Noname manuscript No.
(will be inserted by the editor)

A Scalable Global Optimization Algorithm for Stochastic Nonlinear Programs

Yankai Cao · Victor M. Zavala

Received: date / Accepted: date

Abstract We propose a global optimization algorithm for stochastic nonlinear programs that uses a specialized spatial branch and bound (BB) strategy to exploit the nearly decomposable structure of the problem. In particular, 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 we only need 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). The algorithm is implemented in `SNGO` in Julia and is interfaced to the modeling languages `JuMP` and `Plasmo`. Our implementation contains typical algorithmic features of global optimization solvers such as convexification, outer approximation, feasibility-based bound tightening, optimality-based bound tightening, and local search. Numerical experiments are performed using a stochastic optimization formulation for controller tuning, 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 new approach achieves significant speedups.

Keywords Stochastic NLP · Global Optimization · Scalable

1 Introduction

We study algorithms to find global solutions for nonconvex nonlinear programs (NLPs) arising in two-stage stochastic programming settings. 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

Corresponding Author: victor.zavala@wisc.edu

Yankai Cao · Victor M. Zavala
Department of Chemical and Biological Engineering
University of Wisconsin-Madison, 1415 Engineering Dr, Madison, WI 53706, USA
Tel.: 608-262-6934
E-mail: victor.zavala@wisc.edu

[16], ANTIGONE [14], and SCIP [12]) do not scale well with the number of scenarios because branching is 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) [7], which solves a sequence of lower bounding problems obtained from outer approximation (obtained by solving the so-called master problem) and upper bounding problems (obtained by solving the second-stage problems at candidate values for the first-stage variables). The convergence of GBD is not guaranteed for nonconvex problems because the outer approximation might cut regions of the space and the recourse (second-stage) cost is not guaranteed to be polyhedral. The work in [11] 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 integer. Another class of methods is based on Lagrangian relaxation (LR) [6]. The convergence of Lagrangian relaxation cannot be guaranteed for nonconvex problems due to the potential presence of a duality gap. Consequently, LR lower bounds are usually exploited within a BB framework. The approaches reported in [3] and [10] use this approach to solve stochastic MILPs and stochastic MINLPs, respectively. A limitation of these approaches is that they need to branch on the entire variable space. The approach proposed in [4] establishes convergence for partly convex stochastic programming problems (i.e., problems that are convex when the first stage variables are fixed).

In this paper we introduce a specialized BB scheme for 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. A key advantage of this approach is that both lower bounding and upper bounding problems can be decomposed into scenario subproblems that are solved independently to global optimality using off-the-shelf schemes. 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 the gap between the upper and lower bounding problems is the expected value of perfect information, which is typically moderate in actual applications. The algorithm is implemented in SNGO, which is a Julia-based package that is interfaced to the structured modeling language Plasmo. 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: Section 2 introduces basic nomenclature and lower/upper bounding problems. Section 3 introduces the branch and bound algorithm and provides a convergence proof. Section 4 shows the implementation of this algorithm. Section 5 illustrates the numerical performance on a stochastic optimization formulation for controller tuning, a parameter estimation formulation for microbial community models, and stochastic versions of GLOBALLib instances. The paper closes with final remarks and directions of future work in Section 6.

2 Basic Nomenclature and Setting

We consider the two-stage stochastic program of the form:

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

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

$$\begin{aligned} Q_s(x) = &\min_{y_s} \; f_s(x, y_s) \\ &\text{s.t.} \quad g_{s,j}(x, y_s) \le 0 \;, j = 1, \dots, m_s. \end{aligned} \tag{2.2}$$

We refer to this problem as $P_s(x)$. Here, $y_s \in \mathbb{R}^{n_{y_s}}$ are the second stage variables. The scenario objectives $f_s : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{y_s}} \to \mathbb{R}$ and constraints $g_{s,j} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_{y_s}} \to \mathbb{R}$ are assumed to be twice continuously differentiable and potentially nonconvex. We define the feasible set for the recourse variables as $Y_s(x) := \{y_s | g_{s,j}(x, y_s) \le 0, j = 1, \dots, m_s\}$. The recourse function $Q_s(\cdot)$ implicitly defines a feasible set of x 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 $Q_s(x) = \infty$.

The feasible set defined by all second stage problems 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 [2]). 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 branch and bound algorithm we solve the following problem with respect to the partition set $X \subseteq X_0$:

$$z(X) = \min_{x \in X} \sum_{s \in \mathcal{S}} Q_s(x) \tag{2.3}$$

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

$$\min_{x \in X, y_s} \sum_{s \in \mathcal{S}} f_s(x, y_s) \tag{2.4a}$$

$$\text{s.t.} \;\; g_{s,j}(x, y_s) \le 0, \quad j = 1, \dots, m_s, s \in \mathcal{S}. \tag{2.4b}$$

We can lift the node problem (2.3) by replicating the first stage variables across scenarios and then force them to be the same with the non-anticipativity constraints. This gives the lifted problem:

$$\min_{x_s \in X} \sum_{s \in \mathcal{S}} Q_s(x_s) \tag{2.5a}$$

$$\text{s.t.} \;\; x_s = x_{s+1}, \quad s = 1, \dots, S - 1. \tag{2.5b}$$

It is easy to verify that problems (2.3), (2.4), (2.5) are equivalent.

2.1 Lower Bounding Problem

If the nonanticipativity constraints of (2.5) are removed, we obtain the lower bounding problem for the stochastic program of the form:

$$\beta(X) := \min_{x_s \in X} \sum_{s\in\mathcal{S}} Q_s(x_s). \tag{2.6}$$

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

$$\beta_s(X) := \min_{x_s \in X} Q_s(x_s), \tag{2.7}$$

or equivalently:

$$\begin{aligned} \beta_s(X) = \min_{x_s \in X, y_s} \; & f_s(x, y_s) \\ \text{s.t.} \;\; & g_{s,j}(x, y_s) \le 0 \;, j = 1, \dots, m_s, \end{aligned} \tag{2.8}$$

with $\beta(X) = \sum\limits_{s\in\mathcal{S}} \beta_s(X)$. It is also obvious that $\beta(X) \le z(X)$ because the feasible region of (2.5) is a subset of the feasible region of (2.6). 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 (2.3) and the lower bounding problem (2.6) 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 the subproblem is infeasible, the primal node problem is infeasible.

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 $\alpha_s(X) = Q_s(\hat{x})$, with $\alpha(X) = \sum\limits_{s\in\mathcal{S}} \alpha_s(X)$. These subproblems are also solved to *global optimality*. It is easy to see that $z(X) \le \alpha(X)$ holds for any $\hat{x} \in X$. We highlight that, a classic branch and bound scheme obtains an upper bound by solving the extensive form (2.4) only to *local* optimality. Such a local solution provides an upper bound for $\alpha(X)$.

3 Convergence of Branch and Bound Scheme

This section establishes convergence for a spatial branch and bound (BB) scheme constructed using the proposed lower and upper bounding problems. A key feature of the proposed BB scheme is that it only branches on the first-stage variables (because the recourse variables at handled implicitly in the evaluation of the recourse functions). 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 $\mathcal{X}$ the set $X \in \mathcal{X}$ satisfying $\beta(X) = \beta_k$.
    Update $k \leftarrow k + 1$.
4. Branching
    Partition $X$ into subsets $X_1$ and $X_2$ with $X_1 \cap X_2 = \emptyset$ and add each subset to $\mathcal{X}$ to create separate child
    nodes.
3. 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

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 $\lim\limits_{k\to\infty} \alpha_k = \lim\limits_{k\to\infty} \beta_k = z$. 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 the seminal work in [8] 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, $X_{k_{q+1}} \subset X_{k_q}$ [8].*

We use $\delta(X)$ to denote the diameter of set X. In our context, the diameter of the box set $\{x \,|\, x_l \leq x \leq x_u\}$ is $\delta(X) = ||x_u - x_l||_\infty$.

Definition 1 *A subdivision is called* **exhaustive** *if $\lim\limits_{q\to\infty} \delta(X_{k_q}) = 0$, for all decreasing subsequences X_{k_q} generated by the subdivision [8].*

A subdivision can be guaranteed to be exhaustive if x_i with the longest range is selected for partition.

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, then 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 $\bar{X} \cap K \neq \emptyset$ [8].*

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 $B_r(\bar{x}) = \{x| \parallel x - \bar{x} \parallel \leq r\}$, satisfying $B_r(\bar{x}) \cap K_i = \emptyset$. Since $\lim\limits_{q\to\infty} \delta(X_{k_q}) = 0$, 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. $\square$

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 $\bar{X} \cap K \neq \emptyset$, $\lim\limits_{q\to\infty} \beta(X_{k_{q'}}) = z(\bar{X} \cap K)$, where $\bar{X} = \bigcap\limits_q M_{k_q}$. [8].*

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 $\lim\limits_{q\to\infty} \beta(X_{k_q}) = z(\bar{X} \cap K) = \sum\limits_{s\in\mathcal{S}} Q_s(\bar{x})$. Take $\tilde{x}_{k_q,s} \in \text{argmin}\{Q_s(x_s) : x_s \in X_{k_q}\}$, since X_{k_q} shrinks to $\bar{x}$, $\lim\limits_{q\to\infty} \tilde{x}_{k_q,s} = \bar{x}$. Since $\bar{x} \subset X_{k_q}$, it follows that $Q_s(\tilde{x}_{k_q,s}) \leq Q_s(\bar{x})$. From the lower semicontinuity of Q_s, it follows that $Q_s(\bar{x}) \leq \lim\limits_{q\to\infty} Q_s(\tilde{x}_{k_q,s})$. Therefore, $Q_s(\bar{x}) = \lim\limits_{q\to\infty} Q_s(\tilde{x}_{k_q,s}) = \lim\limits_{q\to\infty} \beta_s(X_{k_q})$. Take the sum over s, we obtain $\lim\limits_{q\to\infty} \beta(X_{k_q}) = \sum\limits_{s\in\mathcal{S}} Q_s(\bar{x})$. $\square$

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. [8].*

Algorithm 1 is bound improving since, at every step, the 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 $\lim\limits_{k\to\infty} \beta_k = z$.[8]*

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

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

To prove 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 is satisfied if, for fixed x^*, the Mangasarian-Fromovitz constraint qualification (MFCQ) holds for the scenario subproblems $P_s(x^*),\ s \in \mathcal{S}$ [5].

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

Proof: Because $Q(\cdot)$ is Lipschitz continuous around x^*, there is a ball denoted as $B_r(x^*) = \{x | \parallel x - x^* \parallel \leq r\}$, satisfying $Q(x) - Q(x^*) \leq K\|x - x^*\|$ for all $x \in B_r(x^*)$ and where K is a Lipschitz constant. Therefore, for every point $x \in B_{r'}(x^*)$, we have that $Q(x) - Q(x^*) \leq \epsilon$ holds with $r' = \min(r, \epsilon/K)$. Because the subdivision is exhaustive we have that, after a finite number of iterates $\bar{k}$, the partition considered satisfies $X_{\bar{k}} \subseteq B_{r'}(x^*)$. Because any point $x \in X_{\bar{k}}$ satisfies $Q(x) - Q(x^*) \leq \epsilon$, we have that $\alpha_{\bar{k}} \leq \alpha(X_{\bar{k}}) \leq Q(x^*) + \epsilon$. Because the value of ϵ is arbitrary, we have $\lim\limits_{k\to\infty} Q(x_k) = z$. □

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

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

$$\lim_{k\to\infty} \alpha_k = \lim_{k\to\infty} \beta_k = z. \tag{3.9}$$

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) of the form:

$$z^{LP}(X) = \min_{x\in X, y_s, w_s} \sum_{s\in\mathcal{S}} \bar{f}_s(x, y_s, w_s) \tag{4.10a}$$

$$\text{s.t.}\ \ \bar{g}_{s,j}(x, y_s, w_s) \leq 0, \quad j = 1, \ldots, m_s, s \in \mathcal{S}. \tag{4.10b}$$

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. The LP is also a stochastic programming problem because the underestimators are generated for each scenario subproblem. Since $z^{LP}(X) \leq z(X)$ holds by construction, the solution of the LP relaxation can be used as an alternative lower bound. `SNGO` also adds αBB cuts [15] and reformulation-linearization technique (RLT) cuts [13] 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 then solves the lower bounding problem, which is decomposed into S subproblems of the form (2.7). These subproblems are initialized using the solution of the LP relaxation. 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 $\tilde{x}_s \in \operatorname{argmin}\{Q_s(x_s) : x_s \in X\}$ 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 vaild, or both are vaild. If $\tilde{x}_s \in X_1$, then this implies that $\tilde{x}_s \in \operatorname{argmin}\{Q_s(x_s) : x_s \in X_1\}$. 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 resolved. The solution of lower bounding subproblems is thus stored and passed to children nodes.

To compute upper bounds, SNGO first solves the extensive form (2.4) with a local NLP solver. If the local solver returns a feasible solution, the first stage solution from the local solver can be set as $\hat{x}$ 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 $\hat{x}$. Having $\hat{x}$, the upper bounding problem is solved by solving S seperate 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 3) and then every L_e levels (default of 2) .

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 of first stage variables and solving $2 \cdot n_x$ LPs of the form:

$$\max / \min_{x \in X, y_s, w_s} \; x_i \tag{4.11a}$$

$$\text{s.t.} \;\; \bar{g}_{s,j}(x, y_s, w_s) \leq 0, \quad j = 1, \ldots, m_s, s \in \mathcal{S}. \tag{4.11b}$$

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. 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.

The SNGO implication involves the following major computational steps: 1) solution of S lower bounding subproblems to global optimality, 2) solution of S upper bounding subproblems to global optimality, 3) solution of LP relaxation, 4) solution of extensive form to local optimality, 5) solution of $2 \cdot n_x$ LPs for OBBT, and 6) solution of $2 \cdot n_x$ LPs for branching variable selection. Experiments show that over 90% of the solution time is spent in steps 1) and 2). Assuming the solution time of a subproblem is constant, the solution time per node increases linearly with the number of scenarios. With the number of variables to branch on (i.e., the number of first stage variables) fixed, the number of nodes needed is not expected to explode with the increase in the number of scenarios. We also note that steps 1) 2) 5) 6) are directly parallelizable and 3) 4) can also be parallelized by using solvers such as PIPS and like PIPS-NLP. In this paper, however, we use a fully serial implementation because we aim to compare algorithmic performance with off-the-shelf solvers on an equal basis. We also highlight that parallelization in our context is challenging because of memory management and load imbalancing issues.

We use Plasmo[1] to express the stochastic NLPs under study. Plasmo 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 [9]. Given a graph structure with models and connections, Plasmo 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 Figure 1 illustrates how to implement stochastic problems in Plasmo. As can be seen, the individual scenario models are created and appended to the parent node on-the-fly to create a two-level graph structure. From the snippet we also how to use Plasmo to create a flattened NLP to be solved by off-the-shelf solvers like SCIP [12]. This allows the user to compare computational performance.

5 Computational Experiments

We evaluate the performance of SNGOc by using stochastic NPLs arising from applications such as optimal controller tuning and parameter estimation formulation for microbial community models, and a test set containing stochastic variants of the GLOBALlib set. We compare the computational results against 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

[1] https://github.com/jalving/Plasmo.jl.git

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

```
#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)
```

one of the following conditions: (1) relative optimality gap satisfies $\frac{\alpha_k - \beta_k}{|\beta_k|} \leq 1\%$, (2) absolute optimality gap satisfies $\alpha_k - \beta_k \leq 0.01$, or (3) the search reaches a 12-hour time limit. We use a computing server with Intel(R) Xeon(R) CPU E5-2698 v3 processors running at 2.30GHz 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 (5.12).The first stage variables are the controller parameters (gain K, 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 $s \in \mathcal{S}$. The state trajectories are discretized using an implicit Euler scheme and we note that the number of state variables grows linearly with the number of scenarios. The largest problem solved includes 40 scenarios and has a total of 3203 variables, 3200 constraints, 2323 nonlinear variables, and 3880 nonlinear terms.

Table 1 compares the performance of `SNGO`, `SCIP`, `IPOPT`. We note that, when the number of scenarios is 10 and 40, `SNGO` can solve problems to a gap of 1% while `SCIP` cannot solve the problem within 12 hours. For the problem with 20 scenarios, `SCIP` can solve the problem but it takes over 7 hours while `SNGO` solves the problem in 22 minutes. The key advantage of `SNGO` over `SCIP` is that it only needs to branch on the first 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 22,410 times more than that those visited by `SNGO`. On the other hand, since at each node `SNGO` need to solve subproblems to global optimality, the computational cost of SNGO for each BB node is 1,102 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 highly robust in solving small to medium-sized problems but it is clear that direct branching on all variables is not scalable.

Compared to the local solution found by IPOPT, we see that the solution from SNGO for 10 scenario problems sees an improvement of 17% in the objective value. Interestingly, as the scenarios are increased, the solution found by IPOPT is a global solution. IPOPT, however, cannot certify that this is the case.

$$\min_{x_s(t),K_\text{p},K_\text{i},K_\text{d}} \sum_{s\in\mathcal{S}} \int_0^T e_s(t)^2 dt \tag{5.12a}$$

$$\text{s.t. } \frac{dx_s(t)}{dt} = -\tau_{x,s} x_s(t)^2 + \tau_{u,s} u_s(t) + \tau_{d,s} d_s, s \in \mathcal{S} \tag{5.12b}$$

$$e_s(t) = x_s(t) - \bar{x}_s, s \in \mathcal{S} \tag{5.12c}$$

$$u_s(t) = K_\text{p} e_s(t) + K_\text{i} \int_0^t e_s(\tau)\, d\tau + K_\text{d} \frac{de_s(t)}{dt}, s \in \mathcal{S} \tag{5.12d}$$

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

Problem	SNGO			SCIP 4.0			IPOPT
# S	Time(s)	Gap	# Nodes	Time(s)	Gap	# Nodes	Improve
10	942	1%	58	43200	1.29%	2248314	17.1%
20	1330	1%	37	27045	1.00%	829181	0.0%
40	3862	1%	39	43200	2.16%	595294	0.0%

5.2 Estimation for Microbial Growth Models

We now consider the problem of estimating parameters in microbial growth 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 an scenario and the parameters to be estimated and the variable linking partitions as first stage (complicating) variables. The problem formulation has the form:

$$\min_{x_k(t),\alpha,\beta} \sum_{k\in\mathcal{K}} \int_{t_k}^{t_{k+1}} (x_k(t) - \bar{x}_k)^2 dt \tag{5.13a}$$

$$\text{s.t. } \frac{dx_k(t)}{dt} = \alpha x_k(t)^2 + \beta x_k(t), \quad t \in [t_k, t_{k+1}], \quad k \in \mathcal{K} \tag{5.13b}$$

$$x_{k+1}(t_{k+1}) = x_k(t_{k+1}), \quad k \in \mathcal{K}, \tag{5.13c}$$

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, 943 nonlinear 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 hours, while `SNGO` can solve most of the problems within 20 minutes. The shortest solution time is 8 minutes and the longest solution time is 2 hours. Interestingly, the solution from `SNGO` and `IPOPT` are the same in all data sets. Again, although we do not see an improvement in the objective value over `IPOPT`, `SNGO` provides a certificate that the solution is globally optimal. From an estimation stand-point this is important because it indicates that no better set of model parameters can be found.

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

Problem	`SNGO`		`SCIP`		`IPOPT`
Name	Time(s)	Gap	Time(s)	Gap	Improve
sp.1	7248	1%	43200	592%	0.0%
sp.2	1382	1%	43200	8297%	0.0%
sp.3	1411	1%	43200	14.2%	0.0%
sp.4	2181	1%	1059	0.8%	0.0%
sp.5	591	1%	43200	4052.2%	0.0%
sp.6	1303	1%	43200	1031.6%	0.0%
sp.7	482	1%	321	1.00%	0.0%
sp.8	520	1%	43200	59.37%	0.0%
sp.9	503	1%	43200	25.27%	0.0%
sp.10	1377	1%	43200	113.86%	0.0%
sp.11	730	1%	299	0.29%	0.0%
sp.12	519	1%	43200	280.55%	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 select 28 problems with 20 to 50 variables, and added random perturbations to the right hand size 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 to 50 variables not selected because `SCIP` cannot solve a single scenario problem. The size of the problems depends on the number of scenarios and is outlined in Table 5. The total number of variables when the number of scenarios is 1,000 ranges from 21,005 to 44,005 (these are large-scale instances).

Table 3 and Table 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 25 out of 28 problems. For problems with 1000 scenarios, `SCIP` can only solve 5 problems while `SNGO` can solve 14 problems.

By comparing the results of Table 3 and Table 4 we can again see favorable scalability of `SNGO`. For 13 problems (ex2_1_10, ex8_4_1, hydro, immun, st_fp7b, st_rv2, st_rv3, st_rv7, st_rv8, harker, pollut, ramsey, srcpm), the solution time increases by less than 20 times when the number of scenarios increases from 100 to 1000 (a factor of 10). For 6 problems (abel, ex5_2_5, st_fp7a, st_fp8, and ex8_4_8, ex8_4_8_bnd), the increase in solution time is more dramatic. For the rest 9 problems `SNGO` reaches the time limit but the gap is kept below 20% in most cases (only two instances have a larger gap). There are a few problems where the improvements over `IPOPT` are significant (reaching up to 60,000%). As with the PID controller tuning problem, it is interesting to observe that the improvements over `IPOPT` become significantly less prominent as the number of scenarios increase. This seems to indicate that adding scenarios to the problem helps to better define the problem and eliminates local minima.

Figure 2 shows the progression of the lower and upper bounds for four problems `ex2_1_8, ex5_3_2, ex8_4_1, chenery`. 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). We use crosses to indicate when the bounds from lower and upper bounding problems are the same as the bounds from convexification and local search. Figure 2 shows that, while the lower and upper bounding problems proposed play a significant role, the bounds obtained from convexification and local search also help accelerate the solution process. The gap between the primal problem and the lower bounding problem (the expected value of perfect information (EVPI), is typically small in applications. This can verified by the observation that the gaps at the first iteration of `SNGO` for these four problems are $36.1\%, 14.28\%, 1.4\%, 9.52\%$; while the initial gaps of `SCIP` are $101.5\%, 62.6\%, \geq 10000\%, 10.61\%$. 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 blocks and that branching needs to be performed only on the first-stage variables. We provide a proof of convergence and numerical evidence significantly outperforms the state-of-the-art solver `SCIP` . As a part of future work, we are interested in extend the work to stochastic mixed integer nonlinear programs and to develop a parallel implementation.

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

Problem	`SNGO`		`SCIP`		`IPOPT`
Name	Time(s)	Gap	Time(s)	Gap	Improve
abel	496	1%	10	0.0%	0.0%
ex2_1_10	117	1%	3067	0.8%	0.0%
ex2_1_7	43200	1.3%	43200	222.5%	0.0%
ex2_1_8	6429	1%	43200	54.4%	35.0%
ex5_2_5	93	1%	43200	484.74%	0.0%
ex5_3_2	2384	1%	43200	50.88%	0.0%
ex8_4_1	33429	1%	43200	$\geq$ 10000%	0.0%
hydro	53	1%	7	0.0%	0.0%
immun	3	1%	43200	$\geq$ 10000%	100.07%
st_fp7a	1140	1%	43200	97.5%	55304.6%
st_fp7b	1326	1%	43200	55.1%	38.7%
st_fp7c	901	1%	43200	80.1%	148.1%
st_fp7d	10976	1%	43200	297.1%	4426.3%
st_fp7e	18821	1%	43200	247.8%	61037.5%
st_fp8	279	1%	43200	7.5%	3.6%
st_m1	43200	1.3%	43200	8.1%	0.0%
st_m2	43200	5.8%	43200	25.21%	0.0%
st_rv2	386	1%	43200	4.3%	956.1%
st_rv3	58	1%	43200	7.7%	0.0%
st_rv7	92	1%	43200	2.8%	0.0%
st_rv8	2133	1%	43200	8.9%	0.6%
chenery	14220	1%	43200	10.6%	0.0%
ex8_4_8	19	1%	43200	$\geq$ 10000%	f
ex8_4_8_bnd	8	1%	43200	$\geq$ 10000%	0.0%
harker	261	1%	153	0.0%	0.0%
pollut	14	1%	46	0.0%	0.0%
ramsey	7	1%	2	0.0%	0.0%
srcpm	42	1%	2	0.0%	0.0%

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. Operations Research Letters **33**(1), 42–54 (2005)
2. Birge, J.R., Louveaux, F.: Introduction to stochastic programming. Springer Science & Business Media (2011)
3. CarøE, C.C., Schultz, R.: Dual decomposition in stochastic integer programming. Operations Research Letters **24**(1), 37–45 (1999)

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

Problem	`SNGO`		`SCIP`		`IPOPT`
Name	Time(s)	Gap	Time(s)	Gap	Improve
abel	43200	3.38%	38536	0.0%	0.0%
ex2_1_10	1414	1.0%	5237	0.8%	0.0%
ex2_1_7	43200	1.5%	43200	231.2%	0.0%
ex2_1_8	43200	1.2%	43200	109.9%	24.9%
ex5_2_5	43200	f	43200	f	0.0%
ex5_3_2	43200	1.03%	43200	63.2%	0.0%
ex8_4_1	15782	1.0%	43200	$\geq$ 10000%	0.0%
hydro	520	1.0%	43200	$\geq$ 10000%	0.0%
immun	42	1.0%	43200	$\geq$ 10000%	0.0%
st_fp7a	43200	74.0%	43200	101.9%	0.0%
st_fp7b	1132	1.0%	43200	57.0%	38.7%
st_fp7c	41527	1.0%	43200	83.2%	124.4%
st_fp7d	43200	102%	43200	308.1%	0.0%
st_fp7e	43200	12.8%	43200	257.4%	0.0%
st_fp8	43200	6.9%	43200	5.8%	0.0%
st_m1	43200	3.0%	43200	3490%	0.0%
st_m2	43200	8.1%	43200	$\geq$ 10000%	0.0%
st_rv2	412	1.0%	43200	4.8%	3.7%
st_rv3	493	1.0%	43200	8.2%	0.0%
st_rv7	608	1.0%	43200	3.7%	0.2%
st_rv8	35828	1.0%	43200	f	0.1%
chenery	43200	9.2%	43200	10.6%	0.0%
ex8_4_8	43200	f	43200	$\geq$ 10000%	f
ex8_4_8_bnd	43200	f	43200	$\geq$ 10000%	f
harker	2936	1.0%	43200	$\geq$ 10000%	0.0%
pollut	114	1.0%	1474	0.1%	0.0%
ramsey	37	1.0%	508	0.0%	0.0%
srcpm	253	1.0%	1218	0.0%	0.0%

4. Dür, M., Horst, R.: Lagrange duality and partitioning techniques in nonconvex global optimization. Journal of Optimization Theory and Applications **95**(2), 347–369 (1997)
5. Fiacco, A.V., Ishizuka, Y.: Sensitivity and stability analysis for nonlinear programming. Annals of Operations Research **27**(1), 215–235 (1990)
6. Fisher, M.L.: The lagrangian relaxation method for solving integer programming problems. Management science **27**(1), 1–18 (1981)
7. Geoffrion, A.M.: Generalized benders decomposition. Journal of optimization theory and applications **10**(4), 237–260 (1972)
8. Horst, R., Tuy, H.: Global optimization: Deterministic approaches. Springer Science & Business Media (2013)
9. 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)
10. Karuppiah, R., Grossmann, I.E.: A lagrangean based branch-and-cut algorithm for global optimization of nonconvex mixed-integer nonlinear programs with decomposable structures. Journal of global optimization **41**(2), 163–186 (2008)

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	3000	1500	31005	30000	15000
ex2_1_10	2105	2000	1100	21005	2000	11000
ex2_1_7	2105	2000	1100	21005	20005	11000
ex2_1_8	2505	2400	1100	25005	24000	11000
ex5_2_5	3305	19500	2000	33005	195000	20000
ex5_3_2	2305	2400	1700	23005	24000	1700
ex8_4_1	2305	4000	1100	23005	40000	11000
hydro	3205	1200	2500	32005	12000	25000
immun	2205	600	800	22005	6000	8000
st_fp7a	2105	2000	1100	21005	20000	11000
st_fp7b	2105	2000	1100	21005	20000	11000
st_fp7c	2105	2000	1100	21005	20000	11000
st_fp7d	2105	2000	1100	21005	20000	11000
st_fp7e	2105	2000	1100	21005	20000	11000
st_fp8	2505	2400	2100	25005	24000	21000
st_m1	2105	2000	1200	21005	20000	12000
st_m2	3105	3000	2200	31005	30000	22000
st_rv2	2105	2000	1100	21005	20000	11000
st_rv3	2105	2000	2100	21005	20000	21000
st_rv7	3105	3000	2100	31005	30000	21000
st_rv8	4105	4000	2100	41005	40000	21000
chenery	4405	5600	3900	44005	56000	39000
ex8_4_8	4305	13000	3100	43005	130000	31000
ex8_4_8_bnd	4305	13000	3100	43005	130000	31000
harker	2105	2000	800	21005	20000	8000
pollut	4305	4000	900	43005	40000	9000
ramsey	3405	2200	2300	34005	22000	23000
srcpm	4005	500	2800	40005	5000	28000

11. Li, X., Tomasgard, A., Barton, P.I.: Nonconvex generalized benders decomposition for stochastic separable mixed-integer nonlinear programs. Journal of optimization theory and applications **151**(3), 425 (2011)
12. Maher, S.J., Fischer, T., Gally, T., Gamrath, G., Gleixner, A., Gottwald, R.L., Hendel, G., Koch, T., Lübbecke, M.E., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schenker, S., Schwarz, R., Serrano, F., Shinano, Y., Weninger, D., Witt, J.T., Witzig, J.: The scip optimization suite 4.0. Tech. Rep. 17-12, ZIB, Takustr.7, 14195 Berlin (2017)
13. Misener, R., Floudas, C.A.: Glomiqo: Global mixed-integer quadratic optimizer. Journal of Global Optimization **57**(1), 3–50 (2013)
14. Misener, R., Floudas, C.A.: Antigone: algorithms for continuous/integer global optimization of nonlinear equations. Journal of Global Optimization **59**(2-3), 503–526 (2014)
15. Misener, R., Smadbeck, J.B., Floudas, C.A.: Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into glomiqo 2. Optimization Methods and Software **30**(1), 215–249 (2015)
16. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Mathematical Programming **103**(2), 225–249 (2005)

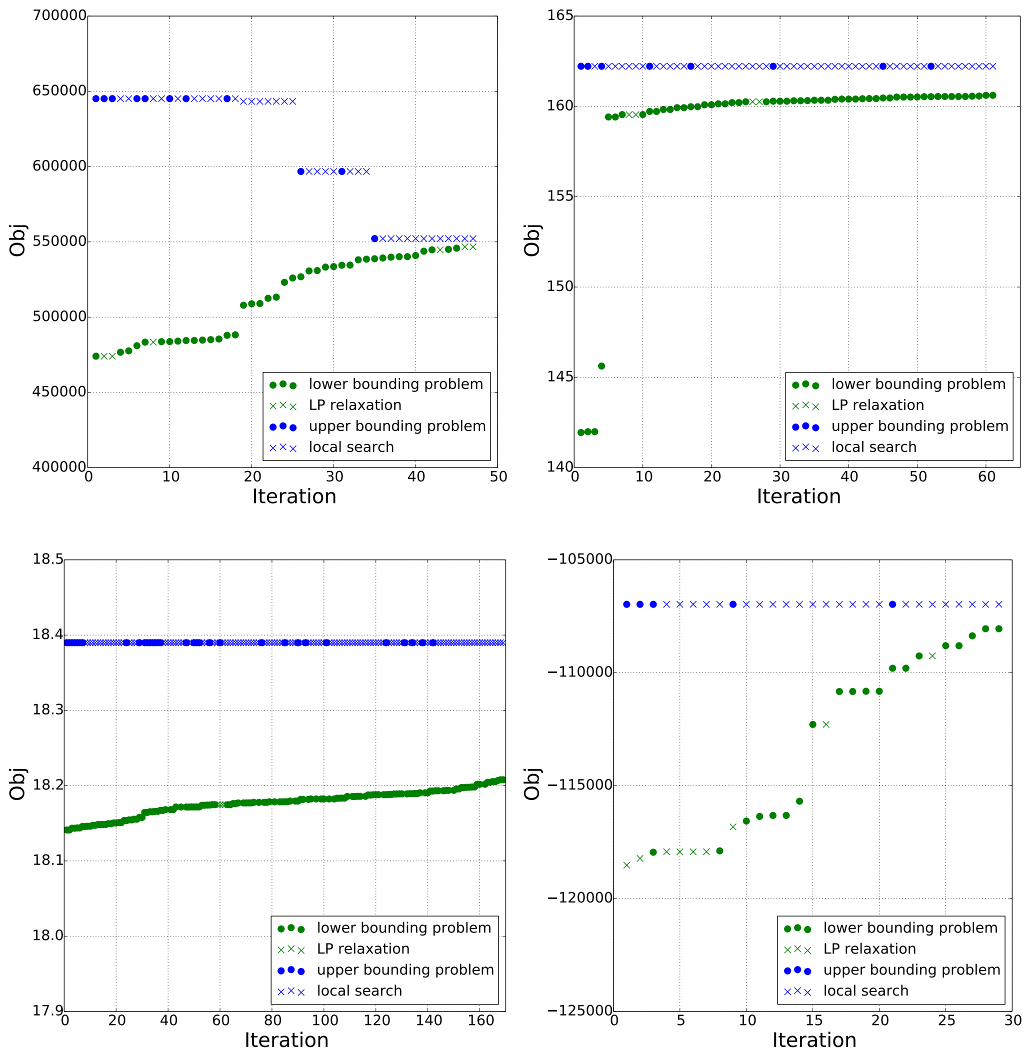

Fig. 2 Evolution of lower and upper bounds for four problem instances.