Math. Program., Ser. A 99: 563–591 (2004)

Digital Object Identifier (DOI) 10.1007/s10107-003-0467-6

Mohit Tawarmalani · Nikolaos V. Sahinidis

Global optimization of mixed-integer nonlinear programs: A theoretical and computational study*

Received: January 24, 2000 / Accepted: July 17, 2003
Published online: August 18, 2003 –

Abstract. This work addresses the development of an efficient solution strategy for obtaining global optima of continuous, integer, and mixed-integer nonlinear programs. Towards this end, we develop novel relaxation schemes, range reduction tests, and branching strategies which we incorporate into the prototypical branch-and-bound algorithm.

In the theoretical/algorithmic part of the paper, we begin by developing novel strategies for constructing linear relaxations of mixed-integer nonlinear programs and prove that these relaxations enjoy quadratic convergence properties. We then use Lagrangian/linear programming duality to develop a unifying theory of domain reduction strategies as a consequence of which we derive many range reduction strategies currently used in nonlinear programming and integer linear programming. This theory leads to new range reduction schemes, including a learning heuristic that improves initial branching decisions by relaying data across siblings in a branch-and-bound tree. Finally, we incorporate these relaxation and reduction strategies in a branch-and-bound algorithm that incorporates branching strategies that guarantee finiteness for certain classes of continuous global optimization problems.

In the computational part of the paper, we describe our implementation discussing, wherever appropriate, the use of suitable data structures and associated algorithms. We present computational experience with benchmark separable concave quadratic programs, fractional 0–1 programs, and mixed-integer nonlinear programs from applications in synthesis of chemical processes, engineering design, just-in-time manufacturing, and molecular design.

1. Introduction

Research in optimization attracted attention when significant advances were made in linear programming in the late 1940's. Motivated by applications, developments in nonlinear programming followed quickly and concerned themselves mostly with local optimization guaranteeing globality only under certain convexity assumptions. However, problems in engineering design, logistics, manufacturing, and the chemical and biological sciences often demand modeling via nonconvex formulations that exhibit multiple local optima. The potential gains to be obtained through global optimization of these problems motivated a stream of recent efforts, including the development of deterministic and stochastic global optimization algorithms.

M. Tawarmalani: Krannert School of Management, Purdue University,
e-mail: mtawarma@mgmt.purdue.edu

N. V. Sahinidis: Department of Chemical and Biomolecular Engineering, University of Illinois at Urbana-Champaign, e-mail: nikos@uiuc.edu

* The research was supported in part by ExxonMobil Upstream Research Company, National Science Foundation awards DMII 95-02722, BES 98-73586, ECS 00-98770, and CTS 01-24751, and the Computational Science and Engineering Program of the University of Illinois.

Solution strategies explicitly addressing mixed-integer nonlinear programs (MINLPs) have appeared rather sporadically in the literature and initially dealt mostly with problems that are convex when integrality restrictions are dropped and/or contain nonlinearities that are quadratic [24, 22, 16, 23, 10, 6, 7, 5]. A few recent works have indicated that the application of deterministic branch-and-bound algorithms to the global optimization of general classes of MINLPs is promising [30, 37, 12, 36]. In this paper, we demonstrate that branch-and-bound is a viable tool for global optimization of purely continuous, purely integer, as well as mixed-integer nonlinear programs if supplemented with appropriate range reduction tools, relaxation schemes, and branching strategies.

The application of the prototypical branch-and-bound algorithm to continuous spaces is outlined in Section 2 and contrasted to application in discrete spaces. The remainder of the paper is devoted to developing theory and methodology for various steps of the branch-and-bound algorithm as applied to MINLP problems. In Section 3, we synthesize a novel linear programming lower bounding scheme by combining factorable programming techniques [26] with the sandwich algorithm [13, 29]. In the context of the latter algorithm, we propose a new variant that is based on a projective error rule and prove that this variant exhibits quadratic convergence.

In Section 4, we develop a new framework for range reduction and show that many existing range reduction schemes in integer and nonlinear programming [41, 18, 17, 20, 34, 30, 31, 35, 43] are special cases of our approach. Additionally, our reduction framework naturally leads to a novel learning reduction heuristic that eliminates search space in the vicinity of fathomed nodes, thereby improving older branching decisions and expediting convergence.

In Section 5, we present a new rectangular partitioning procedure for nonlinear programs. In Section 6, we describe how the branch-and-bound tree can be efficiently navigated and expanded. Finally, in Section 7, we describe computational experience with BARON, the Branch-And-Reduce Optimization Navigator, which is the implementation of the proposed branch-and-bound global optimization algorithm. Computations are presented for continuous, integer, and mixed-integer nonlinear programs demonstrating that a large class of difficult nonconvex optimization problems can be solved in an entirely automated fashion with the proposed techniques.

2. Branch-and-bound in continuous spaces

Consider the following mixed-integer nonlinear program:

$$\text{(P)} \qquad \begin{aligned} \min \ & f(x) \\ \text{s.t.} \ & g(x) \leq 0 \\ & x \in X \subseteq \mathbb{Z}^{n_d} \times \mathbb{R}^{n-n_d} \end{aligned}$$

where $f : X \mapsto \mathbb{R}$ and $g : X \mapsto \mathbb{R}^m$. We restrict attention to factorable functions f and g, *i.e.,* functions that are recursive sums and products of univariate functions. This class of functions suffices to describe most application areas of interest [26]. Examples of factorable functions include $f(x_1, x_2) = x_1 x_2$, $f(x_1, x_2) = x_1/x_2$, $f(x_1, x_2, x_3, x_4) = \sqrt{\exp(x_1 x_2 + x_3 \ln x_4) x_3^3}$, $f(x_1, x_2, x_3, x_4) = \left(x_1^2 x_2^{0.3} x_3\right)/x_4^2 + \exp\left(x_1^2 x_4/x_2\right) - x_1 x_2$,

$f(x) = \sum_{i=1}^{n} \ln_i(x_i)$, and $f(x) = \sum_{i=1}^{T} \prod_{j=1}^{p_i} \left(c_{ij}^0 + c_{ij}x\right)$ with $x \in \mathbb{R}^n$, $c_{ij} \in \mathbb{R}^n$ and $c_{ij}^0 \in \mathbb{R}$ $(i = 1, \ldots, T;\ j = 1, \ldots, p_i)$.

Initially conceived as an algorithm to solve combinatorial optimization problems [21, 9], branch-and-bound has evolved to a method for solving more general multi-extremal problems like P [11, 19]. To solve P, branch-and-bound computes lower and upper bounds on the optimal objective function value over successively refined partitions of the search space. Partition elements are generated and placed on a list of open partition elements. Elements from this list are selected for further processing and further partitioning, and are deleted when their lower bounds are no lower than the best known upper bound for the problem.

A bounding scheme is called *consistent* if any unfathomed partition element can be further refined and, for any sequence of infinitely decreasing partition elements, the upper and lower bounds converge in the limit. Most algorithms ensure consistency through an *exhaustive* partitioning strategy, *i.e.,* one that guarantees that partition elements converge to points or other sets over which P is easily solvable. A selection scheme is called *bound-improving* if a partition with the current lowest lower bound is chosen after a finite number of steps for further bounding and partitioning. If the bounding scheme is consistent and the selection scheme is bound-improving, branch-and-bound is convergent [19].

Remark 1. In $0-1$ programming, branching on a binary variable creates two subproblems in both of which that variable is fixed. On the contrary, branching on a continuous variable in nonlinear programming may require infinitely many subdivisions. As a result, branch-and-bound for $0-1$ programs is finite but merely convergent for continuous nonlinear programs. Results pertaining to finite rectangular branching schemes for continuous global optimization problems have been rather sparse in the literature [35, 3, 1]. Finiteness for these problems was obtained based on branching schemes that are exhaustive and branch on the incumbent, and/or explicitly took advantage of special problem structure.

Remark 2. While dropping the integrality conditions leads to a linear relaxation of an MILP, dropping integrality conditions may not suffice to obtain a useful relaxation of an MINLP. Indeed, the resulting NLP may involve nonconvexities that make solution difficult. It is even possible that the resulting NLP may be harder to solve to global optimality than the original MINLP.

Remark 3. Linear programming technology has reached a level of maturity that provides robust and reliable software for solving linear relaxations of MILP problems. Nonlinear programming solvers, however, often fail even in solving convex problems. At the theoretical level, duality theory provides necessary and sufficient conditions for optimality in linear programming, whereas the Karush-Kuhn-Tucker (KKT) optimality conditions, that are typically exploited by nonlinear programming algorithms, are not even necessary unless certain constraint qualifications hold.

In the sequel, we address the development of branch-and-bound algorithms for MINLPs. Branch-and-bound is more of a prototype of a global optimization algorithm than a formal algorithmic specification since it employs a number of schemes that may be

tailored to the application at hand. In order to derive an efficient algorithm, it is necessary to study the various techniques for relaxation construction, domain reduction, upper bounding, partitioning, and node selection. Also, a careful choice of data structures and associated algorithms is necessary for developing an efficient implementation. We develop theory and methodology behind these algorithmic components in the next four sections.

3. Lower bounding

In this section, we address the question of constructing a lower bounding procedure for the factorable programming problem P stated in Section 2. The proposed methodology starts by using the following recursive algorithm to decompose factorable functions in terms of sums and products of univariate functions that are subsequently bounded to yield a relaxation of the original problem:

Algorithm Relax f(x)
If $f(x)$ is a function of single variable $x \in [x^l, x^u]$, then
    Construct under- and over-estimators for $f(x)$ over $[x^l, x^u]$,
else if $f(x) = g(x)/h(x)$, then
        Fractional_Relax (f, g, h)
    end of if
else if $f(x) = \prod_{i=1}^{t} f_i(x)$, then
    for $i := 1$ to t do
        Introduce variable y_{f_i}, such that $y_{f_i} =$ Relax $f_i(x)$
    end of for
    Introduce variable y_f, such that $y_f =$ Multilinear_Relax $\prod_{i=1}^{t} y_{f_i}$
else if $f(x) = \sum_{i=1}^{t} f_i(x)$, then
    for $i := 1$ to t do
        Introduce variable y_{f_i}, such that $y_{f_i} =$ Relax $f_i(x)$
    end of for
    Introduce variable y_f, such that $y_f = \sum_{i=1}^{t} y_{f_i}$
else if $f(x) = g(h(x))$, then
    Introduce variable $y_h =$ Relax $h(x)$
    Introduce variable $y_f =$ Relax $g(y_h)$
end of if

Note that while decomposing $f(x)$ as a product/sum of functions, a variable is introduced if and only if another variable corresponding to the same function has not been introduced at an earlier step.

Theorem 1 ([26]). *Consider a function $f(x) = g(h(x))$ where $h : \mathbb{R}^n \mapsto \mathbb{R}$ and $g : \mathbb{R} \mapsto \mathbb{R}$. Let $S \subseteq \mathbb{R}^n$ be a convex domain of h and let $a \leq h(x) \leq b$ over S. Assume $c(x)$ and $C(x)$ are, respectively, a convex underestimator and a concave overestimator of $h(x)$ over S. Let $e(\cdot)$ and $E(\cdot)$ be the convex and concave envelopes of $g(h)$ for $h \in [a, b]$. Also, let $h_{\min}$ and $h_{\max}$ be such that $g(h_{\min}) = \inf_{h\in[a,b]}(g(h))$*

and $g(h_{\max}) = \sup_{h\in[a,b]}(g(h))$. *Finally, let* $\operatorname{mid}[r, s, t]$ *denote the second largest number amongst* r, s *and* t. *Then,* $e\left(\operatorname{mid}\left[c(x), C(x), h_{\min}\right]\right)$ *is a convex underestimating function and* $E\left(\operatorname{mid}\left[c(x), C(x), h_{\max}\right]\right)$ *is a concave overestimating function for* $f(x)$.

Theorem 2. *The underestimator and overestimator in Theorem 1 are implied in the relaxation derived through the application of Algorithm Relax as long as* $e(\cdot)$ *and* $E(\cdot)$ *are used as the under- and over-estimators for* $g(y_h)$.

Proof. Consider the $(n+2)$-dimensional space of points of the form (x, h, f). Construct the surface $M = \{(x, h, f) \mid f = g(h)\}$, the set $F = \{(x, h, f) \mid e(h) \le f \le E(h)\}$, and the set $H = \{(x, h, f) \mid c(x) \le h \le C(x), x \in S\}$. Then, the feasible region as constructed by Algorithm Relax is the projection of $F \cap H$ on the space of x and f variables assuming $h(x)$ does not occur elsewhere in the factorable program. The resulting set is convex since intersection and projection operations preserve convexity. For a given x, let us characterize the set of feasible points, (x, f). Note that $(x, f) \in F \cap H$ as long as $f \ge f_{\min} = \min\{e(h) \mid h \in [c(x), C(x)]\}$. If $h_{\min} \le c(x)$, the minimum value of $e(h)$ is attained at $c(x)$, by quasiconvexity of e. Similarly, if $c(x) \le h_{\min} \le C(x)$, then the minimum is attained at $h_{\min}$ and, if $C(x) \le h_{\min}$, then the minimum is attained at $C(x)$. The resulting function is exactly the same as described in Theorem 1. The overestimating function follows by a similar argument. □

We chose to relax the composition of functions in the way described in Algorithm Relax instead of constructing underestimators based on Theorem 1 because our procedure is simpler to implement, and may lead to tighter relaxations if $h(x)$ occurs more than once in P. On the other hand, the scheme implied in Theorem 1 introduces one variable lesser than that in Algorithm Relax for every univariate function relaxed by the algorithm.

We now address the relaxation of special functions, beginning with the simple monomial $y = \prod_{i=1}^{t} y_{f_i}$. Various relaxation strategies are possible for simple monomials as well as more general multilinear functions. For example, in [40], a complete characterization of the generating set of the convex polyhedral encloser of a multilinear function over any hypercube is provided in terms of its extreme points. While this implies trivially the polyhedral description of the convex encloser of the product of t variables (cf. Theorem 4.8, p. 96 [27]), this description is exponential in t. To keep the implementation simple, we instead employ a recursive arithmetic interval scheme for bounding the product.

Algorithm Multilinear_Relax $\prod_{\mathbf{i=1}}^{\mathbf{t}} \mathbf{y_{r_i}}$ (Recursive Arithmetic)

for $i := 2$ to t do
  Introduce variable $y_{r_1,\ldots,r_i} =$ Bilinear_Relax $y_{r_1,\ldots,r_{i-1}} y_{r_i}$.
end of for

The fractional term is relaxed using the algorithm developed in [39].

Algorithm Fractional_Relax $(\mathbf{f}, \mathbf{g}, \mathbf{h})$ (Product Disaggregation)

Introduce variable y_f
Introduce variable $y_g =$ Relax $g(x)$

if $h(x) = \sum_{i=1}^{t} h_i(x)$, then
  for $i := 1$ to t do
    Introduce variable $y_{f,h_i} = \text{Relax}\left(y_f h_i(x)\right)$
  end of for
  Introduce relation $y_g = \sum_{i=1,\ldots,t} y_{f,h_i}$
else
  Introduce variable $y_h = \text{Relax } h(x)$
  Introduce relation $y_g = \text{Bilinear\_Relax } y_f y_h$
end of if

The bilinear term is relaxed using its convex and concave envelopes [26, 2].

Algorithm Bilinear_Relax $\mathbf{y_i y_j}$ (Convex/Concave Envelope)

$$
\begin{array}{lcl}
\text{Bilinear\_Relax } y_i y_j & \geq & y_i^u y_j + y_j^u y_i - y_i^u y_j^u \\
\text{Bilinear\_Relax } y_i y_j & \geq & y_i^l y_j + y_j^l y_i - y_i^l y_j^l \\
\text{Bilinear\_Relax } y_i y_j & \leq & y_i^u y_j + y_j^l y_i - y_i^u y_j^l \\
\text{Bilinear\_Relax } y_i y_j & \leq & y_i^l y_j + y_j^u y_i - y_i^l y_j^u
\end{array}
$$

We are now left with the task of outer-estimating the univariate functions. At present, we do not attempt to detect individual functional properties, but decompose a univariate function as a recursive sum and product of monomial, logarithmic, and power terms. This keeps the implementation simple and accounts for most of the functions typically encountered in practical problems. More functions may be added to those listed above without much difficulty.

Algorithm Univariate_Relax $\mathbf{f(x_j)}$ (Recursive Sums and Products)

if $f(x_j) = cx_j^p$ then
  Introduce variable $y_f = \text{Monomial\_Relax } cx_j^p$
else if $f(x_j) = cp^{x_j}$ then
  Introduce variable $y_f = \text{Power\_Relax } cp^{x_j}$
else if $f(x_j) = c\log(x_j)$ then
  Introduce variable $y_f = \text{Logarithmic\_Relax } c\log(x_j)$
else if $f(x_j) = \prod_{i=1}^{t} f_i(x_j)$, then
  for $i := 1$ to t do
    Introduce variable y_{f_i}, such that $y_{f_i} = \text{Relax } f_i(x)$
  end of for
  Introduce variable y_f, such that $y_f = \text{Multilinear\_Relax } \prod_{i=1}^{t} y_{f_i}$
else if $f(x_j) = \sum_{i=1}^{t} f_i(x_j)$, then
  for $i := 1$ to t do
    Introduce variable y_{f_i}, such that $y_{f_i} = \text{Relax } f_i(x_j)$
  end of for
  Introduce variable y_f, such that $y_f = \sum_{i=1}^{t} y_{f_i}$
end of if

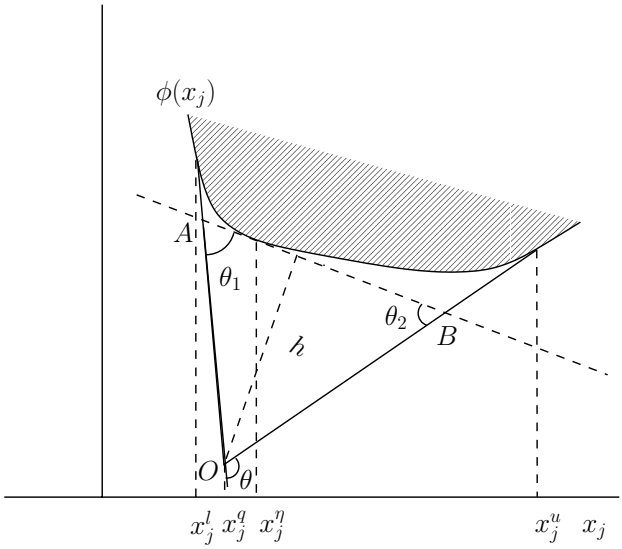

Fig. 1. Outer-approximation of convex functions.

We do not detail each of the procedures Monomial_Relax , Power_Relax and Logarithmic_Relax, as they may be easily derived by constructing the corresponding convex/concave envelopes over $[x_j^l, x_j^u]$. In [38], we address some common characteristic problems that arise in the derivation of these envelopes.

3.1. Outer-approximation

The relaxations to factorable programs derived above are typically nonlinear. However, nonlinear programs are harder to solve and are associated with more numerical issues than linear programs. For this reason, we propose the polyhedral outer-approximation of the above-derived nonlinear relaxations, thus generating an entirely linear programming based relaxation.

Consider a convex function $g(x)$ and the inequality $g(x) \leq 0$. With $s \in \partial g(\overline{x})$, we have the subgradient inequality:

$$0 \geq g(x) \geq g(\overline{x}) + s(x - \overline{x}). \tag{1}$$

We are mainly interested in outer-approximating univariate convex functions that lead to the inequality: $\phi(x_j) - y \leq 0$ (see Figure 1). Then, (1) reduces to:

$$0 \geq \phi(x_j) - y \geq \phi(\overline{x}_j) + s(x_j - \overline{x}_j) - y.$$

To construct a polyhedral outer-approximation, we need to locate points x_j in $[x_j^l, x_j^u]$ such that the subgradient inequalities approximate $\phi(x_j)$ closely.

The problem of outer-approximating convex functions is closely related to that of polyhedral approximations to convex sets [15, 14]. It is known that the distance between

a planar convex figure and its best approximating $n-$gon is $O(1/n^2)$ under various error measures like Hausdorff, and area of the symmetric difference. For convex functions, with vertical distance as the error measure, the convergence rate of the error of the approximation can be shown to be, at best, $O(1/n^2)$ by considering a parabola $y = x^2$ [29].

The sandwich algorithm is a template of outer-approximation schemes that attains the above asymptotic performance guarantee in a variety of cases [13, 29]. At a given iteration, this algorithm begins with a number of points at which tangential outer-approximations of the convex function have been constructed. Then, at every iterative step, the algorithm identifies the interval with the maximum outer-approximation error and subdivides it at a suitably chosen point. Five known strategies for identifying such a point are as follows:

1. *Interval bisection:* bisect the chosen interval.
2. *Slope bisection:* find the supporting line with a slope that is the mean of the slopes at the end points.
3. *Maximum error rule:* Construct the supporting line at the $x-$ordinate of the point of intersection of the supporting lines at the two end points.
4. *Chord rule:* Construct the supporting line with the slope of the linear overestimator of the function.
5. *Angle bisection:* Construct the supporting line with the slope of the angular bisector of the outer angle θ of ROS.

It was shown in [29] that, if the vertical distance between the convex function and the outer-approximation is taken as the error measure, then the first four procedures lead to an optimal performance guarantee. If x_j^{l*} is the right-hand derivative at x_j^l and x_j^{u*} is the left-hand derivative at x_j^u, then, for $n \geq 2$, interval and slope bisection schemes produce a largest vertical error ϵ such that:

$$\epsilon \leq \frac{9(x_j^u - x_j^l)(x_j^{u*} - x_j^{l*})}{8n^2}.$$

The maximum error and chord rules have a better performance guarantee of:

$$\epsilon \leq \frac{(x_j^u - x_j^l)(x_j^{u*} - x_j^{l*})}{n^2}.$$

Note that, since the initial $x_j^{u*} - x_j^{l*}$ can be rather large, the sandwich algorithm may still lead to high approximation errors for low values of n.

Let $\mathrm{G}(f)$ denote the set of points (x, y) such that $y = f(x)$. Projective error is another error measure which is commonly used to describe the efficiency of an approximation scheme. If $\phi^o(x_j)$ is the outer-approximation of $\phi(x_j)$, then

$$\epsilon_p = \sup_{p^{\phi^o} \in \mathrm{G}(\phi^o)} \inf_{p^{\phi} \in \mathrm{G}(\phi)} \left\{ \| p^{\phi} - p^{\phi^o} \| \right\}. \tag{2}$$

It was shown in [13] that the projective error of the angular bisection scheme decreases quadratically. In particular, if the initial outer angle of the initial function is θ, then:

$$\epsilon_p \leq \frac{9\theta(x_j^u - x_j^l)}{8(n-1)^2}.$$

We suggest another scheme where the supporting line is drawn at the point in $G(\phi)$ which is closest to the intersection point of the tangents at the two endpoints. We call this the *projective error rule*. We next show that the resulting sandwich algorithm exhibits quadratic convergence.

Lemma 1. *If a, b, c, and d are four positive numbers such that $ac = bd$, then $ac + bd \leq ad + bc$.*

Proof. $ac = bd$ implies that $a \geq b$ if and only if $c \leq d$. Therefore, $(b - a)(c - d) \geq 0$. Expanding, we obtain the desired inequality. □

Theorem 3. *Consider a convex function $\phi(x_j) : [x_j^l, x_j^u] \mapsto \mathbb{R}$ and the outer-approximation of ϕ formed by the tangents at x_j^l and x_j^u. Let $R = \big(x_j^l, \phi(x_j^l)\big)$ and $S = \big(x_j^u, \phi(x_j^u)\big)$. Assume that the tangents at R and S intersect at O and let θ be $\pi - \angle ROS$, and L be the sum of the lengths of RO and OS. Let $k = L\theta/\epsilon_p$. Then, the algorithm needs at most*

$$N(k) = \begin{cases} 0, & k \leq 4; \\ \lceil \sqrt{k} - 2 \rceil, & k > 4, \end{cases}$$

supporting lines for the outer-approximation to approximate $\phi(x_j)$ within ϵ_p.

Proof. We argue that the supremum in (2) is attained at O (see Figure 2). Indeed, since $\phi(x_j)$ is a convex function, as we move from R to S the derivative of ϕ increases, implying that the length of the perpendicular between the point on the curve and RO increases attaining its maximum at A. For any two points X and Y on the curve, we denote by L_{XY}, θ_{XY}, and W_{XY} the length of the line segment joining X and Y, the angle $\pi - \angle XOY$, and $L_{XO} + L_{OY}$, respectively.

If $\phi(x_j)$ is not approximated within ϵ_p, then

$$\epsilon_p < L_{AO} = L_{O_1A} \tan\theta_{RA} = L_{O_1O} \sin\theta_{RA} \leq L_{RO}\theta_{RA}. \tag{3}$$

Similarly, $\epsilon_p < L_{OS}\theta_{AS}$.

We prove the result by induction on $W_{RS}\theta_{RS}/\epsilon_p$. For the base case:

$$\begin{aligned} W_{RS}\theta_{RS} &= \big(L_{RO} + L_{OS}\big)\big(\theta_{RA} + \theta_{AS}\big) \\ &\geq \big(L_{O_1O} + L_{OO_2}\big)\big(\sin\theta_{RA} + \sin\theta_{AS}\big) \\ &\geq 2\big(L_{O_1O}\sin\theta_{RA} + L_{OO_2}\sin\theta_{AS}\big) \qquad \text{(from Lemma 1)} \\ &= 4L_{AO}. \end{aligned}$$

It follows that, if $W_{RS}\theta_{RS} \leq 4\epsilon_p$, the curve is approximated within tolerance.

To prove the inductive step, we need to show that:

$$\begin{aligned} N(W_{RS}\theta_{RS}/\epsilon_p) \geq \max\ & 1 + N(W_{RA}\theta_{RA}/\epsilon_p) + N(W_{AS}\theta_{AS}/\epsilon_p) \\ \text{s.t. } & W_{RA} = L_{RO_1} + L_{O_1A} \\ & W_{AS} = L_{AO_2} + L_{O_2S} \\ & W_{RS} = L_{RO_1} + L_{O_1A}/\cos\theta_{RA} + L_{AO_2} + L_{O_2S}/\cos\theta_{AS} \\ & \theta_{RS} = \theta_{RA} + \theta_{AS} \\ & L_{O_1A}\tan\theta_{RA} > \epsilon_p \\ & L_{AO_2}\tan\theta_{AS} > \epsilon_p \\ & W_{RA}, W_{AS}, L_{RO_1}, L_{O_1A}, L_{AO_2}, L_{O_2S}, \theta_{RA}, \theta_{AS} \geq 0. \end{aligned}$$

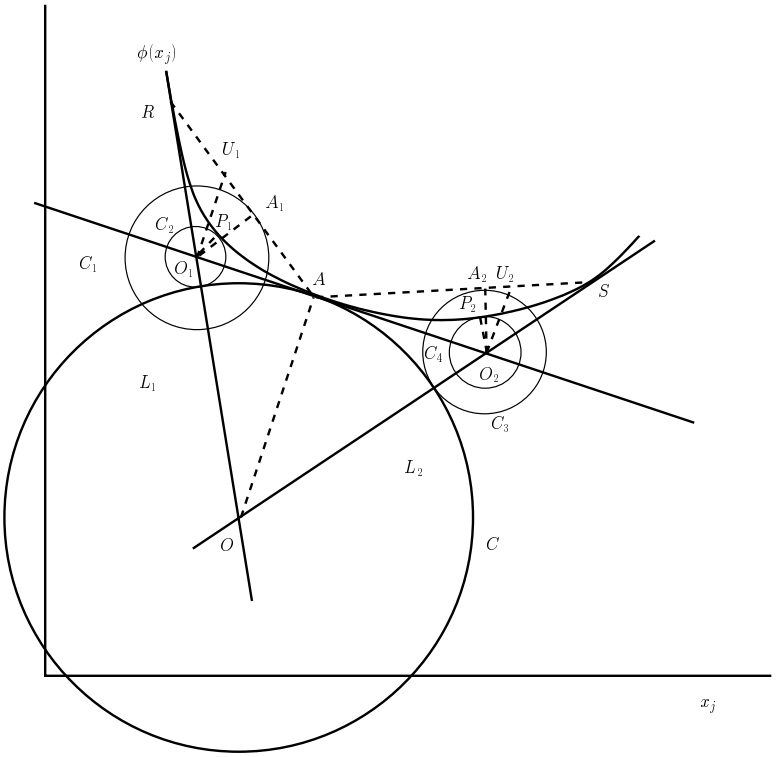

Fig. 2. Proof of Theorem 3.

Since $N(\cdot)$ is an increasing function of $(\cdot)$, $W_{RA} \leq L_{RO_1} + L_{O_1A}/\cos\theta_{RA} = L_{RO}$, and $W_{AS} \leq L_{AO_2} + L_{O_2S}/\cos\theta_{AS} = L_{OS}$, we relax the above mathematical program by introducing variables L_{RO} and L_{OS} and prove the following tighter result:

$$\begin{aligned} N(W_{RS}\theta_{RS}/\epsilon_p) \geq \max\ & 1 + N(L_{RO}\theta_{RA}/\epsilon_p) + N(L_{OS}\theta_{AS}/\epsilon_p) \\ \text{s.t. } & \theta_{RS} = \theta_{RA} + \theta_{AS} \\ & W_{RS} = L_{RO} + L_{OS} \\ & L_{RO}\theta_{RA} > \epsilon_p \qquad \text{(see (3))} \\ & L_{OS}\theta_{AS} > \epsilon_p \\ & L_{RO}, L_{OS}, \theta_{RA}, \theta_{AS} \geq 0. \end{aligned}$$

Since $\sqrt{L_{RO}\theta_{RA}/\epsilon_p} \geq 1$ and $\sqrt{L_{OS}\theta_{AS}/\epsilon_p} \geq 1$, we can apply the induction hypothesis. We only need to show that:

$$\left\lceil \sqrt{W_{RS}\theta_{RS}/\epsilon_p} - 2 \right\rceil \geq 1 + \max\left\{ \left\lceil \sqrt{L_{RO}\theta_{RA}/\epsilon_p} - 2 \right\rceil + \left\lceil \sqrt{L_{OS}\theta_{AS}/\epsilon_p} - 2 \right\rceil \right\}.$$

Since $\lceil c-2\rceil \geq 1 + \lceil a-2\rceil + \lceil b-2\rceil$ when $c \geq a+b$, we prove the result by showing that:

$$\sqrt{W_{RS}\theta_{RS}/\epsilon_p} \geq \max\left\{ \sqrt{L_{RO}\theta_{RA}/\epsilon_p} + \sqrt{L_{OS}\theta_{AS}/\epsilon_p} \right\},$$

where $L_{RO} + L_{OS} = W_{RS}$, and $\theta_{RA} + \theta_{AS} = \theta_{RS}$. Note that:

$$\begin{aligned}
&\left(\sqrt{L_{RO}\theta_{RA}} + \sqrt{L_{OS}\theta_{AS}}\right)^2 \\
&\quad \leq \left(\sqrt{L_{RO}\theta_{RA}} + \sqrt{L_{OS}\theta_{AS}}\right)^2 + \left(\sqrt{L_{RO}\theta_{AS}} - \sqrt{L_{OS}\theta_{RA}}\right)^2 \\
&\quad = (L_{RO} + L_{OS})(\theta_{RA} + \theta_{AS}) \\
&\quad = W_{RS}\theta_{RS}.
\end{aligned}$$

Dividing by ϵ_p and taking the square root, we get:

$$\sqrt{L_{RO}\theta_{RA}/\epsilon_p} + \sqrt{L_{OS}\theta_{AS}/\epsilon_p} \leq \sqrt{W_{RS}\theta_{RS}/\epsilon_p}.$$

Hence, the result is proven. □

Implicit in the statement of the above theorem is the assumption that $\theta < \pi$ which is needed to ensure that the tangents at the end-points of the curve intersect. This is not a restrictive assumption since the curve can be segmented into two parts to ensure that the assumption holds.

The above result shows that the convergence of the approximation gap is quadratic in the number of points used for outer-approximation. This result can be extended to other error measures, such as the area of the symmetric difference between the linear overestimator and the outer-approximation simply because the area of the symmetric difference grows super-linearly with the projective distance h. We provide a more in-depth analysis of the algorithm in [38].

Remark 4. Examples were constructed in [29] to show that there exist functions for which each of the above outer-approximation schemes based on interval, slope, and angle bisection, as well as the maximum and chord error rules perform arbitrarily worse compared to the others. We are not aware of any function for which the proposed projective error rule will produce weak approximations.

4. Domain reduction

Domain reduction, also known as range reduction, is the process of cutting regions of the search space that do not contain an optimal solution. Various techniques for range reduction have been developed in [41, 18, 17, 20, 30, 31, 35, 43]. We develop a theory of range reduction tools in this section and then derive earlier results in the light of these new developments.

4.1. Theoretical framework

Consider a mathematical programming model:

$$\text{(T)} \qquad \begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq b \\ & x \in X \subseteq \mathbb{R}^n \end{aligned}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$, $g : \mathbb{R}^n \mapsto \mathbb{R}^m$, and X denotes the set of "easy" constraints. The standard definition of the Lagrangian subproblem for T is:

$$\inf_{x\in X} l(x, y) = \inf_{x\in X}\big\{f(x) - y\big(g(x) - b\big)\big\}, \text{ where } y \leq 0. \tag{4}$$

Instead, we define the Lagrangian subproblem as:

$$\inf_{x\in X} l'(x, y_0, y) = \inf_{x\in X}\big\{-y_0 f(x) - yg(x)\big\}, \text{ where } (y_0, y) \leq 0.$$

The additional dual variable y_0 homogenizes the problem and allows us to provide a unified algorithmic treatment of range-reduction problems. Since yb is a constant as far as the minimization in (4) is concerned, we account for it in the Lagrangian master problem.

Assume that b_0 is an upper bound on the optimal objective function value of T and consider the following range-reduction problem:

$$h^* = \inf_{x,u_0,u}\big\{h(u_0, u) \mid f(x) \leq u_0 \leq b_0, g(x) \leq u \leq b, x \in X\big\}, \tag{5}$$

where h is assumed to be a linear function. Problem (5) can be restated as:

$$(\mathrm{R}_{\mathrm{u}}^{\mathrm{C}}) \qquad \begin{aligned} h^* = \inf_{x,u_0,u}\ & h(u_0, u) \\ \text{s.t. }\ & -y_0\big(f(x) - u_0\big) - y\big(g(x) - u\big) \leq 0 \quad \forall\, (y_0, y) \leq 0 \\ & (u_0, u) \leq (b_0, b),\ x \in X. \end{aligned}$$

Computing h^* is as hard as solving T. Therefore, we lower bound h^* with the optimal value of the following problem:

$$(\overline{\mathrm{R}_{\mathrm{u}}^{\mathrm{C}}}) \qquad \begin{aligned} h_L = \inf_{u_0,u}\ & h(u_0, u) \\ \text{s.t. }\ & y_0u_0 + yu + \inf_{x\in X}\big\{-y_0 f(x) - yg(x)\big\} \leq 0 \quad \forall\, (y_0, y) \leq 0 \\ & (u_0, u) \leq (b_0, b). \end{aligned}$$

Much insight into our definition of the domain reduction problem and its potential uses is gained by restricting $h(u_0, u)$ to $a_0u_0 + au$ where $(a_0, a) \geq 0$ and $(a_0, a) \neq 0$. Interesting applications arise when (a_0, a) is set to one of the principal directions in $\mathbb{R}^{m+1}$.

Using Fenchel-Rockafellar duality, the following algorithm is derived in [38] to iteratively obtain lower and upper bounds on the range-reduction problem with a linear objective. The algorithm is similar to the iterative algorithm commonly used to solve the Lagrangian relaxation of T. The treatment herein can be generalized to arbitrary lower semicontinuous functions $h(u, u_0)$. We refer the reader to [38] for a detailed discussion, related proofs, and generalizations.

Algorithm SimpleReduce

Step 0. Set $K = 0$, $u_0^0 = b_0$, $u^0 = b$.

Step 1. Solve the relaxed dual of (5):

$$\begin{aligned} h_U^K = \max_{(y_0, y)} \quad & (y_0 + a_0)b_0 + (y + a)b - z \\ \text{s.t.} \quad & z \geq y_0 u_0^k + y u^k \qquad k = 0, \ldots, K-1 \\ & (y_0, y) \leq -(a_0, a) \end{aligned}$$

Let the solution be $\left(y_0^K, y^K\right)$.

Step 2. Solve the Lagrangian subproblem:

$$\begin{aligned} \inf_x l'(x, y_0^K, y^K) = -\max_{x, u_0, u} \quad & y_0^K u_0 + y^K u \\ \text{s.t.} \quad & f(x) \leq u_0 \\ & g(x) \leq u \\ & x \in X \end{aligned}$$

Let the solution be (x^K, u_0^K, u^K). Note that $(y_0, y) \leq 0$ implies that $u_0^K = f(x^K)$ and $u^K = g(x^K)$.

Step 3. Augment and solve the relaxed primal problem:

$$\begin{aligned} h_L^K = \min_{(u_0, u)} \quad & a_0 u_0 + a u \\ \text{s.t.} \quad & y_0^k u_0 + y^k u + \inf_x l'(x, y_0^k, y^k) \leq 0 \qquad k = 1, \ldots, K \\ & (u_0, u) \leq (b_0, b) \end{aligned}$$

Step 4. Termination check: If $h_U^K - h_L^K \leq$ tolerance, stop; otherwise, set $K = K + 1$ and goto Step 1.

The reader will notice that the Algorithm SimpleReduce reduces to the standard Lagrangian lower bounding procedure for problem T when $(a_0, a) = (1, 0)$. In this case, y_0 can be fixed to -1 and the relaxed master problem is equivalent to maintaining the maximum bound obtained from the solved Lagrangian subproblems: $\max_{k=1,\ldots,K} \inf_{x \in X} l(x, y)$. Similarly, Algorithm SimpleReduce reduces to the standard Lagrangian lower bounding procedure for the problem:

$$(\mathrm{R}_{\mathrm{g_i}}) \qquad \begin{aligned} \min \quad & g_i(x) \\ \text{s.t.} \quad & g(x) \leq b \\ & f(x) \leq b_0 \\ & x \in X \subseteq \mathbb{R}^n \end{aligned}$$

when $a_i = 1$ and $a_j = 0$ for all $j \neq i$. Since the Lagrangian subproblem is independent of (a_0, a) and homogenous in (y_0, y), the cuts $z \geq y_0 u_0^k + y u^k$ and $y_0^k u_0 + y^k u + \inf_x l(x, y_0^k, y^k) \leq 0$ derived during the solution of the Lagrangian relaxation at any node in a branch-and-bound tree can be used to hot-start the Lagrangian procedure that determines lower bounds for $g_i(x)$ as long as (y_0^k, y^k) can be scaled to be less than or equal to (a_0, a). Note that it may not always be possible to solve the Lagrangian subproblem efficiently, which may itself be NP-hard. In such cases, any suitable relaxation of the Lagrangian subproblem can be used instead to derive a weaker lower bound on $h(u_0, u)$.

4.2. Applications to polyhedral sets

We shall, as an example, specialize some of the results of the previous section to polyhedral sets using linear programming duality theory. Consider the following dual linear programs:

$$\begin{array}{lll} \text{(PP)} & \min & cx \\ & \text{s.t.} & Ax \leq b \end{array} \quad \underset{\Longleftrightarrow}{\text{Duality}} \quad \begin{array}{lll} \text{(PD)} & \max & \lambda b \\ & \text{s.t.} & \lambda A = c \\ & & \lambda \leq 0 \end{array}$$

where $A \in \mathbb{R}^{n \times m}$. Construct the following perturbation problem and its dual:

$$\begin{array}{lll} \text{(PU)} & \min & u_i \\ & \text{s.t.} & Ax \leq b + e_i u_i \\ & & cx \leq U \end{array} \quad \underset{\Longleftrightarrow}{\text{Duality}} \quad \begin{array}{lll} \text{(PR)} & \max & rb + sU \\ & \text{s.t.} & rA + sc = 0 \\ & & -re_i = 1 \\ & & r, s \leq 0. \end{array}$$

Note that the optimal objective function value of PU provides a lower bound for $A_i x - b_i$ assuming that U is an upper bound on cx. In particular, PU is the range-reduction problem defined in (5) when $h(u_0, u) = u_i$. Then, PR models the dual of PU which is equivalent to the following Lagrangian relaxation:

$$\max_{y \geq 0, y_i = 1} \min_{x \in X} y(Ax - b) + b_i + y_0(cx - U)$$

In this case, $l'(x, y_0^k, y^k) = y_0 cx + yAx$ and moving the terms independent of x outside the inner minimization we obtain the Lagrangian relaxation:

$$\max_{y \geq 0, y_i = 1} -y_0 U - yb + b_i + \min_{x \in X} y_0 cx + yAx$$

Algorithm SimpleReduce then reduces to the outer-approximation procedure to solve the Lagrangian dual (see [4]). Let $(\bar{r}, \bar{s})$ be a feasible solution to the above problem. Then, either $s = 0$ or $s < 0$.

- Case 1: $s < 0$. The set of solutions to PR are in one-to-one correspondence with the solutions to PD by the following relation:

$$s = 1/\lambda_i, \quad r = -\lambda s.$$

- Case 2: $s = 0$.

$$\begin{array}{lll} \text{(PRM)} & \max & rb \\ & \text{s.t.} & rA = 0 \\ & & -re_i = 1 \\ & & r \leq 0 \end{array} \quad \underset{\Longleftrightarrow}{\text{Duality}} \quad \begin{array}{lll} \text{(PUM)} & \min & u_i \\ & \text{s.t.} & Ax \leq b + e_i u_i. \end{array}$$

  The dual solutions are in direct correspondence with those of PUM.

Note that PR and PD both arise from the same polyhedral cone:

$$rA + sc = 0$$
$$r, s \leq 0$$

with different normalizing planes $s = -1$ and $-re_i = 1$. Therefore, apart from the hyperplane at infinity, the correspondence between both sets is one-to-one. As a result, dual solutions of PP (not necessarily optimal) can be used to derive lower bounds for PU. The dual solutions are useful in tightening bounds on the variables when some or all of the variables are restricted to be integer. Also, when the model includes additional nonlinear relations, the domain-reduction is particularly useful since relaxation schemes for nonlinear expressions often utilize variable bounds as was done in Section 3.

4.3. Relation to earlier works

4.3.1. Optimality-Based range reduction. The following result is derived as a simple corollary of Algorithm SimpleReduce.

Theorem 4. *Suppose the Lagrangian subproblem in (4) is solved for a certain dual multiplier vector $y \leq 0$. Then, for each i such that $y_i \neq 0$, the cut $g_i(x) \geq \big(b_0 - \inf_x l(x, y)\big)/y_i$ does not chop off any optimal solution of T.*

Proof. A lower bound on $g_i(x)$ is obtained by solving the relaxed primal problem in Step 3 of Algorithm SimpleReduce with $(a_0, a) = -e_i$, where e_i is the i^{th} column of the identity matrix. It is easily seen that the optimal solution sets $u_j = b_j$ for $j \neq i$ and the optimal solution value is $\big(b_0 - \inf_x l(x, y)\big)/y_i$. □

We show next that simple applications of the above result produce the range-reduction schemes of [31, 43]. Assuming that appropriate constraint qualifications hold, the strong duality theorem of convex programming asserts that the optimal objective function value, L, of a convex relaxation with optimal dual multipliers, y^*, is less than or equal to the optimal value of the corresponding Lagrangian subproblem, $\inf_x l(x, y^*)$. The cut

$$g_i(x) \geq (U - L)/y_i^*$$

proposed in Theorem 2 of [31] is therefore either the same as that derived in Theorem 4 or weaker than it. Theorem 3 and Corollaries 1, 2, and 3 in [31] as well as the marginals-based reduction techniques in [41, 20] and the early integer programming literature [27] follow similarly. Also similarly, follows Corollary 4 in [31] which deals with *probing*, the process of temporarily fixing variables at their bounds and drawing inferences based on range reduction. More general probing procedures are developed in [38] as a consequence of the domain reduction strategy of Section 4.1.

In [43], a lower bound is derived on the optimal objective function of T by solving an auxiliary contraction problem. We now derive this result as a direct application of Theorem 4. Consider an auxiliary problem constrained by $f(x) - U \leq 0$ with an objective function $f_a(x)$, for which an upper bound f_a^u is known. Let $l_a(x, y)$ be the Lagrangian function for the auxiliary problem. Consider a dual solution, y_a, with a positive dual

multiplier, y_{af}, corresponding to $f(x) - U \leq 0$. By Theorem 4, the following cut does not chop off an optimal solution:

$$y_{af}\{f(x) - U\} \leq f_a^u - \inf_x l_a(x, y).$$

Using the auxiliary contraction problem below:

$$\begin{array}{ll} \min & x_i \\ \text{s.t.} & g(x) \leq 0 \\ & f(x) - U \leq 0 \\ & x \in X \subseteq \mathbb{R}^n, \end{array}$$

Theorem 3, Theorem 4, Corollary 1, and Corollary 2 of [43] follow immediately.

4.3.2. Feasibility-based range reduction. Consider the constraint set $\sum_{j=1}^n a_{ij}x_j \leq b_i$, $i = 1, \ldots, m$. Processing one constraint at a time, the following variable bounds are derived:

$$\begin{cases} x_h \leq \dfrac{1}{a_{ih}}\left(b_i - \displaystyle\sum_{j\neq h} \min\{a_{ij}x_j^u, a_{ij}x_j^l\}\right), & a_{ih} > 0 \\ x_h \geq \dfrac{1}{a_{ih}}\left(b_i - \displaystyle\sum_{j\neq h} \min\{a_{ij}x_j^u, a_{ij}x_j^l\}\right), & a_{ih} < 0 \end{cases} \tag{6}$$

where x_j^u and x_j^l are the tightest initially available upper and lower bounds for x_j. This procedure was referred to as "feasibility-based range reduction" in [35] and has been used extensively in mixed-integer linear programming (cf. [34]). We now derive this scheme as a consequence of the duality framework of Section 4.1. Note that this feasibility-based range reduction may be viewed as a rather simple application of Fourier-Motzkin elimination to the following problem–obtained by relaxing all but the i^{th} constraint:

$$\begin{array}{ll} \min & x_h \\ \text{s.t.} & a_i x \leq b_i \\ & x \leq x^u \\ & x \geq x^l. \end{array}$$

Assuming $a_{ih} < 0$ and x_h is at not at its lower bound, the optimal dual solution can easily be gleaned from the above linear program to be:

$$\begin{array}{l} y = 1/a_{ih} \\ r_j = -\max\{a_{ij}/a_{ih}, 0\} \text{ for all } j \neq h \\ s_j = \min\{a_{ij}/a_{ih}, 0\} \text{ for all } j \neq h \\ r_h = s_h = 0 \end{array}$$

where y, r_j, and s_j are the dual multipliers corresponding to $a_i x \leq b_i$, $x_j \leq x_j^u$, and $x_j \geq x_j^l$, respectively. Then, the following relaxed primal master problem is constructed:

$$\min\{x_h \mid -x_h + yu + rv - sw \leq 0, u \leq b, v \leq x^u, w \geq x^l\}. \tag{7}$$

Consequently, the bounds presented in (6) follow easily from (7).

4.4. New range reduction procedures

4.4.1. Duality-based reduction. When a relaxation is solved by dual ascent procedures such as dual Simplex or Lagrangian relaxations, dual feasible solutions are generated at every iteration of the algorithm. These solutions can be maintained in a set, S, of solutions feasible to the following system:

$$\begin{aligned} rA + sc &= 0 \\ r, s &\leq 0 \end{aligned}$$

As described in Section 4.2, the feasible solutions to PD as well as PR belong to the polyhedral cone defined by the above inequalities. Whenever a variable bound is updated as a result of partitioning or otherwise, or a new upper bounding solution is found for the problem, new bounds on all problem variables are obtained by evaluating the objective function of PR–with the appropriate perturbation–using the dual solutions stored in S.

In fact, if the matrix A also depends on the bounds of the variables as is the case when nonlinear programs are relaxed to linear outer-approximations (see Section 3), then a slightly cleverer scheme can be employed by treating the bounds as "easy" constraints and including them in the Lagrangian subproblem. Consider, for example, a linear relaxation which takes the following form:

$$\begin{aligned} \min \quad & c(x^L, x^U)x \\ \text{s.t.} \quad & l(x^L, x^U) \leq A(x^L, x^U)x \leq u(x^L, x^U) \\ & x^L \leq x \leq x^U \end{aligned} \tag{8}$$

Assume that (8) contains the objective function cut $c(x^L, x^U)x \leq U$. Construct a matrix Π, each row of which is a vector of dual multipliers of (8) (possibly from the set S described above). Feasibility-based tightening is done on the surrogate constraints generated by pre-multiplying the linear constraint set with this Π matrix. Clearly, there exists a vector π of dual multipliers that proves the best bound which can be derived from the above relaxation. The feasibility-based range reduction is a special case of duality-based tightening when Π is chosen to be the identity matrix. The marginals-based range reduction is a special case of the proposed duality-based range-reduction where the optimal value dual solution is augmented with -1 as the multiplier for the objective function cut.

4.4.2. Learning reduction heuristic. In order to guarantee exhaustiveness, most rectangular partitioning schemes resort to periodic bisection of the feasible space along each variable axis. Let us assume that, after branching, one of the child nodes turns out to be inferior. Then, it is highly probable that a larger region could have been fathomed if a proper branching point was chosen initially. In cases of this sort, a *learning reduction procedure* can introduce an error-correcting capability by expanding the region that is provably inferior. In particular, consider a node N where the following relaxation is generated:

$$\begin{array}{ll} \min\limits_{x} & \big(f_N(x^L, x^U)\big)(x) \\ \text{s.t.} & \big(g_N(x^L, x^U)\big)(x) \leq 0 \\ & x^L \leq x \leq x^U \end{array}$$

where $\big(f_N(x^L, x^U)\big)(\cdot)$ and $\big(g_N(x^L, x^U)\big)(\cdot)$ are convex functions. We assume that the relaxation strategy is such that, when a variable is at its bound, the dependent functions are represented exactly. Consider now the partitioning of node N into N_1 and N_2 by branching along x_j at x_j^B. Define

$$x^a = [x_1^U, \ldots, x_{j-1}^U, x_j^B, x_{j+1}^U, \ldots, x_n^U]$$

and

$$x^b = [x_1^L, \ldots, x_{j-1}^L, x_j^B, x_{j+1}^L, \ldots, x_n^L].$$

N_1 and N_2 are constructed so that $x^L \leq x \leq x^a$ for all $x \in N_1$ and $x^b \leq x \leq x^U$ for all $x \in N_2$. Assume further that node N_1 is found to be inferior and y^* is the vector of optimal dual multipliers using the relaxation generated at node N_1. The constraint $x \leq x_j^B$ is assumed to be active in the solution of N_1.

If we can construct the optimal dual solution $y^*_{N_2^P}$ for the following program:

$$(\mathrm{P_{N_2}}) \qquad \begin{array}{ll} \min\limits_{x} & \big(f_{N_2}(x^b, x^U)\big)(x) \\ \text{s.t.} & \big(g_{N_2}(x^b, x^U)\big)(x) \leq 0 \\ & x^b \leq x \leq x^U \\ & x_j \geq x_j^B, \end{array}$$

then we can apply the techniques developed in earlier sections for domain reduction. Construction of $y^*_{N_2^P}$ is expensive since it involves lower bounding the nonconvex problem at N_2. However, when the relaxation technique has certain properties, $y^*_{N_2^P}$ can be computed efficiently. We assumed that the relaxations at nodes N_1 and N_2 are identical when $x_j = x_j^B$ and that $x \leq x_j^B$ is active. These assumptions imply that not only is the optimal solution of the relaxation at N_1 optimal to $\mathrm{P_{N_2}}$, but the geometry of the KKT conditions at optimality is identical apart from the activity of the constraint $x_j \leq x_j^B$. Therefore, it is possible to identify the set of active constraints for $\mathrm{P_{N_2}}$. Once this is done, $y^*_{N_2^P}$ can be computed by solving a set of linear equations. In fact, the only change required to $y^*_{N_1}$ is in the multiplier corresponding to $x_j \leq x_j^B$.

There is, however, a subtle point that must be realized while identifying the set of active constraints. A function $r(x)$ in the original nonconvex program may be relaxed to $\max\{r_1(x), \ldots, r_k(x)\}$, where each $r_i(\cdot)$ is separable and convex. In this case, the function attaining the supremum can change between the nodes N_1 and N_2. We illustrate this through the example of the epigraph of a bilinear function $z \leq x_j w$. We relax z using the bilinear envelopes as follows [25]:

$$z \leq \min\{x_j^L w + w^U x_j - x_j^L w^U, x_j^U w + w^L x_j - x_j^U w^L\}.$$

In particular, at node N_1:

$$z \leq \min\{x_j^L w + w^U x_j - x_j^L w^U, x_j^B w + w^L x_j - x_j^B w^L\}$$

and at node N_2:

$$z \leq \min\{x_j^B w + w^U x_j - x_j^B w^U, x_j^U w + w^L x_j - x_j^U w^L\}.$$

When $x = x_j^B$ in the relaxation at node N_1, the dominating constraint is $z \leq x_j^B w + w^L x_j - x_j^B w^L$ whereas, in $\mathrm{P}_{\mathrm{N}_2}$, the dominating constraint is $z \leq x_j^B w + w^U x_j - x_j^B w^U$. If μ is the dual multiplier for constraint $z \leq x_j^B w + w^L x_j - x_j^B w^L$ in $y^*_{N_1}$, then μ should be the dual multiplier of $z \leq x_j^B w + w^U x_j - x_j^B w^U$ in $y^*_{N_2^P}$. Adding $(w^U - w^L)(x_j^B - x_j) \geq 0$, yields the desired constraint:

$$z \leq x_j^B w + w^L x_j - x_j^B w^L.$$

Thus, it is possible that the multiplier corresponding to the upper bounding constraint $x_j \leq x_j^B$ increases when a dual solution is constructed for $\mathrm{P}_{\mathrm{N}_2}$ from a dual solution of N_1. Such an increase would not occur if the only difference between the relaxations at N_1 and N_2 is the bound on x_j, which is most often the case with relaxations of integer linear programs.

5. Node partitioning schemes

Rectangular partitioning—to which we restrict our attention—splits the feasible region, at every branch-and-bound iteration, into two parts by intersecting it with the half-spaces of a hyperplane orthogonal to one of the coordinate axes. In order to define this hyperplane uniquely, the coordinate axis perpendicular to the hyperplane and the intercept of the hyperplane on the axis must be specified. The choice of the coordinate axis and the intercept are referred to as the branching variable and branching point selection, respectively.

5.1. Branching variable selection

The selection of the branching variable determines, to a large extent, the structure of the tree explored and affects the performance of a branch-and-bound algorithm significantly. We determine the branching variable through "violation transfer." First, violations of nonconvexities by the current relaxation solution are assigned to problem variables. Then, the violations are transferred between variables utilizing the functional relationships in the constraints and objective. Finally, violations are weighted to account for branching priorities and the potential relaxation improvement after branching, and a variable leading to the maximum weighted violation is chosen for branching.

Algorithm Relax of Section 3 reformulates the general problem such that multidimensional functions are replaced with either univariate functions or bilinear functions. We provide branching rules for both cases in this section.

Consider a variable y that depends on x_j through the relation $y = f(x_j)$. Let (x_j^*, y^*) denote the value attained by (x_j, y) in the relaxation solution. Define $X_j^y = f^{-1}(y^*)$. The violation of x_j is informally defined to be the length of the interval obtained as the intersection of $[x_j^L, x_j^U]$ and the smallest interval around x_j^* which contains, for every variable y dependent on x_j^*, an $x_j^y \in X_j^y$.

The violations on all variables are computed in a two-phase process, the forward-transfer phase and the backward-transfer phase. In the forward transfer, the violations on the dependent variables are computed using the relaxation solution values for the independent variables using all the introduced relations. In the backward-transfer phase, these violations are transmitted back to the independent variables. For each dependent variable y we derive an interval $[y^{lv}, y^{uv}]$. Similarly, we derive an interval $[x_j^{lv}, x_j^{uv}]$ for x_j.

Algorithm Violation Transfer

Step 0. Set $y^{lv} = y^{uv} = y^*$. If x_j is not required to be integer-valued, set $x_j^{lv} = x^{uv} = x_j^*$. Otherwise, set $x_j^{lv} = \lfloor x_j^* \rfloor$ and $x_j^{uv} = \lceil x_j^* \rceil$.

Step 1. Forward transfer:

$$y^{lv} = \max\left\{y^L, \min\{y^{lv}, f(x_j^{lv}), f(x_j^{uv})\}\right\}$$
$$y^{uv} = \min\left\{y^U, \max\{f(x_j^{lv}), f(x_j^{uv}), y^{uv}\}\right\}$$

Step 2. Backward transfer:

$$x_j^{lv} = \max\left\{x_j^L, \min\{x_j^{lv}, f^{-1}(y^{lv}), f^{-1}(y^{uv})\}\right\}$$
$$x_j^{uv} = \min\left\{x_j^U, \max\{f^{-1}(y^{lv}), f^{-1}(y^{uv}), x_j^{uv}\}\right\}$$

Step 3. $v_y = y^{uv} - y^{lv}$, $v_{x_j} = x_j^{uv} - x_j^{lv}$.

For bilinear functions, the violation transfer scheme is more involved. We refer the reader to [38] for details.

Define

$$x^l = [x_1^*, \ldots, x_{j-1}^*, x_j^{lv}, x_{j+1}^*, \ldots, x_n^*]$$

and

$$x^u = [x_1^*, \ldots, x_{j-1}^*, x_j^{uv}, x_{j+1}^*, \ldots, x_n^*].$$

Consider the Lagrangian function $l(x, \pi) = f(x) - \pi g(x)$. Let f^l and f^u, g^l and g^u be the estimated minimum and maximum values of f and g, respectively, over $[x^l, x^u]$. Also, let π^* denote the set of marginals obtained at the relaxation of the current node. Then, the weighted violation of x_j is computed as

$$\beta_{x_j}\left\{f^u - f^l + \sum_{i=1}^{m} \pi_i^*\left(\max\{0, g^l\} - g^u\right) + \alpha_{x_j} v_{x_j}\right\},$$

where α_{x_j} and β_{x_j} are appropriately chosen constants (branching priorities). In the case of linear relaxations of the form $\min\{cx \mid Ax \leq b\}$, the above may be approximated as:

$$\beta_{x_j} \left\{ |c_j| v_{x_j} + \sum_{i=1}^{m} |\pi_i^* A_{ij}| v_{x_j} + \alpha_{x_j} v_{x_j} \right\}.$$

The branching variable is chosen to be the one with the largest weighted violation.

5.2. Branching point selection

Following the choice of the branching variable, the branching point is typically chosen by an application of the bisection rule, ω-rule, or their modified versions [31, 35, 38]. In Section 3.1, we showed that, for any univariate convex function, the gap between the over- and under-estimators reduces as $O(1/n^2)$, provided that the branching point is chosen via the interval bisection rule, slope bisection rule, projective rule, maximum error rule, chord rule, or the angular bisection rule. The branch-and-bound algorithm operates in a manner similar to the sandwich algorithm as it identifies and subsequently refines regions where the function is desired to be approximated more closely. However, in branch-and-bound, those regions are chosen where the lower bounding function attains the minimum value. In contrast, in the sandwich algorithm, the error measure between the function and its approximating functions is used for the selection of the interval for further refinement. Still, in the course of our branch-and-bound, the approximation error of a function reduces by $O(1/n^2)$. Here, n is the number of times partitioning was done along the variable axis corresponding to the independent variable in all ancestors of the current node.

As an alternative strategy, a bilinear program for branching point selection is given in [38]. For a given branching variable, this formulation identifies a branching point that locally maximizes the improvement in the lower bound for the current node.

6. Tree navigation and expansion

At every iteration, branch-and-bound chooses a node from the list of open nodes for processing. The choice of this node governs the structure of the branch-and-bound tree explored before convergence is attained. This choice also determines to a large extent the memory requirements of the algorithm.

The node selection rule we employ is a composite rule based on a priority ordering of the nodes. Priority orderings based on lower bound, violation, and order of creation are dynamically maintained during the search process. The violation of a node is defined as the summation of the errors contributed by each variable of a branch-and-bound node (see Section 5.1). The default node selection rule switches between the node with the minimum violation and the one with least lower bound. When memory limitations become stringent, we temporarily switch to the LIFO rule.

While solving large problems using branch-and-bound methods, it is not uncommon to generate a large number of nodes in the search tree. Let us say that the number of

open nodes in the branch-and-bound tree at a given point in time is L. The naive method of traversing the list of nodes to select the node satisfying the priority rule takes $\Theta(L)$ time. Unlike most other procedures conducted at every node of the branch-and-bound tree, the node selection process consumes time that is super-polynomial in the size of the input. It therefore pays to use more sophisticated data structures for maintaining priority queues. In particular, we maintain the priority orderings using the heap data structure for which insertion and deletion of a node may be done in $O(\log L)$ time and retrieval of the node satisfying the priority rule takes only $O(1)$ time. In particular, for combinatorial optimization problems, the node selection rule is guaranteed to spend time linear in the size of the input.

Once the current node has been solved and a branching decision has been taken, the branching decision is stored and the node is partitioned only when it is chosen subsequently by the node selection rule. This approach is motivated by two reasons. First, it results in half the memory requirements as compared to the case in which the node is partitioned immediately after branching. Secondly, postponement offers the possibility of partial optimization of a node's relaxation. For example, consider a relaxation that is solved using an outer-approximation method based on Lagrangian duality, Benders decomposition, or a dual ascent procedure such as bundle methods. In such cases, it may not be beneficial to solve the relaxation to optimality if, after a certain number of iterations, a lower bound is proven for this node which significantly reduces the priority of this node. When this node is picked again for further exploration, the relaxation may be solved to optimality. A similar scheme is followed if range reduction leads to significant tightening of the variable bounds.

7. Implementation and computational results

The Branch-and-Reduce Optimization Navigator (BARON) [32] is an implementation of our branch-and-bound algorithm for global optimization problems. BARON is modular with problem-specific enhancements (modules) that are isolated from the core implementation of the branch-and-bound algorithm.

BARON's core consists of nine major components: preprocessor, navigator, data organizer, I/O handler, range reduction utilities, sparse matrix utilities, solver links, automatic gradient and function evaluator, and debugging facilities:

- The navigator—the principal component of BARON—coordinates the transitions between the preidentified computation states of a branch-and-bound algorithm, including node preprocessing, lower bounding, range reduction, upper bounding, node postprocessing, branching, and node selection. The execution of the navigator can be fine-tuned using a wide variety of options available to the user. For example, the user may choose between various node selection schemes like LIFO, best bound, node violation, and other composite rules based on these priority orders. The navigator calls the module-specific range reduction, lower bounding, upper bounding, feasibility checker, and objective function evaluator during the appropriate phases of the algorithm.
- The data organizer is tightly integrated with the navigator and maintains the branch-and-bound tree structure, priority queues, and bases information to hot-start the

solver at every node of the search tree. These data structures are maintained in a linear work array. Therefore, the data organizer provides its own memory management facilities (allocator and garbage collector) which are suited to the branch-and-bound algorithm. Data compression techniques are used for storing bounded integer arrays like the bounds of integer variables and the LP basis. Each module is allowed to store its problem-specific information in the work-array before the computation starts at every node of the branch-and-bound tree.

- The I/O handler provides the input facilities for reading in the problem and the options that affect the execution of the algorithm. A context-free grammar has been developed for the input of factorable nonlinear programming problems where the constraint and objective functions are recursive sums and products of univariate functions of monomials, powers, and logarithms. In addition, each module supports its own input format through its input routine. Output routines provide the timing details and a summary of the execution in addition to the solutions encountered during the search process. The output routines may also be instructed to dump detailed information from every step of the branch-and-bound algorithm for debugging purposes.
- The range reduction facilities include tightening based on feasibility, marginals, probing, and duality schemes as well as interval arithmetic utilities for logarithmic, monomial, power, and bilinear functions of variables.
- BARON has interfaces to various solvers like OSL, CPLEX, MINOS, SNOPT, and SDPA. BARON also provides automatic differentiation routines to interface to the nonlinear solvers. The interfaces save the requisite hot-starting information from the solver and use it at subsequent nodes in the tree. Depending on the degree of change in the problem fed to the solver, an appropriate update scheme is automatically chosen by each interface. The interfaces also provide the I/O handler with the solver-specific information that may be desired by the user for debugging purposes.

In addition to the core component, the BARON library includes solvers for mixed-integer linear programming, separable concave quadratic programming, indefinite quadratic programming, separable concave programming, linear multiplicative programming, general linear multiplicative programming, univariate polynomial programming, $0-1$ hyperbolic programming, integer fractional programming, fixed-charge programming, power economies of scale, mixed-integer semidefinite programming, and factorable nonlinear programming. In particular, what we refer to as the factorable NLP module of BARON is the most general of the modules and addresses the solution of factorable NLPs with or without additional integrality requirements. Without any user intervention, BARON constructs linear/nonlinear relaxations and solves relaxations while at the same time performing local optimization using widely available solvers.

Extensive computational experience with the proposed algorithm on over 500 problems is reported in [38]. Below, we present computational results on three representative classes of problems: a purely continuous nonlinear class, a purely integer nonlinear class, and miscellaneous mixed-integer nonlinear programs. The machine on which we perform all of our computations is a 332 MHz RS/6000 Model 43P running AIX 4.3 with 128MB RAM and a LINPACK score of 59.9. For all problems solved, we report the total CPU seconds taken to solve the problem (T_{tot}), the total number of nodes in the

branch-and-bound tree (N_{tot}), and the maximum number of nodes that had to be stored in memory during the search (N_{mem}). Unless otherwise noted, computations are carried out with an absolute termination tolerance (difference between upper and lower bounds) of $\epsilon_a = 10^{-6}$. BARON converged with a global optimal solution within this tolerance for all problems reported in this paper.

7.1. Separable concave quadratic programs (SCQPs)

In Table 1, we present computational results for the most difficult of the separable concave quadratic problems in [35]. In this table, m and n, respectively, denote the number of constraints and variables of the problems solved. We use the BARON SCQP module with and without probing and the NLP module without probing to solve these problems. As seen in this table, our general purpose factorable NLP module takes approximately the same number of branch-and-bound iterations to solve the problems as the specialized SCQP module.

The problems tackled in Table 1 are relatively small. We next provide computational results on larger problems that are generated using the test problem generator of [28]. The problems generated are of the form:

$$\begin{aligned} \min \quad & \frac{1}{2}\theta_1 \sum_{j=1}^{n} \lambda_j (x_j - \bar{\omega}_j)^2 + \theta_2 \sum_{j=1}^{k} d_j y_j \\ \text{s.t.} \quad & A_1 x + A_2 y \leq b \\ & x \geq 0,\ y \geq 0 \end{aligned}$$

where $x \in \mathbb{R}^n$,$\lambda \in \mathbb{R}^n$, $\bar{\omega} \in \mathbb{R}^n$, $y \in \mathbb{R}^k$, $d \in \mathbb{R}^k$, $b \in \mathbb{R}^m$, $A_1 \in \mathbb{R}^{m\times n}$, $A_2 \in R^{m\times k}$, $\theta_1 = -0.001$, and $\theta_2 = 0.1$. The results of Table 2 were obtained by solving five randomly generated instances for every problem size. The algorithms compared used a relative termination tolerance of $\epsilon_r = 0.1$. Our algorithm is up to an order of magnitude faster than that of [42] for large problems.

Table 1. Computational results for small SCQPs.

			BARON SCQP no probing			BARON SCQP probing			BARON NLP no probing		
Problem	m	n	T_{tot}	N_{tot}	N_{mem}	T_{tot}	N_{tot}	N_{mem}	T_{tot}	N_{tot}	N_{mem}
FP7a	10	20	0.17	77	5	0.28	71	4	1.38	51	6
FP7b	10	20	0.19	93	7	0.25	59	4	1.38	51	6
FP7c	10	20	0.19	81	7	0.28	71	4	1.29	45	6
FP7d	10	20	0.19	81	4	0.25	65	4	1.26	49	6
FP7e	10	20	0.38	197	16	0.56	141	12	8.87	339	30
RV1	5	10	0.04	35	3	0.04	11	3	0.29	29	3
RV2	10	20	0.16	103	7	0.20	37	5	1.37	65	8
RV3	20	20	0.40	233	15	0.28	73	10	3.22	141	11
RV7	20	30	0.32	155	6	0.23	41	4	3.63	109	7
RV8	20	40	0.47	187	13	0.45	58	8	3.1	81	8
RV9	20	50	1.25	526	38	1.19	184	31	15.9	331	34
M1	11	20	0.36	189	3	0.61	151	3	2.11	99	7
M2	21	30	0.93	291	3	1.67	233	2	5.57	159	2

Table 2. Comparative computational results for large SCQPs.

Algorithm Computer			GOP96 [42] HP9000/730	GOP/MILP [42] HP9000/730				BARON SCQP RS6000/43P					
			T_{tot}	T_{tot}	T_{tot}			N_{tot}			N_{mem}		
m	n	k	avg	avg	min	avg	max	min	avg	max	min	avg	max
50	50	50	0.12	0.12	0.10	0.10	0.10	1	2	3	1	1	2
50	50	100	0.15	0.14	0.10	0.22	0.30	1	13	24	1	5	9
50	50	200	6.05	1.57	0.30	0.70	1.60	9	51	170	5	17	45
50	50	500	—	14.13	1.00	1.86	2.80	21	68	108	9	16	30
50	100	100	0.22	1.37	0.30	0.72	1.10	20	62	127	6	23	48
50	100	200	0.36	11.98	0.50	1.48	3.30	17	98	246	4	20	39
100	100	100	0.31	0.31	0.40	0.46	0.50	3	7	14	2	4	8
100	100	200	0.38	0.36	0.70	0.78	0.90	14	26	37	5	8	11
100	100	500	—	80.03	1.61	4.40	15.2	43.7	740	2314	44	161	425
100	150	400	—	180.2	1.20	18.8	82.6	8	927	4311	3	163	721

7.2. Cardinality constrained hyperbolic programming

Cardinality constrained hyperbolic programs are of the following form:

$$
\begin{aligned}
\text{(CCH)} \qquad \max \quad & \sum_{i=1}^{m} \frac{a_{i0} + \sum_{j=1}^{n} a_{ij} x_j}{b_{i0} + \sum_{j=1}^{n} b_{ij} x_j} \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_j = p \\
& \sum_{j=1}^{n} b_{i0} + b_{ij} x_j > 0 \qquad i = 1, \ldots, m \\
& x_j \in \{0, 1\} \qquad\qquad\quad j = 1, \ldots, n,
\end{aligned}
$$

where $a \in \mathbb{R}^{n+1}$ and $b \in \mathbb{R}^{n+1}$ for $i = 1, \ldots, m$, $p \in \mathbb{Z}$, $0 \leq p \leq n$.

As we detail in [39], we can convert the pure nonlinear integer program CCH to an MILP using the reformulation strategy shown below where $v_{ij} = g_i x_j$ is enforced by using the bilinear envelopes of [25]:

$$
\begin{aligned}
\text{(R7)} \max \quad & \sum_{i=1}^{m} g_i \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_j = p \\
& \sum_{j=1}^{n} v_{ij} = g_i p \qquad\qquad i = 1, \ldots, m \\
& b_{i0} g_i + \sum_{j=1}^{n} b_{ij} v_{ij} = a_{i0} + \sum_{j=1}^{n} a_{ij} x_j \quad i = 1, \ldots, m \\
& v_{ij} \leq g_i^u x_j \qquad\qquad i = 1, \ldots, m;\ j = 1, \ldots, n;\ b_{ij} < 0 \\
& v_{ij} \leq g_i + g_i^l x_j - g_i^l \quad i = 1, \ldots, m;\ j = 1, \ldots, n;\ b_{ij} < 0 \\
& v_{ij} \geq g_i + g_i^u x_j - g_i^u \quad i = 1, \ldots, m;\ j = 1, \ldots, n;\ b_{ij} > 0 \\
& v_{ij} \geq g_i^l x_j \qquad\qquad i = 1, \ldots, m;\ j = 1, \ldots, n;\ b_{ij} > 0 \\
& x \in \{0, 1\}^n
\end{aligned}
$$

We derive tight bounds on g_i using an algorithm devised by Saipe in [33]. Then, we use BARON to carry out the above reformulation at every node of the tree to solve CCH and compare the performance of our algorithm against that of CPLEX on the MILP. As Table 3 shows, range contraction and reformulation at every node results in a much superior approach.

7.3. Mixed-integer nonlinear programs from `minlplib`

In this section, we address the solution of mixed-integer nonlinear programs. In particular, we focus our attention on a collection of MINLPs from `minlplib` [8], which provides complete model descriptions and references to original sources for all problems solved in this section.

The computational results are presented in Table 4 where, for each problem, we report the globally optimal objective function value, the number of problem constraints (m), the total number of problem variables (n), and the number of discrete variables (n_{d}), in addition to CPU and node information. Problems nvs01 through nvs24 originate from [16] who used local solutions to nonconvex nonlinear programs at each node and conducted branch-and-bound on the integer variables. Even though the problem sizes are small, the solutions we obtained for problems nvs02, nvs14, nvs20, and nvs21 correspond to significantly better objective function values than those reported in the original source of these problems [16], demonstrating the benefits of global optimization.

Amongst the remaining problems of this table, one finds a nonlinear fixed-charge problem (st_e27), a reliability problem (st_e29), a mechanical fixture design problem (st_e31), a heat exchanger network synthesis problem (st_e35), a pressure design problem (st_e38), a truss design problem (st_e40), a problem from the design of just-in-time manufacturing systems (jit), and a particulary challenging molecular design problem (primary).

8. Conclusions

The subject matter of this paper has been the development and implementation of a global optimization algorithm for mixed-integer nonlinear programming. Our algorithmic developments included a new outer-approximation scheme that provides an entirely

Table 3. Computational results for CCH.

Problem			CPLEX 6.0			BARON HYP		
m	n	p	T_{tot}	N_{tot}	N_{mem}	T_{tot}	N_{tot}	N_{mem}
5	20	12	1.07	40	39	1.9	11	4
10	20	8	105	1521	1518	70.5	234	54
10	20	12	7.4	96	95	11.2	31	7
20	30	14	22763	47991	47076	1811	1135	216
20	20	12	207	666	632	83	85	18
20	30	16	17047	38832	36904	1912	1353	256
5	50	25	19372	506225	184451	2534	6959	1230
5	50	27	—	—	—	1803	5079	893

"—" in this table indicates a problem for which CPLEX did not converge after a day.

Table 4. Selected MINLP problems from [8].

Problem		Obj.	m	n	n_d	T_{tot}	N_{tot}	N_{mem}
nvs01		12.47	3	3	2	0.05	29	5
nvs02	*	5.96	3	8	5	0.04	13	6
nvs03		16.00	3	2	2	0.01	3	2
nvs04		0.72	0	2	2	0.02	1	1
nvs05		5.47	9	8	2	1.91	241	52
nvs06		1.77	0	2	2	0.03	11	5
nvs07		4.00	2	3	3	0.02	3	2
nvs08		23.45	3	3	2	0.39	21	5
nvs09		−43.13	0	10	10	0.16	39	7
nvs10		−310.80	2	2	2	0.04	12	4
nvs11		−431.00	3	3	3	0.07	34	8
nvs12		−481.20	4	4	4	0.11	70	13
nvs13		−585.20	5	5	5	0.40	208	30
nvs14	*	−40358.20	3	8	5	0.03	8	5
nvs15		1.00	1	3	3	0.02	11	3
nvs16		0.70	0	2	2	0.03	23	12
nvs17		−1100.40	7	7	7	12.00	3497	388
nvs18		−778.40	6	6	6	2.75	1037	122
nvs19		−1098.40	8	8	8	43.45	9440	1022
nvs20	*	230.92	2	3	2	1.64	156	19
nvs21	*	−5.68	2	3	2	0.05	31	3
nvs22		6.06	9	8	4	0.08	14	6
nvs23		−1125.20	9	9	9	168.23	25926	3296
nvs24		−1033.20	10	10	10	1095.87	130487	15967
st_e27		2.00	2	1	1	0.02	3	2
st_e29		−0.94	6	2	2	0.18	47	11
st_e31		−2.00	135	112	24	3.75	351	56
st_e32		−1.43	18	35	19	13.7	906	146
st_e35		64868.10	39	32	7	16.4	465	57
st_e36		−246.00	2	2	1	2.59	768	72
st_e38		7197.73	3	4	2	0.38	5	2
st_e40		30.41	7	4	3	0.15	24	6
jit		173,983	33	26	4	0.67	63	10
primary		−1.2880	164	82	58	375	15930	1054

*: Solutions found for these problems are better than those earlier reported by [16].

LP-based relaxation of MINLPs, a new theoretical framework for domain reduction, and new branching strategies. Computational results with our branch-and-bound implementation in BARON demonstrated the flexibility that such a computational model offers in solving MINLPs from a variety of disciplines through an entirely automated procedure.

Acknowledgements. The authors are grateful to two anonymous referees whose constructive comments helped improve the presentation of this manuscript significantly.

References

1. Ahmed, S., Tawarmalani, M., Sahinidis, N.V.: A finite branch and bound algorithm for two-stage stochastic integer programs. Mathematical Programming. Submitted, 2000
2. Al-Khayyal, F.A., Falk, J.E.: Jointly constrained biconvex programming. Math. Oper. Res. **8**, 273–286 (1983)

3. Al-Khayyal, F.A., Sherali, H.D.: On finitely terminating branch-and-bound algorithms for some global optimization problems. SIAM J. Optim. **10**, 1049–1057 (2000)
4. Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: Nonlinear Programming, Theory and Algorithms. Wiley Interscience, Series in Discrete Math. Optim. 2nd edition, 1993
5. Bienstock, D.: Computational study of a family of mixed-integer quadratic programming problems. Math. Program. **74**, 121–140 (1996)
6. Borchers, B., Mitchell, J.E.: An improved branch and bound for mixed integer nonlinear programs. Comput. Oper. Res. **21**, 359–367 (1994)
7. Borchers, B., Mitchell, J.E.: A computational comparison of branch and bound and outer approximation algorithms for $0-1$ mixed integer nonlinear programs. Comput. Oper. Res. **24**, 699–701 (1997)
8. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib–A Collection of Test Models for Mixed-Integer Nonlinear Programming. INFORMS J. Comput. **15**, 114–119 (2003)
9. Dakin, R.J.: A tree search algorithm for mixed integer programming problems. Comput. J. **8**, 250–255 (1965)
10. Duran, M.A., Grossmann, I.E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Math. Prog. **36**, 307–339 (1986)
11. Falk, J.E., Soland, R.M.: An algorithm for separable nonconvex programming problems. Manag. Sci. **15**, 550–569 (1969)
12. Floudas, C.A.: Deterministic Global Optimization: Theory, Algorithms and Applications. Kluwer Academic Publishers, Dordrecht, 1999
13. Fruhwirth, B., Burkard, R.E., Rote, G.: Approximation of convex curves with applications to the bicriteria minimum cost flow problem. European J. Oper. Res. **42**, 326–338 (1989)
14. Gruber, P.M.: Aspects of approximation of convex bodies. In: Gruber, P. M. Gruber, Wills, J. M., (eds.), Handbook of Convex Geometry. North-Holland, 1993
15. Gruber, P.M., Kenderov, P.: Approximation of convex bodies by polytopes. Rendiconti Circ. Mat. Palermo, Serie II. **31**, 195–225 (1982)
16. Gupta, O.K., Ravindran, A.: Branch and bound experiments in convex nonlinear integer programming. Manag. Sci. **31**, 1533–1546 (1985)
17. Hamed, A.S.E., McCormick, G.P.: Calculation of bounds on variables satisfying nonlinear inequality constraints. J. Global Opt. **3**, 25–47 (1993)
18. Hansen, P., Jaumard, B., Lu, S.-H.: An analytic approach to global optimization. Math. Prog. **52**, 227–254 (1991)
19. Horst, R., Tuy, H.: Global Optimization: Deterministic Approaches. Springer Verlag, Berlin, Third edition, 1996
20. Lamar, B.W.: An improved branch and bound algorithm for minimum concave cost network flow problems. J. Global Opt. **3**, 261–287 (1993)
21. Land, A.H., Doig, A.G.: An automatic method for solving discrete programming problems. Econometrica **28**, 497–520 (1960)
22. Lazimy, R.: Mixed-integer quadratic programming. Math. Prog. **22**, 332–349 (1982)
23. Lazimy, R.: Improved algorithm for mixed-integer quadratic programs and a computational study. Math. Prog. **32**, 100–113 (1985)
24. McBride, R.D., Yormark, J.S.: An implicit enumeration algorithm for quadratic integer programming. Manag. Sci. **26**, 282–296 (1980)
25. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I – Convex underestimating problems. Math. Prog. **10**, 147–175 (1976)
26. McCormick, G.P.: Nonlinear Programming: Theory, Algorithms and Applications. John Wiley & Sons, 1983
27. Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley Interscience, Series in Discrete Math. Opt., 1988
28. Phillips, A.T., Rosen, J.B.: A parallel algorithm for constrained concave quadratic global minimization. Math. Prog. **42**, 421–448 (1988)
29. Rote, G.: The convergence rate of the sandwich algorithm for approximating convex functions. Comput. **48**, 337–361 (1992)
30. Ryoo, H.S., Sahinidis, N.V.: Global optimization of nonconvex NLPs and MINLPs with applications in process design. Computers & Chemical Engineering **19**, 551–566 (1995)
31. Ryoo, H.S., Sahinidis, N.V.: A branch-and-reduce approach to global optimization. J. Global Opt. **8**, 107–139 (1996)
32. Sahinidis, N.V.: BARON: A general purpose global optimization software package. J. Global Opt. **8**, 201–205 (1996)
33. Saipe, A.L.: Solving a (0, 1) hyperbolic program by branch and bound. Naval Res. Logistics Quarterly **22**, 497–515 (1975)

34. Savelsbergh, M.W.P.: Preprocessing and probing for mixed integer programming problems. ORSA J. Comput. **6**, 445–454 (1994)
35. Shectman, J.P., Sahinidis, N.V.: A finite algorithm for global minimization of separable concave programs. J. Global Opt. **12**, 1–36 (1998)
36. Sherali, H.D., Wang, H.: Global optimization of nonconvex factorable programming problems. Math. Prog. **89**, 459–478 (2001)
37. Smith, E.M.B., Pantelides, C.C.: Global optimisation of general process models. In: Grossmann, I.E., (ed.), Global Optimization in Engineering Design. Kluwer Academic Publishers, Boston, MA, 1996, pp. 355–386
38. Tawarmalani, M.: Mixed Integer Nonlinear Programs: Theory, Algorithms and Applications. PhD thesis, Department of Mechanical & Industrial Engineering. University of Illinois, Urbana-Champaign, IL, August 2001
39. Tawarmalani, M., Ahmed, S., Sahinidis, N.V.: Global optimization of $0-1$ hyperbolic programs. J. Global Opt. **24**, 385–417 (2002)
40. Tawarmalani, M., Sahinidis, N.V.: Convex extensions and convex envelopes of l.s.c. functions. Math. Prog. **93**, 247–263 (2002)
41. Thakur, L.S.: Domain contraction in nonlinear programming: Minimizing a quadratic concave function over a polyhedron. Math. Oper. Res. **16**, 390–407 (1990)
42. Visweswaran, V., Floudas, C.A.: Computational results for an efficient implementation of the GOP algorithm and its variants. In: Grossmann, I.E., (ed.), Global Optimization in Engineering Design. Kluwer Academic Publishers, Boston, MA, 1996, pp. 111–153
43. Zamora, J.M., Grossmann, I.E.: A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. J. Global Opt. **14**, 217–249 (1999)