = 500$, our observation can be clearly derived.

Therefore, analyzing our results according to Figure 8 suggests the following claim. When scaling the number of processes N, there exists an optimal setting of the opening angle θ such that PPLS can perform at its best, depending on whether the target is to minimize time or to maximize quality. To illustrate in a more quantitative manner the gain we obtain with such a claim, let us consider the following scenario. Let T_N^θ and HV_N^θ be respectively the mean hypervolume of the approximation set computed by PPLS and its mean completion time. Assume PPLS is deployed over $N = 24$ parallel processes using its original configuration [10], i.e., $\theta = 0$. If we were given more computing resources $N > 24$, would it be possible for PPLS to compute an approximation set with a mean hypervolume at least as good as $HV_{N=24}^{\theta=0}$ but in a significantly faster time than $T_{N=24}^{\theta=0}$? Alternatively, would it be possible to compute an approximation set having a significantly better mean hypervolume within at most the same target mean time $T_{N=24}^{\theta=0}$? Using the original

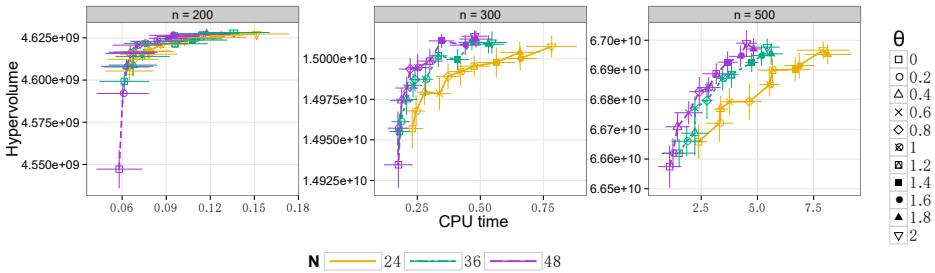


Figure 8: Trade-offs between CPU runtime (to be minimized, x-axis) *vs.* approximation set quality (hypervolume indicator to be maximized, y-axis) of PPLS. For every $N \in \{24, 36, 48\}$ (see the legend in the bottom), we show a performance line using the output of the different θ-values (the point shape refer to θ, see the right legend). Subfigures are with respect to different UBQP instances of size $n \in \{200, 300, 400\}$ (respectively from left to right).

Table 2: Illustration of the gain in Hypervolume values when using the best configuration $\theta^\star$ of the opening angle.

	$N = 24$	$N = 36$			$N = 48$		
n	$HV_N^{\theta=0}$	$HV_N^{\theta=0}$	$\theta^\star$	$HV_N^{\theta^\star}$	$HV_N^{\theta=0}$	$\theta^\star$	$HV_N^{\theta^\star}$
200	4.6053	4.5992	1	4.6217	4.5472	1.2	4.6230
300	14.9570	14.9551	0.8	14.9871	14.9346	1	14.9945
500	66.6581	66.6198	0.6	66.7720	66.5745	0.8	66.8267

configuration of PPLS, we can verify that the answer is no for $N \in \{36, 48\}$. However, using our modified PPLS, this can be achieved for some small values of θ (around 1°) as shown in Table 2, where $\theta^\star$ refers to the choice of the opening angle allowing the best mean hypervolume within mean time not significantly worst than $T_{N=24}^{\theta=0}$, where statistical significance is measured using a Wilcoxon test at significance level 0.05. Furthermore, the hypervolume values $HV_N^{\theta^\star}$ are found to be significantly better for a fixed $N \in \{36, 48\}$ than when using $\theta = 0$.

5 CONCLUSION AND PERSPECTIVES

In this paper, we address the load imbalance and the scalability issues caused by the definition of sub-region decomposition in original PPLS. Beside providing evidence on the accuracy of the proposed modifications, our experimental analysis is intended to shed more lights into the behavior of PPLS, and hopefully to provide some hints for future possible improvements. For instance, a particularly challenging question is to better deal with fine-grained parallelism at extreme scales. In particular, one idea would be to design an automatic technique for the proper setting of the value of θ independently of the tackled problem instance and the considered parallel scale. Since the idea of enlarging the search sub-regions is motivated by helping each PLS processes to escape local optima while staying focused on different target parts of the PF, a particularly promising alternative could be to further introduce communication between the parallel PLS processes. We believe that the online adaptation of the opening angle θ with a carefully designed local cooperation between the multiple PLS processes shall lead to an extremely scalable decomposition-based parallel design of PLS. Besides, confirming and extending our findings for other problem instances with more than two objectives and coming from other problem domains would be a nice piece of research to conduct in the future.

Acknowledgments. The work was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (RGC Project No. A-CityU101/16) and the French national research agency (ANR-16-CE23-0013-01).

REFERENCES

[1] Mădălina M Drugan and Dirk Thierens. 2012. Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. *Journal of Heuristics* 18, 5 (2012), 727–766.
[2] Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. 2015. Anytime Pareto local search. *EJOR* 243, 2 (2015), 369–385.
[3] Martin Josef Geiger. 2011. Decision support for multi-objective flow shop scheduling by the Pareto iterated local search methodology. *Computers & Industrial Engineering* 61, 3 (2011), 805–812.
[4] Arnaud Liefooghe, Jérémie Humeau, Salma Mesmoudi, Laetitia Jourdan, and El-Ghazali Talbi. 2012. On dominance-based multiobjective local search: Design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics* 18, 2 (2012), 317–352.
[5] Arnaud Liefooghe, Sébastien Verel, and Jin-Kao Hao. 2014. A hybrid metaheuristic for multiobjective unconstrained binary quadratic programming. *Applied Soft Computing* 16 (2014), 10–19.
[6] Arnaud Liefooghe, Sébastien Verel, Luis Paquete, and Jin-Kao Hao. 2015. Experiments on local search for bi-objective unconstrained binary quadratic programming. In *EMO-2015 8th International Conference on Evolutionary Multi-Criterion Optimization*, Vol. 9018. 171–186.
[7] K. Miettinen. 1999. *Nonlinear Multiobjective Optimization.* Kluwer, Boston, USA.
[8] Luis Paquete, Tommaso Schiavinotto, and Thomas Stützle. 2007. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research* 156, 1 (2007), 83–97.
[9] Oliver Schütze, Adanay Martín, Adriana Lara, Sergio Alvarado, Eduardo Salinas, and Carlos A. Coello Coello. 2016. The directed search method for multi-objective memetic algorithms. *Computational Optimization and Applications* 63, 2 (2016), 305–332.
[10] Jialong Shi, Qingfu Zhang, Bilel Derbel, Arnaud Liefooghe, and Sébastien Verel. 2017. Using parallel strategies to speed up Pareto local search. In *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 62–74.
[11] Jialong Shi, Qingfu Zhang, and Jianyong Sun. 2017. PPLS/D: Parallel Pareto local search based on decomposition. *arXiv preprint arXiv:1709.09785* (2017).
[12] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE TEC* 11, 6 (2007), 712–731.
[13] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE TEC* 7, 2 (2003), 117–132.