Optimization and Engineering
https://doi.org/10.1007/s11081-023-09859-z

RESEARCH ARTICLE


# Implementation of an oracle-structured bundle method for distributed optimization

**Tetiana Parshakova**[1] · **Fangzhao Zhang**[2] · **Stephen Boyd**[2]

Received: 30 January 2023 / Revised: 20 July 2023 / Accepted: 2 October 2023


## Abstract

We consider the problem of minimizing a function that is a sum of convex agent functions plus a convex common public function that couples them. The agent functions can only be accessed via a subgradient oracle; the public function is assumed to be structured and expressible in a domain specific language (DSL) for convex optimization. We focus on the case when the evaluation of the agent oracles can require significant effort, which justifies the use of solution methods that carry out significant computation in each iteration. To solve this problem we integrate multiple known techniques (or adaptations of known techniques) for bundle-type algorithms, obtaining a method which has a number of practical advantages over other methods that are compatible with our access methods, such as proximal subgradient methods. First, it is reliable, and works well across a number of applications. Second, it has very few parameters that need to be tuned, and works well with sensible default values. Third, it typically produces a reasonable approximate solution in just a few tens of iterations. This paper is accompanied by an open-source implementation of the proposed solver, available at https://github.com/cvxgrp/OSBDO.

**Keywords** Convex optimization · Distributed optimization · Bundle-type method · Cutting-plane method · Finite memory

✉ Tetiana Parshakova
tetianap@stanford.edu

[1] Institute for Computational and Mathematical Engineering, Stanford University, 475 Via Ortega, Stanford, CA 94305, USA

[2] Department of Electrical Engineering, Stanford University, 350 Jane Stanford Way, Stanford, CA 94305, USA

Published online: 30 November 2023

# 1 Oracle-structured distributed optimization

## 1.1 Oracle-structured optimization problem

We consider the optimization problem

$$\text{minimize } h(x) = f(x) + g(x), \tag{1}$$

with variable $x \in \mathbf{R}^n$, where $f, g : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$ are the oracle and structured objective functions, respectively. We assume the problem has an optimal point $x^\star$, and we denote the optimal value of the problem (1) as $h^\star = h(x^\star)$. We use infinite values of $f$ and $g$ to encode constraints or the domains, with $\mathbf{dom}\, f = \{x \in \mathbf{R}^n \mid f(x) < \infty\}$ and similarly for $g$. We assume that $\mathbf{dom}\, f \supseteq \mathbf{dom}\, g \neq \emptyset$, *i.e.*, every point in the domain of $g$ is in the domain of $f$, and $g$ has at least one point in its domain.

### 1.1.1 The oracle objective function

We assume that the oracle function $f$ is block separable,

$$f(x) = \sum_{i=1}^{M} f_i(x_i), \quad x = (x_1, \ldots, x_M),$$

with $f_i : \mathbf{R}^{n_i} \to \mathbf{R} \cup \{\infty\}$ closed convex, where $n_1 + \cdots + n_M = n$. We refer to $x_i$ as the variable and $f_i$ as the objective function of agent $i$. Our access to $f_i$ is only via an oracle that evaluates $f_i(x_i)$ and a subgradient $q_i \in \partial f_i(x_i)$ at any point $x_i \in \mathbf{dom}\, f_i$.

### 1.1.2 The structured objective function

We assume the structured function $g$ is closed convex. While $f$ is block separable, $g$ (presumably) couples the block variables $x_1, \ldots, x_M$. We assume that $g$ is given in a structured form, using the language of Nesterov, which means we have a complete description of it. We assume that we can minimize $g$ plus some additional structured function of $x$. As a practical matter this might mean that $g$ is expressed in a domain-specific language (DSL) for convex optimization, such as CVXPY (Diamond and Boyd 2016; Agrawal et al. 2018), based on disciplined convex programming (DCP) (Grant et al. 2006).

### 1.1.3 Example

Problem (1) is very general, and includes as special cases many convex optimization problems arising in applications. To give a simple and specific example, it includes the so-called consensus problem (Boyd et al. 2011, Ch. 7),

$$\begin{array}{l} \text{minimize } \sum_{i=1}^{M} f_i(x_i) \\ \text{subject to } x_1 = \cdots = x_M, \end{array}$$

where $x_i \in \mathbf{R}^p$ are the variables, and $f_i$ are given convex functions. We put this in the form (1) with $n_i = p$, $n = Mp$, $x = (x_1, \ldots, x_M)$, and

$$g(x) = \begin{cases} 0 & x_1 = \cdots = x_M \\ \infty & \text{otherwise,} \end{cases}$$

the indicator function of the consensus constraint $x_1 = \cdots = x_M$.

#### 1.1.4 Optimality condition

The optimality condition for problem (1) is: $x$ is optimal if and only if

$$\partial h(x) = \partial f(x) + \partial g(x) \ni 0, \tag{2}$$

where $\partial h(x)$ denotes the subdifferential of $h$ at $x$. In some applications we may be interested in finding a subgradient $q^\star \in \partial f(x^\star)$ for which $-q^\star \in \partial g(x^\star)$. Such a subgradient can sometimes be interpreted as a vector of optimal prices. The method we describe in this paper will also compute (an estimate of) $q^\star$.

#### 1.1.5 Our focus

We seek an algorithm that solves the distributed oracle-structured problem (1), respecting our access assumptions. Several generic methods can be used, such as proximal subgradient methods or their accelerated extensions, described below. Most research on methods for the composite minimization problem focus on the case where $f$ is differentiable and $g$ has a simple, typically analytically computable, proximal operator, and algorithms that involve very minimal computation beyond evaluation of the gradient of $f$ and the proximal operator of $g$, such as a few vector-vector operations. Our focus here, however, is on the case where the agent oracles can be expensive to evaluate, which has several implications. First, it means that we focus on algorithms that in practice find good suboptimal points in relatively few iterations. Second, it justifies algorithms that solve an optimization problem involving $g$ in each iteration, instead of carrying out just a few vector-vector operations.

#### 1.1.6 Our contribution

Our contribution is to assemble a number of known methods, such as diagonal preconditioning, level bundle methods, and others into an algorithm that works well on a variety of practical problems, with no parameter tuning. By working well, we mean that modest accuracy, say on the order of 1%, is achieved typically in tens of iterations. In the language of the bundle method literature, our final algorithm is a disaggregate partially exact bundle method.

Our open-source implementation, along with all data to reproduce all of our numerical experiments, is available at https://github.com/cvxgrp/OSBDO. OSBDO stands for oracle-structured bundle distributed optimization.

### 1.2 Previous and related work

There is a vast literature on general distributed optimization, but fewer authors consider the specific subgradient oracle plus structured function access we consider here. Several general methods can be used to solve the problem (1) using our access methods, including subgradient, cutting-plane, and bundle-type methods.

#### 1.2.1 Subgradient methods

Subgradient methods were originally developed by Shor and others in the 1970s (Shor 2012). Early work that describes subgradients and convex optimization includes (Rockafellar 1981). (To use a subgradient-type method for our problem, we would need to compute a subgradient of $g$, which is readily done.) Subgradient methods typically require a large number of iterations, and are employed when the computational cost of each iteration is low, which is not the case in our setting. Subgradient methods also involve many algorithm parameters that must be tuned, such as a step-size sequence. Modern variations include AdaGrad (Duchi et al. 2011), an adaptive subgradient method which shows good practical results in the online learning setting, but for our setting still requires far too many iterations to achieve even modest accuracy.

#### 1.2.2 Proximal subgradient methods

A closely related method that is better matched to our specific access restrictions is the proximal subgradient method, which in each iteration requires a subgradient of $f$ and an evaluation of the proximal operator of $g$. (This proximal step is readily carried out since $g$ is structured.) Most of the original work on these types of methods focuses on the case where $f$ is differentiable, and the method is called the proximal gradient method. The proximal gradient method can be described as an operator-splitting method (Parikh and Boyd 2014, Ch. 4.2); some early work includes (Bruck 1975; Chen and Rockafellar 1997). Since then there have been two relevant developments. The papers (Passty 1979; Lions and Mercier 1979) handle the case where $f$ is nondifferentiable, and the gradient is replaced with a subgradient, so the method is called the proximal subgradient method. The stochastic case is addressed in Schechtman (2022), where a stochastic proximal subgradient method is developed. Another advance is the development of simple generic acceleration methods, originally introduced by Nesterov (1983). Other work addresses issues such as inexact computation of the proximal operator (Birgin et al. 2003), or inexact computation of the subgradient (Burachik et al. 2015). Proximal gradient and subgradient methods are widely used in different applications; see, *e.g.*, the book (Combettes and Pesquet 2011), which covers proximal gradient method applications in signal processing.

Compared to the method we propose, proximal subgradient methods fail to take into account previously evaluated function values and subgradients (other than through their effect on the current iterate), and our ability to build up a model of each agent function separately. These are done in the method we propose, which requires only a modest increase in complexity over evaluating just the proximal operator of $g$, and give substantial improvement in practical convergence.

### 1.2.3 Cutting-plane methods

Cutting-plane methods can be traced back to Cheney and Goldstein (1959) and Kelley's cutting-plane method (Kelley 1960). These methods maintain a piecewise affine lower bound or minorant on the objective, and improve it in each iteration using the subgradient and value of the function at the current iterate. Each iteration requires the solution of a linear program (LP), with size that increases with iterations. Cutting-plane methods are extended to handle convex mixed-integer problems in Westerlund and Pettersson (1995). Limited-memory or constraint-dropping versions, that drop terms in the minorant so the LP solved in each iteration does not grow in size, are given in Elzinga and Moore (1975); Dem'yanov and Vasil'ev (1985). Constraint dropping for general outer approximation algorithms are also considered in Gonzaga and Polak (1979). Many other variations on cutting-plane methods have been developed, including the analytic center cutting-plane method introduced by Atkinson and Vaidya (1995). A review of cutting-plane methods used in machine learning can be found in Sra et al. (2012).

### 1.2.4 Bundle methods

Bundle methods are closely related to cutting-plane methods. The first difference is the addition of a stabilization term, among which a proximal regularization term is most common and leads to the so-called proximal bundle method (Kiwiel 1990; Frangioni 2020). Typically in each iteration of the proximal bundle method a quadratic program (QP) must be solved. Other stabilization forms include the level bundle method (Lemaréchal et al. 1995; Frangioni 2020) and the trust-region bundle method (Marsten et al. 1975; Frangioni 2020). The second difference between cutting-plane and bundle methods is the logic that updates the current point only if a certain sufficient descent condition holds.

Bundle methods were first developed as dual methods in Lemaréchal (1975) and Mifflin (1977); the primal form of bundle methods was mainly studied in the 1990s. The first convergence proof for the proximal bundle method is given in Lemaréchal (1978). A comprehensive review of the history and development of bundle methods can be found in (Hiriart-Urruty and Lemaréchal 1996, Ch. XIV,XV). Later variations of bundle methods include variable metric bundle methods which accumulate second order information about function curvature in the proximal regularization term (Mifflin 1996; Chen and Fukushima 1999; Lukšan and Vlček 1999; Burke and Qian 2000; Haarala et al. 2004), bundle methods that handle inexact function values or subgradient values (Hintermüller 2001; Kiwiel 1985, 1995, 2006; Hare et al. 2016; van Ackooij et al. 2017; Lv et al. 2018; de Oliveira and Solodov 2020), bundle methods with semidefinite cutting sets replacing the traditional polyhedral cutting planes (Helmberg and Rendl 2000), and bundle methods with generalized stabilization (Kiwiel 1999; Frangioni 2002; Ben Amor et al. 2009; Frangioni and Gorgone 2014b; de Oliveira and Solodov 2016). Bundle methods are also widely used in non-convex optimization (Schramm and Zowe 1992; Kiwiel 1996; Lukšan and Vlček 1998; Fuduli et al. 2004; Haarala et al. 2007).

Incremental bundle methods (Emiel and Sagastizábal 2010) are based on a principle of selectively skipping oracle calls for some agents, while replacing them by an approximation. Essentially, the function is evaluated when lower and upper estimates on function values are not sufficient (van Ackooij and Frangioni 2018). This setting was analyzed within a framework of inexact oracles (Emiel and Sagastizábal 2010; de Oliveira et al. 2014; de Oliveira and Eckstein 2015; van Ackooij et al. 2016), *i.e.*, noisy oracles that are "asymptotically exact". Further, there has been an interest in asynchronous bundle methods as well (Iutzeler et al. 2020; Fischer 2022), where the agent oracle calls have varying finite running times.

Our method for parameter discovery relies on a standard level bundle method (Lemaréchal et al. 1995) for a few initial iterations. We then set the value of the proximal parameter to a Lagrange multiplier and proceed with the proximal bundle method. In a separate study, de Oliveira and Solodov (2016) propose an algorithm that automatically chooses between proximal and level bundle approaches at every iteration. Another related line of research is based on variable metric bundle methods (Kiwiel 1990; Lemaréchal et al. 1995; Lemaréchal and Sagastizábal 1997; Kiwiel 2000; Rey and Sagastizábal 2002; Frangioni 2002; de Oliveira et al. 2014; van Ackooij and Frangioni 2018). In general, these approaches are not crafted for solving composite function minimization satisfying our access conditions.

Our setting with difficult-to-evaluate components has been considered in bundle methods before (Emiel and Sagastizábal 2010). In those studies, the authors use the inexact evaluations to skip the "hard" components. We, on the other hand, focus on exact oracle evaluations or queries for each component $f_i$ alongside the structured function $g$, which is known as partially exact bundle method (van Ackooij et al. 2017). Our structured function $g$ presents a way to incorporate constraints into bundle methods, which has also been considered before (Kiwiel 1990).

In our method we disaggregate the minorant of $f$, an idea which has been extensively studied in the prior literature (Frangioni 2002; Lemaréchal et al. 2009; Frangioni and Gorgone 2014a, b; Frangioni 2020). When combined with the structured function $g$, which is not approximated by a minorant but handled exactly, we arrive at what is referred to as a disaggregate partially exact bundle method.

### 1.2.5 Software packages

Despite the large literature on related methods, relatively few open-source software packages are available. We found only one open-source implementation of a bundle method that is compatible with our access requirements. `BundleMethod.jl` (Kim et al. 2021) is a Julia package with implementations of proximal bundle method (Kiwiel 1990) and trust region bundle method (Kim et al. 2019). We found that in all our examples OSBDO works substantially better.

We also mention here other existing open-source implementations of bundle methods that do not meet our access requirements. A Fortran implementation (Mäkelä 2003) with Julia interface for multi-objective proximal bundle method (Mäkelä 2003; Mäkelä et al. 2016), available at Karmitsa (2016), requires the objective function to have full domain. A Fortran implementation (Teo et al. 2010), uses bundle methods for unconstrained regularized risk minimization. A limited memory bundle method

(Karmitsa 2007; Haarala et al. 2007; Karmitsa and Mäkelä 2010) is available as a Fortran implementation at Karmitsa (2007), solves nonsmooth large-scale minimization problems, either unconstrained (Karmitsa 2007; Haarala et al. 2007) or bound constrained (Karmitsa and Mäkelä 2010). An unconstrained proximal bundle method (Díaz and Grimmer 2023) with Julia implementation is available at Díaz (2021).

### 1.3 Outline

We describe our assembled bundle method for oracle-structured distributed optimization in Sect. 2. In Sect. 3 we take a deeper look at the agent functions $f_i$, which often involve additional variables that are optimized by the agent, and not used by the algorithm, which we call private variables. We present several numerical examples in Sect. 4. A convergence proof for our algorithm is given in Sect. A.

## 2 Disaggregate partially exact bundle method

Here we describe our assembled bundle method to solve the oracle-structured distributed optimization problem (1). We use the superscript $k$ to denote a vector or function at iteration $k$, as in $x_i^k$ or $x^k = (x_1^k, \ldots, x_m^k)$, our estimates of $x_i^\star$ and $x^\star$ at iteration $k$. Each iteration involves querying the agent objective functions, *i.e.*, evaluating $f_i(\tilde{x}_i^k)$ and a subgradient $q_i^k \in \partial f_i(\tilde{x}_i^k)$ at a query point $\tilde{x}_i^k$, for $i = 1, \ldots, k$, along with some computation that updates the iterates $x^k$. (We will describe what the specific query points are later.)

### 2.1 Minorants

The basic idea in a bundle or cutting-plane method is to maintain and refine a minorant of each agent function, denoted $\hat{f}_i^k : \mathbf{R}^{n_i} \to \mathbf{R}$. (Minorant means that these functions satisfy $\hat{f}_i^k(x_i) \leq f_i(x_i)$ for all $x_i \in \mathbf{R}^{n_i}$.) They are constructed from initial (given) minorants $\hat{f}_i^0$, and evaluations of the value and subgradients in previous iterations, as

$$\hat{f}_i^{k+1}(x_i) = \max\left(\hat{f}_i^k(x_i),\ f_i(\tilde{x}_i^{k+1}) + (q_i^{k+1})^T(x_i - \tilde{x}_i^{k+1})\right), \quad i = 1, \ldots, M. \tag{3}$$

Here we use the basic subgradient inequality

$$f_i(\tilde{x}_i^{k+1}) + (q_i^{k+1})^T(x_i - \tilde{x}_i^{k+1}) \leq f_i(x_i)$$

for all $x_i \in \mathbf{R}^{n_i}$, which shows that the lefthand side, which is an affine function of $x_i$, is a minorant of $f_i$. We also note that the minorant (3) is tight at $\tilde{x}_i^{k+1}$, *i.e.*,

$$\hat{f}_i^{k+1}(\tilde{x}_i^{k+1}) = f_i(\tilde{x}_i^{k+1}), \quad i = 1, \ldots, M.$$

From these agent objective minorants we obtain a minorant of the oracle objective $f$,

$$\hat{f}^{k+1}(x) = \hat{f}_1^{k+1}(x_1) + \cdots + \hat{f}_M^{k+1}(x_M), \tag{4}$$

and in turn, a minorant of $h$,

$$\hat{h}^{k+1}(x) = \hat{f}^{k+1}(x) + g(x). \tag{5}$$

The minorant of $f$ in (4) is referred to as a disaggregated minorant, to distinguish it from forming a minorant directly of $f$.

#### 2.1.1 Initial minorant

The simplest initial minorant $\hat{f}_i^0$ is a constant, a known lower bound on the agent objective $f_i(x_i)$. Later we will see how more sophisticated minorants for the agent objectives can be obtained. With a simple constant initial minorant for each agent, the minorant $\hat{f}^k$ is also piecewise affine function, since it is a sum of $m$ terms, each the maximum of $k$ affine functions.

#### 2.1.2 Lower bound on optimal value

Minorants allow us to compute a lower bound on $h^\star$, the optimal value of the problem (1). Since $\hat{h}^k$ is a minorant of $h$, we have

$$L^k = \min_x \hat{h}^k(x) \leq h^\star. \tag{6}$$

Evaluating $L^k$ involves solving an optimization problem, minimizing $g$ plus the piecewise affine function $\hat{f}^k$. Of course $h(x^k)$ is an upper bound on $h^\star$, so we have

$$L^k \leq h^\star \leq h(x^k).$$

#### 2.1.3 Gap-based stopping criterion

There is a multitude of stopping criteria that have been proposed for bundle methods (Lemaréchal et al. 1995; Lemaréchal and Sagastizábal 1997; Lemaréchal et al. 1996; Frangioni 2020; Lemaréchal 2001; Hiriart-Urruty and Lemaréchal 1996, 2013). Many of them are based on approximately satisfying the optimality conditions. For example, we can terminate when the norm of the aggregate subgradient and aggregate linearization error are both small (Lemaréchal and Sagastizábal 1997; Lemaréchal et al. 1996; Frangioni 2020; Hiriart-Urruty and Lemaréchal 1996, Ch. XIV). Stopping can also be done based on the predicted function decrease (Hiriart-Urruty and Lemaréchal 1996; Frangioni 2020), which differs depending on the type of stabilization in the bundle method (Hiriart-Urruty and Lemaréchal 1996, Ch. XV). We refer the reader to the books (Hiriart-Urruty and Lemaréchal 1996, 2013) for more details on stopping criterion for bundle methods. There are other reasonable choices of stopping criteria, for

example based on the minorant error $h(x^k) - \hat{h}^k(\tilde{x}^{k+1})$ and a subgradient of $\hat{h}^k$ at $\tilde{x}^{k+1}$ see, *e.g.*, (Lemaréchal 2001, Ch. 5.1).

Since we are in the regime where the agents are expensive to evaluate, solving a minimization problem (6) is not an issue, and especially if it is not done every iteration, *i.e.*, we compute the lower bound $L^k$ every few iterations. The lower bound (6) gives us a gap-based stopping criterion. In particular, this stopping criterion was previously used in level bundle methods (Lemaréchal et al. 1995; Hiriart-Urruty and Lemaréchal 1996, Ch. XIV). We stop if either the absolute gap is small,

$$h(x^k) - L^k \leq \epsilon^{\text{abs}}, \tag{7}$$

where $\epsilon^{\text{abs}} > 0$ is a given absolute gap tolerance, or the relative gap is small,

$$h(x^k) - L^k \leq \epsilon^{\text{rel}} \min\{|h(x^k)|, |L^k|\} \quad \text{and} \quad h(x^k)L^k > 0, \tag{8}$$

where $\epsilon^{\text{rel}} > 0$ is a given relative gap tolerance. (The sign condition guarantees that $\min\{|h(x^k)|, |L^k|\} \leq |h^\star|$.) This guarantees that we terminate with a point $x^k$ with objective value that is either within absolute difference $\epsilon^{\text{abs}}$, or within relative distance $\epsilon^{\text{rel}}$, of $h^\star$.

For later reference, we define the relative gap as

$$\omega^k = \begin{cases} \frac{h(x^k)-L^k}{\min\{|h(x^k)|,|L^k|\}} & h(x^k)L^k > 0 \\ \infty & \text{otherwise.} \end{cases} \tag{9}$$

We define the true relative gap as

$$\omega^k_{\text{true}} = \frac{h(x^k) - h^\star}{|h^\star|}, \tag{10}$$

for $h^\star \neq 0$. The relative gap is an upper bound on the true relative gap, *i.e.*, we have $\omega^k \geq \omega^k_{\text{true}}$. But $\omega^k$ is known at iteration $k$, whereas $\omega^k_{\text{true}}$ is not.

### 2.2 Oracle-structured bundle method

The basic bundle method for (1) is given below. It includes two parameters $\eta$ and $\rho$ (discussed later), and is initialized with an initial guess $x^0$ of $x$, and the initial minorants of the agent objective functions $\hat{f}_i^0$.

**Algorithm 2.1** BUNDLE METHOD FOR ORACLE- STRUCTURED DISTRIBUTED OPTIMIZATION

**given** $x^0 \in \mathbf{dom}\, h$, $h(x^0)$, initial minorants $\hat{f}_i^0$, parameters $\eta \in (0, 1)$ and $\rho > 0$.

**for** $k = 0, 1, \ldots$

1. *Check stopping criterion.* Quit if (7) or (8) holds.
2. *Tentative update.* $\tilde{x}^{k+1} = \text{argmin}_x \left(\hat{h}^k(x) + (\rho/2)\|x - x^k\|_2^2\right)$.
   Record $\hat{h}^k(\tilde{x}^{k+1})$ and $g(\tilde{x}^{k+1})$.
3. *Query agents.* Evaluate $f_i(\tilde{x}_i^{k+1})$ and $q_i^{k+1} \in \partial f_i(\tilde{x}_i^{k+1})$, for $i = 1, \ldots, M$.

4. Compute $h(\tilde{x}^{k+1}) = f_1(\tilde{x}_1^{k+1}) + \cdots + f_M(\tilde{x}_M^{k+1}) + g(\tilde{x}^{k+1})$.
5. Compute $\delta^k = h(x^k) - \left(\hat{h}^k\left(\tilde{x}^{k+1}\right) + (\rho/2)\left\|\tilde{x}^{k+1} - x^k\right\|_2^2\right)$.
6. *Update iterate.* Set $x^{k+1} = \begin{cases} \tilde{x}^{k+1} & h(x^k) - h\left(\tilde{x}^{k+1}\right) \geq \eta\delta^k \\ x^k & \text{otherwise.} \end{cases}$
7. *Update minorants.*
   Update $\hat{f}_i^{k+1}$ using (3), for $i = 1, \ldots, M$.
   Update $\hat{f}^{k+1}$ and $\hat{h}^{k+1}$ using (4) and (5).

### 2.2.1 Comments

In step 1 we evaluate $L^k$, which involves solving an optimization problem, *i.e.*, minimizing $\hat{h}^k(x)$. We recognize step 2 as evaluating the proximal operator of $\hat{h}^k$ at $x^k$ (Parikh and Boyd 2014). This step also involves solving an optimization problem, minimizing $\hat{h}^k(x)$ plus the quadratic (proximal) term $(\rho/2)\|x - x^k\|_2^2$. We note that we always have $\tilde{x}^{k+1} \in \mathbf{dom}\, g \subseteq \mathbf{dom}\, f$, so $h(\tilde{x}^{k+1})$ is always finite. Step 3, the agent query, can be done in parallel across all agents. Steps 4 and 5 involve only simple arithmetic using quantities already computed in steps 2 and 3. Steps 1, 5, and 6 use the quantity $h(x^k)$, but that has already been computed, since $x^k$ is equal to a previous $\tilde{x}^j$ for some $j \leq k$, and so was computed in step 4 of a previous iteration.

The substantial computation in each iteration of the bundle method is the evaluation of the lower bound in step 1, evaluating the proximal operator of $\hat{h}$ in step 2, and querying each agent oracle in step 3. The computation of $L^k$ in step 1 is only used for the stopping criterion, so this step can be carried out only every few steps, to reduce the average computational burden.

### 2.2.2 Descent method

The quantity $\delta^k$ computed in step 5 is nonnegative. To see this, we note from step 2 that $x = \tilde{x}^{k+1}$ minimizes $\hat{h}^k(x) + (\rho/2)\|x - x^k\|_2^2$, so

$$\begin{aligned} \hat{h}^k(\tilde{x}^{k+1}) + (\rho/2)\|\tilde{x}^{k+1} - x^k\|_2^2 &\leq \hat{h}^k(x^k) + (\rho/2)\|x^k - x^k\|_2^2 \\ &= \hat{h}^k(x^k) \\ &= h(x^k), \end{aligned} \tag{11}$$

from which $\delta^k \geq 0$ follows. From step 6 we see that the bundle method is a descent method, *i.e.*, $h(x^{k+1}) \leq h(x^k)$. More specifically, $h(x^{k+1}) < h(x^k)$ if the tentative step is accepted, *i.e.*, $x^{k+1} = \tilde{x}^{k+1}$, and $h(x^{k+1}) = h(x^k)$ if the tentative step is not accepted, *i.e.*, $x^{k+1} = x^k$.

### 2.2.3 Convergence

In "Appendix A" for completeness we give a proof that the bundle method converges, *i.e.*, $h(x^k) \to h^\star$ as $k \to \infty$. Convergence proofs for bundle methods have a long history, dating back to the 1970 s (Lemaréchal 1978).

#### 2.2.4 Choice of parameters

The bundle method is not particularly sensitive to the choice of $\eta$; the value $\eta = 0.01$ works well. The value of $\rho$, however, can have a strong influence on the practical performance of the algorithm. We discuss choices of $\rho$ later in §2.4.

#### 2.2.5 Dual variable

An estimate of an optimal dual variable $q^\star \in \partial f(x^\star)$ can be found when we compute the lower bound $L^k$ in step 1. To do this we compute $L^k$ by solving the modified problem

$$\begin{array}{ll} \text{minimize} & \hat{f}^{k+1}(x) + g(\tilde{x}) \\ \text{subject to} & \tilde{x} = x, \end{array}$$

with variables $x \in \mathbf{R}^n$ and $\tilde{x} \in \mathbf{R}^n$. (This is the consensus form; see, *e.g.*, (Boyd et al. 2011, Sect. 7).) The optimal dual variable associated with the constraint is our estimate of $q^\star$.

### 2.3 Diagonal preconditioning

Practical convergence of the bundle method is greatly enhanced by diagonal preconditioning. This means that we choose a diagonal matrix $D$ with positive entries, and define the (scaled) variable $\overline{x} = D^{-1}x$. Then we solve the problem (1) with variable $\overline{x}$ and functions

$$\overline{f}(\overline{x}) = f(D\overline{x}), \qquad \overline{g}(\overline{x}) = g(D\overline{x}).$$

Note that $\overline{g}$ is structured, since $g$ is. We recover the solution of the original problem (1) as $x^\star = D\overline{x}^\star$.

The idea of preconditioning has a long-standing history dating back to 1845 (Jacobi 1845) and is a crucial technique in optimization; see, *e.g.*, (Hestenes and Stiefel 1952; Sinkhorn 1964; Concus et al. 1985; Nocedal and Wright 1999; Bradley 2010; Takapoui and Javadi 2016). It is also closely related to the idea of variable metric methods, where essentially a different (not necessarily diagonal) preconditioning is applied each step. Variable metric bundle methods have been studied in Lemaréchal and Sagastizábal (1994); Bacaud et al. (2001); Helmberg and Pichler (2017). In contrast to our computationally cheap preconditioning technique, which is computed once before the algorithm starts, existing variable metric bundle methods are considerably more complex (Kiwiel 1990; Lemaréchal et al. 1995; Lemaréchal and Sagastizábal 1997; Kiwiel 2000; Rey and Sagastizábal 2002; Frangioni 2002; de Oliveira et al. 2014; van Ackooij and Frangioni 2018); in addition, most are not compatible with our specific access conditions.

Given $\overline{x}$, we query agent $i$ and the point $x_i = D_i\overline{x}_i$, where $D_i$ is the submatrix of $D$ associated with $x_i$. Agent $i$ responds with $f_i(x_i) = \overline{f}_i(\overline{x}_i)$ and $q_i \in \partial f_i(x_i)$.

The algorithm uses the function value without change, since $f_i(x_i) = \overline{f}_i(\overline{x}_i)$, and the scaled subgradient

$$\overline{q}_i = Dq_i \in \partial \overline{f}_i(\overline{x}_i).$$

Diagonal scaling can be thought of as a thin layer or interface between the algorithm, which works with the scaled variable $\overline{x}$ and functions $\overline{f}_i$ and $\overline{g}$, and the agents, which work with the original variables $x_i$ and the original functions $f_i$. In particular, neither the algorithm nor the agents need to know that diagonal scaling is being used, provided the query points and returned subgradients are scaled correctly.

#### 2.3.1 A specific choice for diagonal scaling

The matrix $D$ scales the original variable. Our goal is to scale the variables so they range over similar intervals, say of width on the order of one. To do this, we assume that we have known lower and upper bound on the entries of $x$,

$$l \leq x \leq u, \tag{12}$$

where $l < u$. (Presumably these constraints are included in $g$.) With these variable bounds we can choose

$$D = \mathbf{diag}(u - l). \tag{13}$$

The new variable bounds have the form $\overline{l} \leq z \leq \overline{u}$, with $\overline{u} - \overline{l} = \mathbf{1}$, the vector with all entries one.

### 2.4 Proximal parameter discovery

While the bundle algorithm converges for any positive value of the parameter $\rho$, fast convergence in practice requires a reasonable choice that is somewhat problem dependent; see, *e.g.*, Díaz and Grimmer (2023). Several schemes can be used to find a good value of $\rho$. In one common scheme $\rho$ is updated in each iteration depending on various quantities computed in the iteration. (As an example of such an adaptive scheme, see (Boyd et al. 2011, §3.4.1).) Another general scheme is to start with some steps that are meant to discover a good value of $\rho$, as in Rey and Sagastizábal (2002). For both such schemes, we fix $\rho$ after some modest number $K$ of iterations, so our proof of convergence (which assumes a fixed value of $\rho$) still applies. Our experiments suggest that a natural $\rho$-discovery method, described below, works well in practice. The idea of switching between proximal and level bundle methods has been proposed in de Oliveira and Solodov (2016). This method is more complex than our proposed simple approach, which switches just once, after a fixed number of iterations.

For the discovery steps, we modify the update in step 2, where we minimize

$$\hat{h}^k(x) + (\rho/2)\|x - x^k\|_2^2, \tag{14}$$

*i.e.*, evaluate the proximal operator of $\hat{h}^k$ at $x^k$, to solving the closely related problem

$$\begin{array}{l}\text{minimize } (1/2)\|x - x^k\|_2^2 \\ \text{subject to } \hat{h}^k(x) \leq \eta^k,\end{array} \tag{15}$$

where $\eta^k \in (L^k, h(x^k))$. (This ensures that the problem is feasible, and that the constraint is tight.) The problem (15) finds the projection of the point $x^k$ onto the $\eta^k$-sublevel set of the minorant. The idea of projecting onto the sublevel set instead of carrying out an explicit proximal step can be found in, *e.g.*, (Frangioni 2020, Sect. 2.3).

It is easy to show that any solution of (15) is also a solution of (14), for some value of $\rho$. Indeed we can find this value as $\rho = 1/\lambda$, where $\lambda$ is an optimal dual variable for the constraint in (15). In other words: When we solve (15), we are actually computing the proximal operator, *i.e.*, solving (14), for a value of $\rho$ that we only find after solving it.

Our $\rho$-discovery method uses the update (15) for the first $K = 20$ steps, with

$$\eta^k = \frac{h(x^k) + L^k}{2}.$$

After $K$ steps, we use the standard update (14) with the value of $\rho$ chosen as the geometric mean of the last 5 values found during the discovery steps.

### 2.5 Finite memory

Evidently the optimization problems that must be solved to compute $L^k$ in step 1 and $\tilde{x}^{k+1}$ in step 2 grow in size as $k$ increases. A standard method in bundle or cutting-plane type methods is to use a finite memory or constraint dropping version, also known as bundle compression (Correa and Lemaréchal 1993; de Oliveira and Eckstein 2015). The essential element that underpins theoretical convergence in compressed bundle methods is aggregate linearization (Kiwiel 1983; Correa and Lemaréchal 1993). It is given by the linearization of the minorant,

$$l_i^{k+1}(x_i) = \hat{f}_i^k(\tilde{x}_i^{k+1}) + (\hat{q}_i^{k+1})^T(x_i - \tilde{x}_i^{k+1}),$$

where $\hat{q}_i^{k+1} \in \partial \hat{f}_i^k(\tilde{x}_i^{k+1})$. As a result, a finite memory version replaces the minorant (3) with

$$\hat{f}_i^{k+1}(x_i) = \max\left(l_i^{k+1}(x_i), \max_{j=\max\{0,k-m+2\},\ldots,k}\left(f_i(\tilde{x}_i^{j+1}) + (q_i^{j+1})^T(x_i - \tilde{x}_i^{j+1})\right)\right),$$

for all $i = 1, \ldots, M$. Thus we use only the last $m - 1$ subgradients and values in the minorant, instead of all previous ones with an additional affine term for aggregate linearization. This results in a total of $m$ affine functions. For further information on bundle compression, we refer the reader to Correa and Lemaréchal (1993); Lemaréchal (2001); de Oliveira and Eckstein (2015); Frangioni (2020). Interestingly, the minorant

$\hat{f}_i^{k+1}(x_i)$ can be reduced to just two affine pieces. However, in practical applications, a smaller bundle size leads to slower convergence rates (de Oliveira and Eckstein 2015). (Finite-memory should not be confused with limited-memory, which can refer to a quasi-Newton algorithm that approximates the Hessian using a finite number of function and gradient evaluations.)

With finite memory it is possible that the lower bounds $L^k$ are not monotone non-decreasing. In this case we can keep track of the best (*i.e.*, largest) lower bound found so far, for use in our stopping criterion.

## 3 Agents

In this section we discuss some details of the agent objective functions, and how to compute the value and a subgradient. For simplicity we drop the subscript $i$ that was used denote agent $i$, so in this section, we denote $x_i$ as $x$, $f_i$ as $f$, $q_i$ as $q$, and so on.

### 3.1 Private variables and partial minimization

In many cases the agent function $f$ is defined via partial minimization, as the optimal value of a problem with variable $x$ and additional variables $z$. Specifically, $f(x)$ is the optimal value of the problem

$$\begin{array}{ll} \text{minimize} & F_0(x, z) \\ \text{subject to} & F_i(x, z) \leq 0, \quad i = 1, \ldots, m, \\ & H_i(x, z) = 0, \quad i = 1, \ldots, p, \end{array}$$

with variable $z$. (In this optimization problem, $x$ is a parameter.) Here $F_i$ are jointly convex in $(x, z)$, and $H_i$ are jointly affine in $(x, z)$. This is called partial minimization (Boyd and Vandenberghe 2004, §4.1), and defines $f$ that is convex. To evaluate $f$ we must solve a convex optimization problem. We refer to $x$ as the public variable for the agent, and $z$ as its private variable, since its value (or even its existence) is not known outside the agent.

We now explain how to find a subgradient $q \in \partial f(x)$. We solve the equivalent convex problem

$$\begin{array}{ll} \text{minimize} & F_0(\tilde{x}, z) \\ \text{subject to} & F_i(\tilde{x}, z) \leq 0, \quad i = 1, \ldots, m, \\ & H_i(\tilde{x}, z) = 0, \quad i = 1, \ldots, p, \\ & \tilde{x} = x, \end{array}$$

with variables $z$ and $\tilde{x}$. (As in the problem above, $x$ is a parameter in this optimization problem.) We assume that strong duality holds for this problem (which is guaranteed if the stronger form of Slater's condition holds), with $\nu$ denoting an optimal dual variable for the constraint $\tilde{x} = x$. Then it is easy to show that $q = -\nu$ is a subgradient of $f$ (Boyd et al. 2022) assuming strong duality holds.

### 3.2 Soft constraints and slack variables

We assume that the domain of the agent objective function includes the domain of $g$. This is critical since the agents will always be queried at a point $\tilde{x} \in \mathbf{dom}\, g$, and we need the agent objective function to be finite for any such $\tilde{x}$. In some cases this property does not hold for the natural definition of the agent objective function. Here we explain how to modify an original definition of an agent objective function $f$ so that it does.

Let $\tilde{f}$ be the original agent objective function, which does not satisfy $\mathbf{dom}\, \tilde{f} \subseteq \mathbf{dom}\, g$. It is often possible to replace constraints that appear in the original definition of $\tilde{f}$ with soft constraints, that ensure that $\mathbf{dom}\, \tilde{f} \subseteq \mathbf{dom}\, g$ holds.

There is also a generic method that uses slack variables to ensure that $f$ has full domain (which implies the domain condition). We define

$$f(x) = \min_{\tilde{x}} \left( \tilde{f}(\tilde{x}) + \lambda \|\tilde{x} - x\|_1 \right),$$

where $\lambda > 0$ is a parameter. This $f$ is convex and has full domain. For $\lambda$ large enough and $x \in \mathbf{dom}\, \tilde{f}$, we have $\tilde{x} = x$. We can think of $\tilde{x} - x$ as a slack variable, used to guarantee that $f(x)$ is defined for all $x$. We interpret $\lambda$ as a penalty for using the slack variable.

## 4 Examples

In this section we present a number of examples to illustrate our method. There are better methods to solve each of these examples, which exploit the custom structure of the particular problem. Our purpose here is to show that OSBDO, with default parameters, achieves good practical performance, *i.e.*, attains modest accuracy in some tens of iterations, on a variety of large-scale practical problems.

We use the default parameters for the first four examples, except that we continue iterations after the algorithm would have terminated with the default tolerances $\epsilon^{\text{abs}} = 10^{-3}$ and $\epsilon^{\text{rel}} = 10^{-2}$, to show the continued progress. The final subsection gives experimental results for the bundle method with finite memory (described in §2.5).

### 4.1 Supply chain

Supply chain problems involve the placement and movement of inventory, including sourcing from a supplier and distribution to an end customer. A comprehensive review can be found in Trisna et al. (2016). Here we consider a single commodity network composed of a series of $M$ trans-shipment components. Each of these has a vector $a_i$ of (nonnegative) flows into it, and a vector $b_i$ of (nonnegative) flows out of it. These are connected in series, with the first trans-shipment component's output connected to the second component's input, and so on, so

$$b_1 = a_2, \ \ldots, \ b_{M-1} = a_M. \tag{16}$$

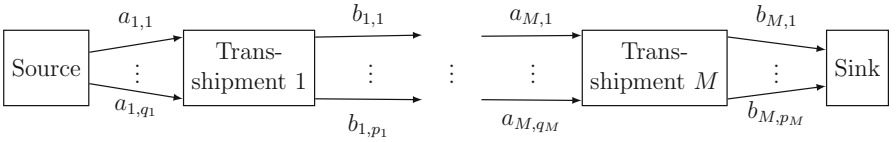


**Fig. 1** Supply chain consisting of a source (left), a sequence of $M$ trans-shipment components (middle), and a sink (right)

The vector $a_1$ gives the flows into the first trans-shipment component, and $b_M$ is the vector of flows out of the last trans-shipment component. This is illustrated in Fig. 1.

We assume the trans-shipment components are lossless, which means that

$$\mathbf{1}^T a_i = \mathbf{1}^T b_i, \quad i = 1, \ldots, M, \tag{17}$$

*i.e.*, the total flow of the commodity into each trans-shipment component equals the total flow leaving it.

Each trans-shipment component has a nonnegative objective function $f_i(a_i, b_i)$ which we interpret as the cost of shipping the commodity from the input flows to the output flows. In addition there is a source (purchase) objective term $\psi^{\mathrm{src}}(a_1)$, the cost of purchasing the commodity, and a sink (delivery) objective term $\psi^{\mathrm{sink}}(b_M)$, which we interpret as the negative revenue derived from delivering the commodity. (So we expect that $\psi^{\mathrm{src}}(a_1)$ is nonnegative, and $\psi^{\mathrm{sink}}(b_M)$ is nonpositive.)

The overall objective is the total of the trans-shipment costs and the source and sink costs,

$$\psi^{\mathrm{src}}(a_1) + f_1(a_1, b_1) + \cdots + f_M(a_M, b_M) + \psi^{\mathrm{sink}}(b_M).$$

Our goal is to choose the input and output flows $a_i$ and $b_i$, subject to the flow conservation constraints (16), so as to minimize this total objective. We assume $\psi^{\mathrm{src}}$, $\psi^{\mathrm{sink}}$, and all $f_i$ are convex, and that $\psi^{\mathrm{src}}$ and $f_i$ are nonnegative.

#### 4.1.1 Oracle-structured form

We put the supply chain problem into the form (1) as follows. We take $x_i = (a_i, b_i)$ for $i = 1, \ldots, M$, with variable range $0 \le x \le u$, where $u$ is an upper bound on the flows. We take $g$ to be the source and sink objective terms, plus the indicator function of the flow conservation constraints (16), variable ranges, and flow balance (17):

$$g(x) = \begin{cases} \psi^{\mathrm{src}}(a_1) + \psi^{\mathrm{sink}}(b_M) & (16),\ (17),\ 0 \le x \le u, \\ \infty & \text{otherwise.} \end{cases}$$

Roughly speaking, $g(x)$ is the gross negative profit, and $f(x)$ is the total shipping cost.

Our initial minorant is the indicator function of the flow conservation constraint (17), *i.e.*, 0 if (17) holds and $\infty$ otherwise. Note that this includes the lower bound $0 \le f_i(x_i)$.

#### 4.1.2 Trans-shipment cost

The trans-shipment cost for agent $i$ is based on a fully bipartite graph, with flow along each edge connecting one of $q_i$ inputs to each of $p_i$ outputs. We represent the edge flows as $X_i \in \mathbf{R}^{p_i \times q_i}$, where $a_i \in \mathbf{R}^{q_i}$ and $b_i \in \mathbf{R}^{p_i}$, and $(X_i)_{jk}$ is the flow from input $k$ to output $j$.

We assume that each edge has a convex quadratic cost of the form

$$(D_i)_{jk}(X_i)_{jk} + (E_i)_{jk}(X_i)_{jk}^2,$$

with $(D_i)_{jk} \geq 0$, and in addition is capacitated, *i.e.*, $0 \leq (X_i)_{jk} \leq (C_i)_{jk}$, with $(C_i)_{jk} \geq 0$. Due to the capacity constraints, the domain condition $\mathbf{dom}\, f_i \supseteq \mathbf{dom}\, g$ need not hold, so in addition we include a slack variable. We define $f_i(x_i)$ as the optimal value of the trans-shipment problem

$$\begin{array}{ll}
\text{minimize} & \sum_{j,k}\left((D_i)_{jk}(X_i)_{jk} + (E_i)_{jk}(X_i)_{jk}^2\right) + \lambda\|r\|_1 \\
\text{subject to} & 0 \leq (X_i)_{jk} \leq (C_i)_{jk}, \quad j = 1, \ldots, p_i, \quad k = 1, \ldots, q_i \\
& \sum_j (X_i)_{jk} = (\tilde{a}_i)_k, \quad k = 1, \ldots, q_i \\
& \sum_k (X_i)_{jk} = (\tilde{b}_i)_j, \quad j = 1, \ldots, p_i, \\
& (\tilde{a}_i, \tilde{b}_i) - r = x_i,
\end{array}$$

with variables $(X_i)_{jk}$, $\tilde{a}_i$, $\tilde{b}_i$, and $r$. Here $\lambda$ is a large positive parameter that penalizes using slack variables, *i.e.*, having $\tilde{a}_i \neq a_i$ or $\tilde{b}_i \neq b_i$.

To evaluate $f_i(x_i)$ we solve the problem above, which is a convex QP. To obtain a subgradient of $f_i$ at $x_i$ we use the negative optimal dual variables associated with the last constraint as $q_i$.

#### 4.1.3 Source and sink costs

We take simple linear source and sink costs,

$$\psi^{src}(a_1) = \alpha^T a_1, \qquad \psi^{sink}(b_M) = \beta^T b_M,$$

with $\alpha \geq 0$ and $\beta \leq 0$. We can interpret $\alpha_k$ as the price of acquiring the commodity at input $k$ of the first trans-shipment component, and $-\beta_k$ as the price of selling the commodity at output $k$ of the last trans-shipment component.

#### 4.1.4 Problem instance

We consider a problem instance with $M = 5$ trans-shipment components, and dimensions of $a_i$, $b_i$ given by

$$(20, 30), \quad (30, 40), \quad (40, 25), \quad (25, 35), \quad (35, 20).$$

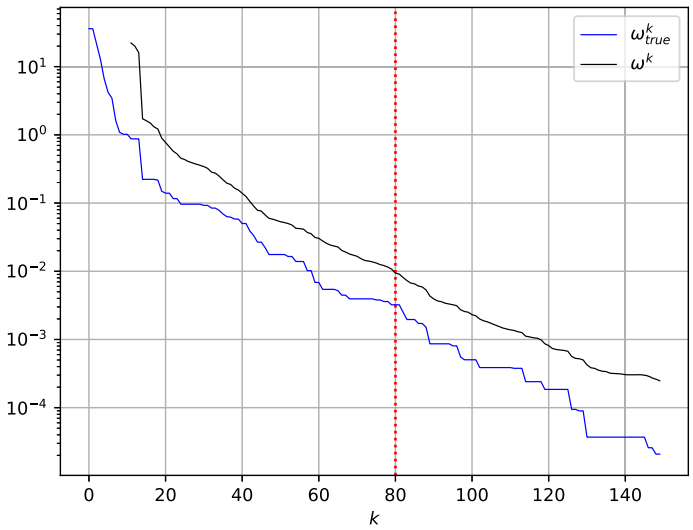


**Fig. 2** Relative gap and true relative gap versus iterations for supply chain example

We choose the edge capacities $(C_i)_{jk}$ from a log normal distribution, with $\log(C_i)_{jk} \sim \mathcal{N}(0, 1)$. From these edge capacities we construct an upper bound on each component of $x_i$, as the maximum of the sum of the capacities of all edges that feed the flow variable, and the sum of the capacities of all edges that flow out of it. This gives us our upper bound $u$. The linear cost coefficients $(E_i)_{jk}$ are log normal, with $\log(E_i)_{jk} \sim \mathcal{N}(0.07, 0.7)$. The quadratic cost coefficients $(D_i)_{jk}$ are obtained from the capacity and linear cost coefficients as

$$(D_i)_{jk} = (E_i)_{jk}/(2(C_i)_{jk}).$$

The sale prices $\alpha$ are uniform on [8, 10] and retail prices $-\beta$ are chosen uniformly on [10, 12]. The total problem size is $n = 300$ public variables (the input and output flows of the trans-shipment components), with an additional 4975 private variables in the agents (the specific flows along all edges of the trans-shipment components). The overall problem is a QP with 5275 variables.

### 4.1.5 Results

Figure 2 shows the relative gap $\omega^k$ and true relative gap $\omega^k_{\text{true}}$, versus iterations. With the default stopping criterion parameters the algorithm would have terminated after 80 iterations, when it can guarantee that it is no more than 1% suboptimal. In fact, it is around 0.3% suboptimal at that point. This is shown as the vertical dashed line in the plot. We can also see that the relative accuracy was in fact better than 1% after only 59 iterations, but, roughly speaking, we did not know it then.

### 4.2 Resource allocation

Resource allocation problems consider how to allocate a limited amount of several resources to a number of participants in order to optimize some overall objective. This kind of problem arises in communication networks (Han and Liu 2008), urban development (Wei et al. 2020), cloud computing (Choi and Lim 2016), and many others. Various algorithms have been proposed to solve this problem, with interesting ones including ant colony algorithm (Yin and Wang 2006), a genetic algorithm (Liu et al. 2005), and a graph-based approach (Zhou et al. 2021). In this section, we demonstrate how bundle method can be exploited to address a distributed version of the generic resource allocation problem.

#### 4.2.1 Resource allocation problem

We consider the optimal allocation of $n$ resources to $N$ participants. We let $r_i \in \mathbf{R}^n_+$ denote the amounts of the resources allocated to participant $i$, for $i = 1, \ldots, N$. The utility derived by participant $i$ is $U_i(r_i)$, where $U_i : \mathbf{R}^n_+ \to \mathbf{R}$ is a concave nondecreasing utility function. The resource allocation problem is to allocate resources to maximize the total utility subject to a limit on the total resources allocated:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^N U_i(r_i) \\ \text{subject to} & r_i \geq 0, \quad i = 1, \ldots, N \\ & \sum_{i=1}^N r_i \leq R, \end{array}$$

with variables $r_1, \ldots, r_N$, where $R \in \mathbf{R}^n_+$ is the total resources to be allocated, *i.e.*, the budget. We denote the optimal value, *i.e.*, the maximum total utility, as a function of $R$ as $U^\star(R)$. It is also concave and nondecreasing. When we solve this problem, an optimal dual variable associated with the last (budget) constraint can be interpreted as the prices of the resources.

#### 4.2.2 Distributed resource allocation problem

We have $M$ groups of participants, each with its own set of participants, resource budget $R_i \in \mathbf{R}^n_+$, and utility $U_i^\star(R_i)$. The distributed resource allocation problem is

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^M U_i^\star(R_i) \\ \text{subject to} & R_i \geq 0, \quad i = 1, \ldots, M \\ & \sum_{i=1}^M R_i \leq R, \end{array}$$

with variables $R_1, \ldots, R_M$, where $R$ is the total budget of resources. This problem has exactly the same form as the resource allocation problem, but here $U_i^\star(R_i)$ is the optimal total utility for group $i$ of participants, whereas in the resource allocation problem, $U_i(r_i)$ is the utility of the single participant $i$.

#### 4.2.3 Oracle-structured form

Each agent is associated with a group in the distributed resource allocation problem. We take $x_i = R_i$, the total resource allocated to the participants in group $i$. We take agent objective functions

$$f_i(x_i) = -U_i^\star(x_i), \quad i = 1, \ldots, M,$$

the optimal (negative) total utility for its group of participants, given resources $x_i$. We take the structured objective function to be

$$g(x) = \begin{cases} 0 & x_1 + \cdots + x_M \leq R, \quad x_i \geq 0, \quad i = 1, \ldots, M \\ \infty & \text{otherwise.} \end{cases}$$

With these agent and structured objectives, the problem (1) is equivalent to the distributed resource allocation problem. The resource allocations to the individual participants within each group are private variables; the public variables are the total resources allocated to each group.

To evaluate $f_i(\tilde{x}_i)$, we solve the resource allocation problem for group $i$. To find a subgradient $q \in \partial f_i(\tilde{x}_i)$, we take the negative of an optimal dual variable in the resource allocation problem, *i.e.*, the negative of the optimal prices. To obtain a range on each variable, we use $0 \leq x_i \leq R$.

#### 4.2.4 Problem instance

Our example uses participant utility functions of the form

$$U_i(r_i) = \mathbf{geomean}\,(A_i r_i + b_i)\,,$$

where $\mathbf{geomean}(u) = \left(\prod_{i=1}^p u_i\right)^{1/p}$ for $u \in \mathbf{R}_+^p$ is the geometric mean function. The entries of $A_i$ are nonnegative, so $U_i$ is concave and nondecreasing. We choose $A_i$ to be column sparse, with around $\frac{n}{10}$ columns chosen at random to be nonzero. The nonzero entries in these columns are chosen as uniform on [0, 1]. We choose entries of $b_i$ to be uniform on $[0, \frac{n}{10}]$. The resource budgets $R_i$ are chosen from $\log R_i \sim \mathcal{N}(\log \frac{n}{10}, 1)$. The initial minorant is given by

$$\hat{f}^0 = \sum_{i=1}^{M} \sum_{j=1}^{N_i} -\mathbf{geomean}(A_{ij} R + b_{ij}),$$

the negative of the utility if all agents were given the full budget of resources.

For the specific instance we consider, we take $n = 50$ resources and $M = 50$ agents, each of which allocates resources to $N_i = 10$ participants. (As a single resource allocation problem we would have 50 resources and 500 participants.) The utility functions use $p = 5$, *i.e.*, each is the geometric mean of 5 affine functions.

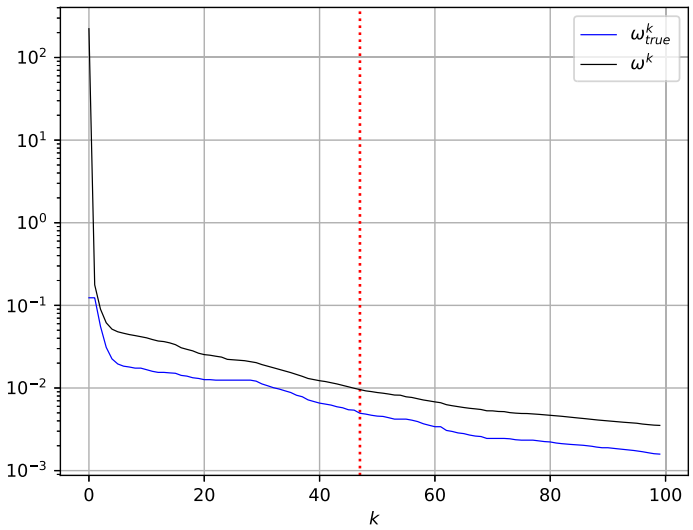


**Fig. 3** Relative gap and true relative gap versus iterations for resource allocation example

#### 4.2.5 Results

Figure 3 shows the relative gap $\omega^k$ and true relative gap $\omega^k_{\text{true}}$ versus iterations. With the default stopping criterion the algorithm would have terminated after 47 iterations (shown as the vertical dashed line), when it can guarantee that it is no more than 1% suboptimal. In fact, it is around 0.5% suboptimal at that point. We can also see that the relative accuracy was better than 1% after only 33 iterations.

### 4.3 Multi-commodity flow

Multi-commodity flow problems involve shipping different commodities on the same network in a way such that the total utility is maximized, while the total flow on each edge stays below its capacity. In Ouorou et al. (2000) authors give a survey of algorithms for this problem. We consider a network defined by a graph with $p$ vertices or nodes and $q$ directed edges, defined by the incidence matrix $A \in \mathbf{R}^{p\times q}$. The network supports the flow of $M$ different commodities. Each commodity has a source node, denoted $r_i \in \{1,\ldots,p\}$, and a sink or destination node, denoted $s_i \in \{1,\ldots,p\}$. The flow of commodity $i$ from the source to destination is given by $d_i \geq 0$. We let $z_i \in \mathbf{R}^q_+$ be the vector of flows of commodity $i$ on the edges. Flow conservation is the constraint

$$Az_i + d_i(e_{r_i} - e_{s_i}) = 0, \quad i = 1,\ldots,M,$$

where $e_k$ denotes the $k$th unit vector, with $(e_k)_k = 1$ and $(e_k)_j = 0$ for $j \neq k$. Flow conservation requires that the flow is conserved at all nodes, with $d_i$ injected at the source node, and $d_i$ removed at the sink node. The utility of flow $i$ is $U_i(d_i)$, where

$U_i$ is a concave nondecreasing function. Our objective is to maximize the total utility $U_1(d_1) + \cdots + U_M(d_M)$.

The total flow on the edges must not exceed the capacities on the edges, given by $c \in \mathbf{R}^q_+$, *i.e.*,

$$z_1 + \cdots + z_M \leq c.$$

(This capacity constraint couples the variables associated with the different commodities.)

The variables in this multi-commodity flow problem are $z_1, \ldots, z_M$ and $d_1, \ldots, d_M$. The data are the incidence matrix $A$, the edge capacities $c$, the commodity source and sink nodes $(r_i, s_i)$, and the flow utility functions $U_i$.

It will be convenient to work with a form of the problem where we split the capacity on each edge into $M$ different capacities for the different commodities. We take

$$z_i \leq c_i, \quad i = 1, \ldots, M,$$

where $c_1 + \cdots + c_M = c$ and $c_i \geq 0$ for $i = 1, \ldots, M$. We interpret $c_i$ as the edge capacity assigned to, or reserved for, commodity $i$. Our multi-commodity flow problem then has the form

$$\begin{array}{ll} \text{maximize} & U_1(d_1) + \cdots + U_M(d_M) \\ \text{subject to} & 0 \leq z_i \leq c_i, \quad i = 1, \ldots, M \\ & Az_i + d_i(e_{r_i} - e_{s_i}) = 0, \quad i = 1, \ldots, M \\ & c_1 + \cdots + c_M = c, \quad c_i \geq 0, \quad i = 1, \ldots, M, \end{array}$$

with variables $z_i$, $d_i$, and $c_i$. This is evidently equivalent to the original multi-commodity flow problem.

#### 4.3.1 Oracle-structured form

We can put the multi-commodity flow problem into oracle-structured form as follows. We take $x_i = c_i$, the edge capacity assigned to commodity $i$. We take the range of $x_i$ as $0 \leq x_i \leq c$. We take the agent cost function $f_i(x_i)$ to be the optimal value of the single commodity flow problem (expressed as a minimization problem)

$$\begin{array}{ll} \text{minimize} & -U_i(d_i) \\ \text{subject to} & 0 \leq z_i \leq x_i \\ & Az_i + d_i(e_{r_i} - e_{s_i}) = 0, \end{array}$$

with variables $z_i$ and $d_i$. (Here $x_i = c_i$ is a parameter.) These functions $f_i(x_i)$ are convex and nonincreasing in $x_i$, the capacity assigned to commodity $i$. To evaluate $f_i$ we solve the single commodity flow problem above; a subgradient $q_i$ is obtained as the negative optimal dual variable associated with the capacity constraint $z_i \leq x_i$.

We take $g(x)$ to be the indicator function of

$$x_1 + \cdots + x_M = c, \quad x_i \geq 0, \quad i = 1, \ldots, M$$

(which includes the ranges of $x_i$).

All together there are $n = Mq$ variables, representing the allocation of edge capacity to the commodities. There are also $M(q+1)$ private variables, which are the flows for each commodity on each edge and the values of the flows of each commodity.

#### 4.3.2 Problem instance

We consider an example with $M = 10$ commodities, and a graph with $p = 100$ nodes and $q = 1000$ edges. Edges are generated randomly from pairs of nodes, with an additional cycle passing through all vertices (to ensure that the graph is strongly connected, *i.e.*, there is a directed path from any node to any other). We choose the source-destination pairs $(r_i, s_i)$ randomly. We choose capacities $c_i$ from a uniform distribution on [0.2, 2]. The flow utilities $U_i$ are linear, *i.e.*, $U_i(d_i) = b_i d_i$, with $b_i$ chosen uniformly on [0.5, 1.5]. This problem instance has $n = 10000$ variables, with an additional 10010 private variables.

#### 4.3.3 Results

Figure 4 shows the relative gap $\omega^k$ and true relative gap $\omega^k_{\text{true}}$ versus iterations. With the default stopping criterion the algorithm would have terminated after 14 iterations (shown as the vertical dashed line), when it can guarantee that it is no more than 1% suboptimal. In fact, it is around 0.6% suboptimal at that point. We can also see that the relative accuracy was better than 1% after only 13 iterations.

### 4.4 Federated learning

Federated learning refers to distributed machine learning, where agents keep their local data and collaboratively train a model using a distributed algorithm. An overview of the development of federated learning is given in Li et al. (2020). In this section we consider the federated learning problem.

We are to fit a model parameter $\theta \in \mathbf{R}^d$ to data that is stored in $M$ locations. Associated with each location is a function $L_i : \mathbf{R}^d \to \mathbf{R}$, where $L_i(\theta)$ is the loss for parameter value $\theta$ for the data held at location $i$. We seek $\theta$ that minimizes

$$\sum_{i=1}^{M} L_i(\theta) + R(\theta),$$

where $R : \mathbf{R}^d \to \mathbf{R} \cup \{\infty\}$ is a regularization function. We assume that $L_i$ and $R$ are convex, so this fitting problem is convex. In federated learning (Kairouz et al. 2021), we solve the fitting problem in a distributed manner, with each location handling its own data.

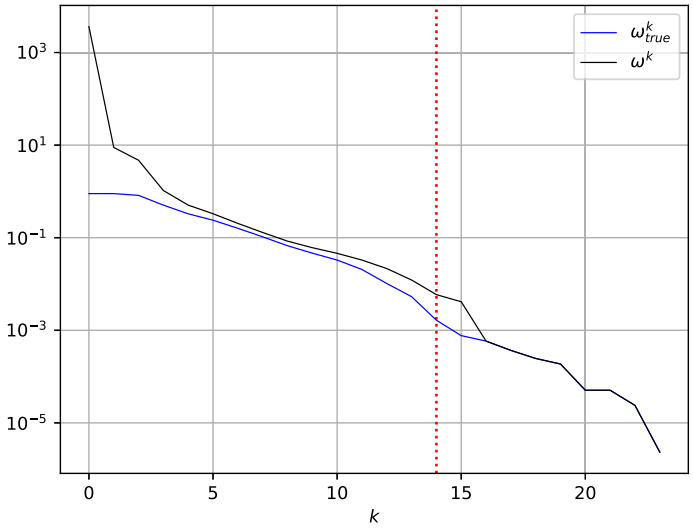


**Fig. 4** Relative gap and true relative gap for multi-commodity flow example

### 4.4.1 Oracle-structured form

We can put the federated learning problem into oracle-structured form by taking $x_i$ to be the parameter estimate at location $i$, $f_i = L_i$, and $g$ the indicator function for consensus plus the regularization,

$$g(x) = \begin{cases} R(x_1) & x_1 = \cdots = x_M \\ \infty & \text{otherwise.} \end{cases}$$

### 4.4.2 Problem instance

We consider a classification problem with logistic loss function,

$$L_i(\theta) = \sum_{j=1}^{n_i} \log\left(1 + \exp(-v_{ij} u_{ij}^T \theta)\right),$$

where $v_{ij} \in \{-1, 1\}$ is the label and $u_{ij} \in \mathbf{R}^d$ is the feature value for data point $j$ in location $i$, and $n_i$ is the number of data points at location $i$. We use $\ell_1$ regularization, *i.e.*, $R(\theta) = \lambda\|\theta\|_1$, where $\lambda > 0$.

Our example takes parameter dimension $d = 500$, $M = 10$ locations, and $n_i = 1000$ data points at each location. In this problem there are no private variables, and the total dimension of $x$ is $n = Md = 5000$.

We generate the data points as follows. The entries of $u_{ij}$ are $\mathcal{N}(0, 1)$, and we take

$$v_{ij} = \mathbf{sign}\left(u_{ij}^T \theta^{\text{true}} + z_{ij}\right),$$

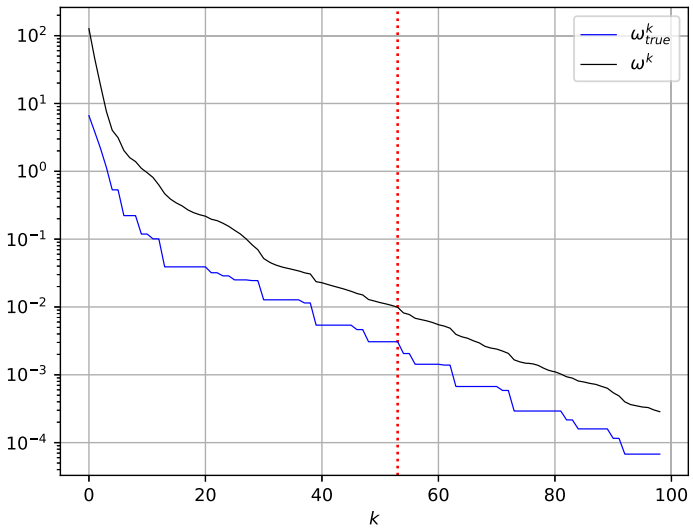


**Fig. 5** Relative gap and true relative gap versus iterations for federated learning example

where $z_{ij} \sim \mathcal{N}(0, 10^{-2})$ and $\theta^{\text{true}}$ is a true value of the parameter, chosen as sparse with around 50 nonzero entries, each $\mathcal{N}(0, 1)$. We choose $\lambda = 5$.

### 4.4.3 Results

Figure 5 shows the relative gap $\omega^k$ and true relative gap $\omega^k_{\text{true}}$ versus iterations. With the default stopping criterion the algorithm would have terminated after 53 iterations (shown as the vertical dashed line), when it can guarantee that it is no more than 1% suboptimal. In fact, it is around 0.3% suboptimal at that point. We can also see that the relative accuracy was better than 1% after only 39 iterations.

## 4.5 Finite-memory experiments

In this subsection we present results showing how limiting the memory to various values affects convergence. In many cases, limiting the value to $m = 20$ or more has negligible effect. As an example, Fig. 6 shows the effect on convergence of memory with values $m = 20$, $m = 30$, $m = 50$, and $m = \infty$ for the federated learning problem described above. In this example finite memory has essentially small effect on the convergence.

As an example of a case where finite memory does affect the convergence, Fig. 7 shows the effect of finite memory with values $m = 20$, $m = 30$, $m = 50$, and $m = \infty$. With $m = 20$, the algorithm shows minimal improvement beyond double the number of iterations required for a method with $m = \infty$ to achieve the default tolerance level; with $m = 30$ there is a modest increase; with $m = 50$ there is a small increase.

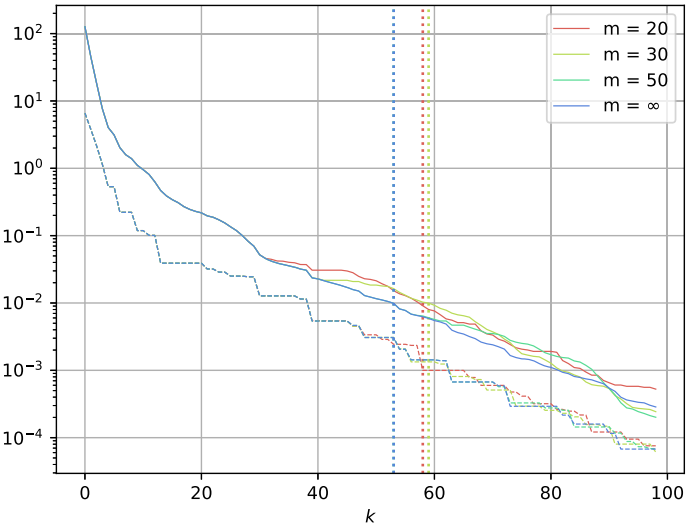


**Fig. 6** Relative gap (solid) and true relative gap (dashed) versus iterations for federated learning example, with finite memory values $m = 20$, $m = 30$, $m = 50$, and $m = \infty$

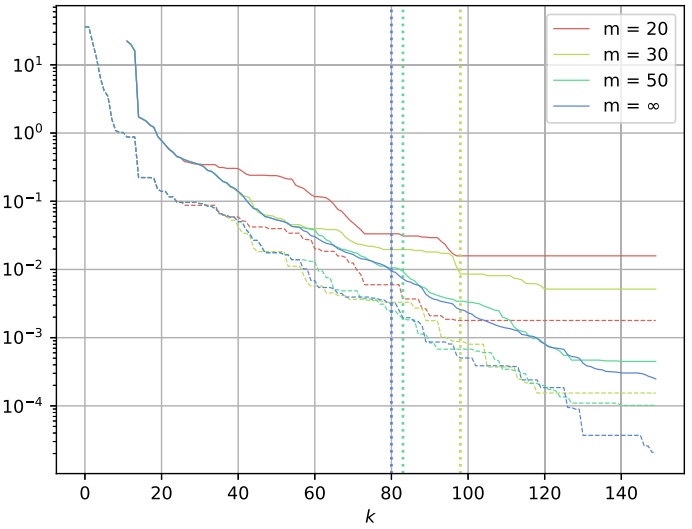


**Fig. 7** Relative gap (solid) and true relative gap (dashed) versus iterations for supply chain example, with finite memory values $m = 20$, $m = 30$, $m = 50$, and $m = \infty$

## 5 Conclusions

We focus on developing a good practical method for distributed convex optimization in a setting where the agents support a value/subgradient oracle, which can take substantial effort to evaluate, and the coupling among the agent variables is given explicitly and exactly as a structured convex problem, possibly including constraints.

(This differs from the more typical setting, where the agent functions are differentiable and can be evaluated quickly, and the coupling function has an analytical proximal operator.) Our assumptions allow us to consider methods that carry out more computation in each iteration, such as cutting-plane or bundle methods, that typically involve the solution of a QP. We have found that a basic bundle-type method, when combined with diagonal scaling and a good algorithm parameter discovery method, gives good practical convergence across a number of problems types and sizes. Here by good practical convergence we mean that with default algorithm parameters, a reasonable approximate solution can be found in a few tens of iterations, and a higher accuracy solution (which is generally not needed in applications) can be obtained in perhaps a hundred or fewer iterations. (Theoretical convergence of the algorithm is always guaranteed.)

Our methods combines multiple variations of known techniques for bundle-type methods into a solver has a number of attractive features. First, it has essentially no algorithm parameters, and works well with the few parameters set to default values. Second, it achieves good practical convergence across a number of problems types and sizes. Third, it can warm start when the coupling changes, by saving the information obtained in previous agent evaluations.

**Acknowledgements** We thank Parth Nobel, Nikhil Devanathan, Garrett van Ryzin, Dominique Perrault-Joncas, Lee Dicker, and Manan Chopra for very helpful discussions about the problem and formulation. The supply chain example was suggested by van Ryzin, Perrault-Joncas, and Dicker. The communication layer for the implementation with structured variables, to be described in a future paper, was designed by Parth Nobel and Manan Chopra. We thank Mateo Díaz for pointing us to some very relevant literature that we had missed in an early version of this paper. We thank three anonymous reviewers who gave extensive and helpful feedback on an early version of this paper. We gratefully acknowledge support from Amazon, Stanford Graduate Fellowship, Office of Naval Research, and the Oliger Memorial Fellowship. This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

## Appendix A: Convergence proof

In this section we give a proof of convergence of the bundle method for oracle-structured optimization. Our proof uses well known ideas, and borrows heavily from Belloni (2005). We will make one additional (and traditional) assumption, that $f$ and $g$ are Lipschitz continuous on **dom** $g$.

We say that the update was accepted in iteration $k$ if $x^{k+1} = \tilde{x}^{k+1}$. Suppose this occurs in iterations $k_1 < k_2 < \cdots$. We let $K = \{k_1, k_2, \ldots\}$ denote the set of iterations where the update was accepted. We distinguish two cases: $|K| = \infty$ and $|K| < \infty$.

### Infinite updates

We assume $|K| = \infty$. First we establish that $\delta^{k_s} \to 0$ as $s \to \infty$. Since $k = k_s$ is an accepted step, from step 6 of the algorithm we have

$$\eta\delta^{k_s} \leq h(x^{k_s}) - h(x^{k_s+1}) = h(x^{k_s}) - h(x^{k_s+1}).$$

Summing this inequality from $s = 1$ to $s = l$ and dividing by $\eta$ gives

$$\sum_{s=1}^{l} \delta^{k_s} \leq \frac{h(x^{k_1}) - h(x^{k_{l+1}})}{\eta} \leq \frac{h(x^0) - h^\star}{\eta},$$

which implies that $\delta^{k_s}$ is summable, and so converges to zero as $s \to \infty$.

Since $\tilde{x}^{k_s+1}$ minimizes $\hat{h}^{k_s}(x) + (\rho/2)\|x - x^{k_s}\|_2^2$, we have

$$\partial \hat{h}^{k_s}\left(\tilde{x}^{k_s+1}\right) + \rho(\tilde{x}^{k_s+1} - x^{k_s}) \ni 0.$$

Using $\tilde{x}^{k_s+1} = x^{k_s+1} = x^{k_{s+1}}$, we have

$$\rho(x^{k_s} - x^{k_{s+1}}) \in \partial \hat{h}^{k_s}\left(x^{k_{s+1}}\right).$$

It follows that

$$h^\star = h(x^\star) \geq \hat{h}^{k_s}(x^\star) \geq \hat{h}^{k_s}(x^{k_{s+1}}) + \rho(x^{k_s} - x^{k_{s+1}})^T(x^\star - x^{k_{s+1}}).$$

We first rewrite this as

$$\begin{aligned}\frac{h^\star - \hat{h}^{k_s}(x^{k_{s+1}})}{\rho} &\geq (x^{k_s} - x^{k_{s+1}})^T(x^\star - x^{k_s}) + (x^{k_s} - x^{k_{s+1}})^T(x^{k_s} - x^{k_{s+1}}))\\ &= (x^{k_s} - x^{k_{s+1}})^T(x^\star - x^{k_s}) + \|x^{k_s} - x^{k_{s+1}}\|_2^2,\end{aligned}$$

and then in the form we will use below,

$$2(x^{k_s} - x^{k_{s+1}})^T(x^\star - x^{k_s}) \leq (2/\rho)\left(h^\star - \hat{h}^{k_s}(x^{k_{s+1}})\right) - 2\|x^{k_s} - x^{k_{s+1}}\|_2^2.$$

Now we use a standard subgradient algorithm argument. We have

$$\begin{aligned}\|x^{k_{s+1}} - x^\star\|_2^2 &= \|x^{k_s} - x^\star\|_2^2 + \|x^{k_{s+1}} - x^{k_s}\|_2^2 + 2(x^{k_s} - x^{k_{s+1}})^T(x^\star - x^{k_s})\\ &\leq \|x^{k_s} - x^\star\|_2^2 + (2/\rho)\left(h^\star - \hat{h}^{k_s}(x^{k_{s+1}})\right) - \|x^{k_{s+1}} - x^{k_s}\|_2^2\\ &= \|x^{k_s} - x^\star\|_2^2 + (2/\rho)\left(h^\star - h(x^{k_s}) + \delta^{k_s}\right).\end{aligned}$$

Summing this inequality from $s = 1$ to $s = l$ and re-arranging yields

$$\begin{aligned}(2/\rho)\sum_{s=1}^{l}\left(h(x^{k_s}) - h^\star\right) &\leq \|x^{k_1} - x^\star\|_2^2 - \|x^{k_{l+1}} - x^\star\|_2^2 + (2/\rho)\sum_{s=1}^{l}\delta^{k_s}\\ &\leq \|x^{k_1} - x^\star\|_2^2 + 2(h(x^0) - h^\star)/\eta\rho.\end{aligned}$$

It follows that the nonnegative series $h(x^{k_s}) - h^\star$ is summable, and therefore, $h(x^{k_s}) \to h^\star$ as $s \to \infty$.

**Finite updates**

We assume $|K| < \infty$, with $p = \max K$ its largest entry. It follows that for any $k > p$, we have $h(x^k) - h\left(\tilde{x}^{k+1}\right) < \eta\delta^k$. Note that $x^k = x^p$ for all $k \geq p+1$. Moreover, using

$$\|\tilde{x}^{k+2} - x^p\|_2^2 = \|\tilde{x}^{k+2} - \tilde{x}^{k+1}\|_2^2 + \left\|\tilde{x}^{k+1} - x^p\right\|_2^2 - 2\left(x^p - \tilde{x}^{k+1}\right)^T (\tilde{x}^{k+2} - \tilde{x}^{k+1})$$

with $\rho(x^p - \tilde{x}^{k+1}) \in \partial\hat{h}^k\left(\tilde{x}^{k+1}\right)$ and $\hat{h}^{k+1}(\tilde{x}^{k+2}) \geq \hat{h}^k(\tilde{x}^{k+2})$, we get

$$\begin{aligned}\delta^k - \delta^{k+1} &\geq \hat{h}^{k+1}(\tilde{x}^{k+2}) - \hat{h}^k\left(\tilde{x}^{k+1}\right) - \rho\left(x^p - \tilde{x}^{k+1}\right)^T (\tilde{x}^{k+2} - \tilde{x}^{k+1}) \\ &\quad + (\rho/2)\left\|\tilde{x}^{k+2} - \tilde{x}^{k+1}\right\|_2^2 \\ &\geq (\rho/2)\left\|\tilde{x}^{k+2} - \tilde{x}^{k+1}\right\|_2^2.\end{aligned}$$

Therefore, $\delta^k \geq \delta^{k+1} + (\rho/2)\left\|\tilde{x}^{k+2} - \tilde{x}^{k+1}\right\|_2^2$ for all $k \geq p+1$. Then from

$$\begin{aligned}\hat{h}^k(x^p) &\geq \hat{h}^k\left(\tilde{x}^{k+1}\right) + \rho(x^p - \tilde{x}^{k+1})^T(x^p - \tilde{x}^{k+1}) \\ &= h(x^k) - \delta^k + (\rho/2)\left\|x^p - \tilde{x}^{k+1}\right\|_2^2,\end{aligned}$$

it follows that $\left\|x^p - \tilde{x}^{k+1}\right\|_2^2 \leq 2\delta^k/\rho \leq 2\delta^p/\rho$.

Now we use the assumption that $f$ and $g$ are Lipschitz continuous with Lipschitz constant $L$ for all $x \in \mathbf{dom}\, g$. Every $q \in \partial\hat{f}^k(x)$ has the form $q = \sum_{t \leq k} \theta_t q^t$, with $\theta_t \geq 0$ and $\sum_t \theta_t = 1$, a convex combination of normal vectors of active constraints at $x$, where $q^t \in \partial f(x^t)$. Therefore, $\hat{h}^k(x) = \hat{f}^k(x) + g(x)$ is $2L$-Lipschitz continuous.

Combining this with

$$\delta^k \leq h(x^k) - \hat{h}^k\left(\tilde{x}^{k+1}\right), \qquad -\eta\delta^k \leq h\left(\tilde{x}^{k+1}\right) - h(x^k),$$

we have

$$(1-\eta)\delta^k \leq h\left(\tilde{x}^{k+1}\right) - h\left(\tilde{x}^k\right) + \hat{h}^k\left(\tilde{x}^k\right) - \hat{h}^k\left(\tilde{x}^{k+1}\right) \leq 4L\left\|\tilde{x}^k - \tilde{x}^{k+1}\right\|_2.$$

Therefore, from

$$\frac{(1-\eta)^2\rho}{32L^2}\sum_{k \geq p}\left(\delta^k\right)^2 \leq \sum_{k \geq p}\left(\delta^k - \delta^{k+1}\right) \leq \delta^p,$$

we can establish that $\delta^k$ converges to zero as $k \to \infty$. This implies

$$\lim_{k\to\infty}\left(\hat{h}^k\left(\tilde{x}^{k+1}\right) + (\rho/2)\left\|\tilde{x}^{k+1} - x^p\right\|_2^2\right) = h(x^p).$$

Also from $\lim_{k\to\infty}\left(h(x^p) - h\left(\tilde{x}^{k+1}\right)\right) = 0$ and $\left\|x^p - \tilde{x}^{k+1}\right\|_2^2 \leq \frac{2\delta^k}{\rho}$, it follows that

$$\lim_{k\to\infty}\hat{h}^k\left(\tilde{x}^{k+1}\right) = h(x^p), \qquad \lim_{k\to\infty}\left\|x^p - \tilde{x}^{k+1}\right\|_2^2 = 0.$$

Hence, we get $0 \in \partial h(x^p)$, which implies $h(x^p) = h^\star$.

## References


Agrawal A, Verschueren R, Diamond S, Boyd S (2018) A rewriting system for convex optimization problems. J Control Decis 5(1):42–60

Atkinson D, Vaidya P (1995) A cutting plane algorithm for convex programming that uses analytic centers. Math Program 69:1–43

Bacaud L, Lemaréchal C, Renaud A, Sagastizábal C (2001) Bundle methods in stochastic optimal power management: a disaggregated approach using preconditioners. Comput Optim Appl 20:227–244

Belloni A (2005) Lecture notes for IAP 2005 course introduction to bundle methods. Operation Research Center, MIT, Version of February, 11

Ben Amor H, Desrosiers J, Frangioni A (2009) On the choice of explicit stabilizing terms in column generation. Discret Appl Math 157(6):1167–1184

Birgin E, Martínez J, Raydan M (2003) Inexact spectral projected gradient methods on convex sets. IMA J Numer Anal 23(4):539–559

Boyd, S, Duchi J, Pilanci M, Vandenberghe L (2022) Stanford EE 364b, lecture notes on subgradients. URL: https://web.stanford.edu/class/ee364b/lectures/subgradients_notes.pdf

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, Cambridge

Boyd S, Parikh N, Chu E (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends Mach Learn 3(1):1–122

Bradley A (2010) *Algorithms for the equilibration of matrices and their application to limited-memory Quasi-Newton methods*. PhD thesis, Stanford University, CA

Bruck R (1975) An iterative solution of a variational inequality for certain monotone operators in Hilbert space. Bull Am Math Soc 81:890–892

Burachik R, Martínez-Legaz J, Rezaie M, Théra M (2015) An additive subfamily of enlargements of a maximally monotone operator. Set-Valued Variat Anal 23:643–665

Burke J, Qian M (2000) On the superlinear convergence of the variable metric proximal point algorithm using Broyden and BFGS matrix secant updating. Math Program 88:157–181

Chen X, Fukushima M (1999) Proximal quasi-Newton methods for nondifferentiable convex optimization. Math Program 85(2):313–334

Chen G, Rockafellar R (1997) Convergence rates in forward-backward splitting. SIAM J Optim 7(2):421–444

Cheney E, Goldstein A (1959) Newton's method for convex programming and Tchebycheff approximation. Numer Math 1:253–268

Choi Y, Lim Y (2016) Optimization approach for resource allocation on cloud computing for IoT. Int J Distrib Sens Netw 12(3):3479247

Combettes P, Pesquet J-C (2011) Proximal splitting methods in signal processing. Fixed-point algorithms for inverse problems in science and engineering. Springer, Berlin, pp 185–212

Concus P, Golub G, Meurant G (1985) Block preconditioning for the conjugate gradient method. SIAM J Sci Stat Comput 6(1):220–252

Correa R, Lemaréchal C (1993) Convergence of some algorithms for convex minimization. Math Program 62:261–275

de Oliveira W, Solodov M (2016) A doubly stabilized bundle method for nonsmooth convex optimization. Math Program 156(1):125–159
de Oliveira W, Solodov M (2020) Bundle methods for inexact data. Numerical nonsmooth optimization. Springer, Berlin, pp 417–459
de Oliveira W, Sagastizábal C, Lemaréchal C (2014) Convex proximal bundle methods in depth: a unified analysis for inexact oracles. Math Program 148:241–277
de Oliveira W, Eckstein J (2015) A bundle method for exploiting additive structure in difficult optimization problems. Optimization Online
Dem'yanov V, Vasil'ev L (1985) Nondifferentiable optimization. Translations series in mathematics and engineering. Springer, New York
Diamond S, Boyd S (2016) CVXPY: a Python-embedded modeling language for convex optimization. J Mach Learn Res 17(83):1–5
Díaz M (2021) proximal-bundle-method. Julia software package available at https://github.com/mateodd25/proximal-bundle-method
Díaz M, Grimmer B (2023) Optimal convergence rates for the proximal bundle method. SIAM J Optim 33(2):424–454
Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12(7):2121–2159
Elzinga J, Moore T (1975) A central cutting plane algorithm for the convex programming problem. Math Program 8:134–145
Emiel G, Sagastizábal C (2010) Incremental-like bundle methods with application to energy planning. Comput Optim Appl 46(2):305–332
Fischer F (2022) An asynchronous proximal bundle method. Optimization Online
Frangioni A (2002) Generalized bundle methods. SIAM J Optim 13(1):117–156
Frangioni A (2020) Standard bundle methods: untrusted models and duality. Numerical nonsmooth optimization. Springer, Berlin, pp 61–116
Frangioni A, Gorgone E (2014) Bundle methods for sum-functions with "easy" components: applications to multicommodity network design. Math Program 145:133–161
Frangioni A, Gorgone E (2014) Generalized bundle methods for sum-functions with "easy" components: applications to multicommodity network design. Math Program 145:133–161
Fuduli A, Gaudioso M, Giallombardo G (2004) Minimizing nonconvex nonsmooth functions via cutting planes and proximity control. SIAM J Optim 14(3):743–756
Gonzaga C, Polak E (1979) On constraint dropping schemes and optimality functions for a class of outer approximations algorithms. SIAM J Control Optim 17(4):477–493
Grant M, Boyd S, Ye Y (2006) Disciplined convex programming. Global optimization. Springer, Berlin, pp 155–210
Haarala M, Miettinen K, Mäkelä M (2004) New limited memory bundle method for large-scale nonsmooth optimization. Optim Methods Softw 19(6):673–692
Haarala N, Miettinen K, Mäkelä M (2007) Globally convergent limited memory bundle method for large-scale nonsmooth optimization. Math Program 109:181–205
Han Z, Liu K (2008) Resource allocation for wireless networks: basics, techniques, and applications. Cambridge University Press, Cambridge
Hare W, Sagastizábal C, Solodov M (2016) A proximal bundle method for nonsmooth nonconvex functions with inexact information. Comput Optim Appl 63(1):1–28
Helmberg C, Rendl F (2000) A spectral bundle method for semidefinite programming. SIAM J Optim 10(3):673–696
Helmberg C, Pichler A (2017) Dynamic scaling and submodel selection in bundle methods for convex optimization. https://www.tu-chemnitz.de/mathematik/preprint/2017/PREPRINT_04.pdf
Hestenes M, Stiefel E et al (1952) Methods of conjugate gradients for solving linear systems. J Res Natl Bur Stand 49(6):409–436
Hintermüller M (2001) A proximal bundle method based on approximate subgradients. Comput Optim Appl 20(3):245–266
Hiriart-Urruty J-B, Lemaréchal C (1996) Convex analysis and minimization algorithms II: advanced theory and bundle methods. Grundlehren der mathematischen Wissenschaften. Springer, Berlin Heidelberg
Hiriart-Urruty J-B, Lemaréchal C (2013) Convex analysis and minimization algorithms I: fundamentals, vol 305. Springer Science & Business Media, Berlin
Iutzeler F, Malick J, de Oliveira W (2020) Asynchronous level bundle methods. Math Program 184:319–348

Jacobi C (1845) Ueber eine neue auflösungsart der bei der methode der kleinsten quadrate vorkommenden lineären gleichungen. Astron Nachr 22(20):297–306
Kairouz P, McMahan H, Avent B, Bellet A, Bennis M, Bhagoji A, Bonawitz K, Charles Z, Cormode G, Cummings R et al (2021) Advances and open problems in federated learning. Found Trends Mach Learn 14(1–2):1–210
Karmitsa N (2016) Proximal bundle method. http://napsu.karmitsa.fi/proxbundle/
Karmitsa N (2007) LMBM—FORTRAN subroutines for large-scale nonsmooth minimization: user's manual. TUCS Tech Rep 77:856
Karmitsa N, Mäkelä M (2010) Limited memory bundle method for large bound constrained nonsmooth optimization: convergence analysis. Optim Methods Softw 25(6):895–916
Kelley J (1960) The cutting-plane method for solving convex programs. J Soc Ind Appl Math 8(4):703–712
Kim K, Petra C, Zavala V (2019) An asynchronous bundle-trust-region method for dual decomposition of stochastic mixed-integer programming. SIAM J Optim 29(1):318–342
Kim K, Zhang W, Nakao H, Schanen M (2021) BundleMethod.jl: Implementation of Bundle Methods in Julia
Kiwiel K (1983) An aggregate subgradient method for nonsmooth convex minimization. Math Program 27:320–341
Kiwiel K (1985) An algorithm for nonsmooth convex minimization with errors. Math Comput 45(171):173–180
Kiwiel K (1990) Proximity control in bundle methods for convex nondifferentiable minimization. Math Program 46(1–3):105–122
Kiwiel K (1995) Approximations in proximal bundle methods and decomposition of convex programs. J Optim Theory Appl 84(3):529–548
Kiwiel K (1996) Restricted step and Levenberg–Marquardt techniques in proximal bundle methods for nonconvex nondifferentiable optimization. SIAM J Optim 6(1):227–249
Kiwiel K (1999) A bundle Bregman proximal method for convex nondifferentiable minimization. Math Program 85(2):241–258
Kiwiel K (2000) Efficiency of proximal bundle methods. J Optim Theory Appl 104(3):589–603
Kiwiel K (2006) A proximal bundle method with approximate subgradient linearizations. SIAM J Optim 16(4):1007–1023
Lemaréchal C (1978) Nonsmooth optimization and descent methods. IIASA Research Report, 78-4
Lemaréchal C (1975) An extension of Davidon methods to non differentiable problems. Math Program Study 3:95–109
Lemaréchal C (2001) Lagrangian relaxation. Computational combinatorial optimization. Springer, Berlin, pp 112–156
Lemaréchal C, Sagastizábal C (1994) An approach to variable metric bundle methods. System modelling and optimization. Springer, Berlin, pp 144–162
Lemaréchal C, Sagastizábal C (1997) Variable metric bundle methods: from conceptual to implementable forms. Math Program 76:393–410
Lemaréchal C, Nemirovskii A, Nesterov Y (1995) New variants of bundle methods. Math Program 69(1):111–147
Lemaréchal C, Ouorou A, Petrou G (2009) A bundle-type algorithm for routing in telecommunication data networks. Comput Optim Appl 44:385–409
Lemaréchal C, Sagastizábal C, Pellegrino F, Renaud A (1996) Bundle methods applied to the unit-commitment problem. In: System modelling and optimization: proceedings of the seventeenth IFIP TC7 conference on system modelling and optimization, 1995. Springer, Berlin, pp 395–402
Li T, Sahu AK, Talwalkar A, Smith V (2020) Federated learning: challenges, methods, and future directions. IEEE Signal Process Mag 37(3):50–60
Lions P, Mercier B (1979) Splitting algorithms for the sum of two nonlinear operators. SIAM J Numer Anal 16(6):964–979
Liu Y, Zhao S, Du X, Li S (2005) Optimization of resource allocation in construction using genetic algorithms. In: 2005 International conference on machine learning and cybernetics, vol 6, pp 3428–3432. IEEE
Lukšan L, Vlček J (1998) A bundle-Newton method for nonsmooth unconstrained minimization. Math Program 83:373–391
Lukšan L, Vlček J (1999) Globally convergent variable metric method for convex nonsmooth unconstrained minimization. J Optim Theory Appl 102:593–613

Lv J, Pang L, Meng F (2018) A proximal bundle method for constrained nonsmooth nonconvex optimization with inexact information. J Global Optim 70(3):517–549
Mäkelä M (2003) Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0. Reports of the Department of Mathematical Information Technology, Series B. Sci Comput B 13:2003
Mäkelä M, Karmitsa N, Wilppu O (2016) Proximal bundle method for nonsmooth and nonconvex multiobjective optimization. Math Model Optim Complex Struct, 191–204
Marsten R, Hogan W, Blankenship J (1975) The boxstep method for large-scale optimization. Oper Res 23(3):389–405
Mifflin R (1977) Semismooth and semiconvex functions in constrained optimization. SIAM J Control Optim 15(6):959–972
Mifflin R (1996) A quasi-second-order proximal bundle algorithm. Math Program 73(1):51–72
Nesterov Y (1983) A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. Proc USSR Acad Sci 269:543–547
Nocedal J, Wright S (1999) Numerical Optimization. Springer, Berlin
Ouorou A, Mahey P, Vial J-Ph (2000) A survey of algorithms for convex multicommodity flow problems. Manage Sci 46(1):126–147
Parikh N, Boyd S et al (2014) Proximal algorithms. Found Trends Optim 1(3):127–239
Passty G (1979) Ergodic convergence to a zero of the sum of monotone operators in Hilbert space. J Math Anal Appl 72(2):383–390
Rey P, Sagastizábal C (2002) Dynamical adjustment of the prox-parameter in bundle methods. Optimization 51(2):423–447
Rey P, Sagastizábal C (2002) Dynamical adjustment of the prox-parameter in bundle methods. Optimization 51(2):423–447
Rockafellar R (1981) The theory of subgradients and its applications to problems of optimization. Heldermann Verlag
Schechtman S (2022) Stochastic proximal subgradient descent oscillates in the vicinity of its accumulation set. Optim Lett, 1–14
Schramm H, Zowe J (1992) A version of the bundle idea for minimizing a nonsmooth function: conceptual idea, convergence analysis, numerical results. SIAM J Optim 2(1):121–152
Shor N (2012) Minimization methods for non-differentiable functions, vol 3. Springer Science & Business Media, Berlin
Sinkhorn R (1964) A relationship between arbitrary positive matrices and doubly stochastic matrices. Ann Math Stat 35(2):876–879
Sra S, Nowozin S, Wright S (2012) Optimization for machine learning. MIT Press, Cambridge
Takapoui R, Javadi H (2016) Preconditioning via diagonal scaling. arXiv preprint arXiv:1610.03871
Teo C, Vishwanathan S, Smola A, Le Q (2010) Bundle methods for regularized risk minimization. J Mach Learn Res, 11(1)
Trisna T, Marimin M, Arkeman Y, Sunarti T (2016) Multi-objective optimization for supply chain management problem: a literature review. Decis Sci Lett 5(2):283–316
van Ackooij W, Frangioni A (2018) Incremental bundle methods using upper models. SIAM J Optim 28:379–410
van Ackooij W, Frangioni A, de Oliveira W (2016) Inexact stabilized Benders' decomposition approaches with application to chance-constrained problems with finite support. Comput Optim Appl 65:637–669
van Ackooij W, Berge V, de Oliveira W, Sagastizábal C (2017) Probabilistic optimization via approximate $p$-efficient points and bundle methods. Comput Oper Res 77:177–193
Wei F, Zhang X, Xu J, Bing J, Pan G (2020) Simulation of water resource allocation for sustainable urban development: an integrated optimization approach. J Clean Prod 273:122537
Westerlund T, Pettersson F (1995) An extended cutting plane method for solving convex MINLP problems. Comput Chem Eng 19:131–136
Yin P, Wang J (2006) Ant colony optimization for the nonlinear resource allocation problem. Appl Math Comput 174(2):1438–1453
Zhou B, Bao J, Li J, Lu Y, Liu T, Zhang Q (2021) A novel knowledge graph-based optimization approach for resource allocation in discrete manufacturing workshops. Robot Comput Integr Manuf 71:102160