ASYNCHRONOUS LEVEL BUNDLE METHODS

FRANCK IUTZELER[★], JÉRÔME MALICK[♯], AND WELINGTON DE OLIVEIRA[†]

Abstract. In this paper, we consider nonsmooth convex optimization problems with additive structure featuring independent oracles (black-boxes) working in parallel. Existing methods for solving these distributed problems in a general form are synchronous, in the sense that they wait for the responses of all the oracles before performing a new iteration. In this paper, we propose level bundle methods handling asynchronous oracles. These methods require original upper-bounds (using upper-models or scarce coordinations) to deal with asynchronicity. We prove their convergence using variational-analysis techniques and illustrate their practical performance on a Lagrangian decomposition problem. Nonsmooth optimization and level bundle methods and distributed computing and asynchronous algorithms

1. Introduction: context, related work and contributions

1.1. Nonsmooth convex optimization and bundle methods. We consider convex optimization problems of the form

$$f_\star := \min_{x \in X} f(x) \qquad \text{with} \qquad f(x) := \sum_{i=1}^{m} f^i(x)\,, \tag{1}$$

where the constraint set X is compact convex, and the functions $f^i : \mathbb{R}^n \to \mathbb{R}$ (for all $i = 1, \dots, m$) are convex and possibly nonsmooth. Typically, nonsmoothness comes from the fact that the f^i are themselves the results of inner optimization problems, as in Lagrangian relaxation (see e.g. [22]), in stochastic optimization (see e.g. [32]), or in Benders decomposition (see e.g. [14]). In such cases, the functions f^i are known only implicitly through oracles providing values and subgradients (or approximations of them) for a given point.

The so-called bundle methods are a family of nonsmooth optimization algorithms adapted to solve such problems. Using oracle information, these methods construct cutting-plane approximations of the objective function together with quadratic stabilization techniques. Bundle methods can be traced back to [21]; we refer to the textbook [16] and the recent surveys [12, 31] for relevant references. Real-life applications of such methods are numerous, ranging from combinatorial problems [5] to energy optimization [30, 6].

1.2. Centralized distributed setting. In this paper, we further consider the situation where (1) is scattered over several machines: each function f^i corresponds to one machine and associated local data. This scattering is due to the privacy of the data, its prohibitive size, or the prohibitive computing load required to treat it.

These machines perform their computations separately and communicate with a master machine which gathers local results to globally productive updates. This distributed optimization set-up covers a diversity of situations, including computer clusters or mobile agents; it is also a framework attracting interest in machine learning (see e.g. [20, 26]).

Distributed optimization has been studied for decades; see the early references [35, 4]. There is a rich recent literature on distributed optimization algorithms (with no shared memory) for machine learning applications; we mention e.g. [40, 24, 27, 1]. In our context, the functions f^i are nonsmooth and only known through oracles; moreover, in many applications, some of the oracles require considerably more computational effort than others. To efficiently exploit the fact that the oracles f^i are independent and could be computed in parallel, we should turn our attention to *asynchronous* nonsmooth optimization algorithms. An asynchronous algorithm is able to carry on computation without waiting for slower machines: machines perform computations based on outdated versions of the main variable, and a master machine gathers the inputs into a productive update. In traditional synchronous algorithms, latency, bandwidth limits, and unexpected drains on resources that delay the update of machines would cause the

[★] Univ. Grenoble Alpes, LJK, F-38000, Grenoble, France.
[♯] CNRS, LJK, F-38000, Grenoble, France.
[†] CMA – Centre de Mathématiques Appliquées, MINES ParisTech, PSL – Research University, F-06904, Sophia Antipolis, France.

entire system to wait. By eliminating the idle times, asynchronous algorithms can be much faster than traditional ones; see e.g. the recent reference [15].

1.3. **Distributed bundle methods.** Though asynchronism is extremely important for efficiency and resilience of distributed computing, no asynchronous algorithm exists for solving (1) in the above general distributed setting. For example, big problems in stochastic optimization (where uncertainty is due to intermittent renewable energy sources) are heavily structured, often amenable to parallel computing by standard decomposition schemes (e.g. by scenarios [29, 6], by production units [10], or even both [37]). However, existing optimization algorithms exploiting this decomposability are all synchronous, expect for the recent [18] (see also references therein) dealing with a specific polyedral problem.

Convincingly, bundle methods are particularly well-suited for asynchronous generalizations: outdated information provided by late machines could indeed be considered as inexact linearizations, and then could be treated as such by one of the existing tools (see e.g. the recent [25], the review [9], and references therein). However, to our knowledge, there is no asynchronous version of bundle methods for the general distributed setting. Indeed, there does exist an asynchronous (proximal) bundle method [11] but it is tailored for the case where m is large and each component depends only on some of the variables. Moreover its implementation and its analysis are intricate and do not follow the usual rationale of the literature. There is also an asynchronous bundle (trust-region) method [18] designed for dual decomposition of two-stage stochastic mixed-integer problems. This algorithm requires to eventually call all the oracles for all the iterates, which we want avoid in our general situation. Another related work is the incremental (proximal) bundle algorithm of [38], that could serve as a basis for an asynchronous generalization. However, this generalization is not provided or discussed in [38], neither are numerical illustrations of the proposed algorithm with its new features.

In this paper, we propose, analyze, and illustrate the first asynchronous bundle method adapted to the general centralized distributed setting, encompassing computer clusters or mobile agents, described previously. We will not build on the two aforementioned *proximal* bundle methods of [11, 38], but rather investigate *level* bundle methods [23, 19]. In contrast with proximal methods where the iterates' moves can be small even with reasonably rich cutting-planes, level bundle methods fully benefit from collected asynchronous information: richer cutting-plane models would tend to generate useful lower bounds, so that the level set would better approximate the solution set, and the next iterate would better approximate an optimal solution. Such behavior is discussed in the related context of uncontrolled inexact linearizations; see the comparisons between proximal and level methods in Section 4 of [25].

Though asynchronous linearizations of the functions and associated lower-bounds can be well exploited in cutting-plane approximations, existing (level) bundle methods cannot be readily extended in an asynchronous setting because of the lack of upper-bounds: the values of the objective function are never available, since the oracles have no reason to be all called upon the same point. The main algorithmic difficulty is thus to find and manage upper-bounds within asynchronous bundle methods. This is quite specific to bundle methods which rely on estimates of functions values or upper-bounds on the optimal values to compute ad-hoc iterates. In contrast, existing asynchronous methods for other distributed problems usually use predefined stepsize and have an analysis based on fixed-point arguments; see e.g. [24, 27, 26].

1.4. **Contributions, organization.** In this work, we present the first asynchronous level bundle methods for efficiently solving (1) in a general distributed master-slave framework where one master machine centralizes the computations of m machines corresponding to the m oracles of functions f_i. To tackle the lack of function evaluation, we propose two options: (i) to exploit Lispschitz properties of the functions to construct upper-bounds or (ii) to coordinate the oracles when necessary for convergence, which has the interest of requiring no extra knowledge on the functions.

The proposed algorithms are thus totally asynchronous in the terminology of Bertsekas and Tsitsiklis (see Chapter 6 of the book [4]). This means that every machine updates infinitely many times and the relative delays between machines are always finite (but the sequence of delays is possibly unbounded). We underline that the algorithms does not rely on any assumption on the computing system, in contrast with almost all existing literature on asynchronous optimization algorithms; notable exceptions include [34] for coordinate descent and [26] for proximal gradient methods). In practice, the algorithm are not more difficult to program than their synchronous counterparts, apart from the implementation of the exchanges between the master and the machines.

We provide a convergence analysis of our algorithms under the assumption that the convex constraint set X is compact. By using elegant tools from set-valued analysis, we offer an original and more straightforward proof of convergence of level bundle algorithms. To simplify the presentation, we first restrict to the case of *exact* oracles (i.e. when the functions f^i are known through subroutines providing their values and subgradient for a given point), and then present the extension to *inexact* oracles (i.e. when the subroutine provides noisy approximations of its value and gradient). The numerical experiments on a Lagrangian decomposition of a mixed-integer problem illustrate the features of the asynchronous algorithms compared to the standard (synchronous) one.

This paper is organized as follows. Section 2 reviews the main ingredients of level bundle methods, presents the standard level bundle method [19], and introduces disaggregated models, on which asynchronous bundle methods are built. Section 3 incorporates to this algorithm the asynchronous communications and the upper-bound estimation to provide a first asynchronous bundle algorithm. Section 4 presents and studies a second asynchronous bundle algorithm using scarce coordination between machines, requiring no extra assumption on the functions. In Section 5, we extend our previous results to the case of noisy inexact oracles. Finally, Section 6 presents some preliminary numerical experiments, illustrating and comparing the algorithms.

2. Level bundle methods: recalls and disaggregated version

This section reviews the main ideas about (synchronous) level bundle methods. In particular, we recall in Section 2.1 the classical algorithm with the associated notation. We also describe in Section 2.2 a disaggregated variant exploiting the decomposability of the objective function of (1). The use of disaggregate models is well-known for proximal bundle methods, but not fully investigated for level bundle methods: we develop here the useful material for the asynchronous algorithms of the next sections.

2.1. **Main ingredients of level bundle methods.**

We briefly present the classical scheme of level bundle methods, dating back to [23, 19], to minimize over X the function f only known through an exact oracle. While presenting the algorithm and its main features, we introduce standard notation and terminology of convex analysis and bundle methods; see e.g. the textbook [16].

Algorithm 1 essentially corresponds to the level bundle method of [19]; its main features are the following ones. The generic bundle algorithm produces a sequence of feasible points $(x_k) \subset X$. For each iterate x_k, the oracle information $f(x_k)$ and $g_k \in \partial f(x_k)$ is assumed to be available. With such information, the method creates linearizations of the form $f(x_k) + \langle g_k, \cdot - x_k\rangle \leq f(\cdot)$. We denote $\langle x, y\rangle := \sum_j x_j y_j$ the usual inner product of two vectors $x, y \in \mathbb{R}^n$, and $\|x\| = \sqrt{\langle x, x\rangle}$ the associated Euclidean norm. Such linearizations are used to create the cutting-plane model for f at iteration k:

$$\check{f}_k(x) := \max_{j \in J_k}\{f(x_j) + \langle g_j, x - x_j\rangle\} \quad \leq \quad f(x) \tag{2}$$

where $J_k \subset \{1, 2, \ldots, k\}$ is a set of indices of points at which the oracle was called. For any k, we introduce the level set $\mathbb{X}_k$ of $\check{f}_k$ associated to level parameter $f_k^{\text{lev}} \in \mathbb{R}$

$$\mathbb{X}_k := \{x \in X : \check{f}_k(x) \leq f_k^{\text{lev}}\} \quad \supset \quad \{x \in X : f(x) \leq f_k^{\text{lev}}\}.$$

This level set defines a region of the feasible set in which the next iterate is chosen: $x_{k+1} \in \mathbb{X}_k$. It also provides a lower bound f_k^{low} for $f_\star$ whenever $\mathbb{X}_k = \emptyset$. Indeed, one can take $f_{k+1}^{\text{low}} = f_k^{\text{lev}}$ as, in this case, $f_\star \geq f_k^{\text{lev}}$.

A common rule to choose the next iterate is by projecting a certain stability center $\hat{x}_k \in (x_k)$ onto $\mathbb{X}_k$, that is

$$x_{k+1} := \arg\min_{x \in \mathbb{X}_k} \frac{1}{2}\|x - \hat{x}_k\|^2 . \tag{3}$$

When X is a polyhedral set, computing the next iterate consists in solving a mere convex quadratic optimization problem. This is also the case when X is a Euclidean ball, as pointed out in [7]. Since the sequence (x_k) is available, an upper bound for the optimal value $f_\star$ is just $f_k^{\text{up}} := \min_{j \in \{1 \ldots, k\}} f(x_j)$. Thus, we have an upper and a lower bound on the optimal value; we can then define the gap

$$\Delta_k := f_k^{\text{up}} - f_k^{\text{low}},$$

The gap gives a natural stopping test $\Delta_k \leq \text{tol}_\Delta$, for some stopping tolerance $\text{tol}_\Delta \geq 0$. Moreover, Δ_k can also be used to update the level parameter as

$$f_k^{\text{lev}} := \alpha f_k^{\text{low}} + (1 - \alpha) f_k^{\text{up}} = f_k^{\text{up}} - \alpha \Delta_k, \qquad \text{with } \alpha \in (0, 1). \tag{4}$$

Remark 1 (About convergence). The convergence of the standard level bundle method presented in Algorithm 1 follows Theorem 3.5 in [19]. The proof uses that $x_{k+1} \in \mathbb{X}_k$ implies $f(x_k) + \langle g_k, x_{k+1} - x_k \rangle \leq f_k^{\text{lev}}$ for $k \in J_k$, that in turn yields

$$\|x_{k+1} - x_k\| \geq \frac{f(x_k) - f_k^{\text{lev}}}{\|g_k\|} \geq \frac{\alpha \Delta_k}{\Lambda}, \tag{5}$$

where the bound $\|g_k\| \leq \Lambda$ can be guaranteed by compactness and convexity of f. Since this inequality requires the value of $f(x_k)$, it is not guaranteed anymore in asynchronous setting. We will pay special attention to this technical point in the algorithms of Section 4.

Algorithm 1 Standard level bundle method

```
 1: Choose $x_1 \in X$ and set $\hat{x}_1 = x_1$
 2: Define $f_1^{\text{up}} = \infty$, $J_0 = \emptyset$ and $\hat{\Delta} = \infty$
 3: Choose a stopping tolerance $\text{tol}_\Delta \geq 0$, a parameter $\alpha \in (0,1)$ and a finite $f_1^{\text{low}} \leq f_\star$
 4: Send $x_1$ to the oracle
 5: for $k = 1, 2, \ldots$ do
                                              ▷ Step 1: receive information from oracle
 6:     Receive $(f(x_k), g_k)$ from the oracle
 7:     Set $J_k = J_{k-1} \cup \{k\}$
 8:     if $f(x_k) < f_k^{\text{up}}$ then
 9:         Update $f_k^{\text{up}} = f(x_k)$ and $x_{\texttt{best}} = x_k$        ▷ update upper bound
10:     end if
                                              ▷ Step 2: test optimality and sufficient decrease
11:     Set $\Delta_k = f_k^{\text{up}} - f_k^{\text{low}}$
12:     if $\Delta_k \leq \text{tol}_\Delta$ then
13:         Return $x_{\texttt{best}}$ and $f_k^{\text{up}}$
14:     end if
15:     if $\Delta_k \leq \alpha\hat{\Delta}$ then
16:         Set $\hat{x}_k = x_{\texttt{best}}$, set $\hat{\Delta} = \Delta_k$, and possibly reduce $J_k$        ▷ critical iterate
17:     end if
                                              ▷ Step 3: compute next iterate
18:     Set $f_k^{\text{lev}} = f_k^{\text{up}} - \alpha\Delta_k$. Run a quadratic solver on problem (3).
19:     if (3) is feasible then
20:         Get the new iterate $x_{k+1} \in \mathbb{X}_k$. Update $f_{k+1}^{\text{low}} = f_k^{\text{low}}$ and $f_{k+1}^{\text{up}} = f_k^{\text{up}}$
21:     else
22:         Set $f_k^{\text{low}} = f_k^{\text{lev}}$ and go to Step 2        ▷ update lower bound
23:     end if
                                              ▷ Step 4: send back information to the oracle
24:     Send $x_{k+1}$ to the oracle
25:     Set $\hat{x}_{k+1} = \hat{x}_k$
26: end for
```

2.2. **Disaggregated level bundle iteration.**

We present here a level bundle algorithm which exploits the additive structure of the objective function (1) and the presence of m oracles[1] providing individual information $(f^i(x), g^i) \in \mathbb{R}^{1+n}$ with $g^i \in \partial f^i(x)$, for every any point $x \in X$. We define $J_k^i \subset \{1, \ldots, k\}$ as the index set of the points in the sequence (x_k) where the oracle i was called, i.e. such that $(f^i(x_j), g_j^i)_j$ is computed. The unique feature of our situation and the main technical point is that the intersection of the index sets may contain only few elements, or even be empty.

We can define individual cutting-plane models for each i and k

$$\check{f}_k^i(x) := \max_{j \in J_k^i} \{f^i(x_j) + \langle g_j^i, x - x_j \rangle\} \quad \leq \quad f^i(x). \tag{6}$$

Instead of approximating the objective function f by its aggregate cutting-plane model $\check{f}_k$ (2), we approximate f by its disaggregated model $\sum_{i=1}^m \check{f}_k^i$. While the idea is standard for proximal bundle methods (see e.g. [2] for an application in electricity management and [13] for an application in network optimization), the use of disaggregated models

[1]To better underline our contributions on asynchronicity, we consider first only exact oracles of the f^i as above. Later in Section 5, we explain how our developments easily extend to the case of inexact oracles providing noisy approximations of $(f^i(x), g^i)$.

has not been fully investigated for level methods: we are only aware of the level algorithm of [39] that employs a disaggregate model indirectly just to request exact information from on-demand accuracy oracles.

If $J_k \subset \cap_{i=1}^m J_k^i$, then the disaggregated model provides a better approximation for f than the aggregated model based on J_k

$$\check{f}_k(x) \quad \leq \quad \sum_{i=1}^m \check{f}_k^i(x) \quad \leq \quad f(x) \quad \forall\, x \in \mathbb{R}^n\,.$$

We can then replace in the level bundle algorithm the quadratic subproblem (3) by the disagregated quadratic subproblem:

$$\left\{\begin{array}{ll} \min_x & \frac{1}{2}\|x-\hat{x}_k\|^2 \\ \text{s.t.} & x \in \mathbf{X}_k^{\mathrm{d}}\,, \end{array}\right. \qquad \text{with} \quad \mathbf{X}_k^{\mathrm{d}} := \left\{ x \in X : \sum_{i=1}^m \check{f}_k^i(x) \leq f_k^{\mathrm{lev}} \right\}, \tag{7}$$

where $\hat{x}_k$ is the stability center and $f^{\mathrm{lev}} \in (f_k^{\mathrm{low}}, f_k^{\mathrm{up}})$ is the level parameter. As in Section 2, if problem (7) is infeasible, then f_k^{lev} is a lower bound for $f_\star$.

Aside from offering better accuracy, the disaggregate model has another advantage in our context: it allows for partial update of the cutting-plane model using individual oracle evaluations, without calling all m of the oracles at the same point. This is an important feature that permits to handle oracles in an asynchronous manner.

We formalize in the next lemma the easy result stating that (7) can be cast as a standard quadratic problem.

Lemma 1 (Disaggregated master problem)**.** *Assume that the feasible set of* (7) *is nonempty. Then the unique solution of* (7) *can be obtained by solving the following quadratic optimization problem (in both variables x and r) and disregarding the r-component of its the solution:*

$$\left\{\begin{array}{lll} \min_{x,r} & \frac{1}{2}\|x-\hat{x}_k\|^2 & \\ s.t. & x \in X,\, r \in \mathbb{R}^m & \\ & f^1(x_j) + \langle g_j^1, x - x_j\rangle \leq r^1 & \forall\, j \in J_k^1 \\ & \vdots & \vdots \\ & f^m(x_j) + \langle g_j^m, x - x_j\rangle \leq r^m & \forall\, j \in J_k^m \\ & \sum_{i=1}^m r^i \leq f_k^{\mathrm{lev}}\,. & \end{array}\right. \tag{8}$$

Proof. Let $\bar{x}$ be the (unique) solution to problem (7), and (x_{k+1}, r_{k+1}) be the solution of (8). Then by defining $\bar{r}^i = \check{f}_k^i(\bar{x})$ we get that $(\bar{x}, \bar{r})$ is feasible for (8) and therefore the optimal value of (8), $\frac{1}{2}\|x_{k+1}-\hat{x}_k\|^2$ is less or equal than $\frac{1}{2}\|\bar{x}-\hat{x}_k\|^2$, the optimal value of (7). Then $\sum_{i=1}^m \check{f}_k^i(x_{k+1}) \leq \sum_{i=1}^m r_{k+1}^i \leq f_k^{\mathrm{lev}}$, showing that x_{k+1} is feasible to problem (7). We therefore have shown that $\|\bar{x}-\hat{x}_k\|^2 \leq \|x_{k+1}-\hat{x}_k\|^2 \leq \|\bar{x}-\hat{x}_k\|^2$. Consequently, every x-component solution x_{k+1} of (8) is also a solution to (7). As the latter problem has a unique solution, then $x_{k+1} = \bar{x}$. □

3. Asynchronous level bundle method by upper-bound estimation

In this section, we present and analyze the first asynchronous level bundle method, using the disagregated master subproblem (8) to incorporate asynchronous linearizations from the oracles. We will index the time by k, the number of iterations of the master. One iteration corresponding to the treatment of the information sent by one oracle. At iteration k, the master receives the information from an oracle, say i, updates the disagregated model, generates a new point, and sends back a point to oracle i.

The asynchronicity in the algorithm makes that the oracles do not necessarily provide information on the last iterate but on a previous one, so that asynchronous bundle method has to deal with delayed linearizations. While we assume that all oracles respond and are incorporated in finite time (which is the standard setting of totally asynchronous in the terminology of [4, Chap. 6]), we do not need to upper bound theses response times. In order to incorporate these delayed cuts, we denote by $\mathrm{a}(i)$ the iteration index of the *anterior* information provided by oracle i : at iteration k, the exchanging oracle i provides the linearization for f^i at the point denoted $x_{\mathrm{a}(i)}$ (see lines 6 and 7 of the algorithm and Figure 1). Our finite response assumption then simply translates to $a(i) \to \infty$ as $k \to \infty$ for all i.

Apart from the above communication with oracles, the main algorithmic difference between the asynchronous level bundle method and the standard level algorithm (Algorithm 1) is the management of upper-bounds f_k^{up}. The strategy presented here (and inspired by [38]) is to estimate upper bounds on $f_\star$ without evaluating all the component

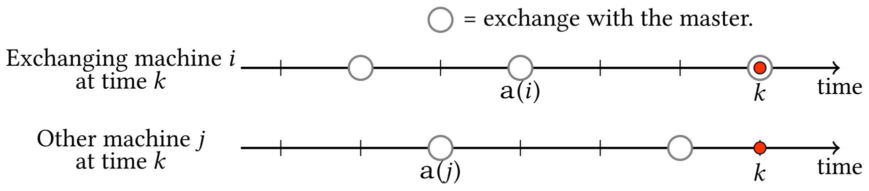
= exchange with the master.
Exchanging machine i at time k
$a(i)$
k
time
Other machine j at time k
$a(j)$
k
time

FIGURE 1. Notations for the delayed cuts added in our asynchronous bundle algorithms.

functions f^i at the same point. To this end, we make the assumption that we know an upper bound $\bar{\Lambda}^i$ on the Lipschitz constant Λ^i of f^i for all $i = 1, \dots, m$. In other words, we assume

$$|f^i(x) - f^i(y)| \leq \bar{\Lambda}^i \|x - y\| \qquad \text{for all } x, y \in X. \tag{9}$$

The recent work [38] builds on the same assumption and proposes to bound $f^i(x)$ at a given point by solving an extra quadratic problem of size $|J^i_k| + 1$ depending on $\bar{\Lambda}^i$. Using this technique, we would obtain an upper-bound of $f(x)$ at the extra cost of solving $m - 1$ quadratic problems at each iteration. We propose in Algorithm 2 a simpler procedure to compute upper bounds f^{up}_k without solving quadratic problems or other extra cost.

Algorithm 2 Asynchronous level bundle method by upper-bounds estimation

```
1: Given $\bar{\Lambda}^i$ satisfying (9), choose $x_1 \in X$, and set $\bar{x}_{\text{best}} = \hat{x}_1 = x_1$
2: Choose a tolerance $\text{tol}_\Delta \geq 0$, a parameter $\alpha \in (0,1)$, and $f^{\text{up}}_1 > f_\star + \text{tol}_\Delta$ and $f^{\text{low}}_1 \leq f_\star$
3: Set $\hat{\Delta} = \infty$ and $J^i_0 = \emptyset$ for all $i = 1, \dots, m$
4: Send $x_1$ to the $m$ oracles
5: for $k = 1, 2, \dots$ do
                                              ▷ Step 1: receive information from an oracle
6:     Receive from oracle $i$ the oracle information on $f^i$ at a previous iterate $x^{k'}$
7:     Set $a(i) = k'$, store $(f^i(x_{k'}), g^i_{k'})$, and set $J^i_k = J^i_{k-1} \cup \{k'\}$
                                              ▷ Step 2: test optimality and sufficient decrease
8:     Set $\Delta_k = f^{\text{up}}_k - f^{\text{low}}_k$
9:     if $\Delta_k \leq \text{tol}_\Delta$ then
10:        Return $\bar{x}_{\text{best}}$ and $f^{\text{up}}_k$
11:    end if
12:    if $\Delta_k \leq \alpha\hat{\Delta}$ then
13:        Set $\hat{x}_k = \bar{x}_{\text{best}}$ and $\hat{\Delta} = \Delta_k$. Possibly reduce $J^j_k$ for all $j = 1, \dots, m$.
14:    end if
                                              ▷ Step 3: compute next iterate
15:    Set $f^{\text{lev}}_k = f^{\text{up}}_k - \alpha\Delta_k$. Run a quadratic optimization software on problem (8)
16:    if QP (8) is feasible then
17:        Get new iterate $x_{k+1} \in \mathbf{X}^{\text{d}}_k$. Update $f^{\text{low}}_{k+1} = f^{\text{low}}_k$
18:    else
19:        Set $f^{\text{low}}_k = f^{\text{lev}}_k$ and go to Step 2          ▷ update lower bound
20:    end if
21:    if $J^j_k \neq \emptyset$ for all $j = 1, \dots, m$ then
22:        $f^{\text{up}}_{k+1} = \min\left\{f^{\text{up}}_k,\ f^{\text{lev}}_k + \sum_{j=1}^{m}\left(\bar{\Lambda}^j\|x_{k+1} - x_{a(j)}\| - \langle g^j_{a(j)}, x_{k+1} - x_{a(j)}\rangle\right)\right\}$
23:        if $f^{\text{up}}_{k+1} < f^{\text{up}}_k$ then
24:            set $\bar{x}_{\text{best}} = x_{k+1}$
25:        end if
26:    else
27:        $f^{\text{up}}_{k+1} = f^{\text{up}}_k$
28:    end if
                                              ▷ Step 4: send back information to the oracle
29:    Send $x_{k+1}$ to machine $i$
30:    Set $\hat{x}_{k+1} = \hat{x}_k$
31: end for
```

Note that the upper bound is (possibly) improved only after all the oracles have responded at least once. This means that the index $a(j)$ (representing the iteration index of the last iterated processed by oracle j) is well defined.

3.1. **Upper-bound estimation in asynchronous case.** The strategy displayed on line 22 of Algorithm 2 to compute upper bounds is based on the following lemma. This yields a handy rule for updating f^{up}_k depending only on the distance (weighted by $\bar{\Lambda}^i$) between the current solution of the master QP (8), x_{k+1}, and the last/anterior points upon which the oracles responded, $x_{\mathrm{a}(i)}$ for $i = 1,..,m$.

Lemma 2. *Suppose that (9) holds true for $i = 1, \dots, m$. At iteration k, whenever the master QP (8) is feasible, with (x_{k+1}, r_{k+1}) its solution, one has*

$$f(x_{k+1}) \leq f_k^{\text{lev}} + \sum_{j=1}^{m} \left(\bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| - \langle g^j_{\mathrm{a}(j)}, x_{k+1} - x_{\mathrm{a}(j)} \rangle \right). \tag{10}$$

Furthermore,

$$\alpha \Delta_k \leq f_k^{\text{up}} - f_{k+1}^{\text{up}} + 2 \sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\|. \tag{11}$$

Proof. Note first that

$$f(x_{k+1}) = \sum_{j=1}^{m} f^j(x_{k+1}) = \sum_{j=1}^{m} f^j(x_{\mathrm{a}(j)}) + \sum_{j=1}^{m} \left(f^j(x_{k+1}) - f^j(x_{\mathrm{a}(j)}) \right).$$

This yields

$$\begin{aligned} f(x_{k+1}) &\leq \sum_{j=1}^{m} \left(r^j_{k+1} - \langle g^j_{\mathrm{a}(j)}, x_{k+1} - x_{\mathrm{a}(j)} \rangle \right) + \sum_{j=1}^{m} \left(f^j(x_{k+1}) - f^j(x_{\mathrm{a}(j)}) \right) \\ &\leq f_k^{\text{lev}} + \sum_{j=1}^{m} \left(\bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| - \langle g^j_{\mathrm{a}(j)}, x_{k+1} - x_{\mathrm{a}(j)} \rangle \right) \ , \end{aligned}$$

where the first inequality comes from the fact that (x_{k+1}, r_{k+1}) is feasible point for the master QP (8). The second inequality uses that $\sum_{j=1}^{m} r^j_{k+1} \leq f_k^{\text{lev}}$ as (x_{k+1}, r_{k+1}) is a feasible point of (8) and the Lipschitz assumption (9) for the functions.

To prove the second part of the result, we proceed as follows:

$$\begin{aligned} f_{k+1}^{\text{up}} &\leq f_k^{\text{lev}} + \sum_{j=1}^{m} \left(\bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| - \langle g^j_{\mathrm{a}(j)}, x_{k+1} - x_{\mathrm{a}(j)} \rangle \right) \\ &\leq f_k^{\text{up}} - \alpha \Delta_k + 2 \sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\|, \end{aligned}$$

where the second inequality is due to the definition of f_k^{lev} and due to the bound on the scalar product provided by the Lipschitz assumption (9). This concludes the proof. □

At each iteration, the asynchronous level algorithm (Algorithm 2) computes upper bounds of the functional values by using inequality (10). The rest of the algorithm corresponds essentially[2] to the standard level algorithm (Algorithm 1) using the disaggregated model (8). In particular, since the values f_k^{up} are provable upper bounds, we still have $f_\star \in [f_k^{\text{low}}, f_k^{\text{up}}]$ for all k and the stopping test $\Delta_k \leq \text{tol}_\Delta$ is valid. The convergence analysis of the next section relies on proving that the sequence of gaps (Δ_k) does tend to 0 when $\text{tol}_\Delta = 0$.

[2]Note that Algorithm 2 still needs initial bounds f_1^{up} and f_1^{low}. These bounds can often be easily estimated from the data of the problem. Otherwise, we can use the standard initialization: call the m oracles at an initial point x_1 and wait for their first responses from which we can compute $f_1^{\text{up}} = f(x_1) = \sum_i f^i(x_1)$ and f_1^{low} as the minimum of the linearization $f(x_1) + \langle g_1, x - x_1 \rangle$ over the compact set X. If we do not want to have this synchronous initial step, we may alternatively estimate f^{lev} and set $f_1^{\text{up}} = +\infty$ and $f_1^{\text{low}} = -\infty$. This would require small changes in the algorithm (in line 15) and in its proof (in Lemma 3). For sake of clarity we stick with the simplest version of the algorithm and the most frequent situation where we can easily estimate f_1^{up} and f_1^{low}.

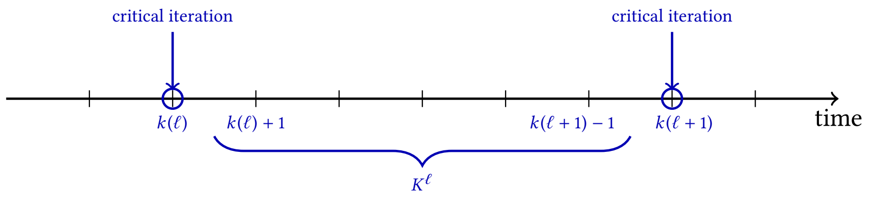

FIGURE 2. Illustration of the set K^ℓ used in convergence analysis

3.2. **Convergence analysis.** This section analyzes the convergence of the asynchronous level bundle algorithm described in Algorithm 2, under the assumption (9). At several stages of the analysis, we use the fact that the sequences of the optimality gap Δ_k and upper bound f_k^{up} are non-increasing by definition, and that the sequence of lower bound f_k^{low} is non-decreasing. More specifically, we update the lower bound only when the master QP (8) is infeasible.

We count with ℓ the number of times the gap significantly decreases, meaning that line 13 is accessed, and denote by $k(\ell)$ the corresponding iteration. We have by construction

$$\Delta_{k(\ell+1)} \leq \alpha\Delta_{k(\ell)} \leq \alpha^2\Delta_{k(\ell-1)} \leq \cdots \leq \alpha^\ell\Delta_1 \quad \forall\, \ell = 1, 2, \ldots \tag{12}$$

We call $k(\ell)$ a critical iteration, and $x_{k(\ell)}$ a critical iterate. We introduce the set of iterates between two consecutive critical iterates as illustrated by Fig 2:

$$K^\ell := \{k(\ell)+1, \ldots, k(\ell+1)-1\}. \tag{13}$$

The proof of convergence of Algorithm 2 consists in showing the algorithm performs infinitely many critical iterations when $\text{tol}_\Delta = 0$. We start with basic properties for iterations in K^ℓ, valid beyond Algorithm 2 for any level bundle method under mild assumptions.

Lemma 3 (Between two consecutive critical iterates)**.** *Consider a level bundle method (such as Algorithm 2) satisfying*

- f_k^{up} *is a non-increasing sequence of upper-bounds on* $f_\star$*;*
- f_k^{low} *is updated only when the master QP is empty, and* f_k^{low} *is chosen as a lower bound greater or equal to* f_k^{lev}*;*
- f_k^{lev} *satisfies* $f_k^{\text{lev}} = \alpha f_k^{\text{low}} + (1-\alpha)f_k^{\text{up}}$*;*
- *all the linearizations are kept between two critical steps.*

Fix an arbitrary ℓ and let K^ℓ be defined by (13)*. Then,* $(\mathbb{X}_k^{\text{d}})$ *is a nested non-increasing sequence of non-empty compact convex sets:* $\mathbb{X}_k^{\text{d}} \subset \mathbb{X}_{k-1}^{\text{d}}$ *for all* $k \in K^\ell$*. Furthermore, for all* $k \in K^\ell$*,*

(i) *the master QP* (8) *is feasible;*
(ii) *the stability center and the lower bound are fixed:* $\hat{x}_k = \hat{x}_{k(\ell)}$ *and* $f_k^{\text{low}} = f_{k(\ell)}^{\text{low}}$*;*
(iii) *the level parameter and the gap can only decrease:* $f_k^{\text{lev}} \leq f_{k(\ell)}^{\text{lev}}$ *and* $\Delta_k \leq \Delta_{k(\ell)}$*.*

Proof. We start with proving (i), (ii) and (iii). Each $\mathbb{X}_k^{\text{d}}$ is non-empty as otherwise the master QP (8) would be infeasible. Indeed, if (8) was infeasible at time k, f_k^{low} receives f_k^{lev} and therefore $f_k^{\text{low}} = f_k^{\text{up}} - \alpha\Delta_k \geq f_k^{\text{up}} - \alpha\Delta_{k(\ell)}$ so $f_k^{\text{up}} - f_k^{\text{low}} \leq \alpha\Delta_{k(\ell)}$, which contradicts the fact that $k \in K^\ell$ (i.e, is not a critical step). This proves (i).

For each $k \in K^\ell$, the stability center is fixed by construction and the lower bound is updated (increased) only when the master QP (8) is found infeasible which is impossible for $k \in K^\ell$ (see above). This establishes (ii).

The inequality on the level parameter comes directly as follows: $f_k^{\text{lev}} = \alpha f_k^{\text{low}} + (1-\alpha)f_k^{\text{up}} = \alpha f_{k(\ell)}^{\text{low}} + (1-\alpha)f_k^{\text{up}} \leq \alpha f_{k(\ell)}^{\text{low}} + (1-\alpha)f_{k(\ell)}^{\text{up}} = f_{k(\ell)}^{\text{lev}}$. Note finally that f_k^{up} is non-increasing and so is Δ_k. We thus have also (iii).

Finally, each $\mathbb{X}_k^{\text{d}}$ is compact as X is compact, and it is also convex as X is convex and the disaggregate cutting-plane model is convex. The fact that $(\mathbb{X}_k^{\text{d}})$ is a nested non-increasing sequence is thus direct from (iii) as the local cutting plane models only get richer with k (as the model is only reduced in critical steps which cannot happen in K^ℓ). □

We now provide a proof of convergence featuring elegant results from variational analysis [28].

Theorem 1 (Convergence)**.** *Assume that X is a convex compact set and that* (9) *holds. Let* $\text{tol}_\Delta = 0$ *in Algorithm 2, then the sequence of gaps vanishes,* $\lim_k \Delta_k = 0$*, and the sequence of best iterates is a minimizing sequence for* (1)*,* $\lim_k f_k^{\text{up}} = f_\star$*. As a consequence, for a strictly positive tolerance* $\text{tol}_\Delta > 0$*, the algorithm terminates after finitely many steps with an approximate solution:* $f_\star \leq f(\bar{x}_{\texttt{best}}) \leq f_\star + \text{tol}_\Delta$*.*

Proof. The convergence $\Delta_k \to 0$ is given by (12) as soon as the counter of critical steps ℓ increases indefinitely. Thus we just need to prove that, after finitely many steps, the algorithm performs a new critical iteration. For the sake of contradiction, suppose that only finitely many critical iterations are performed. Accordingly, let $\bar{\ell}$ be the total number of critical iterations and $k(\bar{\ell})$ be the index of the last critical iteration. Observe that $\hat{x}_k = \hat{x}$ is fixed and $\Delta_k \geq \Delta > 0$ for all $k > k(\bar{\ell})$. We have from Lemma 3, that $(\mathbb{X}_k^{\mathrm{d}})$ is a nested non-increasing sequence of non-empty compact convex sets for $k > k(\bar{\ell})$. Suppose that there is an infinite number of asynchronous iterations after the last critical iteration $k(\ell)$, then $(\mathbb{X}_k^{\mathrm{d}})$ converges to $\mathbb{X}^{\mathrm{d}}$ in the sense of the Painlevé-Kuratowski set convergence [28, Chap. 4.B]:

$$\lim_k \mathbb{X}_k^{\mathrm{d}} = \mathbb{X}^{\mathrm{d}} = \bigcap_k \mathtt{cl}\, \mathbb{X}_k^{\mathrm{d}} .$$

Now, Šmulian's theorem [33, 3] guarantees that the intersection $\mathbb{X}^{\mathrm{d}} = \cap_k \mathtt{cl}\, \mathbb{X}_k^{\mathrm{d}}$ is nonempty. Moreover, $\mathbb{X}^{\mathrm{d}}$ is by definition a convex compact set and, therefore, the projection of $\hat{x}$ onto $\mathbb{X}^{\mathrm{d}}$ is well defined and unique:

$$P_{\mathbb{X}^{\mathrm{d}}}(\hat{x}) = \arg\min_{x \in \mathbb{X}^{\mathrm{d}}} \frac{1}{2}\|x - \hat{x}\|^2.$$

Then [28, Prop. 4.9] implies that $x_{k+1} = \arg\min_{x \in \mathbb{X}_k^{\mathrm{d}}} \frac{1}{2}\|x - \hat{x}\|^2 = P_{\mathbb{X}_k^{\mathrm{d}}}(\hat{x})$ converges to $P_{\mathbb{X}^{\mathrm{d}}}(\hat{x})$. Hence, (x_k) is a Cauchy sequence

$$\forall \varepsilon > 0 \; \exists \bar{k} \in \mathbb{N} \text{ such that } \forall s, t \geq \bar{k} \quad \Longrightarrow \quad \|x_s - x_t\| \leq \varepsilon .$$

By taking $\varepsilon = \frac{\alpha}{4m \max_i \bar{\Lambda}_i}\Delta$ and[3] $k \geq \min\limits_{j=1,\ldots,m} \mathtt{a}(j) \geq \bar{k}$, the inequality in (11) gives

$$\alpha\Delta \leq \alpha\Delta_k \leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + 2\sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathtt{a}(j)}\|$$

$$\leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + 2\sum_{j=1}^{m} \bar{\Lambda}^j \frac{\alpha}{4m \max_i \bar{\Lambda}_i}\Delta \leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + \frac{\alpha}{2}\Delta,$$

showing that $f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} \geq \frac{\alpha}{2}\Delta > 0$ for all $k \geq \min\limits_{j=1,\ldots,m} \mathtt{a}(j) \geq \bar{k}$. This is in contradiction with the fact that the sequence (f_k^{up}) is non-increasing and lower bounded, thus convergent. Hence, the index set K^ℓ is finite and ℓ grows indefinitely if $\mathrm{tol}_\Delta = 0$. Finally, the proof that (f_k^{up}) converges to the optimal value follows easily by noting that

$$f_\star \leq f_k^{\mathrm{up}} = f_k^{\mathrm{low}} + \Delta_k \leq f_\star + \Delta_k$$

from $f_k^{\mathrm{low}} \leq f_\star \leq f_k^{\mathrm{up}}$ and $\Delta_k = f_k^{\mathrm{up}} - f_k^{\mathrm{low}}$, which ends the proof. □

4. Asynchronous level bundle method by coordination

The level bundle algorithm of the previous section is fully asynchronous for solving problem (1). It relies however on having bounds on the Lipshitz constant of the functions f_i, which is a strong practical assumption. To compute upper-bounds f_k^{up}, we propose here an alternative strategy based on coordination of the machines to evaluate $f(x_k)$ if necessary. We present this method in Section 4.1 (Algorithm 3) and we analyze its convergence in Section 4.2.

4.1. **Upper-bounds by coordination.**

In our asynchronous setting, the oracles have no reason to be called upon the same point except if the master decides to coordinate them. We propose a test that triggers coordination of the points sent to the machines, when the proof of convergence is in jeopardy (namely, when (5) does not hold). Thus we introduce a coordination step (see line 32 in Algorithm 3): if the test is valid, this step consists in sending to all oracles the same iterate at the next iteration for which they are involved. We note that some incremental algorithms (as [17]) also have such a coordination step. In such algorithms though, there is usually a coordination step after a fixed number of iterations. Here, in contrast, we require coordination only when the difference between two iterates becomes too small. Our coordination strategy does not generate idle time because the machines are not requested to abort their current jobs.

The second asynchronous algorithm is given in Algorithm 3. Its steps correspond to the ones of Algorithm 2, with more complex communications (Steps 1 and 4). Step 2 (optimality and sufficient decrease test) is unchanged.

[3]As the oracles are assumed to respond in a finite time, the inequality $\min_{j=1,\ldots,m} \mathtt{a}(j) \geq \bar{k}$ is guaranteed to be satisfied for k is large enough.

Finally, Step 3 (next iterate computation) relies on the same master problem (8) in both algorithm, but here the coordination-triggering test replaces by a upper-bound test.

The coordination iterates are denoted by $\bar{x}_k$ in Algorithm 3. Assuming that all oracles always eventually respond (after an unknown time), the coordination allows to compute the full value $f(\bar{x}_k)$ and a subgradient $g \in \partial f(\bar{x}_k)$ at the coordination iterate $\bar{x}_k$ (see line 10, where `rr` ("*remaining to respond*") counts the number of oracles that have not responded yet). The functional value is used to update the upper bound f_k^{up}, as usual for level methods; the subgradient is used to update the bound L approximating the Lipschitz constant of f.

In the algorithm, the coordination is implemented by two vectors of booleans (**to-coordinate** and **coordinating**):

- The role **to-coordinate**$[i]$ is to indicate to machine i that its next computation has to be performed with the new coordination point $\bar{x}_{k+1}$; (at that moment, **to-coordinate**$[i]$ is set to False and **coordinating**$[i]$ is set to True.)
- The role **coordinating**$[i]$ is to indicate to the master that machine i is responding to a coordination step, which is used to update the upper bound (Line 14).

Notice that the major difference between this algorithm with respect to Algorithm 2 and the incremental (proximal) bundle algorithm [38], which both require the knowledge of a bound on Lipschitz constants: here, we just use the estimation of line 16 to guarantee that the bound L used in the test is always greater than all the computed subgradients.

We note that, as usual for level bundle methods, the sequence of the optimality gaps Δ_k is non-increasing by definition. We will further count with ℓ the number of times in which Δ_k decreases *sufficiently*: more precisely, ℓ is increased whenever line 24 is accessed, and $k(\ell)$ denotes the corresponding iteration index. As in the previous section, we call $k(\ell)$ a critical iteration and we consider K^ℓ the set of iterates between two consecutive critical iterates (recall (13) and Fig 2).

4.2. **Convergence analysis.** This section analyzes the convergence of the asynchronous level bundle algorithm described in Algorithm 3. As previously, we have by definition of critical iterates the chain of inequalities (12) and we rely on Lemma 3. The scheme of the convergence proof consists in showing that there exist infinitely many critical iterations. Note though that between two critical steps, we can have several coordination steps: the next lemma shows that two coordination iterates cannot be arbitrary close.

Lemma 4 (Between two coordination iterates). *For a given ℓ and two coordinate iterates $\bar{x}_{k_1}$ and $\bar{x}_{k_2}$ (with $\bar{x}_{k_1} \neq \bar{x}_{k_2}$) and $k_1 < k_2 \in K^\ell$, there holds*

$$\|\bar{x}_{k_1} - \bar{x}_{k_2}\| \geq \alpha \frac{\Delta_{k_2}}{L}. \tag{14}$$

Proof. At the second coordinate iterate $\bar{x}_{k_2} \in \mathbf{X}^{\mathrm{d}}_{k_2-1}$, all the oracles have responded at least one time and all of them have been evaluated at $\bar{x}_{k_1}$. Since the set of constraints of (8) keeps growing as k increase within K^ℓ, we have that $\bar{x}_{k_2}$ satisfies the m constraints generated by linearizations at $\bar{x}_{k_1}$. Summing these m linearizations and using that $\bar{x}_{k_2} \in \mathbf{X}^{\mathrm{d}}_{k_2-1}$ (see Eq. (7)) gives

$$f(\bar{x}_{k_1}) + \langle g_{k_1}, \bar{x}_{k_2} - \bar{x}_{k_1}\rangle = \sum_{i=1}^{m} \left(f^i(\bar{x}_{k_1}) + \langle g^i_{k_1}, \bar{x}_{k_2} - \bar{x}_{k_1}\rangle \right) \leq \sum_{i=1}^{m} \check{f}^i_{k_2-1}(\bar{x}_{k_2}) \leq f^{\mathrm{lev}}_{k_2-1}$$

This yields $-\|g_{k_1}\|\|\bar{x}_{k_2} - \bar{x}_{k_1}\| \leq f^{\mathrm{lev}}_{k_2-1} - f(\bar{x}_{k_1})$ so that $\|\bar{x}_{k_2} - \bar{x}_{k_1}\| \geq (f(\bar{x}_{k_1}) - f^{\mathrm{lev}}_{k_2-1})/\|g_{k_1}\|$.

The value $f(\bar{x}_{k_1})$ was fully computed before the next coordinate iterate, so before k_2. It is then used to update the upper bound, we have $f(\bar{x}_{k_1}) \geq f^{\mathrm{up}}_{k_2}$. This gives $f(\bar{x}_{k_1}) - f^{\mathrm{lev}}_{k_2-1} \geq f^{\mathrm{up}}_{k_2} - (f^{\mathrm{up}}_{k_2-1} - \alpha\Delta_{k_2-1}) \geq \alpha\Delta_{k_2-1} \geq \alpha\Delta_{k_2}$ where we used that (Δ_k) is non-increasing by construction. Finally, using the bound $\|g_{k_1}\| \leq L$ as provided by the algorithm at line 16 completes the proof. □

Theorem 2 (Convergence). *Assume that X is a convex compact set and that* (9) *holds. Let* $\mathrm{tol}_\Delta = 0$ *in Algorithm 3, then the sequence of gaps vanishes,* $\lim_k \Delta_k = 0$, *and the sequence of coordination iterates is a minimizing sequence for* (1), $\lim_k f(\bar{x}_k) = f_\star$. *For a strictly positive tolerance* $\mathrm{tol}_\Delta > 0$, *the algorithm terminates after finitely many steps with an approximate solution.*

Proof. The convergence $\Delta_k \to 0$ is given by (12), as soon as the counter ℓ increases indefinitely. Thus, we need to prove that there are infinitely many critical iterations. We obtain this by showing that, for any ℓ, the set K^ℓ is finite;

Algorithm 3 Asynchronous level bundle method by scarce coordination

```
1: Choose $x_1 \in X$ and set $\bar{x}_1 = \hat{x}_1 = x_1$
2: Choose $\mathrm{tol}_\Delta \geq 0$, $\alpha \in (0,1)$, bounds $f_1^{\mathrm{up}} > f_\star + \mathrm{tol}_\Delta$ and $f_1^{\mathrm{low}} \leq f_\star$, and a constant $L > 0$
3: Set $\hat{\Delta}_k = \infty$, $\mathtt{rr} = m$, $J_0^i = \emptyset$ for all $i = 1, .., m$, $\bar{f} = 0 \in \mathbb{R}$,and $\bar{g} = 0 \in \mathbb{R}^n$
4: Set to_coordinate[$i$] = False and coordinating[$i$] = True for all $i = 1, .., m$
5: Send $x_1$ to the $m$ oracles
6: for $k = 1, 2, \ldots$ do
                                                        ▷ Step 1: receive information from an oracle
7:     Receive from oracle $i$ the oracle information on $f^i$ at a previous iterate $x^{k'}$
8:     Store $(f^i(x_{k'}), g^i_{k'})$, and set $J_k^i = J_{k-1}^i \cup \{k'\}$
9:     if coordinating[$i$] = True then
10:        $\mathtt{rr} \leftarrow \mathtt{rr} - 1$ and coordinating[$i$] ← False
11:        Update $\bar{f} \leftarrow \bar{f} + f^i(x_{k'})$ and $\bar{g} \leftarrow \bar{g} + g^i_{k'}$
12:        if $\mathtt{rr} = 0$ then                                  ▷ Full information at point $\bar{x}_k$
13:            if $\bar{f} < f_k^{\mathrm{up}}$ then
14:                Update $f_k^{\mathrm{up}} = \bar{f}$ and $\bar{x}_{\mathtt{best}} = \bar{x}_k$          ▷ update upper bound
15:            end if
16:            Update $L \leftarrow \max\{L, \|\bar{g}\|\}$
17:        end if
18:    end if
                                                ▷ Step 2: test optimality and sufficient decrease
19:    Set $\Delta_k = f_k^{\mathrm{up}} - f_k^{\mathrm{low}}$
20:    if $\Delta_k \leq \mathrm{tol}_\Delta$ then
21:        Return $\bar{x}_{\mathtt{best}}$ and $f_k^{\mathrm{up}}$
22:    end if
23:    if $\Delta_k \leq \alpha\hat{\Delta}$ then                                         ▷ Critical Step
24:        Set $\hat{x}_k = \bar{x}_{\mathtt{best}}$ and $\hat{\Delta} = \Delta_k$. Possibly reduce index sets $J_k^j$ $(j = 1, \ldots, m)$
25:    end if
                                                            ▷ Step 3: compute next iterate
26:    Set $f_k^{\mathrm{lev}} = f_k^{\mathrm{up}} - \alpha\Delta_k$. Run a quadratic solver on problem (8)
27:    if (8) is feasible then
28:        Get new iterate $x_{k+1} \in \mathbb{X}_k^{\mathrm{d}}$. Update $f_{k+1}^{\mathrm{low}} = f_k^{\mathrm{low}}$, $f_{k+1}^{\mathrm{up}} = f_k^{\mathrm{up}}$
29:    else
30:        Set $f_k^{\mathrm{low}} = f_k^{\mathrm{lev}}$ and go to Step 2                       ▷ update lower bound
31:    end if
32:    if $\mathtt{rr} = 0$ and $\|x_{k+1} - x_k\| < \alpha\frac{\Delta_k}{L}$ then
33:        Set $\bar{x}_{k+1} = x_{k+1}$ and to_coordinate[$j$] = True $(j = 1, .., m)$       ▷ Coordination Step
34:        Reset $\mathtt{rr} = m$, $\bar{f} = 0$, $\bar{g} = 0$
35:    else
36:        Set $\bar{x}_{k+1} = \bar{x}_k$
37:    end if
                                                ▷ Step 4: send back information to the oracle
38:    if to_coordinate[$i$] = True then
39:        Send $\bar{x}_{k+1}$ to machine $i$.
40:        Set to_coordinate[$i$] = False and coordinating[$i$] = True
41:    else
42:        Send $x_{k+1}$ to machine $i$
43:    end if
44:    Set $\hat{x}_{k+1} = \hat{x}_k$
45: end for
```

for this, suppose that $\Delta_k > \Delta > 0$ for all $k \in K^\ell$. We proceed in two steps, showing that (i) the number of coordination steps in K^ℓ is finite; and (ii) the number of asynchronous iterations between two consecutive coordination steps is finite as well.

Part (i). Define $(\bar{x}_s)$ the sequence of coordination steps in K^ℓ. By Lemma 4, we obtain that for any $s < s'$

$$\|\bar{x}_s - \bar{x}_{s'}\| \geq \alpha\frac{\Delta_{s'}}{\Lambda} \geq \frac{\Delta}{\Lambda}.$$

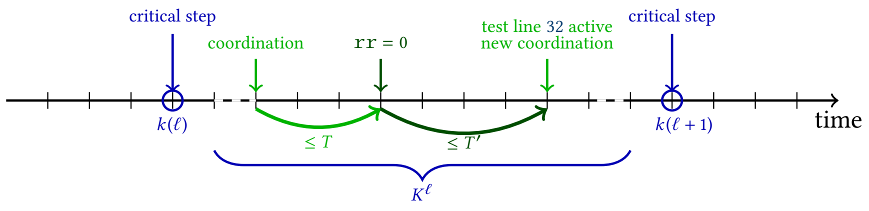

Figure 3. Notations for the proof

If there was an infinite number of coordination steps inside K^ℓ, the compactness of X would allow us to extract a converging subsequence, and this would contradict the above inequality. The number of coordination steps inside K^ℓ is thus finite.

Part (ii). We turn to the number of asynchronous iterations between two consecutive coordination iterations, that is the number of iterations before the test of line 32 is active. This part is illustrated by the green arrows in Fig. 3.

Since all the oracles are responsive, there is a finite number of iterations between two updates of any oracle: as a consequence, at a given iteration k, there exists a T (the dependence on k is dropped for simplicity) such that all oracles will exchange at least twice in $[k, k+T]$; in other words, the first part of the test $\texttt{rr} = 0$ will be verified within a finite number T of iterations.

Now, let us show by contradiction that the second part of the test, $\|x_{k+1} - x_k\| < \alpha\Delta_k/L$, will be verified after a finite number of iterations. As in the proof of Theorem 1, we have from Lemma 3 that $(\mathbb{X}^{\mathrm{d}}_k)$ is a nested non-increasing sequence of non-empty compact convex sets for $k \in K^\ell$. If there was an infinite number of asynchronous iterations before the test is verified, the sequence $\mathbb{X}^{\mathrm{d}}_k$ would converge to an non-empty $\mathbb{X}^{\mathrm{d}}$ in the sense of the Painlevé-Kuratowski (see [28, Chapter 4, Sec B] and [33, 3]). As a consequence,

$$x_{k+1} = \arg\min_{x \in \mathbb{X}^{\mathrm{d}}_k} \frac{1}{2}\|x - \hat{x}\|^2 \longrightarrow P_{\mathbb{X}^{\mathrm{d}}}(\hat{x}) = \arg\min_{x \in \mathbb{X}^{\mathrm{d}}} \frac{1}{2}\|x - \hat{x}\|^2.$$

As (x_k) converges, for any $\varepsilon > 0$, there exists T' such that for any $k \geq T'$, $\|x_{k+1} - x_k\| \leq \varepsilon$. Taking $\varepsilon = \alpha\Delta/(2L)$ with $\Delta_k > \Delta > 0$, the second part of the test is verified after T' iterations which contradicts the infinite number of asynchronous iterations before the test is verified. Thus, there are at most $T + T'$ iterations between two coordination steps.

Combining (i) and (ii), we can then conclude that the algorithm performs only finitely many iterations between two consecutive critical iterations. This in turn shows that there are infinitely many critical iterations, and thus we get convergence using (12). Similarly, for a strictly positive tolerance, there are a finite number of critical steps, and thus a finite number of iterations. The result on the convergence of (f^{up}_k) follows from exactly the same arguments as in the end of the proof of Theorem 1. □

5. Inexact oracles within asynchronous methods

Many applications of optimization to real-life problems lead to objective functions that are assessed through "noisy" oracles, where only some approximations to the function and/or subgradient values are available; see e.g. the recent review [9]. This is the typical case in Lagrangian relaxation of (possibly mixed-integer) optimization problems, in stochastic/robust programming, where the oracles perform some numerical procedure to evaluate functions and subgradients, such as solving optimization subproblems, multidimensional integrations, or simulations.

Level bundle methods are well-known to be sturdy to deal with such inexact oracles; see [9, 8, 36]. In particular, when the feasible set X is compact no special treatment is necessary to handle inexactness in standard level methods [9, Sec. 3]. As we show below, this is also the case for our asynchronous level bundle variants, more precisely: (i) the asynchronous algorithms converge to inexact solutions when used with oracles with bounded error (Section 5.1), and (ii) with a slight modification of the upper bounds, they can converge to exact solutions when used with lower oracles with vanishing error (Section 5.2).

5.1. **Inexact oracles with error bounds.** We assume to have an approximate oracle for f^i delivering, for each given $x \in X$, an inexact linearization on f, namely $(f_x^i, g_x^i) \in \mathbb{R} \times \mathbb{R}^n$ such that

$$\begin{cases} f_x^i = f^i(x) - \eta_x^{v,i} \\ g_x^i \in \mathbb{R}^n \quad \text{such that} \quad f^i(\cdot) \geq f_x^i + \langle g_x^i, \cdot - x\rangle - \eta_x^{s,i} \\ \text{with} \;\; \eta_x^{v,i} \leq \frac{\eta^v}{m} \;\; \text{and} \;\; \eta_x^{s,i} \leq \frac{\eta^s}{m} \;\; \text{for all } x \in \mathbb{R}^n. \end{cases} \tag{15}$$

The subscripts v and s on the oracle errors make the distinction between function *value* and *subgradient* errors. Note that oracles can overestimate function values, as $\eta_x^{v,i}$ can be negative. In fact, both the errors $\eta_x^{v,i}$ and $\eta_x^{s,i}$ can be negative but not simultaneously because they satisfy[4] $\eta_x^{v,i} + \eta_x^{v,i} \geq 0$. Bounds $\eta^v, \eta^s \geq 0$ on the errors should exist but are possibly unknown. When $\eta^s = 0$, we have the so-called lower oracles returning lower linearizations: $f^i(x) - \eta_x^{v,i} = f_x^i \leqslant f^i(x)$ and $f^i(\cdot) \geqslant f_x^i + \langle g_x^i, \cdot - x\rangle$. The exact oracle corresponds to taking $\eta^s = \eta^v = 0$.

Our asynchronous algorithms do not need to be changed to handle inexactness: the information provided by these inexact oracles is used in the same way as done previously where the oracles were exact. Indeed, we can define the inexact disaggregated cutting-plane model as

$$\check{f}_k^i(x) := \max_{j \in J_k^i} \{f_{x_j}^i + \langle g_{x_j}^i, x - x_j\rangle\}$$

and the inexact level set $\mathbb{X}_k^{\mathrm{d}}$ as in (7) (but with the inexact model). We then have the easy following result showing that f_k^{low} is still relevant.

Lemma 5 (Inexact lower bound)**.** *The update of f_k^{low} in Algorithms 2 and 3 guarantees that it is an inexact lower bound, in the sense that*

$$f_k^{\mathrm{low}} \;\leq\; f_\star + \eta^s \qquad \textit{for all } k. \tag{16}$$

In particular, if the oracle is a lower oracle ($\eta^s = 0$) then the algorithms ensure that f_k^{low} is a valid lower bound in every iteration k.

Proof. By summing the m inequalities $f_{x_j}^i + \langle g_{x_j}^i, x - x_j\rangle \leq f^i(x) + \eta_x^{s,i}$, we have that the inexact cutting-plane model satisfies

$$\sum_{i=1}^m \check{f}_k^i(x) \leq f(x) + \eta^s \quad \forall\, x \in X\,.$$

We deduce that

$$\mathbb{X}_k^{\mathrm{d}} = \left\{ x \in X : \sum_{i=1}^m \check{f}_k(x) \leq f_k^{\mathrm{lev}} \right\} \quad \supset \quad \left\{ x \in X : f(x) \leq f_k^{\mathrm{lev}} - \eta^s \right\}.$$

Therefore, if $\mathbb{X}_k^{\mathrm{d}} = \emptyset$ then the right-hand set is empty as well, which means $f_k^{\mathrm{lev}} \leq f_\star + \eta^s$. Thus the definition of f_k^{low} in Algorithms 3 and 2 give indeed an inexact lower bound. □

Similarly, the next lemma shows that f_k^{up} is relevant, up the oracle error.

Lemma 6 (Inexact upper bound)**.** *The definitions of f_k^{up} in Algorithms 2 and 3 guarantee that it is an inexact upper bound; more precisely, at iteration k*

$$f_k^{\mathrm{up}} \;\geq\; f(\bar{x}_{\mathrm{best}}) - \eta^v \;\geq\; f_\star - \eta^v. \tag{17}$$

Proof. We first consider Algorithm 2. As (x_{k+1}, r_{k+1}) solves the inexact version of (8) (where the exact linearizations are replaced by the inexact ones), we get[5]

$$f_{x_{\mathrm{a}(i)}}^i + \langle g_{\mathrm{a}(i)}^i, x_{k+1} - x_{\mathrm{a}(i)}\rangle + \bar{\Lambda}^i \|x_{k+1} - x_{\mathrm{a}(i)}\| \leq r^i + \bar{\Lambda}^i \|x_{k+1} - x_{\mathrm{a}(i)}\|.$$

The oracle's assumptions give $f_{x_{\mathrm{a}(i)}}^i \geq f^i(x_{\mathrm{a}(i)}) - \frac{\eta^v}{m}$. Consequently,

$$f^i(x_{\mathrm{a}(i)}) + \bar{\Lambda}^i \|x_{k+1} - x_{\mathrm{a}(i)}\| \leq r^i + \bar{\Lambda}^i \|x_{k+1} - x_{\mathrm{a}(i)}\| - \langle g_{\mathrm{a}(i)}^i, x_{k+1} - x_{\mathrm{a}(i)}\rangle + \frac{\eta^v}{m}.$$

[4]By substituting $f_x^i = f^i(x) - \eta_x^{v,i}$ in the inequality $f^i(\cdot) \geq f_x^i + \langle g_x^i, \cdot - x\rangle - \eta_x^{s,i}$ and evaluating at x, we get that $f(x) \geq f(x) - \eta_x^{v,i} - \eta_x^{s,i}$. This shows that $\eta_x^{v,i} + \eta_x^{s,i} \geq 0$ and in fact $g_x^i \in \partial_{(\eta_x^{v,i} + \eta_x^{v,i})} f^i(x)$.

[5]As in Section 3, $\mathrm{a}(i)$ is the iteration index of the anterior information provided by oracle i; see Algorithm 2.

Assumption (9) yields $f^i(x_{a(i)}) + \bar{\Lambda}^i \|x_{k+1} - x_{a(i)}\| \geq f^i(x_{k+1})$. By combining these last two inequalities and summing up over $i = 1, \dots, m$ we obtain

$$f(x_{k+1}) \leq f_k^{\text{lev}} + \sum_{i=1}^{m} [\bar{\Lambda}^i \|x_{k+1} - x_{a(i)}\| - \langle g^i_{a(i)}, x_{k+1} - x_{a(i)} \rangle] + \eta^v. \tag{18}$$

Therefore, the rule on line 22 of Algorithm 2 yields (17) by taking $\bar{x}_{\texttt{best}} = x_{k+1}$.

The case of Algorithm 3 is straightforward as

$$f_k^{\text{up}} = \min_{j=1,\dots,k} f_{x_j} \geq \min_{j=1,\dots,k} f(x_j) - \eta^v$$

which also gives (17) by definition of $\bar{x}_{\texttt{best}}$ at iteration k. □

The previous two lemmas thus show that the inexact upper and lower bounds appearing in the asynchronous algorithms with inexact oracles satisfy

$$f_k^{\text{low}} - \eta^s \;\leq\; f_\star \;\leq\; f_k^{\text{up}} + \eta^v \qquad \text{for all } k.$$

We now formalize a theorem to conclude that, as in the standard case, inexactness can be readily handled by our asynchronous level methods.

Theorem 3. *The convergence results for Algorithms 2 and 3, namely Theorems 1 and 2 still hold, up to the oracle error, when inexact oracles (15) are used. More precisely, we obtain* $\text{tol}_\Delta + \eta^v + \eta^s$*-solution of (1):*

- *when* $\text{tol}_\Delta = 0$, *we have* $\lim_k f_k^{\text{up}} \leq f_\star + \eta^v + \eta^s$
- *when* $\text{tol}_\Delta > 0$, *we have* $f_\star \leq f(\bar{x}_{\texttt{best}}) \leq f_\star + \text{tol}_\Delta + \eta^v + \eta^s$

Proof. The proofs are valid verbatim until the end about the convergence of (f_k^{up}). In the inexact case, we combine (17) and (16) to write

$$f_\star - \eta^v \leq f(\bar{x}_{\texttt{best}}) - \eta^v \leq f_k^{\text{up}} = f_k^{\text{low}} + \Delta_k \leq f_\star + \eta^s + \Delta_k.$$

Passing to the limit, this ends the proof. □

As previously mentioned, no special treatment is necessary to handle inexactness in the proposed asynchronous level methods. The obtained solution is optimal within the precision $\text{tol}_\Delta + \eta^v + \eta^s$, the given tolerance plus the (possibly unknown) oracle error bounds. If we target obtaining tol_Δ-solutions, more assumptions on the inexact oracles need to come into play, and minor changes in the algorithms must be made, as explained in the next section.

5.2. **Lower oracles with vanishing error bound.** We consider further the case of (15) with $\eta^s = 0$ and controllable η^v, for which the asynchronous algorithms converge to an optimal solution if we slightly change the upper bound.

We assume here that the error bound η^v is known and controllable in the sense that the algorithm can decrease or increase η^v along the iterative process. Thus we consider the following special case of (15): given a trial point x and an error bound η^v as inputs, the oracle provides

$$\begin{cases} f_x^i = f^i(x) - \eta_x^{v,i} \\ g_x^i \in \mathbb{R}^n \quad \text{such that} \quad f^i(\cdot) \geq f_x^i + \langle g_x^i, \cdot - x \rangle \\ \text{with} \quad \eta_x^{v,i} \leq \frac{\eta^v}{m} \quad \text{for all } x \in \mathbb{R}^n. \end{cases} \tag{19}$$

The fact that we control η^v allows us to incorporate the oracle error in the algorithm, which eventually will yield convergence to optimality. The fact that $\eta^s = 0$ gives that f_k^{low} is always a lower bound (recall Lemma 5).

At iteration k we index the error bound with k and we drive η_k^v below a fraction of the gap at the preceeding decrease. More precisely, we consider the following control: there exists $\kappa \in (0,1)$ such that the m oracles satisfy (19) with

$$0 \leq \eta_k^v \leq \kappa \Delta_{k(\ell)}, \quad \text{for all } k \in K^\ell \tag{20}$$

where $k(\ell)$ corresponds to the last critical iteration (the iteration yielding enough decrease on line 12 of Algorithm 2 or on line 24 of Algorithm 3). This control on η_k^v is standard in methods with on-demand accuracy [8].

Theorem 4. *Consider Algorithm 2 with inexact oracles (19). If the oracles error can be controlled by (20) with $\kappa \in (0, \alpha^2/2)$, and if the update of f_{k+1}^{up} in line 22 is replaced with*

$$\min\left\{ f_k^{\mathrm{up}}, \left(f_k^{\mathrm{lev}} + \sum_{j=1}^{m} \left(\bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| - \langle g_{\mathrm{a}(j)}^j, x_{k+1} - x_{\mathrm{a}(j)} \rangle \right) + \eta_k^v \right) \right\},$$

then the convergence result of Theorem 1 still holds.

Proof. First note that the update rule of f_{k+1}^{up} provides valid upper bounds: just take Eq. (18) from the proof of Lemma 6 with error bound η^v therein replaced with the new (and known) bound η_k^v. The definitions of f_k^{lev}, f_k^{up}, and assumptions (9) give

$$\begin{aligned} f_{k+1}^{\mathrm{up}} &\leq f_k^{\mathrm{lev}} + \sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| - \langle g_{\mathrm{a}(j)}^j, x_{k+1} - x_{\mathrm{a}(j)} \rangle + \eta_k^v \\ &\leq f_k^{\mathrm{up}} - \alpha\Delta_k + 2\sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| + \eta_k^v, \end{aligned}$$

showing that $\alpha\Delta_k \leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + 2\sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| + \eta_k^v$. As in the proof of Theorem 1, one can show by contradiction that there are infinitely many critical steps and thus convergence of Δ_k to 0. Indeed, suppose that there are a finite number of critical steps so that $\Delta_k \geq \Delta > 0$ and $\hat{x}_k$ is fixed after some k, then the nested non-increasing character of sequence $(\mathbb{X}_k^{\mathrm{d}})$ implies that (x_k) converges. In our totally asynchronous set-up (all oracles respond in a finite time horizon), we have, for all j, that $\mathrm{a}(j)$ diverges with k. As (x_k) converges, it is a Cauchy sequence which implies that $x_{k+1} - x_{\mathrm{a}(j)}$ vanishes for all j. Therefore

$$\|x_{k+1} - x_{\mathrm{a}(j)}\| \leq \varepsilon = \frac{\kappa}{2m \max_i \bar{\Lambda}_i} \Delta_{k(\ell)} \quad \text{for } k \text{ large.}$$

Since $\Delta_k > \alpha\Delta_{k(\ell)}$ for all $k \in K^\ell$, this yields

$$\begin{aligned} \alpha^2\Delta_{k(\ell)} < \alpha\Delta_k &\leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + 2\sum_{j=1}^{m} \bar{\Lambda}^j \|x_{k+1} - x_{\mathrm{a}(j)}\| + \eta_k^v \\ &\leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + 2\sum_{j=1}^{m} \bar{\Lambda}^j \frac{\kappa}{2m \max_i \bar{\Lambda}_i} \Delta_{k(\ell)} + \eta_k^v \leq f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} + 2\kappa\Delta_{k(\ell)}, \end{aligned}$$

showing that $f_k^{\mathrm{up}} - f_{k+1}^{\mathrm{up}} > \left(\alpha^2 - 2\kappa\right)\Delta_{k(\ell)} \geq \left(\alpha^2 - 2\kappa\right)\Delta > 0$ for all k large enough. This contradicts the fact that the bounded and non-increasing sequence (f_k^{up}) is convergent. Finally, as $\Delta_k \to 0$ and we have upper and lower bounds, the convergence holds as stated in Theorem 1. □

Theorem 5. *Consider Algorithm 3 with inexact oracles (19). If the oracles error are controlled by (20) with $\kappa \in (0, \alpha^2)$, then the convergence result of Theorem 2 still holds when the update of the upper bound of line 11 is replaced with*

$$\bar{f} \leftarrow \bar{f} + f_{x_{k'}}^i + \frac{\eta_{k'}^v}{m}. \tag{21}$$

Proof. Since the oracle noise is added in (21), f_k^{up} is a valid upper bound. Hence by construction, (f_k^{up}) and Δ_k are non-increasing and we have

$$f_k^{\mathrm{low}} \leq f_\star \leq f(\bar{x}_{\mathrm{best}}) \leq f_k^{\mathrm{up}} \quad \text{for all } k = 1, 2 \ldots$$

Hence, in order to conclude the proof we only need to show that $\lim_k \Delta_k = 0$. The proof of Theorem 2 still holds but with a slightly modified version of Lemma 4 that we show below.

Let $\bar{x}_{k_1}$ and $\bar{x}_{k_2}$ (with $k_1 < k_2$ and $k_1, k_2 \in K^\ell$) be two consecutive coordinate iterates. Since $\bar{x}_{k_2}$ satisfies the m constraints of linearizations at $\bar{x}_{k_1}$, we obtain

$$f_{\bar{x}_{k_1}} + \langle g_{k_1}, \bar{x}_{k_2} - \bar{x}_{k_1} \rangle = \sum_{i=1}^{m} \left(f_{\bar{x}_{k_1}}^i + \langle g_{k_1}^i, \bar{x}_{k_2} - \bar{x}_{k_1} \rangle \right) \leq f_{k_2-1}^{\mathrm{lev}} \leq f_{k_1}^{\mathrm{lev}},$$

which implies, by Cauchy-Schwarz and boundedness of subgradients,

$$L\|\bar{x}_{k_2} - \bar{x}_{k_1}\| \geq \|g_{k_1}\|\|\bar{x}_{k_2} - \bar{x}_{k_1}\| \geq f_{\bar{x}_{k_1}} - f_{k_1}^{\mathrm{lev}} \geq f_{k_1}^{\mathrm{up}} - \eta_{k_1}^v - f_{k_1}^{\mathrm{lev}} = \alpha\Delta_{k_1} - \eta_{k_1}^v.$$

The last inequality above is due to the new rule (21) to update the upper bound. Noticing that $\Delta_{k_1} > \alpha\Delta_{k(\ell)}$ and $\eta^v_{k_1} \leq \kappa\Delta_{k(\ell)}$ by rule (20), we get

$$\|\bar{x}_{k_2} - \bar{x}_{k_1}\| \geq \frac{(\alpha^2 - \kappa)\Delta_{k(\ell)}}{L} > 0$$

by the choice of κ. As k_1 and k_2 are arbitrary indexes in K^ℓ and X is a compact set, the above inequality can hold only for finitely many steps. This shows that each index set K^ℓ is finite and, therefore, that the number of critical iterations ℓ grows indefinitely if $\mathrm{tol}_\Delta = 0$. Hence, (12) gives $\lim_k \Delta_k = 0$. □

Thus we show that the two asynchronous algorithms also share the well-known robustness of synchronous level bundle methods for dealing with inexact oracles with on-demand accuracy [8, 36].

6. Numerical illustrations

We have run preliminary numerical experiments and comparisons between the level algorithms discussed in the paper. These experiments are limited and just illustrate the effectiveness and the potential interest of asynchronicity in bundle methods. A thorough numerical assessing of the interests and limits of the algorithms would deserve a whole study of itself to take into account the various biases from the inputs and the computed system (in particular, the variance of the solution times of the numerical subroutines and the communications between machines). Here we consider a basic implementation of the (distributed) algorithms, a trivial set-up and computing system, and a simple randomly-generated problem. Extensive numerical experiments are beyond the scope of this paper.

6.1. **Experimental set-up.**

Problem. We consider the instance of problem (1) where each function f^i is the optimal value of the following mixed-integer linear program (MILP): for $x \in \mathbb{R}^n$,

$$f^i(x) = \begin{cases} \max & \pi_i\left(\langle c^i, p\rangle - \langle x, A^i p\rangle\right) \\ \text{s.t.} & \|p\|_\infty \leq B,\ \ G^i p \leq h^i \\ & p \in \mathbb{N}^{m_1} \times \mathbb{R}^{m_2} \end{cases} \tag{22}$$

where c^i, A^i, G^i, h^i are random vectors/matrices with suitable sizes (we denote by n_c the number of affine constraints of the problem *i.e.* the number of lines of G^i, kept constant among oracles). Such oracles appear when solving Lagrangian relaxations of difficult mixed-integer optimization problems. The oracle i solves, for given a point x, the above MILP to get an optimal point $p^\star$, which gives

$$f^i(x) = \pi_i(\langle c^i, p^\star\rangle - \langle x, A^i p^\star\rangle) \quad \text{and} \quad g^i = -\pi_i A^i p^\star \in \partial f^i(x).$$

Note that we have a bound on the Lipschitz constant for f^i by

$$\|\pi_i A^i p\|_2^2 \ \leq\ \pi_i^2\Big(\sum_{j=1}^m \|A^i_j\|_1^2\Big)\|p\|_\infty^2 \ \leq\ \pi_i^2\Big(\sum_{j=1}^m \|A^i_j\|_1^2\Big)B^2 \ =:\ \bar{\Lambda}_i. \tag{23}$$

Tested algorithms. We compare the four following algorithms: the first is the existing one; the three others are introduced in this paper.

- **S** Synchronous level bundle algorithm (Algorithm 1)
- **SD** Synchronous Disaggregated algorithm (Algorithm 1 using (8))
- **AU** Asynchronous algorithm by Upper-bounds (Algorithm 2)
- **AC** Asynchronous algorithm by Coordination (Algorithm 3)

The four algorithms use the same initialization and global parameters. The only exception is the level parameter α: we use the simple value $\alpha = 0.5$ except for **AU**, where we use $\alpha = 0.9$. This higher value allows us to compensate the rough upper-bounding (23) to get better levels f^{lev}_k.

Computing setup. All the code is written in Python 2.7.6 and run on a laptop with an Intel Core i7-5600U and 8GB of RAM. Each machine is assigned to a thread. MPI is used as a communication framework (more precisely the mpi4py implementation of OpenMPI 1.6.5). The MILPs (22) of the oracles and the quadratic problems at the master are computed using Gurobi 8.0.0.

Notice that for the standard algorithm **S**, the quadratic problem (3) uses the total function oracle while the disaggregated quadratic problem (8) uses all the oracles separately. In terms of distributed programming, the first one can be performed by map-reduce (with a sum operation in the reduce) while the second needs a separate gathering of the oracle results.

Instance generation. We consider $m=8$ machines/oracles on a problem size $n=20$. We generate moderately imbalanced oracles: six comparable oracles ($m_1=20$, $m_2=40$, $B=5$, $n_c=100$, and $\pi_i=1$) and two slightly bigger ($m_1=50$, $m_2=100$, $B=10$, $n_c=100$, and $\pi_1=\pi_2=0.1$). The matrices A^i, G^i are drawn independently with coefficients taken from the uniform distribution in $(-1,1)$, c^i is taken from the normal distribution with variance 100. Moreover h^i is taken from the uniform distribution in $[0.1,1.1)$, so that $p=0$ is feasible for all problems (22) and $f_0^{\text{low}}=0$ is a valid lower-bound on $f_\star$.

Experiments. With the above set-up, we run preliminary experiments. We observe a high variance of the solution times of Gurobi and the communication between machines: this strongly impacts the time for an oracle to respond, the order of oracles responses for asynchronous algorithms, and therefore the behaviour of the algorithms. Note also that while the asynchronous methods have the same parameters as the standard level bundle methods, the behavior of the algorithm (e.g. the coordination frequency) highly depends on the problem and computing system. A complete computational study is out of the scope of this paper; we focus here only on showing that using asynchronous methods can save time.

Thus, we generate, as described above, one problem instance for which the two bigger oracles (oracles 1 and 2) are computationally more expensive in practice. We consider five runs of the algorithms: the figures reported in next tables are the average (and the standard deviation) of the obtained results. We compare the algorithms, first for a coarse precision target (in Section 6.2), then for a finer precision (in Section 6.3). Finally, we investigate the case of inexact oracles (in Section 6.4).

6.2. **Experiments for coarse precision.**

In this part, we stop the algorithms as soon as $\Delta^k/f_\star < 10\%$ and we display in Table 1 the number of iterations, the total CPU time, and number of oracle calls to reach this criterion. These figures illustrates the interest of disaggregation and asynchronicity. Indeed, we first see that there is a real difference in computing time between the the usual level algorithm and the disagreggated ones proposed in this paper: 249s for the standard level bundle **S** vs 122s for its disaggregated counterpart **SD**, and as low as 53s for the best asynchronous method. We also see that the two asynchronous algorithms converge quickly compared to the synchronous ones: for example, **S** converges in 10 (synchronous) iterations which corresponds to 80 oracle calls whereas **AC** needs 192 oracle calls but its computing time is 5 times lower. We thus observe that synchronous methods are more reliable (less variance) and asynchronous ones may be faster (as they have a better use of wall clock time) even though they may compute more (they make more oracles calls).

Algo	# iters	f1	f2	f3	f4	f5	f6	f7	f8	time
S	10	10	10	10	10	10	10	10	10	249s
(Alg. 1)	= 80	±0	±0	±0	±0	±0	±0	±0	±0	± 4s
SD	8	8	8	8	8	8	8	8	8	122s
	= 64	±0	±0	±0	±0	±0	±0	±0	±0	± 6s
AU	232	14	18	31	33	33	32	31	32	78s
(Alg. 2)	±60	±6	±9	±8	±9	±9	±9	±9	±9	± 39s
AC	192	4	19	28	27	29	28	28	29	53s
(Alg. 3)	±50	±0	±7	±8	±6	±6	±7	±7	±7	± 29s

Table 1. Comparison of the four algorithms (in terms of number of iterations, number of oracles calls, and total computing time) in the case of low precision. We report the average and the standard deviation of the five results.

The two asynchronous algorithms reach the precision more quickly thanks to the asynchronous bundle information used to improve their lower-bounds, as showed in Figure 4. In this figure, we see that synchronicity provides tight

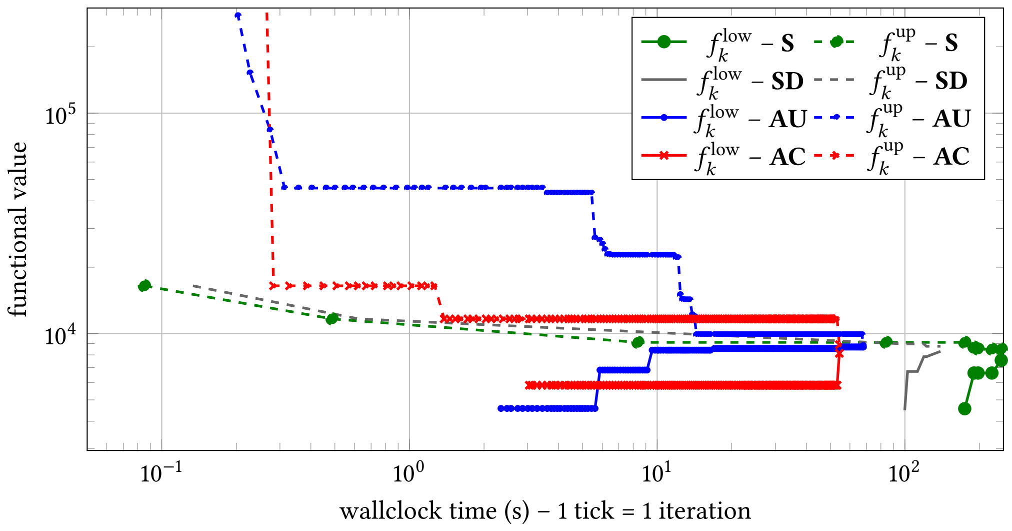

FIGURE 4. Evolution of f_k^{low} and f_k^{up} for on representative run of the algorithms. The two axis (functional values and iterations) are in log-scale.

upper bounds to **S** and **SD** but they need more time to get good lower bounds. The greater variety of cuts added to the disaggregated master QP (8) by asynchronous methods enable them to enjoy better lower bounds than their synchronous counterparts. To close the gap, the asynchrounous methods show different behaviors on upper-bounds. Indeed, we notice that the upper-bounds (23) used by **AU** are weak with respect to the empirical estimation of the associated Lipschitz constants observed from norms of computed subgradients in **AC**; see Figure 5. Thus, the two asynchronous algorithms **AU** and **AC** reach the prescribed coarse precision faster than the synchronous ones, with roughly the same time. However, the loose upper-bounds in **AU** make it less competitive as the precision becomes finer.

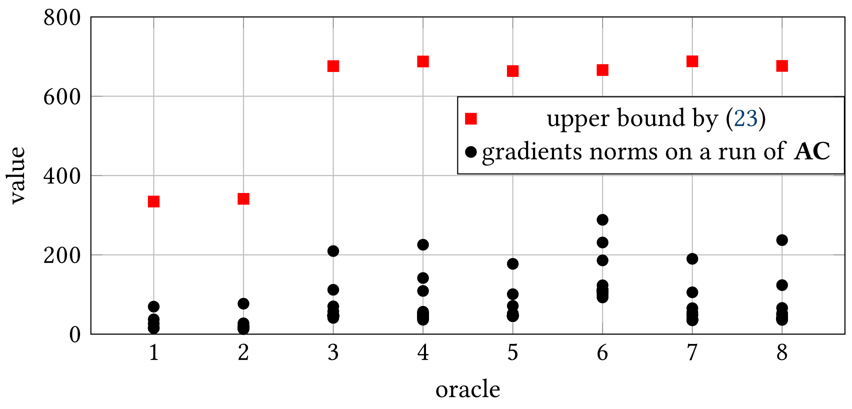

FIGURE 5. Estimation of the Lipschitz constants for the oracles: apriori upper bounds by (23), vs. observed lower bounds by norms of computed subgradient.

6.3. **Experiments with finer precision.** For our second experiments, we stop the algorithms whenever $\Delta^k/f_\star < 1\%$. In order to precise the reach of our methods, we focus here on our flagship asynchronous algorithm with coordination **AC** and compare with the synchronous baseline **S**. We notably illustrate the impact of the proposed coordination strategy by investigating two variants of **AC**: when the test of line 32 is 'on' or 'off', 'off': corresponding to the case where a coordination is triggered as soon as the previous one has been completed. In the following table, we again display the average on 5 runs as well as the standard deviation.

Algo	# iters	f1	f2	f3	f4	f5	f6	f7	f8	time
S	20	20	20	20	20	20	20	20	20	390s
(Alg. 1)	= 160	±0	±0	±0	±0	±0	±0	±0	±0	± 7s
AC	285	6	32	41	40	42	41	41	42	116s
(Alg. 3)	±57	±1	±10	±9	±7	±7	±8	±8	±8	± 37s
AC	294	5	32	42	42	45	41	43	44	128s
test off	±91	±1	±14	±12	±13	±13	±13	±13	±13	± 63s

Table 2. Comparison of the asynchronous algorithm **AC** with the baseline **S** (in terms of number of iterations, number of oracles calls, and total computing time) in the case of high precision. We illustrate the impact of the coordination step of line 32 by looking at **AC** when the test is turned 'off'. We report the average and standard deviation on 5 runs.

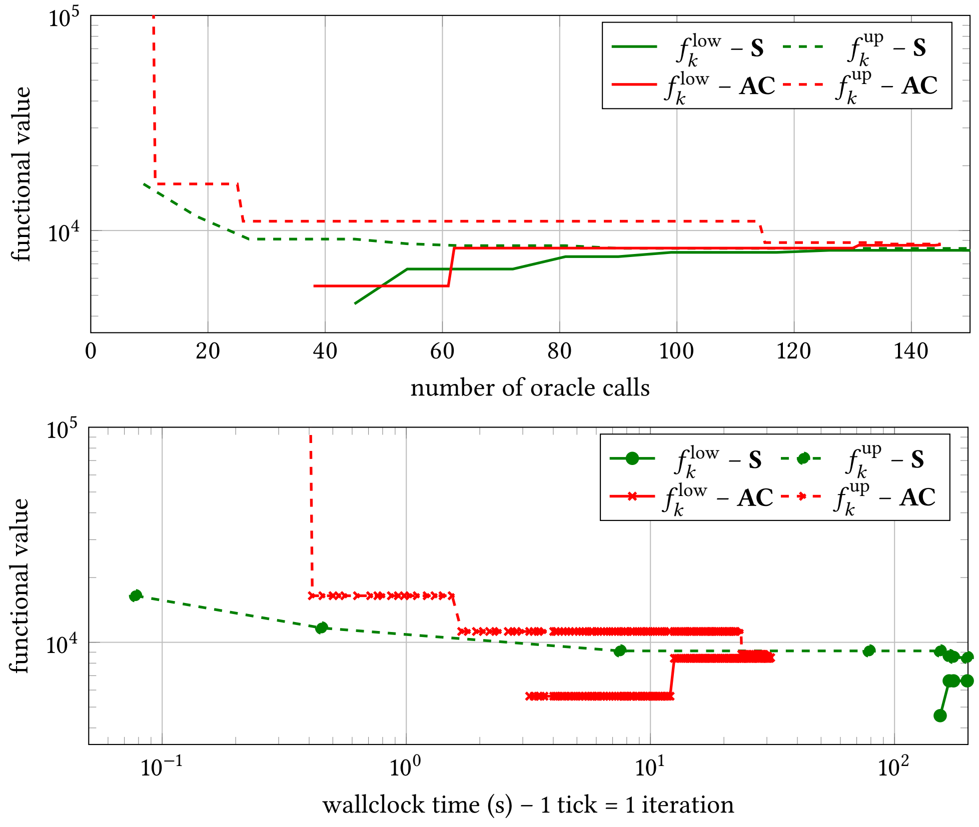

Figure 6. Evolution of f_k^{low} and f_k^{up} for **S** and **AC** for a representative run.

We notice that the asynchronous algorithms achieve a clear speedup compared to the synchronous bundle. This can be explained intuitively by the fact that the first two oracles are more time consuming than the other (as their associated subproblem is harder to solve) while they do not contribute proportionally more in the global model. The asynchronous algorithms thus achieve the sought precision after only 5 or 6 calls from oracle 1 and around 40 for the others while the synchronous one has to get 20 global calls.

In Fig. 6, we plot the values of f_k^{low} and f_k^{up} computed along the methods versus the number of oracle calls. We notice that while the synchronous method improves iteration by iteration (there are 8 calls per iteration), the asynchronous algorithm improves more scarcely but with more significant decreases. Due to the difference in terms of computational cost between the workers, one has to keep in mind that the wallclock time cost of a certain number of oracle calls is smaller in the asynchronous setup, which allows for faster convergence.

6.4. Experiments with inexact oracles.

In this section, we compare our asynchronous algorithms **AU** and **AC** with their on-demand accuracy counterparts from Section 5, named **AU/on-demand** and **AC/on-demand**.

We control the accuracy of the oracles by sending to the workers a target precision along with the trial point. More precisely, at iteration k, we send $\kappa\Delta^k / f_k^{\text{lev}}$ as a target (relative) precision, used by the worker as the precision-control parameter MIPGap of Gurobi. The parameter κ was chosen equal to 0.001 which, given the functional values, corresponds to a relative precision lowering from 10 in the first iterations to 10^{-3} in the final steps; compared to a fixed precision of 10^{-9} for **AU** and **AC**. We stop the algorithms whenever $\Delta^k / f_\star < 3\%$, corresponding to the intermediate precision compared to the two previous sections. The rest of the setup is exactly the same as in the previous section.

In Table 3, we display the average on 5 runs as well as the standard deviation. The use of inexact oracles seems to speed-up the convergence of **AU** and **AC** both in terms of wall-clock time and number of oracle calls (with a more even number of calls across the oracles); this could be due to the fact that larger gaps in the first iterations would make an exact oracle too expensive for the potential gain in the master bundle.

Algo	# iters	f1	f2	f3	f4	f5	f6	f7	f8	time
AU	308	14	24	32	40	51	47	49	50	132s
(Alg. 2)	±104	±6	±11	±15	±14	±15	±15	±15	±15	± 76s
AU	320	40	40	38	40	41	40	41	41	129s
on-demand	±73	±10	±9	±7	±9	±9	±9	±10	±9	±56s
AC	219	4	18	25	30	36	35	35	37	67s
(Alg. 3)	±46	±1	±6	±8	±6	±7	±7	±7	±8	± 26s
AC	97	12	12	11	12	13	12	12	13	14s
on-demand	±9	±1	±1	±2	±1	±1	±1	±1	±1	± 2s

TABLE 3. Comparison of the two asynchronous algorithms and their counterparts with on-demand accuracy (in terms of number of iterations, number of oracles calls, and total computing time). We report the average and the standard deviation on 5 runs.

Acknowledgements We are grateful to the two referees for their rich feedback on the initial version of our paper. We would like to acknowledge the partial financial support of PGMO (Gaspard Monge Program for Optimization and operations research) of the Hadamard Mathematics Foundation, through the project "Advanced nonsmooth optimization methods for stochastic programming".

References

[1] Arda Aytekin, Hamid Reza Feyzmahdavian, and Mikael Johansson, *Analysis and implementation of an asynchronous optimization algorithm for the parameter server*, arXiv preprint arXiv:1610.05507, (2016).

[2] Léonard Bacaud, Claude Lemaréchal, Arnaud Renaud, and Claudia Sagastizábal, *Bundle methods in stochastic optimal power management: A disaggregated approach using preconditioners*, Computational Optimization and Applications, 20 (2001), pp. 227–244.

[3] Nilson C. Bernardes, *On nested sequences of convex sets in Banach spaces*, Journal of Mathematical Analysis and Applications, 389 (2012), pp. 558 – 561.

[4] Dimitri P Bertsekas and John N Tsitsiklis, *Parallel and distributed computation: numerical methods*, vol. 23, Prentice hall Englewood Cliffs, NJ, 1989.

[5] Olivier Briant, Claude Lemaréchal, Ph Meurdesoif, Sophie Michel, Nancy Perrot, and François Vanderbeck, *Comparison of bundle and classical column generation*, Mathematical programming, 113 (2008), pp. 299–344.

[6] Sergio V. B. Bruno, Leonardo A. M. Moraes, and Welington de Oliveira, *Optimization techniques for the Brazilian natural gas network planning problem*, Energy Systems, 8 (2017), pp. 81–101.

[7] Welington de oliveira, *Target radius methods for nonsmooth convex optimization*, Operations Research Letters, 45 (2017), pp. 659 – 664.

[8] Welington de Oliveira and Claudia Sagastizábal, *Level bundle methods for oracles with on-demand accuracy*, Optimization Methods and Software, 29 (2014), pp. 1180–1209.

[9] Welington de Oliveira and Mikhail Solodov, *Bundle methods for inexact data*. tech. report, 2018.

[10] Louis Dubost, Robert Gonzalez, and Claude Lemaréchal, *A primal-proximal heuristic applied to the french unit-commitment problem*, Mathematical programming, 104 (2005), pp. 129–151.

[11] Frank Fischer and Christoph Helmberg, *A parallel bundle framework for asynchronous subspace optimization of nonsmooth convex functions*, SIAM Journal on Optimization, 24 (2014), pp. 795–822.

[12] Antonio Frangioni, *Standard bundle methods: Untrusted models and duality*, tech. report, Universita di Pisa, 2018.

[13] Antonio Frangioni and Enrico Gorgone, *Bundle methods for sum-functions with fieasyfi components: applications to multicommodity network design*, Mathematical Programming, 145 (2014), pp. 133–161.

[14] Arthur M Geoffrion, *Generalized benders decomposition*, Journal of optimization theory and applications, 10 (1972), pp. 237–260.

[15] Robert Hannah and Wotao Yin, *More iterations per second, same quality–why asynchronous algorithms may drastically outperform traditional ones*, arXiv preprint arXiv:1708.05136, (2017).

[16] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal, *Convex analysis and minimization algorithms*, vol. 305 and 306, Springer science & business media, 1993.

[17] Rie Johnson and Tong Zhang, *Accelerating stochastic gradient descent using predictive variance reduction*, in Advances in neural information processing systems, 2013, pp. 315–323.

[18] K. Kim, C. Petra, and V. Zavala, *An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming*, SIAM Journal on Optimization, 29 (2019), pp. 318–342.

[19] Krzysztof C. Kiwiel, *Proximal level bubdle methods for convex nondiferentiable optimization, saddle-point problems and variational inequalities*, Mathematical Programming, 69 (1995), pp. 89–109.

[20] Jakub Konecnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik, *Federated optimization: Distributed machine learning for on-device intelligence*, arXiv preprint arXiv:1610.02527, (2016).

[21] Claude Lemaréchal, *An extension of davidon methods to nondifferentiable problems*, Mathematical programming study, 3 (1975), pp. 95–109.

[22] Claude Lemaréchal, *Lagrangian relaxation*, in Computational combinatorial optimization, Springer, 2001, pp. 112–156.

[23] Claude Lemaréchal, Arkadi Nemirovskii, and Yurii Nesterov, *New variants of bundle methods*, Math. Program., 69 (1995), pp. 111–147.

[24] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael Jordan, Peter Richtarik, and Martin Takac, *Adding vs. averaging in distributed primal-dual optimization*, in International Conference on Machine Learning, 2015, pp. 1973–1982.

[25] Jérôme Malick, Welington de Oliveira, and Sofia Zaourar, *Uncontrolled inexact information within bundle methods*, EURO Journal on Computational Optimization, 5 (2017), pp. 5–29.

[26] Konstantin Mishchenko, Franck Iutzeler, Jerome Malick, and Massih-Reza Amini, *A delay-tolerant proximal-gradient algorithm for distributed learning*, in Proceedings of the 35th International Conference on Machine Learning, vol. 80 of Proceedings of Machine Learning Research, PMLR, 10–15 Jul 2018, pp. 3584–3592.

[27] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin, *Arock: an algorithmic framework for asynchronous parallel coordinate updates*, SIAM Journal on Scientific Computing, 38 (2016), pp. A2851–A2879.

[28] R. Tyrrell Rockafellar and Roger J.-B. Wets, *Variational Analysis*, Springer Verlag, Heidelberg, Berlin, New York, 1998.

[29] R Tyrrell Rockafellar and Roger J-B Wets, *Scenarios and policy aggregation in optimization under uncertainty*, Mathematics of operations research, 16 (1991), pp. 119–147.

[30] Claudia Sagastizábal, *Divide to conquer: decomposition methods for energy optimization*, Mathematical programming, 134 (2012), pp. 187–222.

[31] ———, *A VU-point of view of nonsmooth optimization*, Proc. Int. Cong. of Math, 3 (2018), p. 3785fi?!3806.

[32] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczyński, *Lectures on stochastic programming: modeling and theory*, SIAM, 2009.

[33] Vitold Smulian, *On the principle of inclusion in the space of the type* (b), Rec. Math. [Mat. Sbornik] N.S., 5(47) (1939), pp. 317–328.

[34] Tao Sun, Robert Hannah, and Wotao Yin, *Asynchronous coordinate descent under more realistic assumptions*, in Advances in Neural Information Processing Systems, 2017, pp. 6182–6190.

[35] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans, *Distributed asynchronous deterministic and stochastic gradient optimization algorithms*, IEEE transactions on automatic control, 31 (1986), pp. 803–812.

[36] Wim van Ackooij and Welington de Oliveira, *Level bundle methods for constrained convex optimization with various oracles*, Computation Optimization and Applications, 57 (2014), pp. 555–597.

[37] Wim van Ackooij and Jerome Malick, *Decomposition algorithm for large-scale two-stage unit-commitment*, Annals of Operations Research, 238 (2015), pp. 587–613.

[38] Antonio Frangioni Wim van Ackooij, *Incremental bundle methods using upper models*, Tech. Report 2357, Dipartimento di Informatica, TR . University of Pisa, 2016.

[39] Christian Wolf, Csaba I Fábián, Achim Koberstein, and Leena Suhl, *Applying oracles of on-demand accuracy in two-stage stochastic programming. A computational study*, European Journal of Operational Research, 239 (2014), pp. 437–448.

[40] Ruiliang Zhang and James Kwok, *Asynchronous distributed admm for consensus optimization*, in International Conference on Machine Learning, 2014, pp. 1701–1709.