Journal of Global Optimization (2019) 75:393–416
https://doi.org/10.1007/s10898-019-00769-y

A scalable global optimization algorithm for stochastic nonlinear programs

Yankai Cao[1] · Victor M. Zavala[1]

Received: 13 August 2017 / Accepted: 4 April 2019 / Published online: 16 April 2019

Abstract
We present a global optimization algorithm for two-stage stochastic nonlinear programs (NLPs). The algorithm uses a tailored reduced-space spatial branch and bound (BB) strategy to exploit the nearly decomposable structure of the problem. At each node in the BB scheme, a lower bound is constructed by relaxing the so-called non-anticipativity constraints and an upper bound is constructed by fixing the first-stage variables to the current candidate solution. A key advantage of this approach is that both lower and upper bounds can be computed by solving individual scenario subproblems. Another key property of this approach is that it only needs to perform branching on the first-stage variables to guarantee convergence (branching on the second-stage variables is performed implicitly during the computation of lower and upper bounds). Notably, convergence results for this scheme also hold for two-stage stochastic MINLPs with mixed-integer first-stage variables and continuous recourse variables. We present a serial implementation of the algorithm in `Julia`, that we call `SNGO`. The implementation is interfaced to the structured modeling language `Plasmo.jl`, which facilitates benchmarking and model processing. Our implementation incorporates typical features that help accelerate the BB search such as LP-based lower bounding techniques, local search-based upper bounding techniques, and relaxation-based bounds tightening techniques. These strategies require the solution of extensive forms of the stochastic program but can potentially be solved using structured interior-point solvers (when the problem is an NLP). Numerical experiments are performed for a controller tuning formulation, a parameter estimation formulation for microbial growth models, and a stochastic test set from GLOBALlib. We compare the computational results against `SCIP` and demonstrate that the proposed approach achieves significant speedups.

Keywords Stochastic NLP · Global optimization · Scalable

✉ Victor M. Zavala
victor.zavala@wisc.edu

[1] Department of Chemical and Biological Engineering, University of Wisconsin-Madison, 1415 Engineering Dr, Madison, WI 53706, USA

1 Introduction

We study algorithms for finding global solutions for nonconvex nonlinear programs (NLPs) arising in two-stage stochastic programming. Our work is motivated by the observation that the direct application of spatial branch and bound (BB) techniques (as those implemented in several popular packages such as BARON [21], ANTIGONE [18], and SCIP [17]) do not scale well with the number of scenarios because branching may be performed on all variables (which include first and second-stage variables). Several approaches have been previously proposed to exploit the structure of stochastic programs in order to achieve better scalability. A class of these methods is based on direct application of generalized Benders decomposition (GBD) [10], which solves a sequence of master problems to generate lower bounds and primal problems to generate upper bounds. The master problems are obtained from outer approximation and the primal problems are obtained by fixing the first-stage variables at candidate values. Convergence of GBD is not guaranteed for nonconvex problems. The work in [16] proposes a nonconvex GBD scheme in which a lower bound is generated by solving a convexified problem with GBD and an upper bound is generated by fixing first stage variables and solving the resulting scenario subproblems to global optimality. Finite termination of this approach can be guaranteed if the first stage variables are all bounded integers.

Another class of methods is based on Lagrangian relaxation (LR) [8,11,15]. Lagrangian relaxation is used to generate lower bounds within a BB framework. The approaches reported in [5] and [14] use this approach to solve stochastic MILPs and stochastic MINLPs, respectively. Lagrangian relaxation has also been used to develop reduced-space search approaches that only branch on complicating variables. The approaches reported in the literature, however, do not provide convergence guarantees for general two-stage stochastic problems.

Reduced-space BB approaches with convergence guarantees have been reported for special problem classes. The approach proposed in [6] provides a reduced-space BB scheme for solving partially convex problems (i.e., problems that are convex when a subset of variables are fixed). The authors provide a proof of convergence when branching only in the space of variables that induce nonconvexity. The approaches proposed in [7] and [9] are also reduced-space BB schemes that can be used to solve partially convex problems.

In this paper we introduce a tailored reduced-space BB scheme for general two-stage stochastic NLPs. For each node in the BB scheme, a lower bound is constructed by relaxing the so-called non-anticipativity constraints and an upper bound is constructed by fixing the first-stage variables to a given candidate solution and solving the scenario subproblems. The proposed lower bound is a special case of Lagrangian relaxation with the dual variables fixed to zero. A key advantage of the proposed approach is that both lower bounding and upper bounding problems can be decomposed into scenario subproblems that are solved independently to global optimality. Another key property of this approach is that we only need to perform spatial branching on the first-stage variables to guarantee convergence. The algorithm also exploits the fact that the gap between the upper and lower bounding problems is the so-called expected value of perfect information, which is usually small in applications (relative to the overall magnitude of the objective). Notably, our convergence results also hold for two-stage stochastic MINLPs with mixed-integer first stage variables and continuous two-stage variables. We provide a software implementation of the algorithm that we call `SNGO`. This is a Julia-based package that is interfaced to the modeling language `Plasmo.jl`, which facilitates model processing. Our implementation contains typical algorithmic features of global optimization solvers such as convexification, outer approximation, feasibility-based bound tightening (FBBT), optimality-based bound tightening (OBBT), and local search. SNGO also exploits lower bounds obtained from linear programming (LP) relaxations.

The paper is organized as follows: Sect. 2 introduces basic nomenclature and lower/upper bounding problems. Section 3 introduces the BB algorithm and provides a convergence proof. Section 4 provides implementation details for this algorithm. Section 5 illustrates numerical performance in a stochastic programming formulation for controller tuning, a parameter estimation formulation for microbial community models, and stochastic versions of GLOBALLib instances. The paper closes in Sect. 6 with final remarks and directions of future work.

2 Basic nomenclature and setting

We consider two-stage stochastic programs of the form:

$$z = \min_{x \in X_0} \sum_{s \in \mathcal{S}} Q_s(x). \tag{2.1}$$

Here, $\mathcal{S} := \{1, \ldots, S\}$ is the scenario set, $x \in X_0 \subset \mathbb{R}^{n_x}$ are the first-stage variables, $X_0 := \{x | x_l \leq x \leq x_u\}$ is a closed set, and $Q_s(x)$ is the optimal value of the second-stage problem:

$$\begin{aligned} Q_s(x) = &\min_{y_s} \;\; f_s(x, y_s) \\ &\text{s.t.} \;\;\; g_{s,j}(x, y_s) \leq 0 \;, \; j = 1, \ldots, m_s. \end{aligned} \qquad P_s(x)$$

Here, $y_s \in \mathbb{R}^{n_{y_s}}$ are the second-stage variables. The scenario objectives $f_s : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{y_s}} \rightarrow \mathbb{R}$ and constraints $g_{s,j} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{y_s}} \rightarrow \mathbb{R}$ are assumed to be continuous and potentially nonconvex. Equality constraints such as $c_s(x, y_s) = 0$ can be reformulated by using sets of inequalities. We define the feasible set for the recourse variables as $Y_s(x) := \{y_s | g_{s,j}(x, y_s) \leq 0, j = 1, \ldots, m_s\}$. The recourse function $Q_s(\cdot)$ implicitly defines a feasible set of x, which is in turn defined as $K_s := \{x \mid \exists y_s \in Y_s(x)\}$. If $\nexists y_s \in Y_s(x)$ for some x, we set $Q_s(x) = \infty$. We define the relative interior of a set X as relint(X). We use $\delta(X)$ to denote the diameter of set X. In our context, the diameter of the box set $\{x \mid x_l \leq x \leq x_u\}$ is $\delta(X) = ||x_u - x_l||_\infty$.

The feasible set defined by all second-stage subproblems is denoted as $K = \bigcap_{s \in \mathcal{S}} K_s$. Consequently, the feasible region of the first-stage variables x is $X_0 \cap K$. We make the following blanket assumptions:

Assumption 1 The set K is compact and $X_0 \cap K$ is nonempty.

Assumption 2 The feasible sets $Y_s(x)$ are compact for all $x \in X_0$ and $s \in \mathcal{S}$.

Assumption 2 implies that $Q_s(x)$ is lower semicontinuous in x for all $s \in \mathcal{S}$ (see Theorem 35 in [3]). This also implies that the function $Q(x) := \sum_{s \in \mathcal{S}} Q_s(x)$ is lower semicontinuous in x. Assumption 1 ensures that problem (2.1) attains its minimum according to the generalized Weierstrass theorem.

At each node in a BB algorithm we solve the following problem with respect to the partition set $X \subseteq X_0$:

$$z(X) = \min_{x \in X} \sum_{s \in \mathcal{S}} Q_s(x) \tag{2.2}$$

We refer to this problem as the *primal node problem*. This problem can be written in the extensive form:

$$\min_{x\in X, y_s} \sum_{s\in\mathcal{S}} f_s(x, y_s) \tag{2.3a}$$

$$\text{s.t.}\;\; g_{s,j}(x, y_s) \le 0, \quad j = 1, \ldots, m_s, s \in \mathcal{S}. \tag{2.3b}$$

We can also lift the node problem (2.2) by replicating the first stage variables across scenarios and then enforce non-anticipativity constraints. This gives the lifted problem:

$$\min_{x_s\in X} \sum_{s\in\mathcal{S}} Q_s(x_s) \tag{2.4a}$$

$$\text{s.t.}\;\; x_s = x_{s+1}, \quad s = 1, \ldots, S-1. \tag{2.4b}$$

It is easy to verify that problems (2.2), (2.3), (2.4) are equivalent.

2.1 Lower bounding problem

If the nonanticipativity constraints of (2.4) are removed, we obtain a lower bounding problem of the form:

$$\beta(X) := \min_{x_s\in X} \sum_{s\in\mathcal{S}} Q_s(x_s). \tag{2.5}$$

Clearly, the lower bounding problem can be decomposed into S subproblems of the form:

$$\beta_s(X) := \min_{x_s\in X} \; Q_s(x_s), \tag{2.6}$$

or, equivalently:

$$\begin{aligned} \beta_s(X) = \min_{x_s\in X, y_s} \;\; & f_s(x_s, y_s) \\ \text{s.t.} \;\; & g_{s,j}(x_s, y_s) \le 0\;, j = 1, \ldots, m_s, \end{aligned} \tag{2.7}$$

with $\beta(X) = \sum_{s\in\mathcal{S}} \beta_s(X)$. It is also obvious that $\beta(X) \le z(X)$ because the feasible region of (2.4) is a subset of the feasible region of (2.5). Moreover, we have that $\beta(X_1) \ge \beta(X_2)$ for $X_1 \subset X_2$. The lower bounding problem is also called wait-and-see problem and the gap between the primal problem (2.2) and the lower bounding problem (2.5) is called the expected value of perfect information (EVPI). We highlight that $\beta_s(X)$ is obtained by solving the scenario subproblems to *global optimality*. We also note that, if $\beta_s(X) = \infty$ for some $s \in \mathcal{S}$, then $z(X) = \infty$. In other words, if a subproblem is infeasible, the primal node problem is infeasible.

The lower bounding problem can be seen as the first iteration of Lagrangian relaxation (obtained by dualizing the non-anticipativity constraints and initializing the dual variables to zero). In principle, it is possible to perform multiple Lagrangian relaxation iterations to update the dual variables (i.e., by using subgradient schemes) to obtain a tighter lower bound. Unfortunately, subgradient schemes are difficult to tune and performance is ad-hoc. The approach that we present in this work does not require computation and updates of dual variables.

2.2 Upper bounding problem

An upper bound for the stochastic program can be obtained by fixing the first stage variable at a candidate solution $\hat{x} \in X$. The upper bound is denoted as $\alpha(X) := \sum_{s \in \mathcal{S}} Q_s(\hat{x})$ and we note that the upper bound can be computed by solving S subproblems with optimal values $\alpha_s(X) = Q_s(\hat{x})$ and by setting $\alpha(X) = \sum_{s \in \mathcal{S}} \alpha_s(X)$. In our proposed scheme, the subproblems are also solved to *global optimality*. It is easy to see that $z(X) \leq \alpha(X)$ holds for any $\hat{x} \in X$. We highlight that, a classic BB scheme obtains an upper bound by solving the extensive form (2.3) to *local* optimality. We note this approach requires solving a coupled problem, while our approach requires solutions of decoupled scenario subproblems to *global* optimality. In Sect. 4 we discuss implementations that allow for the possibility of using a local solver to solve the extensive form and possibly obtain a better upper bound. In the convergence analysis that follows we assume that the lower and upper bounding problems are solved *exactly* to global optimality. In Sect. 4 we discuss implementation strategies to set proper solution tolerances.

3 Convergence of branch and bound socheme

This section establishes convergence for a BB scheme constructed using the proposed lower and upper bounding problems. We outline the BB scheme as follows:

```
1. Initialization
    Initialize the iteration index $k \leftarrow 0$.
    Set $\mathcal{X} \leftarrow \{X_0\}$, and tolerance $\epsilon > 0$.
    Compute initial upper and lower bounds $\alpha_k = \alpha(X_0)$, $\beta_k = \beta(X_0)$.
2. Node Selection
    If $\mathcal{X} = \emptyset$, STOP.
    Select and delete from $\mathcal{X}$ a set $X \in \mathcal{X}$ satisfying $\beta(X) = \beta_k$.
    Update $k \leftarrow k + 1$.
3. Branching
    Partition $X$ into subsets $X_1$ and $X_2$ with $\text{relint}(X_1) \cap \text{relint}(X_2) = \emptyset$.
    Add each subset to $\mathcal{X}$ to create separate child nodes.
4. Bounding
    For each child node $X_i$, compute $\beta(X_i)$ and $\alpha(X_i)$.
    If $\beta_s(X_i) = \infty$ for some $s \in \mathcal{S}$, remove $X_i$ from $\mathcal{X}$.
    Let $\beta_k \leftarrow \min\{\beta(X') : X' \in \mathcal{X}\}$ and $\alpha_k \leftarrow \min(\alpha_{k-1}, \alpha(X_1), \alpha(X_2))$.
    Remove all $X'$ from $\mathcal{X}$ with $\beta(X') \geq \alpha_k$.
    If $\beta_k - \alpha_k \leq \epsilon$, STOP.
    Return to Step 2.
```

Algorithm 1: Branch and Bound Scheme

A key feature of the proposed BB scheme is that it only branches on the first-stage variables (because the recourse variables are handled implicitly in the evaluation of the recourse functions).

The BB scheme can be viewed as a rooted tree, where X_0 is the root node at level 0 and X_{k_q} denotes a node at level q explored at iteration k_q. An arc connects a node X_{k_q} at level q

with one of its children $X_{k_{q+1}}$ at level $q+1$. In other words, $X_{k_{q+1}}$ is a direction partition of X_{k_q} satisfying $X_{k_{q+1}} \subset X_{k_q}$. A path in the tree from the root node corresponds to a sequence $\{X_{k_q}\}$ of successively refined partition elements.

It is easy to see that the sequence $\{\alpha_k\}$ is monotonically nonincreasing and that $\{\beta_k\}$ is monotonically nondecreasing. The BB scheme is said to be convergent if $\lim_{k\to\infty} \alpha_k = \lim_{k\to\infty} \beta_k = z$. If the scheme is convergent then it produces a global ϵ-optimal solution in a finite number of steps. We now proceed to prove convergence; to do so, we adapt basic results from Chapter VI of the seminal work in [12] (Definitions IV.6, IV.7, IV.8, IV.10, Theorem IV.3, and Corollary IV.1) to our context.

Lemma 1 *If a BB procedure is infinite, then it generates at least one infinitely decreasing sequence $\{X_{k_q}\}$ of successively refined partition elements, $X_{k_{q+1}} \subset X_{k_q}$* [12].

Definition 1 A subdivision is called **exhaustive** if $\lim_{q\to\infty} \delta(X_{k_q}) = 0$, for all decreasing subsequences X_{k_q} generated by the subdivision [12].

A subdivision can be guaranteed to be exhaustive on X_0 if a first-stage variable x_i that corresponds to the diameter of X is selected for partitioning.

In a BB scheme, a "delete by infeasibility" rule is used to delete infeasible partition sets X (i.e., sets X with $X \cap K = \emptyset$). For example, if the lower bounding problem is infeasible, the node problem is infeasible and the partition set can be deleted from further consideration.

Definition 2 The "delete by infeasibility" rule throughout a BB procedure is called **certain in the limit** if, for every infinite decreasing sequence X_{k_q} of successively refined partition elements with limit $\bar{X}$, we have $\bar{X} \cap K \neq \emptyset$ [12].

Lemma 2 *Given an exhaustive subdivision, the "delete by infeasibility" rule is certain in the limit.*

Proof Under an exhaustive subdivision, X_{k_q} eventually collapses to a single point $\bar{x}$ and we thus have that $\bar{X} = \{\bar{x}\}$. Assume by contradiction that there exists a sequence X_{k_q} converging to an infeasible point $\bar{x}$. Since $\bar{x}$ is infeasible and $\bar{x} \in X_{k_q} \subseteq X_0$, we have that $\bar{x} \notin K$. Consequently, there is at least one set K_i satisfying $\bar{x} \notin K_i$. By the compactness of K_i, the distance between $\bar{x}$ and K_i is nonzero and there is a ball around $\bar{x}$, denoted as $B_r(\bar{x}) = \{x \mid \|x - \bar{x}\| \leq r\}$, satisfying $B_r(\bar{x}) \cap K_i = \emptyset$. Since $\lim_{q\to\infty} \delta(X_{k_q}) = 0$, there is a q_0 such that $X_{k_q} \subset B_r(\bar{x}), \forall q \geq q_0$. At this iteration, $X_{k_{q_0}} \cap K_i = \emptyset$, which implies that $\beta_s(X_{k_{q_0}}) = \infty$. Consequently, the infeasible set will be detected and deleted. Hence, it is impossible that X_{k_q} converges to an infeasible point without being detected by the "delete by infeasibility" rule. □

Definition 3 A lower bounding operation is called **strongly consistent** if, at every iteration, any undeleted partition set can be further refined and if any infinite decreasing sequence X_{k_q} of successively refined partition elements contains a sub-sequence $X_{k_{q'}}$ satisfying $\bar{X} \cap K \neq \emptyset$, $\lim_{q'\to\infty} \beta(X_{k_{q'}}) = z(\bar{X} \cap K)$, where $\bar{X} = \bigcap_q X_{k_q}$ [12].

Lemma 3 *Given an exhaustive subdivision on x, Algorithm 1 is strongly consistent.*

Proof From Lemma 2 we have that $\bar{X} \cap K \neq \emptyset$ holds. With an exhaustive subdivision, X_{k_q} shrinks to a single point $\bar{x}$ and we thus have that $\bar{X} = \{\bar{x}\}$ and $\bar{X} \cap K = \{\bar{x}\}$. The result can thus be proven by showing that $\lim_{q\to\infty} \beta(X_{k_q}) = z(\bar{X} \cap K) = \sum_{s\in\mathcal{S}} Q_s(\bar{x})$. Take $\tilde{x}_{k_q,s} \in \operatorname{argmin}\{Q_s(x_s) \colon x_s \in X_{k_q}\}$, since X_{k_q} shrinks to $\bar{x}$, $\lim_{q\to\infty} \tilde{x}_{k_q,s} = \bar{x}$. Since $\bar{x} \in X_{k_q}$, it follows that $Q_s(\tilde{x}_{k_q,s}) \le Q_s(\bar{x})$. From the lower semicontinuity of Q_s, it follows that $Q_s(\bar{x}) \le \lim_{q\to\infty} Q_s(\tilde{x}_{k_q,s})$. Therefore, $Q_s(\bar{x}) = \lim_{q\to\infty} Q_s(\tilde{x}_{k_q,s}) = \lim_{q\to\infty} \beta_s(X_{k_q})$. Take the sum over s, we obtain $\lim_{q\to\infty} \beta(X_{k_q}) = \sum_{s\in\mathcal{S}} Q_s(\bar{x})$. □

We now proceed to prove convergence of the lower bounds.

Definition 4 A selection operation is said to be **bound improving** if, after a finite number of steps, at least one partition element where the actual lower bounding is attained is selected for further partition. [12].

Algorithm 1 is bound improving since, at every step, a partition where the actual lower bounding is attached is selected for further partition.

Lemma 4 *For a BB scheme using a lower bounding operation that is strongly consistent and using a selection operation that is bound improving, we have that* $\lim_{k\to\infty} \beta_k = z$ [12].

Lemma 5 *Given an exhaustive subdivision on x, Algorithm 1 satisfies* $\lim_{k\to\infty} \beta_k = z$.

Proof This result can be established by combining Lemmas 4 and 3. □

To prove finite-epsilon convergence of the upper bounds, we need to make the following assumption.

Assumption 3 The recourse function $Q(\cdot)$ is Lipschitz continuous in a nonempty neighborhood of a solution x^*.

This assumption requires that the feasible set of the stochastic NLP has a nonempty interior for the first-stage variables, which preclude the application of the following lemma when the formulation has nontrivial equality constraints involving only first stage variables. Typical regularity conditions of the scenario subproblems guaranteeing Lipschitz continuity of $Q(\cdot)$ are discussed in [4]. In particular, Lipschitz continuity follows if the solutions of the subproblems satisfy the strong second order condition and the linear independence constraint qualification (at fixed x^*).

Lemma 6 *Given an exhaustive subdivision on x, Algorithm 1 delivers a sequence* $\{\alpha_k\}$ *satisfying* $\lim_{k\to\infty} \alpha_k = z$.

Proof Because $Q(\cdot)$ is Lipschitz continuous around x^*, there is a ball denoted as $B_r(x^*) = \{x| \parallel x - x^* \parallel \le r\}$, satisfying $Q(x) - Q(x^*) \le K\|x - x^*\|$ for all $x \in B_r(x^*)$ and where K is a Lipschitz constant. Therefore, for $\epsilon > 0$ and every point $x \in B_{r'}(x^*)$, we have that $Q(x) - Q(x^*) \le \epsilon$ holds with $r' = \min(r, \epsilon/K)$.

Because the subdivision is exhaustive we have that, either after a finite number of iterates $\bar{k}$, the partition considered satisfies $X_{\bar{k}} \subseteq B_{r'}(x^*)$, or at iteration $\hat{k}$, the partition $X_{\hat{k}}$ containing x^* is pruned. In the first case, because any point $x \in X_{\bar{k}}$ satisfies $Q(x) - Q(x^*) \le \epsilon$, we have that $\alpha_{\bar{k}} \le \alpha(X_{\bar{k}}) \le Q(x^*) + \epsilon$. In the second case, the node is pruned because $\alpha_{\hat{k}} \le \beta(X_{\hat{k}}) + \epsilon$. Since $x^* \in X_{\hat{k}}$, we have $\beta(X_{\hat{k}}) \le Q(x^*)$ and thus $\alpha_{\hat{k}} \le Q(x^*) + \epsilon$. Because the value of ϵ is arbitrary, we have $\lim_{k\to\infty} \alpha_k = z$. □

Combining Lemmas 5 and 6, we obtain our main result:

Theorem 1 *Given an exhaustive subdivision on x, Algorithm 1 is convergent in the sense that:*

$$\lim_{k\to\infty} \alpha_k = \lim_{k\to\infty} \beta_k = z. \tag{3.1}$$

Remark We note that the assumptions are also expected to hold for two-stage stochastic MINLPs with mixed-integer first-stage variables x but purely continuous recourse variables. In particular, Lipschitz continuity of the recourse function holds under purely continuous recourse variables and typical regularity assumptions. Consequently, the convergence results hold in this case as well.

4 Implementation details

The software implementation of the proposed algorithm is called `SNGO` (Structured Nonlinear Global Optimizer). `SNGO` is implemented in `Julia` and interfaced with the following tools: `Plasmo` for modeling stochastic programs, `IPOPT` for solving an extensive form of the problem to local optimality, `SCIP` to solve subproblems to global optimality, and Gurobi for solving linear programs (LPs). We leverage the syntax and interfaces of the algebraic modeling language `JuMP` in our implementation.

To compute lower bounds, `SNGO` first creates an LP relaxation (obtained from convexification and outer approximation using the auxiliary variable technique of [20]) of the form:

$$z^{LP}(X) = \min_{x\in X, y_s, w_s} \sum_{s\in\mathcal{S}} \bar{f}_s(x, y_s, w_s) \tag{4.1a}$$

$$\text{s.t. } \bar{g}_{s,j}(x, y_s, w_s) \le 0, \quad j = 1, \ldots, \bar{m}_s, s \in \mathcal{S}, \tag{4.1b}$$

where $\bar{f}_s(\cdot)$ and $\bar{g}_{s,j}(\cdot)$ are linear underestimators for $f_s(\cdot)$ and $g_{s,j}(\cdot)$, respectively; and w_s are auxiliary variables. Here, $\bar{m}_s \ge m_s$ holds, since auxiliary constraints might be introduced. These LP relaxations are constructed automatically. After solving the LP, the outer approximation is refined at the solution of the LP problem (resulting in a tighter LP relaxation). This process is repeated until the improvement of the lower bounds is sufficiently small. The LP is also a stochastic program because the underestimators are generated for each scenario subproblem. Note that this is a coupled but structured problem in which the number of variables and constraints increases linearly with the number of scenarios. This problem can, in principle, be solved using a structure-exploiting interior-point solver such as `PIPS`. In our implementation, we solve this problem directly with `Gurobi`. Since $z^{LP}(X) \le z(X)$ holds by construction, the solution of the LP relaxation can be used as an alternative lower bound. `SNGO` also adds αBB cuts [2] and cuts from the reformulation-linearization technique (RLT) [19] automatically to the relaxed LP. The cost of generating and solving the relaxed LP is modest compared to the solution of the nonlinear scenario subproblems in the branch and bound scheme.

After solving the LP relaxation, `SNGO` solves the lower bounding problem, which is decomposed into S subproblems of the form (2.6). The lower bound of the node is set to be the maximum of the lower bounds from the LP relaxation and of the lower bounding problem. We consider different strategies to strengthen the tightness of the lower bounds. First, it is easy to see that the scenario objective is always greater than the optimal value of a subproblem. Consequently, the cut $f_s(x, y_s) \geq \beta_s(X)$ can be added to the node with partition X. Moreover, for all the descendants of this node $X^{'} \subset X$, we have that $\beta_s(X^{'}) \geq \beta_s(X)$ holds; consequently, this cut is still valid. SNGO creates an auxiliary variable denoting the scenario objective and the lower bound of this variable is updated at each node. Second, we note that if we pick $\tilde{x}_s \in \text{argmin}\{Q_s(x_s) : x_s \in X\}$ and assume X is partitioned into two subsets X_1 and X_2 with $X_1 \cap X_2 = \emptyset$, then either $\tilde{x}_s \in X_1$ or $\tilde{x}_s \in X_2$ is valid, or both are valid. If $\tilde{x}_s \in X_1$, then this implies that $\tilde{x}_s \in \text{argmin}\{Q_s(x_s) : x_s \in X_1\}$. Consequently, the solution of the subproblem in a parent node can be reused in the children nodes and the associated subproblems do not need to be re-solved. The solution of lower bounding subproblems is thus stored and passed to the children nodes.

To compute upper bounds, SNGO first solves the extensive form (2.3) with a local NLP solver. At the root node, the local solver is run with a multi-start technique. If the local solver returns a feasible solution, the first stage solution from the local solver can be set to $\hat{x}$ for the upper bounding problem. We also note that, when solving the extensive form, removing redundant duplicates of first stage constraints can aid the local solver. If the local solver fails, then SNGO takes the expected value of first stage solutions from the lower bounding subproblems to obtain $\hat{x}$, as propopsed in [14]. Having $\hat{x}$, the upper bounding problem is solved by solving S separate subproblems $P_s(\hat{x})$ to global optimality. Through experiments we have found that, in many cases, upper bounds reach optimality at an early stage. Consequently, upper bounding problems are solved at the first L_t levels (default of three) and then every L_e levels (default of two) .

SNGO also implements bound tightening techniques including feasibility-based bound tightening (FBBT) and optimality-based bound tightening (OBBT). OBBT is performed by cycling through each component x_i of first stage variables and solving $2n_x$ LPs of the form:

$$\max / \min_{x \in X, y_s, w_s} \quad x_i \tag{4.2a}$$

$$\text{s.t.} \quad \bar{g}_{s,j}(x, y_s, w_s) \leq 0, \quad j = 1, \ldots, \bar{m}_s, s \in \mathcal{S}. \tag{4.2b}$$

Each LP can be decomposed into S subproblems, where subproblem s is of the form:

$$\max / \min_{x \in X, y_s, w_s} \quad x_i \tag{4.3a}$$

$$\text{s.t.} \quad \bar{g}_{s,j}(x, y_s, w_s) \leq 0, \quad j = 1, \ldots, \bar{m}_s, . \tag{4.3b}$$

Each pair of subproblems minimizing or maximizing x_i gives a pair of lower and upper bounds of x_i. Pairs of bounds from different scenarios are summarized by computing the tightest lower and upper bounds. In many other global optimization solvers, OBBT is not performed in every BB node because the computational expense of this procedure is high. For SNGO, however, the solution of the nonlinear subproblems is the main computational bottleneck and the number of first stage variables is usually small. Consequently, OBBT is performed at every BB node.

SNGO uses strong branching [1] to select branching variables. For each component of first stage variables x_i, we compute the branching point x_i^b, divide the current partition set X, into two sets $X_1 = \{x | x \in X, x_i \leq x_i^b\}$ and $X_2 = \{x | x \in X, x_i \geq x_i^b\}$, and compute the lower bounds of the two subsets from LP relaxation $z^{LP}(X_1)$ and $z^{LP}(X_2)$. We compare the improvement of $z^{LP}(X_1)$ and $z^{LP}(X_2)$ over the lower bound of the current node $\beta(X)$, and select the first stage variable x_i that achieves the largest improvement. When the improvement in terms of the lower bounds from the LP relaxation are lower than a threshold, the first stage variable with the longest width is selected. The branching point is decided according to the expected value of solutions from the lower bounding problem, that is $\sum_{s\in\mathcal{S}} \tilde{x}_s/|\mathcal{S}|$. We also enforce that the branching point keeps a minimum distance away from the variables bounds to ensure that the overall subdivision is exhaustive, which is a standard practice in global optimization software such as ANTIGONE [18] and BARON [21]. In our implementation, we project the expected value of solutions from the lower bounding problem to be within $[x_i^l + \theta(x_i^u - x_i^l), x_i^u - \theta(x_i^u - x_i^l)]$, where the default setting for θ is 0.1. To avoid excessive partitioning on one variable, a first stage variable with a range below a certain threshold (i.e., $x_i^u - x_i^l < \gamma$ with $\gamma = 10^{-4}$) is not considered for further branching until the ranges of all first stage variables are below this threshold.

The SNGO implementation follows six major computational steps: (1) solution of LP relaxation, (2) solution of extensive form to local optimality, (3) solution of at most S lower bounding subproblems to global optimality, (4) solution of at most S upper bounding subproblems to global optimality, (5) solution of $2n_x \cdot S$ small LPs for OBBT, and (6) solution of $2n_x$ LPs for branching variable selection. Experiments in Sect. 5 show that, for most problems, over half of the solution time is spent in steps (3) and (4). For some cases (ex2_1_8 and ex8_4_1 in Sect. 5.3), over 90% of the solution time is spent on these two steps. Ideally, the time spent on these steps grows linearly with the number of scenarios, however, the solution time for a subproblem is not consistent and the number of subproblems is not always equal to the number of scenarios at every node. For example, the information from step (1) and (2) might be enough to decide that this node can be pruned and thus no subproblem are solved. With the number of variables to branch on fixed (i.e., the number of first stage variables), the number of nodes needed is not expected to explode with the increase in the number of scenarios. The problems to be solved in steps (1, 2, 6) have an extensive form and the size of the problems grows linearly with the number of scenarios, while the problems to be solved at steps (3, 4, 5) can be decomposed into subproblems and the number of subproblems grows linearly with the number of scenarios. We also note that steps (3, 4, 5, 6) are directly parallelizable and step (1) can also be parallelized by using solvers such as PIPS and PIPS-NLP. In this paper, however, we use a serial implementation because we aim to compare algorithmic performance with off-the-shelf solvers on an equal basis. We also highlight that achieving an efficient parallel implementation is challenging because of memory management and load imbalancing issues.

When deriving convergence results, we assumed that the lower/upper bounding problems are solved exactly. However, in a practical implementation, we need to set up a tolerance for the global solver to solve lower/upper bounding problems. For each subproblem, we set the termination tolerance to be $\frac{\epsilon}{2S}$. For the upper bounding subproblems, the primal bound returned from global solver is used to compute the upper bounds, while for lower bounding subproblems, the dual bound returned from global solver is used to compute the lower bounds.

We use Plasmo.jl[1] to express the stochastic NLPs under study. Plasmo.jl is a Julia-based algebraic modeling framework that facilitates the construction and analysis of large-scale structured optimization models. To do this, it leverages a hierarchical graph abstraction wherein nodes and edges can be associated with individual optimization models that are linked together [13]. Given a graph structure with models and connections, Plasmo.jl can produce either a pure (flattened) optimization model to be solved using off-the-shelf optimization solvers such as IPOPT and SCIP, or it can communicate graph structures to structure-exploiting solvers such as SNGO.

The code snippet shown in Fig. 1 illustrates how to implement stochastic problems in Plasmo.jl. As can be seen, the individual scenario models are created and appended to

[1] https://github.com/zavalab/Plasmo.jl.

```
#call libraries
using Plasmo
using JuMP

# create two-stage model
stom=graphModel()

# define first-stage variables in parent node
@variable(stom, x[1:nx])

# create array of scenario models
nodes=Array(JuMP.Model,n)
for j in 1:S
   # get scenario model and append to parent node
   nodes[j] = get_scenario_model(j)
   @addNode(stom,nodes[j],"s$j")
   # connect children and parent variables
   @constraint(stom, Nonanticipativity[i in 1:nx], x[i] == nodes[j][:x][i])
end
# solve two-stage program with SNGO
SNGO(stom)

# alternatively, create the extensive form and solve with SCIP
ex= extensiveModel(stom)
ex.solver = SCIPSolver()
solve(m)
```

Fig. 1 Snippet of a stochastic NLP implementation in `Plasmo.jl`

the parent node on-the-fly to create a two-level graph structure. The snippet also shows how to use `Plasmo.jl` to create a flattened NLP to be solved by off-the-shelf solvers like `SCIP` [17]. This allows the user to compare computational performance.

5 Computational experiments

We evaluate the performance of `SNGO` by using stochastic NLPs arising from applications such as optimal controller tuning and parameter estimation formulations for microbial community models, and a test set containing stochastic variants of the GLOBALlib set. The Julia scripts of the test cases are available at https://github.com/zavalab/JuliaBox/tree/master/SNGO/examples. We compare the computational results against those of the state-of-the-art global solver `SCIP` 4.0.0, which is linked to SoPlex 3.0.0 and `IPOPT` 3.12.7. Each solver terminates under one of the following conditions: (1) relative optimality gap satisfies $\frac{\alpha_k-\beta_k}{\min\{|\beta_k|,|\alpha_k|\}} \leq 1\%$, (2) absolute optimality gap satisfies $\alpha_k - \beta_k \leq 0.01$, or (3) the search reaches a 12-h time limit. We use a computing server with Intel(R) Xeon(R) CPU E5-2698 v3 processors running at 2.30 GHz to conduct the experiments.

5.1 Optimal controller tuning

We consider the identification of optimal PID controller parameters capable of withstanding diverse scenarios on set-point changes $\bar{x}_s$, model structural uncertainty ($\tau_{x,s}$ and $\tau_{u,s}$), and disturbances d_s. The optimal parameters aim to minimize the expected error between the state and the desired set-point. The formulation is given in (5.1). The first stage variables are the controller parameters (gain K_p, integral gain K_i, and derivative gain K_d) of the controller and the second-stage variables are the state time trajectories $x_s(t)$ for each scenario $s \in \mathcal{S}$.We generate the scenarios using Monte Carlo simulations and assume that $\bar{x}$, τ_x τ_u, and d are independent and uniform random variables. The state trajectories are discretized using an implicit Euler scheme and the integral term in (5.1d) is approximated as the accumulated errors prior to a given time step. We note that the number of state variables grows linearly with the number of scenarios. The largest problem solved includes 60 scenarios and has a total of 4803 variables, 4800 constraints, and 4800 nonlinear nonconvex terms.

Table 1 compares the performance of SNGO, SCIP, IPOPT. We note that, when the number of scenarios is 10, 30, 40, 50 and 60, SNGO can solve problems to a gap of 1% while SCIP cannot solve the problem within 12 h. For the problem with 20 scenarios, SCIP can solve the problem but this requires 7.5 h of solution time while SNGO solves the problem in 30 min. The key advantage of SNGO over SCIP is that it only needs to branch on the three first stage variables while branching over second-stage variables is done implicitly in the solution of the scenario subproblems. SCIP, on the other hand, needs to branch on both first and second-stage variables (2323 in the 20 scenario case) simultaneously. As a result, the number of nodes visited by SCIP is 14,053 times more than those visited by SNGO. On the other hand, since at each node SNGO needs to solve subproblems to global optimality, the computational cost of SNGO for each BB node is 923 times larger than that of SCIP. Despite of this, the computational benefits are significant. We emphasize that the subproblems solved in SNGO are solved with SCIP. We have found SCIP to be robust in solving small to medium-sized problems but it is clear that direct branching on all variables is not scalable.

Table 1 also shows time spent on different tasks of the solver including solving lower bounding subproblems to global optimality (LB1), solving LP relaxations (LB2), solving upper bounding subproblems to global optimality (UB1), solving extensive form NLPs to local optimality (UB2), bound tightening, and branching variable selection (VS). For this problem, solving lower bounding subproblems is relatively expensive while solving upper bounding subproblems is relatively cheap. One reason for this is that the upper bounding subproblems are not solved at every node. Another reason is that this problem has the property that, when the first stage variables are fixed, each subproblem has only one feasible solution. When the number of scenarios is between 10 and 50, the number of nodes is quite consistent and the solution time per node grows almost linearly (as shown in the Fig. 2). However, when the number of scenarios is 60, the node tree explored might be quite different, thus the number of nodes explored and the solution time per node grows quite quickly. We note that SNGO is not able to solve problems with more than 60 scenarios within the time limit.

$$\min_{x_s(t), K_\mathrm{p}, K_\mathrm{i}, K_\mathrm{d}} \sum_{s\in\mathcal{S}} \int_0^T e_s(t)^2 dt \tag{5.1a}$$

$$\text{s.t.} \quad \frac{dx_s(t)}{dt} = -\tau_{x,s} x_s(t)^2 + \tau_{u,s} u_s(t) + \tau_{d,s} d_s, \; s \in \mathcal{S} \tag{5.1b}$$

$$e_s(t) = x_s(t) - \bar{x}_s, \; s \in \mathcal{S} \tag{5.1c}$$

$$u_s(t) = K_\mathrm{p} e_s(t) + K_\mathrm{i} \int_0^t e_s(\tau)\, d\tau + K_\mathrm{d} \frac{de_s(t)}{dt}, \; s \in \mathcal{S} \tag{5.1d}$$

5.2 Estimation for microbial growth models

We now consider the problem of estimating parameters in microbial community models. This problem is not a stochastic program but exhibits the same algebraic structure if the time horizon is partitioned into blocks. We can view each time partition as a scenario and the parameters to be estimated and the variable linking partitions as first stage (complicating)

Table 1 Computational performance of SNGO and SCIP on controller tuning problem

Problem	SNGO									SCIP 4.0		
# S	Time (s)	LB1 (s)	LB2 (s)	UB1 (s)	UB2 (s)	BT (s)	VS (s)	Gap (%)	# Nodes	Time (s)	Gap (%)	# Nodes
10	979	225	233	25	5	152	314	1	67	43,200	1.29	2,248,314
20	1778	515	373	44	8	240	559	1	59	27,045	1.00	829,181
30	3978	1518	880	85	18	440	969	1	61	43,200	2.85	879,500
40	4603	1961	912	96	17	525	1016	1	55	43,200	2.16	595,294
50	5300	1898	1114	128	19	659	1383	1	57	43,200	2.51	352,282
60	16,610	10,014	2043	168	85	1351	2764	1	93	43,200	2.90	377,474

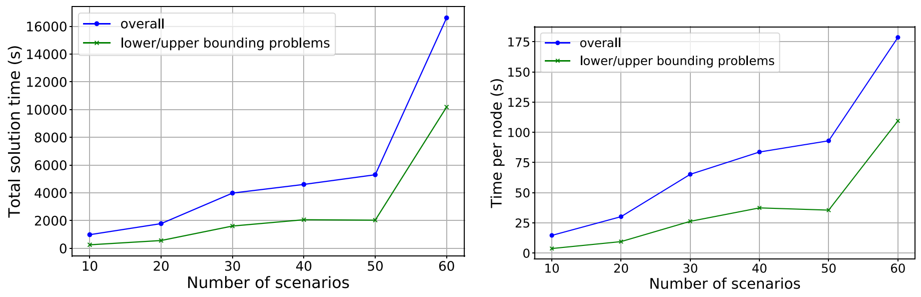


Fig. 2 Total solution time of SNGO and solution time per node to solve robust controller problem with different numbers of scenarios

Table 2 Computational performance of SNGO and SCIP on estimation problems for microbial growth models

Problem	SNGO			SCIP	
Name	Time (s)	Gap (%)	# Nodes	Time (s)	Gap (%)
sp.1	7248	1	291	43,200	592
sp.2	1382	1	35	43,200	8297
sp.3	1411	1	23	43,200	14.2
sp.4	2181	1	71	1059	0.8
sp.5	591	1	15	43,200	4052.2
sp.6	1303	1	33	43,200	1031.6
sp.7	482	1	11	321	1.00
sp.8	520	1	15	43,200	59.37
sp.9	503	1	3	43,200	25.27
sp.10	1377	1	47	43,200	113.86
sp.11	730	1	23	299	0.29
sp.12	519	1	13	43,200	280.55

variables. The problem formulation has the form:

$$\min_{x_k(t),\alpha,\beta} \sum_{k\in\mathcal{K}} \int_{t_k}^{t_{k+1}} (x_k(t)-\bar{x}_k)^2 dt \tag{5.2a}$$

$$\text{s.t. } \frac{dx_k(t)}{dt} = \alpha x_k(t)^2 + \beta x_k(t), \quad t \in [t_k, t_{k+1}], \quad k \in \mathcal{K} \tag{5.2b}$$

$$x_{k+1}(t_{k+1}) = x_k(t_{k+1}), \quad k \in \mathcal{K}, \tag{5.2c}$$

where α, β are the parameters to be estimated, $\mathcal{K}$ is the set of time partitions, and $x_k(\cdot)$ is the state trajectory in partition $k \in \mathcal{K}$. We partition the time domain into 47 blocks to obtain a problem with 48 first stage variables, 1082 total variables, 1080 total constraints, and 1411 nonlinear terms. We solved the problem using 12 different real experimental data sets, corresponding to the growth of different species of bacteria.

Table 2 summarizes the performance of the solvers on these instances. As can be seen, SCIP cannot solve most problems within 12 h, while SNGO can solve most of the problems within 20 min. The shortest solution time is 8 min and the longest solution time is 2 h.

Table 3 Computational performance of SNGO and SCIP on stochastic variants of GLOBALLib instance for $|\mathcal{S}| = 100$

Problem	SNGO			SCIP	
Name	Time (s)	Gap (%)	# Nodes	Time (s)	Gap (%)
abel	147	1	1	10	0.0
ex2_1_10	217	1	7	3067	0.8
ex2_1_7	160	1	1	43,200	222.5
ex2_1_8	4845	1	39	43,200	54.4
ex5_2_5	390	1	1	43,200	484.74
ex5_3_2	1260	1	59	43,200	50.88
ex8_4_1	35,157	1	175	43,200	$\geq$ 10,000
hydro	51	1	1	7	0.0
immun	14	1	1	43,200	$\geq$ 10,000
st_fp7a	171	1	1	43,200	97.5
st_fp7b	153	1	2	43,200	55.1
st_fp7c	951	1	2	43,200	80.1
st_fp7d	105	1	1	43,200	297.1
st_fp7e	133	1	1	43,200	247.8
st_fp8	136	1	1	43,200	7.5
st_m1	43,200	1.5	126	43,200	8.1
st_m2	43,200	5.3	22	43, 200	25.21
st_rv2	48	1	1	43,200	4.3
st_rv3	70	1	1	43,200	7.7
st_rv7	77	1	1	43,200	2.8
st_rv8	3018	1	2	43,200	8.9
chenery	6624	1	23	43,200	10.6
ex8_4_8	282	1	1	43,200	$\geq$ 10,000
ex8_4_8_bnd	1023	1	1	43,200	$\geq$ 10,000
harker	67	1	1	153	0.0
pollut	60	1	1	46	0.0
ramsey	50	1	1	2	0.0
srcpm	50	1	1	2	0.0

5.3 Stochastic GLOBALLib instances

We have also tested the algorithm using stochastic variants of the GLOBALLib instances. To construct such variants, we selected 28 problems with 20–50 variables, and added random perturbations to the right hand side of a subset of the constraints. The first 5 variables of the problem are selected as first stage variables. There are 7 problems (ex5_4_4, ex8_4_2, ex8_6_2, hhfair, launch, maxmin, prolog) also with 20–50 variables not selected because SCIP cannot solve a single scenario problem. The nonconvexities encountered in these 28 problems include bilinear terms, fractional terms, logarithmic terms, and signomial terms, as well as composite functions of these terms and linear terms. The size of the problems depends on the number of scenarios and is outlined in Table 5. The total number of variables

Table 4 Computational performance of SNGO and SCIP on stochastic variants of GLOBALLib instance for $|\mathcal{S}| = 1000$

Problem	SNGO			SCIP	
Name	Time (s)	Gap (%)	# Nodes	Time (s)	Gap (%)
abel	43,200	3.73	137	38,536	0.0
ex2_1_10	3130	1.0	5	43,200	457.4
ex2_1_7	2363	1.0	1	43,200	231.2
ex2_1_8	43,200	1.4	23	43,200	109.9
ex5_2_5	43,200	f	f	43,200	f
ex5_3_2	43,200	5.5	391	43,200	63.2
ex8_4_1	13,456	1.0	1	43,200	$\geq$ 10,000
hydro	1367	1.0	1	43,200	$\geq$ 10,000
immun	202	1.0	1	43,200	$\geq$ 10,000
st_fp7a	2579	1.0	1	43, 200	101.9
st_fp7b	2431	1.0	1	43,200	57.0
st_fp7c	8397	1.0	2	43,200	83.2
st_fp7d	2857	1.0	2	43,200	308.1
st_fp7e	2374	1.0	1	43, 200	257.4
st_fp8	2439	1.0	1	43,200	5.8
st_m1	43,200	3.1	10	43,200	3490
st_m2	43,200	8.7	3	43,200	$\geq$ 10,000
st_rv2	651	1.0	1	43,200	4.8
st_rv3	1217	1.0	1	43,200	8.2
st_rv7	1044	1.0	1	43,200	3.7
st_rv8	30,404	1.0	3	43,200	f
chenery	43,200	9.4	8	43,200	10.6
ex8_4_8	3956	1.0	1	43,200	$\geq$ 10,000
ex8_4_8_bnd	43,200	f	f	43,200	$\geq$ 10,000
harker	2903	1.0	1	43,200	$\geq$ 10,000
pollut	1796	1.0	1	1474	0.1
ramsey	762	1.0	1	508	0.0
srcpm	1490	1.0	1	1218	0.0

when the number of scenarios is 1000 ranges from 21,005 to 44,005 (these are large-scale instances).

Tables 3 and 4 summarize the computational performance when the number of scenarios is 100 and 1000, respectively. We use “f” to indicate when the solver failed to return any bounds or candidate solution. For problems with 100 scenarios, SCIP can only solve 7 problems while SNGO can solve 26 out of 28 problems. For problems with 1000 scenarios, SCIP can only solve 4 problems while SNGO can solve 20 problems. Most of the problems are solved with only one node, this might be related to the way how random perturbations are introduced. However, a naive implementation of lower/upper bounding problems explores significantly more nodes. One reason why SNGO only explores one node for these problems is because bounding tightening and multi-start local search is quite extensive at the root node.

Table 5 Size of problems from stochastic version of GLOBALLib when the number of scenarios is 100 and 1000

Problem	S = 100			S = 1000		
Name	# Vars	# NL terms	# Cons	# Vars	# NL terms	# Cons
abel	3105	0	1500	31,005	0	15,000
ex2_1_10	2105	1000	1100	21,005	10,000	11,000
ex2_1_7	2105	2000	1100	21,005	20,000	11,000
ex2_1_8	2505	2400	1100	25,005	24,000	11,000
ex5_2_5	3305	6000	2000	33,005	60,000	20,000
ex5_3_2	2305	1200	1700	23,005	12,000	17,000
ex8_4_1	2305	1000	1100	23,005	10,000	11,000
hydro	3205	600	2500	32,005	6000	25,000
immun	2205	0	800	22,005	0	8000
st_fp7a	2105	2000	1100	21,005	20,000	11,000
st_fp7b	2105	2000	1100	21,005	20,000	11,000
st_fp7c	2105	2000	1100	21,005	20,000	11,000
st_fp7d	2105	2000	1100	21,005	20,000	11,000
st_fp7e	2105	2000	1100	21,005	20,000	11,000
st_fp8	2505	2400	2100	25,005	24,000	21,000
st_m1	2105	2000	1200	21,005	20,000	12,000
st_m2	3105	3000	2200	31,005	30,000	22,000
st_rv2	2105	2000	1100	21,005	20,000	11,000
st_rv3	2105	2000	2100	21,005	20,000	21,000
st_rv7	3105	3000	2100	31,005	30,000	21,000
st_rv8	4105	4000	2100	41,005	40,000	21,000
chenery	4405	4200	3900	44,005	42,000	39, 000
ex8_4_8	4305	8100	3100	43,005	81,000	31,000
ex8_4_8_bnd	4305	8100	3100	43,005	81,000	31,000
harker	2105	1400	800	21,005	14,000	8000
pollut	4305	4000	900	43,005	40,000	9000
ramsey	3405	2200	2300	34,005	22,000	23,000
srcpm	4005	500	2800	40,005	5000	28,000

By comparing the results of Tables 3 and 4 we can again see favorable scalability of SNGO. For 19 problems (ex2_1_10, ex2_1_7, ex8_4_1, hydro, immun, st_fp7a, st_fp7b, st_fp7c, st_fp7d, st_fp7e, st_fp8, st_rv2, st_rv3, st_rv7, st_rv8, ex8_4_8, pollut, ramsey, srcpm), the solution time increases by less than 30 times when the number of scenarios increases from 100 to 1000 (a factor of 10). For 5 problems (abel, ex5_2_5, ex5_3_2, harker, and ex8_4_8_bnd), the increase in solution time is more dramatic. For the rest 4 problems SNGO reaches the time limit but the gap is kept below 10% in most cases (only two instances have a larger gap).

To further illustrate the scalability of SNGO, Table 6 shows how the time spent on different tasks changes as the number of scenarios increase. Figures 3 and 4 illustrate how the total solution time and the solution time per node change as the number of scenarios increase. We present five relatively difficult instances ex2_1_10, ex2_1_8, ex5_3_2, ex8_4_1, chenery. For problems (ex2_1_8, ex5_3_2, ex8_4_1, chenery), solving the subproblems to global

Table 6 Computational performance of SNGO on five problem instances from GLOBALLib with different numbers of scenarios. We use "–" to denote the situations when the time limit is reached

Problem		SNGO								
Name	# S	Time (s)	LB1 (s)	LB2 (s)	UB1 (s)	UB2 (s)	BT (s)	VS (s)	Gap (%)	# Nodes
ex2_1_10	10	28	2	1	0.6	10	9	1.5	1.0	3
	20	49	4	1.6	0.7	23	11	2.4	1.0	3
	50	131	9	3	0.7	87	18	8	1.0	3
	100	217	48	2	10	107	24	16	1.0	7
	200	889	108	11	21	610	67	49	1.0	5
	500	2411	283	9	55	1730	118	168	1.0	7
	1000	3130	468	24	89	1967	209	248	1.0	5
ex2_1_8	10	213	183	1.7	2	8	9	2	1.0	17
	20	577	475	5	21	34	18	10	1.0	41
	50	2311	2074	14	46	97	39	21	1.0	41
	100	4845	4423	23	97	175	66	34	1.0	39
	200	18,647	16,081	112	348	1708	194	132	1.0	57
	500	34,457	26,047	144	928	5920	397	760	1.0	33
	1000	43,200	–	–	–	–	–	–	–	–

Table 6 continued

Problem		SNGO								
Name	# S	Time (s)	LB1 (s)	LB2 (s)	UB1 (s)	UB2 (s)	BT (s)	VS (s)	Gap (%)	# Nodes
ex5_3_2	10	510	274	28	49	5	106	6	1.0	189
	20	878	509	45	81	8	169	7	1.0	161
	50	6775	4066	258	778	92	1151	59	1.0	593
	100	1260	760	85	57	21	260	3	1.0	59
	200	37,352	22,037	2147	4131	1164	6210	172	1.0	973
	500	43,200	–	–	-	–	–	–	–	–
	1000	43,200	–	–	-	–	–	–	–	–
ex8_4_1	10	43,200	–	–	–	–	–	–	–	–
	20	31,747	24,221	224	6316	29	683	2	1.0	1027
	50	26,073	21,229	205	3985	21	488	4	1.0	263
	100	35, 157	29,209	304	4794	26	653	7	1.0	175
	200	6536	4142	106	1928	11	247	16	1.0	21
	500	7623	7353	48	0	12	192	0	1.0	1
	1000	13,456	12,903	99	0	23	397	0	1.0	1
Chenery	10	394	267	18	5	45	42	9	1.0	7
	20	1740	880	77	57	401	269	3	1.0	57
	50	4769	3047	110	113	808	486	126	1.0	33
	100	6624	4534	214	141	864	596	139	1.0	23
	200	12,507	8258	337	236	2526	791	242	1.0	13
	500	43,200	–	–	–	–	–	–	–	–
	1000	43,200	–	–	–	–	–	–	–	–

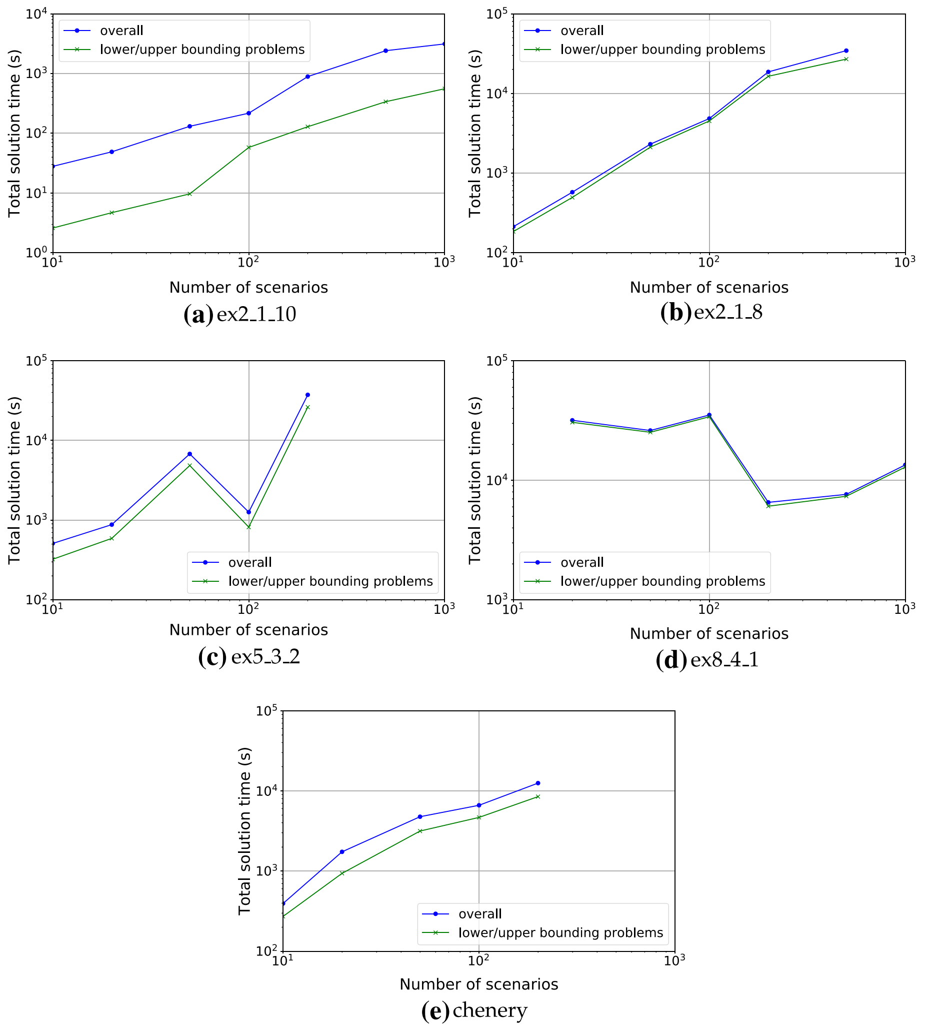
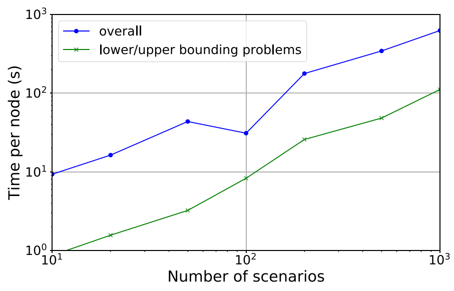


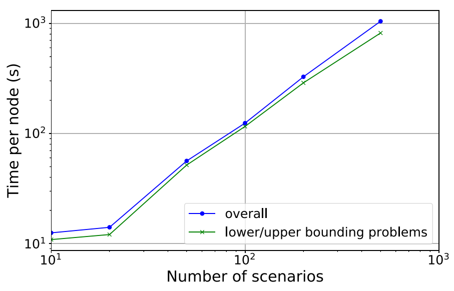
Fig. 3 Total solution time of SNGO to solve five problem instances from GLOBALLib with different numbers of scenarios

optimality requires more than half of the solution time. For problems (ex2_1_10, ex2_1_8, chenery), the number of nodes explored is relatively consistent with the number of scenarios. For each test problem, the solution time per node grows nearly linearly when the number of scenarios are within a certain range (10–1000 for ex2_1_10, 20–500 for ex2_1_8, 10–200 for ex5_3_2, 20–200 for ex8_4_1 and chenery).

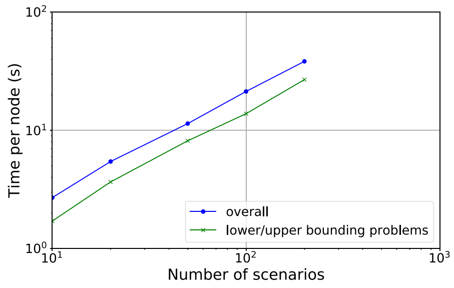
Figure 5 shows the progression of the lower and upper bounds for these five problems. The dots represent iterates under which the bound updates are obtained from lower and upper bounding problems while the crosses represent iterates under which updates are obtained from convexification and local search (i.e., lower bounds are obtained from the LP relaxation). The bounds obtained from convexification and local search at the root node are shown at iteration zero. Figure 5 shows that, while the lower bounding problems proposed play a



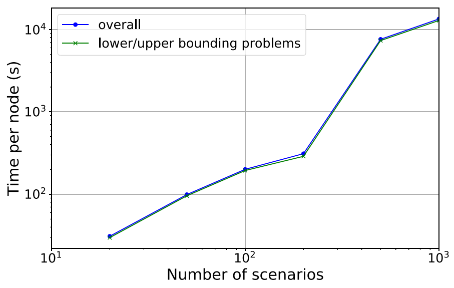
(a) ex2_1_10



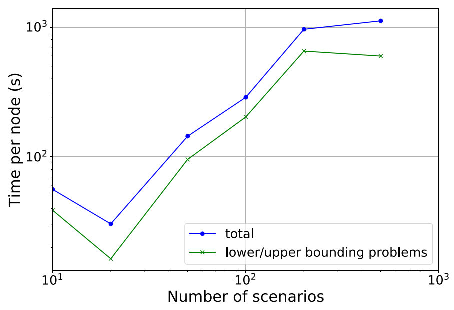
(b) ex2_1_8



(c) ex5_3_2



(d) ex8_4_1



(e) chenery

Fig. 4 Solution time of SNGO per node to solve five problem instances from GLOBALLib with different numbers of scenarios

significant role, the bounds obtained from convexification also help accelerate the solution process. Interestingly, for all test problems, optimal solutions are always found at the root node and the rest of the process is used to prove that these solutions are within the optimality gap. For ex2_1_8, the solution of upper bounding problems finds a significantly better solution than the local search with a multi-start scheme while, for the other problems, optimal solutions are found from local search. The gap between the primal problem and the lower bounding problem (the expected value of perfect information (EVPI)) is typically small in many applications. This can be verified in our test set, where we observe that the gaps at the first iteration are 20.0%, 16.0%, 14.3%, 1.36%, 32.0%; while the initial gaps from convexification and local search observed at iteration zero are 286%, 126%, 43.8%, 354%, $\geq$ 10, 000%. This illustrates that our lower bounding approach provides tight lower bounds.

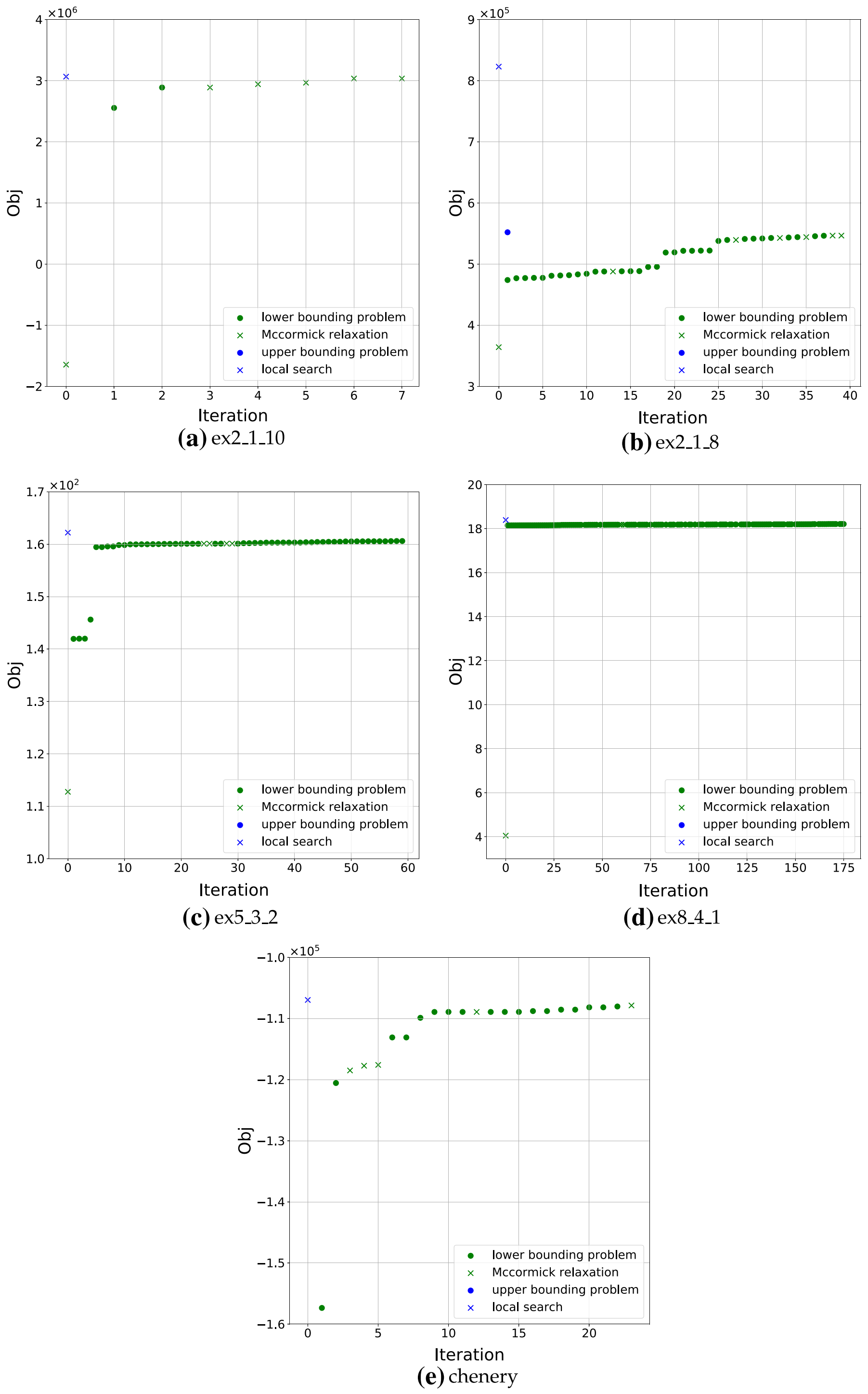


Fig. 5 Evolution of lower and upper bounds for five problem instances from GLOBALLib

We expect that a parallel version of our implementation can help reduce the solution times.

6 Conclusions and future work

We have proposed and implemented a global optimization algorithm for stochastic nonlinear programs. The main advantages of the proposed algorithm are that both lower bounding and upper bounding problems can be decomposed into smaller scenario subproblems and that branching needs to be performed only on the first-stage variables. We provide a proof of convergence and numerical evidence that the proposed approach significantly outperforms the state-of-the-art solver `SCIP`. As a part of future work, we are interested in extending the work to stochastic mixed integer nonlinear programs and to develop parallel implementations.

Acknowledgements We acknowledge financial support by the U.S. Department of Energy under Grant DE-SC0014114. We thank Ophelia Venturelli for providing the model and experimental data for the microbial growth estimation problem. We also thank Felipe Serrano and Ambros Gleixner for providing assistance with the use of `SCIP`.

References

1. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**(1), 42–54 (2005)
2. Androulakis, I.P., Maranas, C.D., Floudas, C.A.: αbb: a global optimization method for general constrained nonconvex problems. J. Glob. Optim. **7**(4), 337–363 (1995)
3. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming, 2nd edn. Springer, New York (2011)
4. Bonnans, J.F., Shapiro, A.: Perturbation Analysis of Optimization Problems. Springer, New York (2013)
5. CarøE, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Oper. Res. Lett. **24**(1), 37–45 (1999)
6. Dür, M., Horst, R.: Lagrange duality and partitioning techniques in nonconvex global optimization. J. Optim. Theory Appl. **95**(2), 347–369 (1997)
7. Epperly, T.G., Pistikopoulos, E.N.: A reduced space branch and bound algorithm for global optimization. J. Glob. Optim. **11**(3), 287–311 (1997)
8. Fisher, M.L.: The lagrangian relaxation method for solving integer programming problems. Manag. Sci. **27**(1), 1–18 (1981)
9. Floudas, C.A., Visweswaran, V.: Primal-relaxed dual global optimization approach. J. Optim. Theory Appl. **78**(2), 187–225 (1993)
10. Geoffrion, A.M.: Generalized benders decomposition. J. Optim. Theory Appl. **10**(4), 237–260 (1972)
11. Guignard, M., Kim, S.: Lagrangean decomposition: a model yielding stronger lagrangean bounds. Math. Program. **39**(2), 215–228 (1987)
12. Horst, R., Tuy, H.: Global Optimization: Deterministic Approaches. Springer, New York (2013)
13. Jalving, J., Abhyankar, S., Kim, K., Hereld, M., Zavala, V.M.: A graph-based computational framework for simulation and optimization of coupled infrastructure networks. Under Review (2016)
14. Karuppiah, R., Grossmann, I.E.: A lagrangean based branch-and-cut algorithm for global optimization of nonconvex mixed-integer nonlinear programs with decomposable structures. J. Glob. Optim. **41**(2), 163–186 (2008)
15. Khajavirad, A., Michalek, J.J.: A deterministic lagrangian-based global optimization approach for quasiseparable nonconvex mixed-integer nonlinear programs. J. Mech. Des. **131**(5), 051,009 (2009)
16. Li, X., Tomasgard, A., Barton, P.I.: Nonconvex generalized benders decomposition for stochastic separable mixed-integer nonlinear programs. J. Optim. Theory Appl. **151**(3), 425 (2011)
17. Maher, S.J., Fischer, T., Gally, T., Gamrath, G., Gleixner, A., Gottwald, R.L., Hendel, G., Koch, T., Lübbecke, M.E., Miltenberger, M., et al.: The scip optimization suite 4.0 (2017)
18. Misener, R., Floudas, C.A.: Antigone: algorithms for continuous/integer global optimization of nonlinear equations. J. Glob. Optim. **59**(2–3), 503–526 (2014)
19. Sherali, H.D., Adams, W.P.: A Reformulation-linearization Technique for Solving Discrete and Continuous Nonconvex Problems, vol. 31. Springer, New York (2013)

20. Smith, E.M., Pantelides, C.C.: A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex minlps. Comput. Chem. Eng. **23**(4–5), 457–478 (1999)
21. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)