Mathematical Programming Computation (2021) 13:683–703
https://doi.org/10.1007/s12532-020-00197-0

FULL LENGTH PAPER

Benders decomposition with adaptive oracles for large scale optimization

Nicolò Mazzi[1] · Andreas Grothey[1] · Ken McKinnon[1] · Nagisa Sugishita[1]

Received: 5 April 2019 / Accepted: 21 October 2020 / Published online: 30 November 2020

Abstract
This paper proposes an algorithm to efficiently solve large optimization problems which exhibit a column bounded block-diagonal structure, where subproblems differ in right-hand side and cost coefficients. Similar problems are often tackled using cutting-plane algorithms, which allow for an iterative and decomposed solution of the problem. When solving subproblems is computationally expensive and the set of subproblems is large, cutting-plane algorithms may slow down severely. In this context we propose two novel adaptive oracles that yield inexact information, potentially much faster than solving the subproblem. The first adaptive oracle is used to generate inexact but valid cutting planes, and the second adaptive oracle gives a valid upper bound of the true optimal objective. These two oracles progressively "adapt" towards the true exact oracle if provided with an increasing number of exact solutions, stored throughout the iterations. These adaptive oracles are embedded within a Benders-type algorithm able to handle inexact information. We compare the Benders with adaptive oracles against a standard Benders algorithm on a stochastic investment planning problem. The proposed algorithm shows the capability to substantially reduce the computational effort to obtain an ϵ-optimal solution: an illustrative case is 31.9 times faster for a 1.00% convergence tolerance and 15.4 times faster for a 0.01% tolerance.

Keywords Benders decomposition · Inexact oracles · Adaptive oracles · Stochastic investment planning

Mathematics Subject Classification 90-08 Computational methods for problems pertaining to operations research and mathematical programming · 90-04 Software, source code, etc. for problems pertaining to operations research and mathematical programming · 90C06 Large-scale problems in mathematical programming · 90C15 Stochastic programming

Research is supported by the Engineering and Physical Sciences Research Council (EPSRC) through the CESI Project (EP/P001173/1).

Extended author information available on the last page of the article

1 Introduction

In this paper we consider problems that can be expressed in the form

$$\min_{\mathbf{x}\in\mathcal{X}} \quad f(\mathbf{x}) + \sum_{i\in\mathcal{I}} \pi_i\, g(x_i, c_i), \tag{1}$$

where $g(x_i, c_i)$ is the optimal solution of the LP subproblem

$$\mathbf{SP}: \qquad g(x_i, c_i) := \min_{y_i\in\mathcal{Y}} \{c_i^\top C y_i \mid Ay_i \le Bx_i\}. \tag{2}$$

The set of problems $\mathcal{I}$ is finite, the x_i are (possibly overlapping) subvectors of $\mathbf{x}$, $\mathcal{Y}$ is a convex polyhedron, and the π_i are non-negative constants. The coefficient matrices A, B, and C are the same in every subproblem so the subproblems differ only through the value of their parameters, x_i and c_i. The c_i are vectors of coefficients and the x_i are vectors of variables for (1) and parameters for (2). However, elements of x_i may be set to fixed values in $\mathbf{x} \in \mathcal{X}$, so become equivalent to parameters specific to subproblem i. The inclusion of the matrices B and C allow the dimensions of x_i and c_i to differ from (and in the case we shall present be much smaller than) the number of explicit constraints and variables y_i in each subproblem.

The function $g(x_i, c_i)$ has properties that can be exploited in the solution procedure. It is a saddle function, convex w.r.t. x_i, and concave w.r.t. c_i. Moreover, we focus on problems where $g(x_i, c_i)$ is also a decreasing function of x_i and an increasing function of c_i. Monotonicity can be a natural property of the problem, e.g. when $C \ge 0$, $y \ge 0$, and $B \ge 0$, or the problem can be usually rearranged to have this property.

An example of a problem with the above structure is an investment planning problem. Here the $\mathbf{x}$ represents investment decisions with corresponding investment cost $f(\mathbf{x})$. The investments affect a set $\mathcal{I}$ of situations, and x_i is the subvector of $\mathbf{x}$ that represents the investments that affect the situation i, c_i specifies the operational costs, y_i defines the operational decisions, and $g(x_i, c_i)$ gives the optimal operating cost. The test case we present in this paper is a stochastic problem, where the situations are the different possible scenarios that might occur in the future, each of which is weighted by the probability π_i of that situation occurring. Note that in a multistage planning problem the sum of the π_i at each stage is equal to 1, so $\sum_{i\in\mathcal{I}} \pi_i \ge 1$.

1.1 Literature review

In the standard Benders decomposition algorithms [2,16] for problem (1) a sequence of relaxations is solved. At each iteration j the relaxed master problem is

$$\textbf{RMP}: \min_{\mathbf{x}\in\mathcal{X},\beta} f(\mathbf{x}) + \sum_{i\in\mathcal{I}} \pi_i \beta_i$$
$$\text{s.t. } \beta_i \geq \theta + \lambda^\top (x_i - x), \qquad \forall (x,\theta,\lambda) \in \Theta_i^{(j-1)},\ \forall i \in \mathcal{I}, \tag{3}$$

where $\Theta_i^{(j-1)}$ is the set of cutting planes associated with subproblem i built up to iteration $j-1$. To generate a new cutting plane at iteration j, we first solve the **RMP** and obtain the optimal solution $\mathbf{x}^{(j)}$. Then, for each $i \in \mathcal{I}$ we call an oracle that, for given $x_i^{(j)}$, returns the optimal objective value $\theta_i^{(j)} = g(x_i^{(j)}, c_i)$ of **SP** and a subgradient $\lambda_i^{(j)}$ w.r.t. $x_i^{(j)}$. Such an oracle that generates an exact value of g and its subgradient will be referred to as *exact*. The newly generated cutting plane is added to $\Theta_i^{(j-1)}$ yielding $\Theta_i^{(j)} := \Theta_i^{(j-1)} \cup \{(x_i^{(j)}, \theta_i^{(j)}, \lambda_i^{(j)})\}$.

The standard Benders algorithm requires calling the exact oracle $|\mathcal{I}|$ times at each iteration j. Consequently, when the number of subproblems is large and the exact oracle is hard to solve (e.g., the problem (2) is very large), the computational efficiency of these methods is badly affected. This motivates the investigation of Benders-type algorithms able to exploit the information of *inexact* oracles. An inexact oracle only provides an estimate of $\theta_i^{(j)}$ and $\lambda_i^{(j)}$, but possibly much faster than an exact oracle.

Inexact oracles can have different characteristics. They may or may not guarantee to generate valid bounds and cutting planes. Additionally it may or may not be possible to control the accuracy of the oracle, and bounds on the approximation error may or may not be known. An example of an inexact oracle is the one proposed in [11]. The authors suggest solving only a subset $\mathcal{I}_{ex} \subset \mathcal{I}$ of subproblems and using fast techniques to evaluate approximate values of $\theta_i^{(j)}$ and $\lambda_i^{(j)}$ for subproblems $i \in \mathcal{I}\backslash\mathcal{I}_{ex}$. Using a concept of collinearity the authors group similar subproblems, solve one subproblem per group, and derive an approximate solution for the remaining subproblems of each group. Similar inexact oracles that do not provide a guaranteed bound can be found in [6–8,14]. Zakeri et al. [18] propose solving each subproblem $i \in \mathcal{I}$ up to a predefined tolerance which is then tightened over time to ensure convergence. The generated cutting planes are all valid and asymptotically exact. Similar inexact but always valid oracles are proposed by [4,5]. The use of any of the above types of oracles leads to algorithms that are fast at the early iterations but are potentially almost as slow as exact oracles towards the end of the iterative procedure. As a matter of fact, in order to obtain high accuracy, the approach of [11] would need to solve almost all the set of subproblems (i.e., $\mathcal{I}_{ex} \approx \mathcal{I}$), and the inexact oracle of [18] would solve the subproblems up to a very tight tolerance.

An alternative approach to constructing inexact oracles is to exploit some known properties of the subproblems that allow valid cuts to be generated for subproblems from solution of different problems. This approach has been used in stochastic dual dynamic programming (Pereira et al. [12]), for example by exploiting convexity of the recourse function w.r.t. the uncertain parameters. Stochastic dual dynamic programming is also used to solve minimax problems, which can arise from the inclusion of risk-aversion within the model. This leads to saddle functions similar to $g(x_i, c_i)$, i.e., convex in some directions (x_i) and concave in others (c_i). Philpott et al. [13] exploit this property and construct an upper bound on the recourse function using an inner

approximation. However, an inner approximation is infeasible if the point of interest is not in the convex hull of known points, so the authors propose to initially compute a solution for every extreme point of the uncertainty set. Work [1] proposes using penalties to deal with infeasibility and obtains valid lower and upper bounds to saddle functions without sampling all the vertices of the uncertainty set.

1.2 Approach and contributions

This paper proposes two new inexact oracles, which we refer to as adaptive oracles. The first adaptive oracle approximates $g(x_i, c_i)$ from below and it is called to generate inexact but valid cuts of those subproblems that are not solved at an iteration j. The second adaptive oracle approximates $g(x_i, c_i)$ from above and it is called to obtain a valid upper bound when a subproblem is not solved. The adaptive oracles exploit properties of $g(x_i, c_i)$ such as convexity w.r.t. x_i and concavity w.r.t. c_i. In addition, they require $g(x_i, c_i)$ to be a monotonic function of both x_i and c_i. The adaptive oracles use the knowledge of m solutions of $g(x_i, c_i)$, known by having solved some subproblems in the previous iterations. Increasing the number m of known solutions, makes both oracles progressively "adapt" towards the true function $g(x_i, c_i)$.

The proposed adaptive oracles are asymptotically exact oracles, since they provide valid upper and lower bounds of $g(x_i, c_i)$, a valid subgradient, and both bounds tend toward $g(x_i, c_i)$ as the number of iterations grows. However, they have properties that, combined, distinguish them from the inexact oracles available in the literature [5–8,10,11,14,15,17,18]. First, the computational effort required to solve the adaptive oracles is independent of the size and complexity of the exact oracle (2), and only depends on the number m of known solutions. Second, a Benders-type algorithm that uses the adaptive oracles converges to an ϵ-optimal solution ($\epsilon \geq 0$) in a finite number of iterations even when only a single subproblem is solved at each iteration. As a consequence, when subproblems are expensive to solve and the set $\mathcal{I}$ of subproblems is large, each iteration is much faster than methods that solve every subproblem at each iteration, and we show this can lead to a significant reduction in total solution time. We test our new method on stochastic LP investment planning problems with up to 1.06×10^8 variables and 3.11×10^8 constraints against a standard Benders algorithm and the Benders algorithm with inexact cuts of [18] (used as a benchmark). Compared to the standard Benders decomposition, the decomposition algorithm of [18] is 1.6, 1.7, and 1.9 times faster at reaching an optimality tolerance ϵ of 1.00%, 0.10%, and 0.01%, while our proposed algorithm is 31.9, 28.5, and 15.4 times faster than standard Benders and 19.5, 16.5, and 8.1 times faster than the Benders algorithm of [18] to reach the same ϵ.

1.3 Paper structure

The remainder of the paper is organized as follows. Section 2 introduces assumptions on problems (1) and (2), needed to apply our adaptive oracles. Section 3 briefly presents a standard formulation of the Benders decomposition algorithm. Section 4 illustrates the intuitions behind the proposed adaptive oracles and derives the associ-

ated mathematical formulation. The adaptive oracles are embedded within a Benders decomposition algorithm in Sect. 5. Section 6 tests the proposed Benders with adaptive oracles against a standard Benders algorithm on a stochastic investment planning problem. Finally, conclusions are drawn in Sect. 7.

2 Assumptions

We consider solving problems of the form of (1). The subproblem **SP** in (2) is assumed to be always feasible (any possible infeasibility having been dealt with by reformulating the subproblem using infeasibility penalties) and bounded for each decision $\mathbf{x}$ that belongs to $\mathcal{X}$ and for all c_i. Accordingly, we say that the function $g(x, c)$ can be computed for all $x \in \mathcal{K}^x$ and $c \in \mathcal{K}^c$, where the region $\mathcal{K}^x$ is obtained by collecting all the possible x_i such that $\mathbf{x} \in \mathcal{X}$, and the region $\mathcal{K}^c$ collecting all c_i for all $i \in \mathcal{I}$. Then, we assume that there exist two vectors $\underline{x}$ and $\underline{c}$ such that $\underline{x} \leq x, \ \forall x \in \mathcal{K}^x$, and $\underline{c} \leq c, \ \forall c \in \mathcal{K}^c$, and that subproblem **SP** is feasible and bounded at the special point $(\underline{x}, \underline{c})$.

Function $g(x, c)$ is a convex function of x and a concave function of c (e.g., see [3]). In addition, we assume that $g(x, c)$ is a non-increasing function of x, and a non-decreasing function of c. If the property of monotonicity does not hold naturally, problem (1) can be modified so that the rearranged subproblem is a monotonic function of both x and c (see "Appendix A").

3 Standard benders

This section briefly describes how a Benders algorithm can exploit the block structure of problem (1) allowing it to be solved in a decomposed fashion.

3.1 Algorithm

In a standard Benders algorithm applied to problem (1), we iteratively approximate the subproblem cost function through a set of cutting planes. The formulation of the relaxed master problem **RMP** is given in (3) and the standard Benders decomposition is summarized in Algorithm 1.

Algorithm 1: `Stand_Bend` (Standard Benders)

choose $\epsilon \geq 0$ (convergence tolerance);
choose $\underline{\beta}$ (a priori lower bound for each β_i);
set $j := 0$ and $\Theta_i^{(0)} := \{(0, \underline{\beta}, 0)\}$ for each $i \in \mathcal{I}$;
repeat
  set $j := j + 1$;
  solve **RMP** and obtain $\mathbf{x}^{(j)}$ and $\beta^{(j)}$;
  set $\mathrm{L}^{(j)} := f(\mathbf{x}^{(j)}) + \sum_{i \in \mathcal{I}} \pi_i \beta_i^{(j)}$;
  for $i \in \mathcal{I}$ **do**
    solve subproblem i at $(x_i^{(j)}, c_i)$ and obtain $\theta_i^{(j)}$ and $\lambda_i^{(j)}$;
  end
  set $\mathrm{U}^{(j)} := \min\left(\mathrm{U}^{(j-1)}, f(\mathbf{x}^{(j)}) + \sum_{i \in \mathcal{I}} \pi_i \theta_i^{(j)}\right)$;
  for $i \in \mathcal{I}$ **do**
    $\Theta_i^{(j)} := \Theta_i^{(j-1)} \cup \{(x_i^{(j)}, \theta_i^{(j)}, \lambda_i^{(j)})\}$;
  end
until $\mathrm{U}^{(j)} - \mathrm{L}^{(j)} \leq \epsilon$;

3.2 Convergence of Algorithm 1

Definition 1 Let $\mathcal{L}_i$ be the set of cutting planes $\{(x_i^{(\ell)}, \theta_i^{(\ell)}, \lambda_i^{(\ell)}), \forall \ell \in \mathcal{L}_i\}$ associated with subproblem i already added to the relaxed master problem. We say that a new exact cut (x, θ, λ) is *locally-improving* if and only if

$$\theta > \max_{\ell \in \mathcal{L}_i} \left\{ \theta_i^{(\ell)} + \lambda_i^{(\ell)\top} \left(x - x_i^{(\ell)} \right) \right\}.$$

Lemma 1 *At each iteration, either* Algorithm *terminates with an ϵ-optimal solution, or the algorithm adds at least one locally-improving exact cut to the reduced master problem.*

Proof If none of the exact cuts $(x_i^{(j)}, \theta_i^{(j)}, \lambda_i^{(j)})$ generated at iteration j is locally-improving, it follows that

$$\theta_i^{(j)} \leq \max_{\ell < j} \left\{ \theta_i^{(\ell)} + \lambda_i^{(\ell)\top} \left(x_i^{(j)} - x_i^{(\ell)} \right) \right\}, \quad \forall i \in \mathcal{I}, \tag{4}$$

and given that the right side of (4) is equal to $\beta_i^{(j)}$, it follows that the lower bound and the upper bound have converged exactly. □

Lemma 2 *There exists a finite number of locally-improving exact cutting planes that can be added to the relaxed master problem.*

Proof Since each subproblem $g(x_i, c_i)$ is an LP, there exist a finite number of faces (of all dimensions from 0, i.e. vertices, edges, …, facets), and each exact cutting plane gives an exact representation of (at least) one of these faces. A new exact cut can be locally-improving if and only if it is associated with a face that has not been exactly

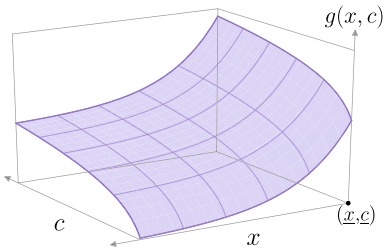

Fig. 1 Illustration of the saddle function $g(x, c)$, convex and non-increasing w.r.t x and concave and non-decreasing w.r.t. c

represented yet. Then, given that the number $|\mathcal{I}|$ of subproblems is finite, it follows that there exists a finite number of locally-improving exact cuts that can be added to the relaxed master problem. □

Theorem 1 *For any* $\epsilon \geq 0$, Algorithm 1 *finds an* ϵ*-optimal solution to problem* (1) *in a finite number of iterations.*

Proof Lemma 2 proves that there exists a finite number of locally-improving exact cuts that can be added to the reduced master problem, and Lemma 1 proves that Algorithm 1 adds at least one locally-improving exact cut at each iteration (or it has converged). It follows that Algorithm 1 finds an ϵ-optimal solution to problem (1) in a finite number of iterations. □

Remark 1 Convergence proofs for Benders decomposition (e.g., see [18]) usually rely on the existence of a finite number of basis matrices for each subproblem. In contrast, Theorem 1 relies on the existence of a finite number of faces and does not require that each exact cut corresponds to a basis matrix.

4 Adaptive oracles

This section presents the adaptive oracles used within the proposed novel Benders-type decomposition algorithm. Figure 1 illustrates a saddle function $g(x, c)$, convex and non-increasing w.r.t x, and concave and non-decreasing w.r.t. c. This function is used to illustrate the intuition behind the proposed adaptive oracles.

4.1 Graphical interpretation of the adaptive oracles

For a subproblem i that is not solved at iteration j, we call two adaptive oracles to retrieve the information needed to perform a Benders-type iteration. The first adaptive oracle provides $\underline{\theta}_i^{(j)}$ and $\underline{\lambda}_i^{(j)}$ such that $(x_i^{(j)}, \underline{\theta}_i^{(j)}, \underline{\lambda}_i^{(j)})$ generates a valid cutting plane. The second adaptive oracle yields an upper bound $\overline{\theta}_i^{(j)}$ on $g(x_i^{(j)}, c_i)$, which is then used to compute a valid upper bound on the optimal solution of the relaxed master problem at $\mathbf{x}^{(j)}$.

Before presenting the mathematical formulation of the two adaptive oracles, we give a graphic and explanatory example using the saddle function $g(x, c)$ shown in Fig. 1. Figure 2 illustrates how to obtain a valid lower bound on $g(x, c)$, and Fig. 2

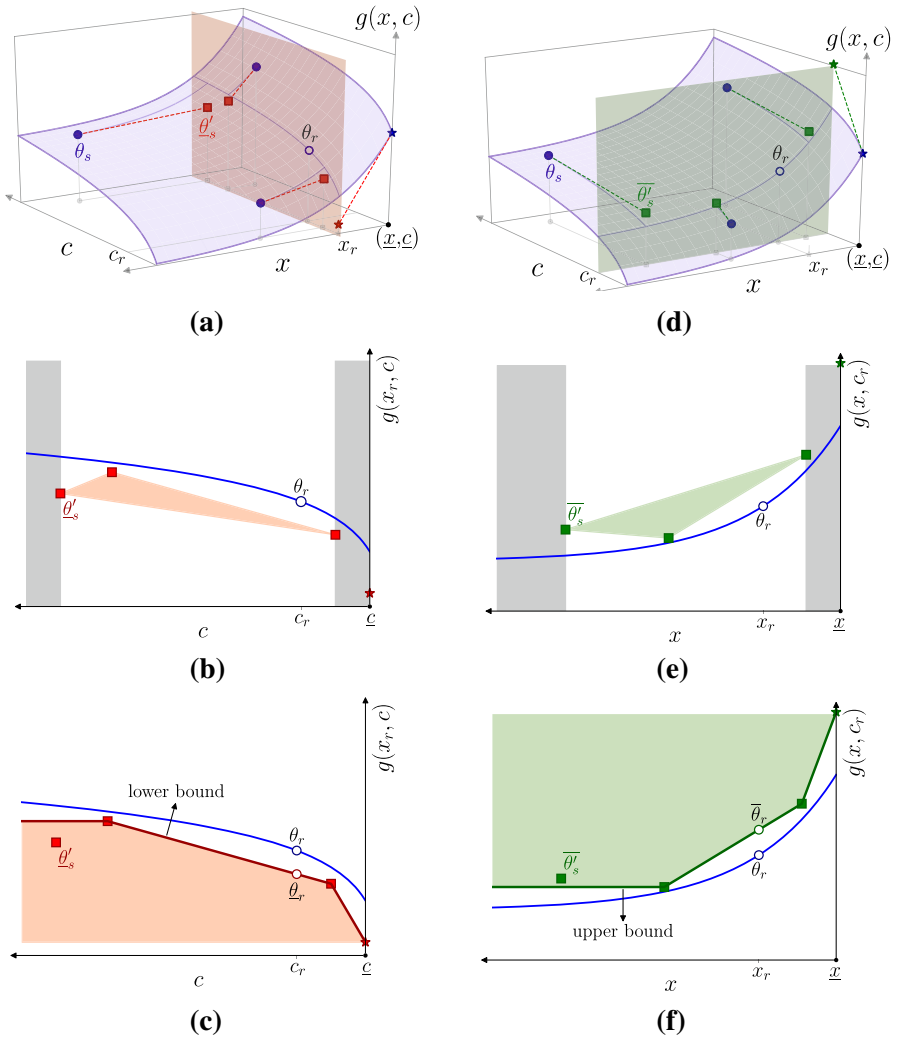

Fig. 2 Illustrative example of how to obtain a valid lower and upper bound $\underline{\theta}_r$ on $\theta_r = g(x_r, c_r)$

shows how to obtain a valid upper bound on $g(x, c)$. Assume that we know the exact value $\theta_s = g(x_s, c_s)$ at points $\{(x_s, c_s), s = 1, \ldots, m\}$, shown with blue filled dots in Figs. 2a and d, and we want to obtain valid lower and upper bounds on $g(x, c)$ at a new point (x_r, c_r), shown with a blue empty dot. Note that Figs. 2a and d also include the special point $(\underline{x}, \underline{c})$, marked with a blue filled star, which is needed at the end of the examples.

Each red dashed line in Fig. 2a shows the tangent at a blue point (x_s, c_s) w.r.t. x (i.e., for fixed c_s). Since $g(x, c_s)$ is convex w.r.t. x, the tangent lies below $g(x, c_s)$. The red squares are the points $(\underline{\theta}'_s, x_r, c_s)$ on the tangent where $x = x_r$. If the gradients are λ_s, then $\underline{\theta}'_s = \theta_s + \lambda_s^\top (x_r - x_s)$. Since $g(x_r, c)$ is concave w.r.t. c and all the

red square points lie in $x = x_r$ plane and below $g(x_r, c)$, the convex hull of these points also lies below $g(x_r, c)$. Figure 2b illustrates this convex hull as a red shaded area, and the gray shaded areas indicate values of c where a convex combination can not be found. Figure 2c shows that if the special point $(\underline{x}, \underline{c})$ is included among the known points (x_s, c_s), the non-decreasing property of $g(x, c)$ w.r.t. c can be exploited to extend the convex hull and obtain a valid lower bound for all $c \geq \underline{c}$. In particular, the upper envelope of the convex hull gives the tightest lower bound on $g(x_r, c)$.

The green dashed lines in Fig. 2d are the tangents (with gradient ϕ_s) at each blue point (x_s, c_s) w.r.t. c, which lie above $g(x_s, c)$ since $g(x_s, c)$ is concave w.r.t. c. The green squares $(\overline{\theta'_s}, x_s, c_r)$ are the points on the tangents where $c = c_r$, i.e., $\overline{\theta'_s} = \theta_s + \phi_s^\top (c_r - c_s)$. The convex hull of these points lies above $g(x, c_r)$, given that $g(x, c_r)$ is convex w.r.t. x and the green square points lie above $g(x, c_r)$. Figure 2e shows this convex hull as a green shaded area, likewise two gray shaded areas where a convex combination can not be found. If the special point $(\underline{x}, \underline{c})$ is included we extend the convex hull and obtain a valid upper bound for all $x \geq \underline{x}$ by exploiting the non-increasing property of $g(x, c)$ w.r.t. x (see Fig. 2f). Here, the lower envelope of the convex hull gives the tightest upper bound on $g(x, c_r)$.

Note that the methodology for obtaining a valid upper bound is the counterpart of the methodology for obtaining a valid lower bound. As an example, one could use the lower bounding oracle to get a valid lower bound on $g(x, c)$ and the same oracle to also obtain a valid upper bound. To do that, one can compute a valid lower bound on $\hat{g}(c, x) = -g(x, c)$ and use it (with a change in sign) as a valid upper bound on $g(x, c)$.

4.2 Adaptive oracles

This section gives a mathematical formulation of the intuitions illustrated in Sect. 4.1. Consider the saddle function $g(x, c)$, and suppose we have already found m optimal solutions at $\{(x_s, c_s), s = 1, .., m\}$. At each point s, we know the true value θ_s, a subgradient λ_s w.r.t. x, and a subgradient ϕ_s w.r.t. c. We are interested in building a valid cutting plane and obtaining a valid upper bound at a new point (x, c), without solving the associated subproblem.

Adaptive oracle for valid cutting plane

Let $\mathcal{A}_m^{\text{LB}}(x, c)$ be defined as

$$\mathcal{A}_m^{\text{LB}}(x, c): \quad \underline{\theta}(x, c) = \max_{\mu \geq 0} \theta(x, c) = \sum_{s=1}^{m} \mu_s \left(\theta_s + \lambda_s^\top (x - x_s)\right)$$
$$\text{s.t.} \sum_{s=1}^{m} \mu_s c_s \leq c, \quad \sum_{s=1}^{m} \mu_s = 1. \tag{5}$$

Lemma 3 *Assume that the set of known points* $\{(x_s, c_s), s = 1, \ldots, m\}$ *includes the special point* $(\underline{x}, \underline{c})$*, and let* $\mu(x, c)$ *be a feasible solution of* (5) *and* $\lambda(x, c) = \sum_{s=1}^{m} \mu_s(x, c)\lambda_s$*. Then,*

(a) $\mathcal{A}_m^{\mathrm{LB}}(x, c)$ *is feasible for all* $x \in \mathcal{K}^x$, $c \in \mathcal{K}^c$,
(b) $\theta(x, c) \leq \underline{\theta}(x, c) \leq g(x, c)$, *for all* $x \in \mathcal{K}^x$, $c \in \mathcal{K}^c$,
(c) $\underline{\theta}(x_r, c_r) = g(x_r, c_r)$, *for all* $r = 1, .., m$,
(d) $\theta(x, c) + \lambda(x, c)^\top \left(\hat{x} - x\right) \leq g(\hat{x}, c)$, *for all* $\hat{x} \in \mathcal{K}^x$, $x \in \mathcal{K}^x$, $c \in \mathcal{K}^c$.

Proof (a) Set the variable μ_s associated with $(\underline{x}, \underline{c})$ equal to 1, and all the others to 0. The first constraint of (5) becomes $\underline{c} \leq c$, which is true by definition, and the other constraints also hold.

(b) The definition of $\theta(x, c)$ leads to

$$\begin{aligned}\theta(x, c) = \sum_{s=1}^{m} \mu_s(x, c)\,(\theta_s + \lambda_s^\top (x - x_s)) \leq \sum_{s=1}^{m} \mu_s(x, c) g(x, c_s) \\ \leq g(x, \sum_{s=1}^{m} \mu_s(x, c) c_s) \\ \leq g(x, c).\end{aligned}$$

The first inequality holds since $\theta_s + \lambda_s^\top (x - x_s)$ is an underestimator of the convex function $g(x, c_s)$ and $\mu(x, c) \geq 0$, the second inequality holds since the $\mu(x, c)$ define a convex combination and $g(x, c)$ is concave w.r.t. c, and the third holds since $\sum_{s=1}^{m} \mu_s(x, c)\, c_s \leq c$ and $g(x, c)$ is non-decreasing w.r.t. c.

(c) Setting $\mu_r = 1$ gives a feasible solution with objective value $\theta_r + \lambda_r^\top (x_r - x_r) = \theta_r$. Since μ_r is feasible for $\mathcal{A}_m^{\mathrm{LB}}(x_r, c_r)$, it follows that $\underline{\theta}(x_r, c_r) \geq \theta_r = g(x_r, c_r)$. This and *b*) imply $\underline{\theta}(x_r, c_r) = g(x_r, c_r)$.

(d) Since the weights $\mu(x, c)$ are feasible for $\mathcal{A}_m^{\mathrm{LB}}(x, c)$, they are also feasible for $\mathcal{A}_m^{\mathrm{LB}}(\hat{x}, c)$. Hence,

$$\begin{aligned} g(\hat{x}, c) \geq & \sum_{s=1}^{m} \mu_s(x, c)\,(\theta_s + \lambda_s^\top \left(\hat{x} - x_s\right)) \\ = & \sum_{s=1}^{m} \mu_s(x, c)\,(\theta_s + \lambda_s^\top (x - x_s)) + \sum_{s=1}^{m} \mu_s(x, c)(\lambda_s^\top \left(\hat{x} - x\right)) \\ = & \ \theta(x, c) + \lambda(x, c)^\top \left(\hat{x} - x\right). \end{aligned}$$

The first equality follows since

$$\lambda_s^\top \left(\hat{x} - x_s\right) = \lambda_s^\top (x - x_s) + \lambda_s^\top \left(\hat{x} - x\right),$$

and the second from the definition of $\theta(x, c)$ and $\lambda(x, c)$. □

Adaptive oracle for valid upper bound

Let $\mathcal{A}_m^{\mathrm{UB}}(x, c)$ be defined as

$$\mathcal{A}_m^{\mathrm{UB}}(x, c): \quad \overline{\theta}(x, c) = \min_{\nu \geq 0} \theta(x, c) = \sum_{s=1}^{m} \nu_s \left(\theta_s + \phi_s^{\top}(c - c_s)\right)$$
$$\text{s.t.} \sum_{s=1}^{m} \nu_s x_s \leq x, \quad \sum_{s=1}^{m} \nu_s = 1. \tag{6}$$

Lemma 4 *Assume that the set of known points* $\{(x_s, c_s), s = 1, \ldots, m\}$ *includes the special point* $(\underline{x}, \underline{c})$. *Then,*

(a) $\mathcal{A}_m^{\mathrm{UB}}(x, c)$ *is feasible for all* $x \in \mathcal{K}^x$, $c \in \mathcal{K}^c$,
(b) $\theta(x, c) \geq \overline{\theta}(x, c) \geq g(x, c)$, *for all* $x \in \mathcal{K}^x$, $c \in \mathcal{K}^c$,
(c) $\overline{\theta}(x_r, c_r) = g(x_r, c_r)$, *for all* $r = 1, .., m$.

Proof Lemma 4 can be proved similarly to Lemma 3 (parts a, b, and c) since obtaining a valid upper bound is the exact counterpart than obtaining a valid lower bound. □

Notes on adaptive oracles

Note that every feasible solution μ of $\mathcal{A}_m^{\mathrm{LB}}(x, c)$ gives a valid cutting plane (x, θ, λ). Of these possible cuts, the one corresponding to the optimal solution, i.e., $(x, \underline{\theta}, \underline{\lambda})$, is the tightest at x. If $\mathcal{A}_m^{\mathrm{LB}}(x, c)$ and/or $\mathcal{A}_m^{\mathrm{UB}}(x, c)$ are solved to a feasible but not optimal solution, the generated cutting plane and upper bound are still valid.

A graphical interpretation of $\underline{\theta}$ and θ of $\mathcal{A}_m^{\mathrm{LB}}(x, c)$ is shown in Fig. 2c, where the set $(\underline{\theta}, c)$ is the red continuous curve, and the set (θ, c) is the red shaded area. Figure 2f gives an interpretation of $\overline{\theta}$ and θ of $\mathcal{A}_m^{\mathrm{UB}}(x, c)$, where the set $(\overline{\theta}, c)$ is the green continuous curve, and the set (θ, c) is the green shaded area.

5 Benders decomposition with adaptive oracles

This section presents the Benders-type algorithm incorporating the inexact information of the adaptive oracles of Sect. 4.

5.1 Convergence of Algorithm 2

Lemma 5 *At each iteration, either* Algorithm 2 *finds an* ϵ*-optimal solution, or the algorithm adds at least one locally-improving exact cut to the reduced master problem.*

Proof At iteration j Algorithm 2 adds at least one locally-improving exact cut to the reduced master problem ($\xi^{(j)} = 1$) or all the subproblems are solved for the current solution ($\mathcal{E}^{(j)} = \varnothing$). If none of the exact cuts $(x_i^{(j)}, \theta_i^{(j)}, \lambda_i^{(j)})$ generated at iteration

Algorithm 2: `Adapt_Bend` (Benders with Adaptive Oracles)

choose ϵ (convergence tolerance);
choose $\underline{\beta}$ (lower bound for each β_i);
set $j := 0$ and $\Theta_i^{(0)} := \{(0, \underline{\beta}, 0)\}$ for each $i \in \mathcal{I}$;
solve $g(\underline{x}, \underline{c})$ to obtain θ, λ, and ϕ;
set $\mathcal{S} := \{(\underline{x}, \underline{c}, \theta, \lambda, \phi)\}$;
repeat
  set $j := j + 1$;
  solve **RMP** and obtain $\mathbf{x}^{(j)}$ and $\beta^{(j)}$;
  set $\underline{\mathrm{L}}^{(j)} := f(\mathbf{x}^{(j)}) + \sum_{i \in \mathcal{I}} \pi_i \beta_i^{(j)}$;
  set $\xi^{(j)} := 0$ and $\mathcal{E}^{(j)} := \mathcal{I}$;
  repeat
    set $\mathcal{I}_{ex} :=$ non-empty subset of $\mathcal{E}^{(j)}$;
    set $\mathcal{E}^{(j)} := \mathcal{E}^{(j)} \backslash \mathcal{I}_{ex}$;
    for $i \in \mathcal{I}_{ex}$ **do**
      solve subproblem i at $(x_i^{(j)}, c_i)$ and obtain $\theta_i^{(j)}$, $\lambda_i^{(j)}$, and $\phi_i^{(j)}$;
      if $(x_i^{(j)}, \theta_i^{(j)}, \lambda_i^{(j)})$ is locally-improving **then** $\xi^{(j)} := 1$ set
      $\mathcal{S} := \mathcal{S} \cup \{(x_i^{(j)}, c_i, \theta_i^{(j)}, \lambda_i^{(j)}, \phi_i^{(j)})\}$;
      set $\underline{\theta}_i^{(j)} := \theta_i^{(j)}$, $\underline{\lambda}_i^{(j)} := \lambda_i^{(j)}$, and $\overline{\theta}_i^{(j)} := \theta_i^{(j)}$;
    end
  until $\xi^{(j)} = 1$ **or** $\mathcal{E}^{(j)} = \varnothing$;
  for $i \in \mathcal{E}^{(j)}$ **do**
    solve $\mathcal{A}^{\mathrm{LB}}_{|\mathcal{S}|}(x_i^{(j)}, c_i)$ and obtain $\underline{\theta}_i^{(j)}$ and $\underline{\lambda}_i^{(j)}$;
    solve $\mathcal{A}^{\mathrm{UB}}_{|\mathcal{S}|}(x_i^{(j)}, c_i)$ and obtain $\overline{\theta}_i^{(j)}$;
  end
  set $\overline{\mathrm{U}}^{(j)} := \min\left(\overline{\mathrm{U}}^{(j-1)}, f(\mathbf{x}^{(j)}) + \sum_{i \in \mathcal{I}} \pi_i \overline{\theta}_i^{(j)}\right)$;
  for $i \in \mathcal{I}$ **do**
    set $\Theta_i^{(j)} := \Theta_i^{(j-1)} \cup \{(x_i^{(j)}, \underline{\theta}_i^{(j)}, \underline{\lambda}_i^{(j)})\}$
  end
until $\overline{\mathrm{U}}^{(j)} - \underline{\mathrm{L}}^{(j)} \leq \epsilon$;

j is locally-improving ($\xi^{(j)} = 0$), it follows that

$$\theta_i^{(j)} \leq \max_{\ell < j} \left\{ \underline{\theta}_i^{(\ell)} + \underline{\lambda}_i^{(\ell)\top} \left(x_i^{(j)} - x_i^{(\ell)} \right) \right\}, \quad \forall i \in \mathcal{I}, \tag{7}$$

and given that the left- and right-hand sides of (7) are equal to $\overline{\theta}_i^{(j)}$ and $\beta_i^{(j)}$, respectively, it follows that the lower bound and the upper bound have converged exactly. □

Theorem 2 *For any $\epsilon \geq 0$, Algorithm 2 finds an ϵ-optimal solution to problem (1) in a finite number of iterations.*

Proof Lemma 2 proves that there exists a finite number of locally-improving exact cuts that can be added to the reduced master problem, and Lemma 5 proves that Algorithm 2 adds at least one locally-improving exact cut at each iteration (or it has converged). It

follows that Algorithm 2 finds an ϵ-optimal solution to problem (1) in a finite number of iterations. □

6 Case Study

We test the proposed Benders algorithm with adaptive oracles on power system stochastic investment planning problems. Serial and parallel versions of the algorithms are implemented in JULIA 1.4. Two Linux Desktop computers Intel i7 6-core processor clocking at 2.40 GHz and 16 GB of RAM are used for running the code. The serial implementation is run on a single machine, and the parallel implementation uses both machines simultaneously. The optimization models are implemented in JUMP (JULIA package) and solved with GUROBI 9.0. The JULIA code implementing Algorithms 1 (`Stand_Bend`) and 2 (`Adapt_Bend`) for the proposed case study is provided at https://github.com/nimazzi/Stand_and_Adapt_Bend [9].

6.1 Investment planning model

We consider a power system investment planning problem with a time horizon of 15 years. The deterministic version of the problem has 3 decision nodes: one refers to decisions to be taken in the first stage, one to decisions in 5 years time, and one to decision in 10 years time. The stochastic version is obtained by modeling different possible scenarios for the future of the system in 5 and 10 years. At each node we also compute the cost of operating the system for the following 5 years for given installed capacity. We consider a construction time of 5 years, so new assets installed in the first stage will only be available in 5 and 10 years, and new capacity installed in 5 years will only be available in 10 years. We model a set $\mathcal{P}$ of technologies: 6 thermal units, 3 storage units, and 3 renewable generation units. The cost for operating the system is computed by solving an hourly economic dispatch for 5 years.

We formulate the stochastic investment planning problem as

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) + \sum_{i\in\mathcal{I}} \pi_i \, g(x_i, c_i), \tag{8}$$

where $\mathcal{I}$ is the set of stochastic decision nodes, each associated with a probability π_i. The function

$$g(x_i, c_i) = \min_{y_i\in\mathcal{Y}} \{c_i^\top C y_i \mid A y_i \leq B x_i\}, \quad \forall i \in \mathcal{I}, \tag{9}$$

gives the cost of operating the system over 5 years. The vector of parameters x_i is given by

$$x_i = \left(\left\{ x_{pi}^{acc}, \forall p \in \mathcal{P} \right\}, -v_i^D, v_i^E \right), \quad \forall i \in \mathcal{I},$$

where x_{pi}^{acc} is the accumulated capacity of technology p at node i. Parameters v_i^D and v_i^E are the relative level of energy demand and the yearly CO_2 emission limit, respectively. Note that $g(x_i, c_i)$ is non-increasing w.r.t. x_i^{acc} and v_i^E, and non-decreasing w.r.t. v_i^D, so using $-v_i^D$ instead satisfies the non-increasing requirement. The vector of uncertain cost coefficients c_i is

$$c_i = (c_i^{nucl}, c_i^{co_2}), \quad \forall i \in \mathcal{I},$$

where c_i^{nucl} is the uranium fuel price and $c_i^{co_2}$ the CO_2 emission price, and $g(x_i, c_i)$ is non-decreasing w.r.t. both c_i^{nucl} and $c_i^{co_2}$. All the remaining parameters, e.g., A, B, and C, are the same for every node $i \in \mathcal{I}$. Finally, the function $f(\mathbf{x})$ yields the expected total investment and fixed cost, and it is computed as

$$f(\mathbf{x}) = \sum_{i \in \mathcal{I}} \pi_i \sum_{p \in \mathcal{P}} \left(c_{pi}^{inv} x_{pi}^{inst} + c_{pi}^{fix} x_{pi}^{acc} \right),$$

where the variable x_{pi}^{inst} is the newly installed capacity of technology p at node i. Parameters c_{pi}^{inv} and c_{pi}^{fix} are the unitary investment and fixed costs of technology p at node i. The accumulated capacity x_{pi}^{acc} at node i is computed as the sum of the historical capacity x_{pi}^{hist} and the newly installed capacity $x_{pi'}^{inst}$ at nodes i' ancestors to i. Finally, the initial special point $(\underline{x}, \underline{c})$ is set to

$$\underline{x}, \underline{c} = \left(\left\{ \min_i x_{pi}^{hist}, \forall p \in \mathcal{P} \right\}, -\max_i v_i^D, \min_i v_i^E \right), \left(\min_i c_i^{nucl}, \min_i c_i^{co_2} \right)$$

We consider three possible sources of uncertainty, i.e., v_i^E, $c_i^{co_2}$, and c_i^{nucl}. Each uncertain parameter has 3 possible outcomes in 5 years, each of which is linked to 3 additional possible outcomes in 10 years. The result is 9 possible trajectories for each source of uncertainty, all with the same probability. We consider 4 different cases of the investment problem. Case 0 is the deterministic version, where v_i^E, $c_i^{co_2}$, and c_i^{nucl} are deterministic parameters (weighted average of the scenarios). Then, case 1 has 1 uncertain parameter (v_i^E), case 2 has 2 uncertain parameters (v_i^E and $c_i^{co_2}$), and case 3 has 3 uncertain parameters (v_i^E, $c_i^{co_2}$, and c_i^{nucl}). The number of decision nodes and the size of the deterministic equivalent for the 4 versions of the problem is summarized in Table 1. As a benchmark, we try to solve the deterministic equivalent problem for cases 0, 1, 2, and 3 on our Linux Desktop computer. Case 0 is successfully solved in 4 minutes, and case 1 in 58 minutes. The deterministic equivalent of cases 2 and 3 is not solved since the JULIA instance is killed due to a memory overload.

6.2 Results

We use Algorithms 1 (`Stand_Bend`) and 2 (`Adapt_Bend`) to solve the stochastic investment planning problem (8), whose operational subproblems (9) are LP problems with 4.11×10^5 constraints and 1.40×10^5 variables. All the subproblems are solved

Table 1 Decision nodes $\mathcal{I}$ and size of the deterministic equivalent problem for different cases of the study

Case	Number of decision nodes			$\lvert\mathcal{I}\rvert$	Variables	Constraints
	Present	In 5 years	In 10 years			
0	1	1	1	3	2.80×10^5	8.24×10^5
1	1	3	9	13	1.68×10^6	4.94×10^6
2	1	9	81	91	1.26×10^7	3.71×10^7
3	1	27	729	757	1.06×10^8	3.11×10^8

with GUROBI barrier method (`"Method"=2`), while we do not impose a predefined solution method for the **RMP** and the oracles (`"Method"=-1`). In the serial implentation of the algorithms we set `"Threads"=0` for the **RMP**, the subproblems, and the oracles, i.e., GUROBI decides the amount of threads to use. In the parallel implementation of the algorithms we impose `"Threads"=1` for the subproblems and the oracles.

For Algorithm 2 at each iteration j we choose the set $\mathcal{I}_{ex} = \{i_k,\ k = 1, .., w\}$ of w subproblems that are solved exactly. Each element i_k is the (or one of the) $i \in \mathcal{E}_k^{(j)}$ for which the difference $\pi_i\overline{\theta}_i^{(j-1)} - \pi_i\underline{\theta}_i^{(j-1)}$ is maximum, where the $\mathcal{E}_k^{(j)}$ form a partition of $\mathcal{E}^{(j)}$. The idea is to group in each subset $\mathcal{E}_k^{(j)}$ subproblems that are potentially similar to each others, to diversify the exact solutions that are added to the oracles. To select the w subsets of $\mathcal{E}^{(j)}$ we use the `kmeans` function of the CLUSTERING package with $(v_i^D, v_i^E, c_i^{nucl}, c_i^{\mathrm{co}_2})$ as the input parameters.

As a benchmark, we also implement the Benders algorithm with inexact cuts as presented in [18], and we refer to it as Algorithm 3 or `Zaker_Bend`. The core idea is to solve subproblems up to primal-dual feasible solution with an optimality gap lower than $\delta^{(j)}$ which is large for early iterations and progressively tightened ($\delta^{(j)} \to 0$ for $j \to \infty$) as the algorithm approaches the optimal solution. The dual solution and dual objective are used to build valid cutting planes, and the primal objective is used to build a valid upper bound. To obtain such primal-dual feasible solution, we solve subproblems with barrier method, we disable crossover (`"Crossover"=0`), and we impose the optimality gap to $\delta^{(j)}$ (`"BarConvTol"`$=\delta^{(j)}$), in accordance with [18]. The optimality gap is set to 1 (maximum value of `"BarConvTol"`) for the first iteration, i.e., $\delta^{(1)} = 1$, then we use the update rule $\delta^{(j)} := \frac{1}{10}\delta^{(j-1)}$ suggested by [18]. We set a minimum threshold for $\delta^{(j)}$ to 10^{-8} which is the default value for `"BarConvTol"`.

Table 2 gives the optimal investment decisions to be taken in the first investment stage for cases 0–3. The solutions are obtained solving (8) with Algorithm 1 up to an 0.01%-optimal solution. Including a more accurate description of the stochastic processes involved in the decision-making process progressively modifies the optimal decisions of the system planner. Using the full stochastic model (case 3) the planner takes considerably different decisions compared to the ones yielded by its deterministic counterpart (case 0). In case 0 the system planner builds 13.7 GW of CCGT, 1.4 GW of diesel, and 19.0 GW of onshore wind. In case 3 the investment in CCGT (11.4 GW)

Table 2 Optimal investments in the first investment stage for cases 0–3

Tech. $p \in \mathcal{P}$	Type	New installed capacity (GW)			
		Case 0	Case 1	Case 2	Case 3
Coal	Thermal	0.0	0.0	0.0	0.0
Coal&CCS	Thermal	0.0	0.0	0.8	1.2
OCGT	Thermal	0.0	0.0	0.0	0.0
CCGT	Thermal	13.7	13.3	11.8	11.4
Diesel	Thermal	1.4	1.8	2.0	2.0
Nuclear	Thermal	0.0	0.0	0.0	0.0
PumpL	Storage	0.0	0.0	0.0	0.0
PumpH	Storage	0.0	0.0	0.0	0.0
Lithium	Storage	0.0	0.0	0.0	0.0
Onshore wind	Renewable	19.0	19.0	19.0	19.0
Offshore wind	Renewable	0.0	0.0	0.0	0.0
PV solar	Renewable	0.0	0.0	0.0	0.0

is less and there is a slight increase in the amount of diesel (2.0 GW). In addition, the planner invests in 1.2 GW of coal with CCS which is not used at all in case 0.

Note that other technologies (e.g., nuclear and off-shore wind) are chosen in many scenarios of the second investment stage (in 5 years) and that varying amounts of most of the technologies are explored during the iterative solution procedure, even if these are not used in the optimal solution.

Serial implementation

This section discusses the results of the serial implementation of the algorithms, where only one JULIA instance is launched. For Algorithm 2 we set $w = 1$ so one single subproblem is solved exactly at each iteration.

Table 3 shows the effort in terms of iterations and computation time (the timer starts after the JULIA code has been precompiled) to reach an ϵ-optimal solution using Algorithms 1 (`Stand_Bend`), 2 (`Adapt_Bend`), and 3 (`Zaker_Bend`) for case studies 0–3. We report the results for values of the optimality tolerance ϵ of 1.00%, 0.10%, and 0.01%. For case 0 the computation efficiencies of Algorithms 1, 2, and 3 are similar. In this small case study Algorithms 1 and 3 solve 2 subproblems each iteration and Algorithm 2 solves 1 subproblem. However, Algorithm 2 needs more iterations to reach the target tolerance. As an example, to obtain a tolerance lower than 0.01% Algorithm 1 requires 26 iterations and Algorithm 2 needs 50. Increasing the size of the problem makes the comparison more interesting. For example, for case 2 Algorithm 1 needs 13 and 24 iterations to reach tolerance of 1.00% and 0.01%, respectively. To reach the same tolerances Algorithm 3 performs 12 and 21 iterations, respectively, and Algorithm 2 performs 67 and 190 iterations. However, an iteration of Algorithms 1 and 3 solves 90 subproblems while Algorithm 2 solves only 1. The results show that Algorithm 1 takes 64 minutes to yield an 0.01%-optimal solution compared

to 38 minutes for Algorithm 3, i.e., 1.7 times faster. To reach the same tolerance Algorithm 2 needs less than 8 minutes, i.e., 8.1 and 4.8 times faster than Algorithm 1 and 3, respectively. For case 3 this difference is even larger as Algorithms 1 and 3 solve the subproblem 756 times at each iteration. Accordingly, even if they reach a 1.00% tolerance in only 12 iterations, this requires 4 hours and 15 minutes, and 2 hours and 36 minutes, respectively. On the other hand, Algorithm 2 reaches the same tolerance in 8 minutes, 31.9 times faster than Algorithm 1 and 19.5 times faster than Algorithm 3. If a tighter tolerance is needed the difference of performances progressively reduces but it is still significant for $\epsilon = 0.01\%$. Indeed, Algorithm 1 requires around 9 hours and 18 minutes, Algorithm 3 takes 4 hours and 53 minutes (1.9 times faster than Algorithm 1), and Algorithm 2 needs 36 minutes (15.4 and 8.1 times faster than Algorithm 1 and 3, respectively).

Table 4 shows the (relative) computation time of the main steps of Algorithm 2, i.e., solving the relaxed master problem, solving the subproblems, and solving the adaptive oracles, to reach an ϵ-optimal solution for case studies 0–3. We report the results for value of the optimality tolerance ϵ of 1.00%, 0.10%, and 0.01%. In cases 0, 1, and 2 almost all the computation time is spent solving subproblems (one each iteration). However, in case 3 more than 10% of the computation time to obtain an 0.01%-optimal solution is spent solving the master problem, 27% solving the adaptive oracles, and only 65% solving subproblems. These results suggest that the proposed rule of solving one subproblem at each iteration may not be the best when the number $|\mathcal{I}|$ of subproblems grows significantly and that a selection rule that dynamically decides the number of subproblems to solve at each iteration may then achieve a better balance.

Parallel implementation

This section discusses the results of the parallel implementation of Algorithms 1 and 2, where together with the main JULIA instance we also start up a pool of workers via function `addprocs` of the DISTRIBUTED package. The RMP is solved in the main JULIA instance, while the subproblems and the oracles are solved by the pool of workers via the function `pmap`. For Algorithm 2 we set w equal to the number of workers in the pool, so one single subproblem is solved exactly by each worker at each iteration.

Table 5 compares the time and the number of iterations to reach an 0.01%-optimal solution of case 3 using Algorithms 1 and 2 as a function of the size of the workers pool. Algorithm 1 takes 23 iterations and 10 hours and 57 minutes with one worker in the pool, whereas Algorithm 2 takes 537 iterations and 1 hour, i.e., 10.8 times faster than Algorithm 1. The number of iterations for Algorithm 2 is slightly different from the values shown in Table 3, even though when the pool contains one worker only it should be equivalent to the serial implementation. The only difference from the serial implementation is the number of threads used to solve subproblems and oracles. Changing that value can make GUROBI converge to slightly different solutions (within optimality and feasibility tolerances) and alter the way the Benders algorithm finds an ϵ-optimal solution. Increasing the number of workers in the pool slightly changes the relative speed-up obtained with Algorithm 2, mainly due to some load balancing

Table 3 Comparative results for Algorithm 1 (`Stand_Bend`), Algorithm 2 (`Adapt_Bend`), and Algorithm 3 (`Zaker_Bend`)

	ϵ (%)	`Stand_Bend`		`Zaker_Bend`			`Adapt_Bend`		
		Iters	Time (s)	Iters	Time(s)	Time ratio	Iters	Time (s)	Time ratio
Case 0	1.00	15	61	22	51	1.2	30	48	1.3
	0.10	23	100	31	79	1.3	36	60	1.7
	0.01	26	114	37	102	1.1	50	90	1.3
Case 1	1.00	13	237	11	129	1.8	44	87	2.7
	0.10	16	309	17	230	1.3	51	106	2.9
	0.01	20	401	19	266	1.5	70	155	2.6
Case 2	1.00	13	1876	12	1131	1.7	67	146	12.8
	0.10	17	2527	18	1896	1.3	122	286	8.8
	0.01	24	3816	21	2289	1.7	190	473	8.1
Case 3	1.00	12	15, 298	12	9351	1.6	181	480	31.9
	0.10	17	23, 097	16	13, 380	1.7	271	811	28.5
	0.01	23	33, 494	20	17, 570	1.9	555	2170	15.4

Table 4 Analysis of Algorithm 2 (`Adapt_Bend`) computation time

	ϵ (%)	Relative computation time		
		Master problem (%)	Exact subproblems (%)	Adaptive oracles (%)
Case 0	1.00	0.14	99.22	0.03
	0.10	0.12	99.37	0.03
	0.01	0.08	99.55	0.04
Case 1	1.00	0.11	99.43	0.12
	0.10	0.10	99.50	0.12
	0.01	0.08	99.58	0.14
Case 2	1.00	0.28	98.40	1.02
	0.10	0.33	98.03	1.42
	0.01	0.43	97.48	1.92
Case 3	1.00	4.69	80.17	14.07
	0.10	6.13	75.32	17.63
	0.01	10.61	65.07	26.63

Table 5 Comparative results of the parallel implementation for Algorithm 1 (`Stand_Bend`) and Algorithm 2 (`Adapt_Bend`) to solve case 3 up to $\epsilon = 0.01\%$

Workers	`Stand_Bend`		`Adapt_Bend`		
	Iters	Time (s)	Iters	Time (s)	Time ratio
1	23	39459	537	3639	10.8
2	23	22856	286	2026	11.3
3	23	14432	216	1622	8.9
4	23	11846	160	1246	9.5

issues when solving subproblems. When there are 4 workers in the pool Algorithm 1 needs 3 hours and 17 minutes, while Algorithm 2 takes 20 minutes, 9.5 times faster.

It is worth highlighting that running the parallel code on a single Linux Desktop machine with more than two workers in the pool showed some severe slow down, probably due to cache issues given the large size of the subproblems to solve.

7 Conclusions

This paper presents a novel concept of inexact oracles that can be used to speed up the computational time of Benders-type decomposition algorithms for a class of large scale optimization problems. We propose two adaptive oracles that yield inexact and progressively more accurate information when subproblems are not solved. The first adaptive oracle builds valid cutting planes, and the second adaptive oracle provides a valid upper bound of the objective function. The two oracles exploit properties of the Benders subproblem such as convexity w.r.t. right-hand side coefficients and concavity w.r.t. cost coefficients, as well as monotonicity w.r.t. to both coefficients.

We use the novel adaptive oracles within a Benders-type decomposition algorithm to solve a stochastic investment planning problem. The results show substantial improvements in terms of computational efficiency (w.r.t. a standard Benders algorithm),

especially if a large optimality gap is allowed. The largest problem we solve has 756 subproblems, each of which has 4.11×10^5 linear constraints and 1.40×10^5 linear variables. Our algorithm is 31.9 times faster than the standard Benders algorithm to reach a 1.00% optimal solution and 15.4 times faster if the optimality tolerance is tightened to 0.01%.

A Reformulation to achieve monotonicity w.r.t. x and c.

We consider the case when $g(x, c) := \min_{y \in \mathcal{Y}} \{c^\top C y \mid Ay \leq Bx\}$ is not a monotonic function of x and c. First, we formulate a modified version of $g(x, c)$ as

$$\begin{aligned} \tilde{g}(x, c) = \min_{y \in \mathcal{Y}} \quad & c^\top C y + \gamma^\top x \\ \text{s.t.} \quad & Ay \leq Bx \end{aligned} \tag{10}$$

and choose a value of γ that makes the objective function $c^\top C y + \gamma^T x$ monotonic w.r.t. x. The master function objective is changed to $f(\mathbf{x}) - \sum_{i \in \mathcal{I}} \gamma^T x_i$ to cancel out this additional term. Then, we formulate a relaxed version of $\tilde{g}(x, c)$ as follows

$$\begin{aligned} \tilde{g}_c(x, c^\alpha, c^\beta) = \min_{y \in \mathcal{Y}, z^\alpha, z^\beta} \quad & c^{\alpha\top} z^\alpha + c^{\beta\top} z^\beta + \gamma^\top x \\ \text{s.t.} \quad & z^\alpha - z^\beta = Cy, \quad Ay \leq Bx \\ & 0 \leq z^\alpha \leq Z^\alpha, \quad 0 \leq z^\beta \leq Z^\beta \end{aligned} \tag{11}$$

where parameters Z^α and Z^β ensure that the relaxed subproblem is not unbounded. The value of Z^α and Z^β is to be chosen so that they do not restrict the feasible region w.r.t. (10). Note that $\tilde{g}_c(x, c^\alpha, c^\beta)$ is a concave and increasing function of both c^α and c^β, and that $\tilde{g}_c(x, c^\alpha, c^\beta) = \tilde{g}(x, c)$ if $c^\alpha = c$ and $c^\beta = -c$. Computing $\tilde{g}_c(\underline{x}, \underline{c}, -\underline{c})$ gives a solution that can be used in the adaptive oracle for obtaining a valid lower bound on $\tilde{g}(x, c), \ \forall x \in \mathcal{K}^x, \forall c \in \mathcal{K}^c$.

References

1. Baucke, R., Downward, A., Zakeri, G.: A deterministic algorithm for solving stochastic minimax dynamic programs. Optimization Online (2018). http://www.optimization-online.org/DB_HTML/2018/02/6449.html

2. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik **4**(1), 238–252 (1962)
3. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer, Berlin (2011)
4. Fábián, C.I.: Bundle-type methods for inexact data. Central Eur. J. Oper. Res. **8**(1), 35–55 (2000)
5. Fábián, C.I., Szőke, Z.: Solving two-stage stochastic programming problems with level decomposition. Comput. Manag. Sci. **4**(4), 313–353 (2007)
6. Hintermüller, M.: A proximal bundle method based on approximate subgradients. Comput. Optim. Appl. **20**(3), 245–266 (2001)
7. Kiwiel, K.C.: A proximal bundle method with approximate subgradient linearizations. SIAM J. Optim. **16**(4), 1007–1023 (2006)
8. Malick, J., de Oliveira, W., Zaourar, S.: Uncontrolled inexact information within bundle methods. EURO J. Comput. Optim. **5**(1–2), 5–29 (2017)
9. Mazzi, N.: Stand_and_adapt_bend julia code (2020). https://doi.org/10.5281/zenodo.4117799
10. Oliveira, W., Sagastizábal, C.: Level bundle methods for oracles with on-demand accuracy. Optim. Methods Softw. **29**(6), 1180–1209 (2014)
11. Oliveira, W., Sagastizábal, C., Scheimberg, S.: Inexact bundle methods for two-stage stochastic programming. SIAM J. Optim. **21**(2), 517–544 (2011)
12. Pereira, M.V., Pinto, L.M.: Multi-stage stochastic optimization applied to energy planning. Math. Program. **52**(1–3), 359–375 (1991)
13. Philpott, A., de Matos, V., Finardi, E.: On solving multistage stochastic programs with coherent risk measures. Oper. Res. **61**(4), 957–970 (2013)
14. Solodov, M.V.: On approximations with finite precision in bundle methods for nonsmooth optimization. J. Optim. Theory Appl. **119**(1), 151–165 (2003)
15. van Ackooij, W., Frangioni, A., de Oliveira, W.: Inexact stabilized Benders' decomposition approaches with application to chance-constrained problems with finite support. Comput. Optim. Appl. **65**(3), 637–669 (2016)
16. Van Slyke, R.M., Wets, R.: L-shaped linear programs with applications to optimal control and stochastic programming. SIAM J. Appl. Math. **17**(4), 638–663 (1969)
17. Wolf, C., Fábián, C.I., Koberstein, A., Suhl, L.: Applying oracles of on-demand accuracy in two-stage stochastic programming-a computational study. Eur. J. Oper. Res. **239**(2), 437–448 (2014)
18. Zakeri, G., Philpott, A.B., Ryan, D.M.: Inexact cuts in Benders decomposition. SIAM J. Optim. **10**(3), 643–657 (2000)

Affiliations

Nicolò Mazzi[1] · Andreas Grothey[1] · Ken McKinnon[1] · Nagisa Sugishita[1]

✉ Nicolò Mazzi
nicolo.mazzi@ed.ac.uk

Andreas Grothey
A.Grothey@ed.ac.uk

Ken McKinnon
K.McKinnon@ed.ac.uk

Nagisa Sugishita
s1576972@sms.ed.ac.uk

[1] School of Mathematics, University of Edinburgh, James Clerk Maxwell Building, Edinburgh EH9 3FD, UK