To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–26

EAGO.jl: Easy Advanced Global Optimization in Julia

M. E. Wilhelm and M. D. Stuber*

Process Systems and Operations Research Laboratory, Dept. of Chemical and Biomolecular Engineering, University of Connecticut, 191 Auditorium Rd, Unit 3222, Storrs, CT, 06269-3222, USA

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

An extensible open-source deterministic global optimizer (EAGO) programmed entirely in the Julia language is presented. EAGO was developed to serve the need for supporting higher-complexity user-defined functions (e.g., functions defined implicitly via algorithms) within optimization models. EAGO embeds a first-of-its-kind implementation of McCormick arithmetic in an Evaluator structure allowing for the construction of convex/concave relaxations using a combination of source code transformation, multiple dispatch, and context-specific approaches. Utilities are included to parse user-defined functions into a directed acyclic graph representation and perform symbolic transformations enabling dramatically improved solution speed. EAGO is compatible with a wide variety of local optimizers, the most exhaustive library of transcendental functions, and allows for easy accessibility through the JuMP modeling language. Together with Julia's minimalist syntax and competitive speed, these powerful features make EAGO a versatile research platform enabling easy construction of novel meta-solvers, incorporation and utilization of new relaxations, and extension to advanced problem formulations encountered in engineering and operations research (e.g., multilevel problems, user-defined functions). The applicability and flexibility of this novel software is demonstrated on a diverse set of examples. Lastly, EAGO is demonstrated to perform comparably to state-of-the-art commercial optimizers on a benchmarking test set.

Keywords: Deterministic Global Optimization; Nonconvex Programming; McCormick Relaxations; Optimization Software; Branch-and-Bound; Julia

AMS Subject Classification: 90C26; 90C34; 90C57; 90C90

1. Introduction and Motivation

Mathematical optimization problems are ubiquitous in scientific and technical fields. Applications range from aerospace and chemical process systems to finance. However, even relatively simple physical processes such as mixing, may introduce significant nonconvexity into problem formulations [60]. As such, nonconvex programs often represent the most faithful representations of the system of interest. Multiple approaches have been developed to address these cases. Heuristics such as evolutionary algorithms, may approximate good solutions for select problems. However, heuristics may fail to guarantee that even a feasible solution is detected in finite time [2]. In scenarios where a guarantee is necessary, such as determining a reactor's maximum safe operating temperature, complete nonconvex methods must be applied. As nonconvex optimization problems are NP-hard, the development of appropriate optimizers remains an active area of research [29].

*Corresponding author. Email: stuber@alum.mit.edu

Author's final accepted version. Published version: Wilhelm, M.E. and Stuber, M.D. EAGO.jl: Easy advanced global optimization. Optimization Methods & Software. (2020) DOI: 10.1080/10556788.2020.1786566

State-of-the-art complete global optimizers generally require that a problem can be constructed in an algebraic modeling language (AML). These languages provide a specialized interface that allows users to construct optimization problems in a manner that can be interpreted by optimizers; translating high-level syntax into the requisite C/Fortran code and reducing chances for user-input errors. In addition to serving this central function, AMLs have evolved to provide additional features such as embedding automatic differentiation (AD) schemes and passing standard expression forms to optimizers [23]. This last feature is required by nearly every complete nonconvex optimizer available today. These regular expressions are then parsed by the optimizer via the introduction of auxiliary variables until all expressions in the model correspond to a form in the optimizers reference library of relaxations. In many cases, this reference library is limited to a few expressions: BARON [55] does not support trigonometric functions, ANTIGONE [37] and Couenne [7] limit expressions to those allowed by AMPL [23] or GAMS [24] environments. In addition, these complete optimizers do not support user-defined functions.

Many real-world optimization problems arise naturally from computer simulations and take non-canonical forms. Problems defined in terms of ordinary differential equation (ODE) and differential-algebraic equation (DAE) systems are pervasive in process systems engineering. Moreover, for many early-stage design problems, optimization of a model with an embedded simulation represents only one of many valuable tasks. Others may center around validating simulations, illustrating system dynamics, and assessing sensitivities [9]. It is often necessary to recast these problems into an AML prior to optimization. The manual reformulation of a model may be quite challenging for both subject matter experts and non-experts alike. While application-specific simulation packages attempt to bridge this gap, only gPROMS [4, 5, 49] provides access to state-of-the-art global optimizers. As a propriety software offering, gPROMS is not readily extensible to user-defined scripts, and suffers from the same lack of adaptability as other AMLs. Therefore, users that require non-canonical formulations must resort back to using AMLs or their own painstakingly implemented code. A few AMLs have mitigated this issue by offering simpler and more intuitive interfaces. Further enhancements to usability have been provided by subject area-specific extensions in AMLs, such as Pyomo [28] and JuMP [21].

As an alternative to AML extensions, set-valued arithmetic may be used to construct bounds. In this approach, convex and concave relaxations of non-canonical functions are constructed using composition rules by overloading methods or objects. In each of these approaches, a set-valued data type is constructed and arithmetic rules such as $+$, $\div$, $\sin(\cdot)$, etc., are defined for this data type. Interval arithmetic [40, 46] was one of the first such approaches, but subsequent work has extended this to affine arithmetic [18], and most recently McCormick arithmetic [36, 39, 56]. While interval methods are quite general and can be readily used in global optimization, McCormick arithmetic offers quadratic convergence rates for a wide class of functions with tighter bounds [12, 42]. In addition, McCormick operator theory has been developed to allow for the relaxation of implicit functions, such as those arising in parametric ODE- and PDE-constrained systems and nonlinear algebraic systems lacking closed-form parametric solutions [39, 56, 61, 64]. These approaches allow for the relaxation of a wide variety of user-defined functions that arise in numerous application areas. For instance, a process flow diagram may be reformulated into a simulation that is solved in a sequential block fashion.

The usage of McCormick relaxations has been limited by two significant factors. First, no publicly available optimizers made use of McCormick composition rules to construct convex and concave relaxations. EAGO was the first optimizer to remedy this issue (by a matter of years) along with a recent release of a C++ implementation, MAiNGO [14]. Secondly, the bounds furnished by set-valued arithmetic may be markedly weaker than

polyhedral approaches, slowing solution speed [48]. One of the reasons for this disadvantage is that operator-overloading approaches cannot recognize and exploit internal problem structure during the presolve phase and exploit this information throughout the course of the branch-and-bound algorithm when the recognition of linear terms, common subexpressions, and convexity properties—as well as the introduction of auxiliary variables—may be beneficial [60, 66, 67].

Our package EAGO—an acronym for **E**asy **A**dvanced **G**lobal **O**ptimization—seeks to provide a unified framework for both auxiliary variable methods and set-valued arithmetic methods with a greatly simplified user interface. Included in EAGO are a development toolkit, a state-of-the-art deterministic global optimizer, and an API for optimizing problems defined as functions in Julia script. The primary appeal of constructing EAGO in the Julia language [11] is that it strikes a balance between the speed requirement of scientific computing and the high-level syntax. Using Julia, algorithms can be written in a syntax simpler than MATLAB$^{\text{TM}}$ while achieving execution speeds as fast as C and Fortran [11]. A simple package management system is included in Julia's base distribution making propagation of software quite simple. In addition to the above advantages, EAGO exploits Julia's Lisp-like abstract syntax tree (AST) for handling expressions to implement tasks that would be extremely difficult, and in some cases impossible in other scientific programming languages. As a consequence, optimization problems may be formulated in Julia script and passed to an advanced global optimizer using a function no more complicated than MATLAB's `fmincon` [16]. This interface serves two purposes. First, it is expected to act as a bridge to new users, who may be unfamiliar with efficient problem formulations or AML syntax, by allowing them to try out a global optimizer on a specific class of test problems before investing the time and energy in learning an AML. In addition, it provides subject matter experts with unparalleled flexibility and the advanced capabilities needed to address the highly-complex problems on the forefront of optimization research.

In this paper, we detail the EAGO deterministic global optimization package and its novel implementation. In Section 2, we establish the mathematical conventions used in the paper. In Section 3, we describe a nonconvex optimizer developed using this toolkit. In Section 4, we provide preliminary benchmarking data illustrating the relative competitiveness of EAGO to extant approaches. In Section 5, we discuss EAGO's extensibility and flexibility. Lastly, we conclude in Section 6 by summarizing EAGO's capabilities and suggesting directions for future research.

2. Mathematical Notation

EAGO provides a toolkit for assembling and modifying branch-and-bound algorithms along with their variants. Specifically, it provides a series of tools for addressing nonconvex optimization problems of the form[1]:

$$\begin{aligned} f^* = \min_{\mathbf{y}\in Y\subset\mathbb{R}^n} & f(\mathbf{y}) \\ \text{s.t. } & \mathbf{h}(\mathbf{y}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{y}) \leq \mathbf{0} \end{aligned} \tag{1}$$

[1]Equivalent forms of this problem are supported via the JuMP AML.

where $f : Y \to \mathbb{R}$ is the objective function, and $\mathbf{g} : Y \to \mathbb{R}^{n_g}$ and $\mathbf{h} : Y \to \mathbb{R}^{n_h}$ are constraint functions. The lower and upper bounds on the variables are given by $\mathbf{y}^L, \mathbf{y}^U \in \mathbb{R}^n$, respectively. A subset of the equations in $\mathbf{h}$ may form linear constraints $\mathbf{A}\mathbf{y} = \mathbf{b}$ specified by the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with the vector $\mathbf{b} \in \mathbb{R}^m$. In order to ensure that the above problem is well-posed, it is typical to assume that all functions are continuous and suitably bounded [29]. In this paper, we refer to variables that appear in any nonlinear expressions as *nonlinear* variables and variables that appear in any nonconvex equality, inequalities, or objective as *nonconvex* variables. The convexity tests used in most optimization routines check sufficient conditions such as determining that natural interval extension of the Hessian of the augmented Lagrangian is everywhere positive-definite [47]. As a consequence, expressions referred to as *convex* should be understood to be expressions that have passed sufficient conditions for convexity.

In addition to the above terminology, the following conventions are followed throughout the paper. Given $\mathbf{x}^L, \mathbf{x}^U \in \mathbb{R}^n$ with $\mathbf{x}^L \leq \mathbf{x}^U$, $X = [\mathbf{x}^L, \mathbf{x}^U]$ will represent an n-dimensional interval that is a nonempty compact set defined as $X = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U\}$ with $\mathbf{x}^L$ and $\mathbf{x}^U$ the lower and upper bounds of the interval, respectively. Additionally, let $\mathbb{IR}^n$ be the set of all n-dimensional real intervals and for any $D \subset \mathbb{R}^n$, $\mathbb{I}D = \{X \in \mathbb{IR}^n : X \subset D\}$ is the set of all interval subsets of D. The mapping $F : \mathbb{I}D \to \mathbb{IR}^n$ is *inclusion-isotonic* provided that $X \subset Y$ implies that $F(X) \subset F(Y)$. The image of X under the mapping $\mathbf{f} : D \to \mathbb{R}^n$ will be denoted by $\mathbf{f}(X)$ whereas an inclusion-isotonic interval extension of $\mathbf{f}$ on X will be denoted by $F(X) = [\mathbf{f}^L(X), \mathbf{f}^U(X)]$. From the Fundamental Theorem of Interval Analysis [41, p.47] we have $\mathbf{f}(X) \subset F(X)$, $\forall X \in \mathbb{I}D$.

3. Global Optimization Framework

The EAGO package maintains a high-performance deterministic global optimizer. At the time of writing this, the best optimizer available is detailed below. Further upgrades to EAGO's default optimizer will be described in the release notes on future tagged versions of the EAGO.jl package available through Julia's package manager. The EAGO optimizer is specialized to treat programs with nonlinear objectives and constraints via a simulation approach rather than via the auxiliary variable method. The distinction between the approaches is primarily the result of how the factorable representation of the problem is treated. In the following section, we discuss the implementation of the default EAGO optimizer. The relaxations supported are detailed in Section 3.2. A description of the presolve steps is presented in Section 3.3. An overview of the domain reduction techniques used is given in Section 3.5. In Section 3.6, the construction of the relaxed lower-bounding problem is discussed. In Section 3.7, we conclude by discussing the formulation of the upper-bounding problem.

3.1 *A Flexible Branch-and-Bound Routine*

EAGO includes an implementation of the spatial branch-and-bound algorithm [29]. When used with specific lower-bounding and node-selection routines, this algorithm furnishes an ϵ-optimal global solution after a finite number of iterations. Numerous approaches to generating bounds and selecting nodes have been shown to provide these guarantees. Further development of such routines remains the subject of active research. For a detailed discussion of the branch-and-bound algorithm, the reader is referred to the excellent review presented in [29]. Most complete global optimizers also perform some additional processing routines on each node to shrink the domain size [50]. EAGO allows the user

to define preprocessing and postprocessing functions as necessary. Information furnished from computing the relaxations can be readily accessed from either of these functions.

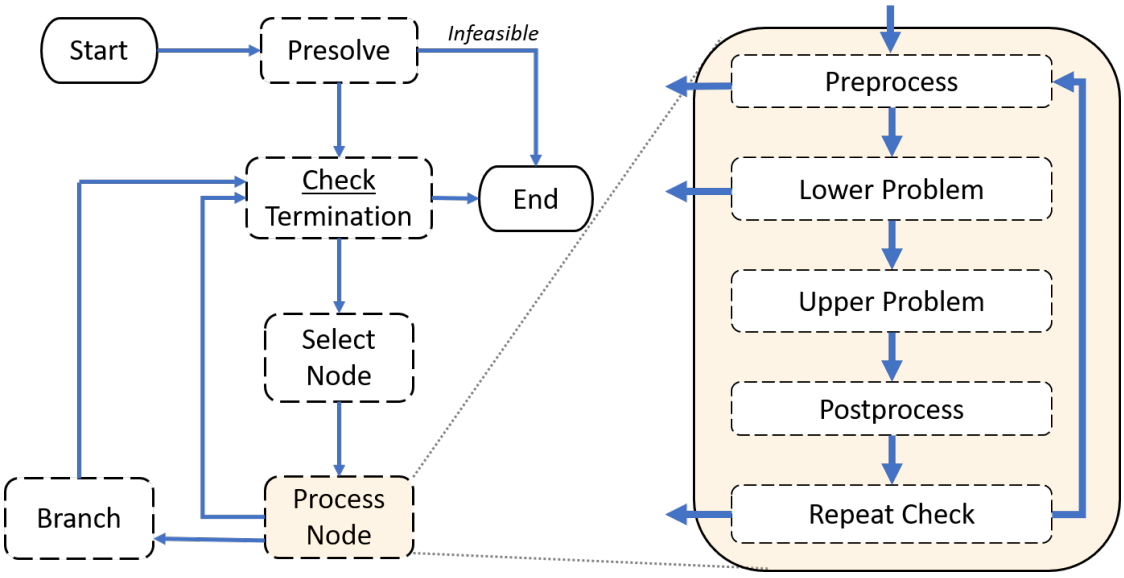

Figure 1. A block flow diagram depicting the main flexible branch-and-bound routine implemented in EAGO.

We will utilize the following notation within the flexible branch-and-bound framework (Alg. 1) presented below. Let $Y \in \mathbb{IR}^n$ be the box constraints on the decision variable $\mathbf{y}$ of program (1). Let the global lower and upper bound of f^* in (1) at iteration k be denoted by α_k and β_k, respectively. Let $Y_l \in \mathbb{I}Y$ correspond to a node in the branch-and-bound tree where f_l^{LBD} and f_l^{UBD} are lower and upper bounds of f on Y_l, respectively. Lastly, let $f^* = f(\mathbf{y}^*)$ be the ϵ-optimal objective function value where $\mathbf{y}^*$ is a feasible point corresponding to a solution of (1) at termination of the algorithm.

ALGORITHM 1 (Flexible Branch-and-Bound Framework)

(1) Initialization
- *(a) Set the stack to* $\Sigma = \{Y\}$.
- *(b) Set the iteration number* $k := 0$*, set tolerances, set the global upper bound* $\alpha_0 := +\infty$*, and the global lower bound* $\beta_0 := -\infty$.

(2) Termination
- *(a) If* $\Sigma = \emptyset$*, the algorithm terminates with* (1) *infeasible.*
- *(b) Check if a termination condition is satisfied.*[2] *If the absolute tolerance condition is satisfied, terminate with* $f^* := \alpha^k$ *as the* ϵ*-optimal estimate for the optimal objective function value and* $\mathbf{y}^*$ *as a feasible point at which* f^* *is attained.*
- *(c) Delete from* Σ *all nodes with* $f_l^{LBD} > \alpha_k$ *and set* $\beta_k := \min_{Y^l \in \Sigma} f_l^{LBD}$.

(3) Node Selection
- *(a) Select and delete a node* Y_l *from the stack* Σ *according to a selection heuristic.*

(4) Preprocessing
- *(a) Apply preprocessing routines*[3]*. If infeasibility on* Y_l *is established, go to Step 2, else set* Y_l *to the bounds determined by preprocessing routines.*

[2]To ensure an ϵ-optimal solution is reached in finite time, the termination condition must be based on an absolute tolerance. Other conditions are included to deal with numerical issues. Additional assumptions must also be satisfied such as pointwise convergence of the relaxations and Lipschitz continuity of the functions involved.
[3]Typically, domain reduction algorithms may be applied at this step.

(5) Lower-Bounding Problem
  (a) Construct and solve the lower-bounding problem globally on Y_l.
  (b) If the problem is infeasible, set $f_l^{LBD} := +\infty$. Otherwise, set f_l^{LBD} to the value of the solution. If $f_l^{LBD} < \alpha^k$ and the feasible optimal solution found, $\check{\mathbf{y}}$, is also feasible in (1), then set $\alpha_k := f_l^{LBD}$ and $\mathbf{y}^* := \check{\mathbf{y}}$.
(6) Upper-Bounding Problem (optional)
  (a) Solve (1) locally on Y_l.
  (b) If a feasible solution is found with $f_l^{UBD} < \alpha_k$, $\alpha_k := f_l^{UBD}$ and set $\mathbf{y}^*$ to the solution found.
(7) Postprocessing
  (a) Apply postprocessing routine[3] and adjust bounds of Y_l, accordingly.
(8) Fathoming
  (a) If $f_l^{LBD} = +\infty$ or $f_l^{LBD} > \alpha_k$, go to Step 2.
(9) Repetition
  (a) Checks if the repetition condition is satisfied[4]. If so, proceed to Step 4.
(10) Branching
  (a) Select a branching dimension i according to a selection heuristic. Partition node Y_l in this dimension to form nodes $Y_{l'}$ and $Y_{l''}$.
  (b) Set the lower bounds of the new nodes, $f_{Y_{l'}}^{LBD}, f_{Y_{l''}}^{LBD} := f_l^{LBD}$, and add these nodes to the working stack Σ.
  (c) Advance the iteration number, $k := k + 1$, and go to Step 2.

While multiple software implementations of branch-and-bound exist, the degree to which the user can adapt the actual branch-and-bound algorithm to specialized problem formulations is often limited [1, 51]. For example, Juniper only provides a framework for branching on integer variables [35]. EAGO's flexible branch-and-bound routine is novel in that it circumvents these limitations by allowing the user to redefine any method used by the main routine. Each block depicted in Figure 1 can be set to user-defined subroutines. This is done by defining a subtype `EAGO.ExtensionType` and then extending the EAGO's base method to specialize on this newly defined type. A demonstration of this usage is provided in Section S2.2 of the Supplementary Materials. This modular architecture allows for the easy implementation of custom optimization schemes.

3.2 *Relaxations*

The relaxation libraries provided with EAGO are based on McCormick relaxation composition rules. The McCormick relaxation of the bilinear function was first introduced by McCormick in [36]. This relaxation which consists of bounding the bilinear term using a series of affine inequalities which many commercially available global optimizers such as ANTIGONE and BARON [17, 27, 37, 55]. In the past decade, a significant effort has been made to further generalize this approach to arbitrary nonlinear functions. An operator-overloading scheme for constructing McCormick-based relaxations of functions described by a class of algorithms was detailed by Mitsos et al. [39] and has been outlined in Section S1 of the Supplementary Materials. Variations on this manner of constructing relaxations through operator overloading have been termed *McCormick relaxations*; a convention we adopt herein to maintain consistency with the extant body of literature.

Theoretical developments for generalizing the McCormick relaxation framework and constructing convex and concave composite relaxations using arbitrary convex and con-

[4]If domain reduction in postprocessing yields a significant improvement, repetition may be beneficial.

cave functions were established by Scott et al. [56]. More recently, tighter composition rules for multiplication and maxima operators were presented in [45, 70]. Methods of generating relaxations of implicit functions were developed by Stuber et al. [64]. Wechsung et al. [73] described a method of propagating McCormick relaxations backwards on a direct acyclic graph (DAG) representation of a problem. A method for tightening interval bounds was described in [44]. Alternative differentiable relaxations were introduced in [33, 34]. Additionally, McCormick relaxations have been shown to converge quadratically under reasonable assumptions [12, 42, 43]; a requirement for avoiding clustering with branch-and-bound [31]. While multiple papers have shown the utility of solving problems via McCormick relaxations of factorable functions, the availability of an optimizer using these relaxations is quite limited.

EAGO fills this void as a pioneering open-source optimization solver and research platform supporting McCormick-based relaxations of general factorable functions. Envelopes of simple expressions are used to generate relaxations of nonconvex intermediate functions. The following functions are currently supported in EAGO with correctly-rounded interval bounds:

- **Arithmetic**: +, -, ×, /, sqr, power, ln, log, sqrt, exp
- **Nonsmooth**: abs, sign, min, max
- **Trigonometric**: cos, sin, tan, sec, csc, cot
- **Inverse Trigonometric**: acos, asin, atan, asec, acsc, acot
- **Hyperbolic**: cosh, sinh, tanh, sech, csch, coth
- **Inverse Hyperbolic**: acosh, asinh, atanh, asech, acsch, acoth

Additional scaled versions of common operators and transcendental functions such as log10(·) and sind(·) are also supported. As such, relaxations of nearly all functions available in standard AD libraries are included in EAGO. Rules for computing subgradients of McCormick relaxations and gradients of the differentiable McCormick relaxations are also included.

The calculation of convex and concave envelopes for convexoconcave (a univariate function, $q : D \to \mathbb{R}$, for which $\exists p \in D$ such that q is convex on $\{d \in D \mid d \leq p\}$ and concave on $\{d \in D \mid d \geq p\}$), concavoconvex (the negation of a convexoconcave function), and general periodic functions requires the computation of anchor points at which line segments meet. These anchor points, that depend only on interval bounds, are calculated using a one-dimensional root-finding algorithm, such as a secant method or Newton's method. EAGO computes roots to an absolute tolerance of 10^{-10}. The computation of these anchor points and propagating correctly-rounded interval bounds may represent the main computational expense when generating relaxations as illustrated by Figure 2.

One of the powerful features of EAGO is its ability to decouple various components required to evaluate a relaxation via a source code transformation. A Wengert list [26] (synonymously *tape* or *trace*) is generated using a source code transformation technique that supports field access, nested-tape generation, and array operators that would be challenging for most state-of-the-art AD software packages [53]. The representation provided by the JuMP AML is used to create a specialized `Evaluator` for nonlinear and user-defined expressions. The `Evaluator` structure included in EAGO allows for the reuse of intermediate values obtained by computationally expensive protocols. Interval bounds and anchor points for each factor are only recomputed if the relaxation is being constructed on a new domain or if both forward and reverse passes are performed (which may alter interval bounds). This expedites additional evaluations such as those required to add additional cutting planes to form outer approximations or as part of callback function evaluations requested by a local NLP optimizer. The interval subgradient cut

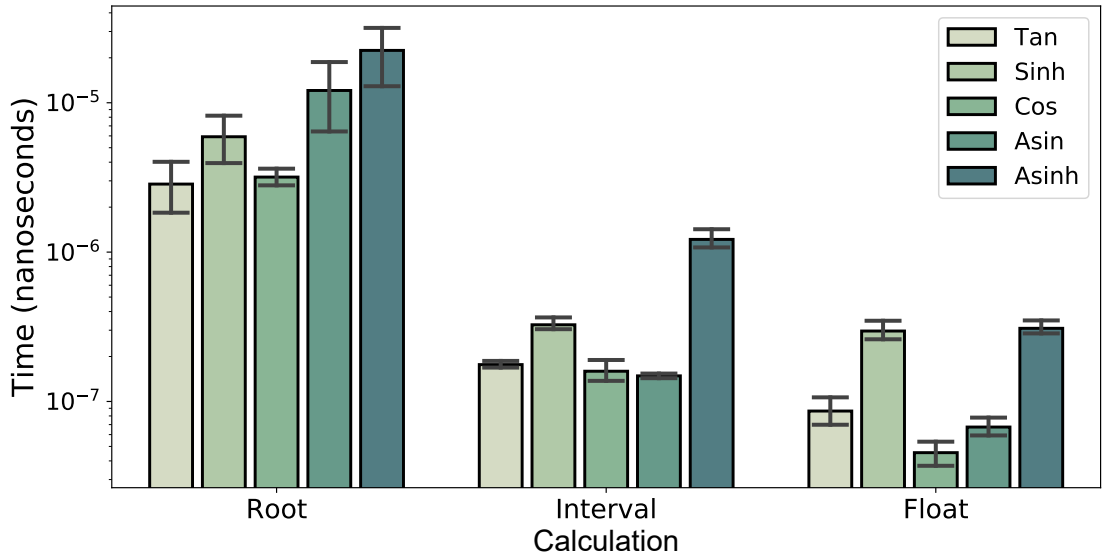

Figure 2. A breakdown of the run-times for computing relaxations associated with implementing an objected-oriented approach by operator.

detailed in Proposition 3.1, Section 3.5.4, is only performed during the first forward pass, after all forward-reverse passes are performed, or evaluation occurs on a new domain. This prevents the distinct relaxation from being evaluated at each point. Flowcharts depicting the distinction between this novel approach and a pure overloading-based implementation are provided in Figure 1 of the Supplementary Materials.

To supplement the source code transformation approach, a full multiple dispatch-based McCormick relaxation implementation is included. This library defines methods for McCormick operators that dispatch on the `struct MC` data type. Each instance of the `struct MC` stores the convex relaxation in the field `cv`, the concave relaxation in the field `cc`, the interval bounds in the field `Intv`, and subgradients of convex and concave relaxations in the `cv_grad` and `cc_grad` fields, respectively. The user may force EAGO to use the multiple dispatch implementation to compute relaxations in order to reduce memory requirements using keyword arguments. Additionally, for some select expressions where source code transformations are not expected to yield improvements to computational speed, EAGO defaults to a multiple dispatch implementation.

3.3 *Presolving*

Presolving may dramatically improve solution times by detecting special structures, rearranging algebraic terms to tighten relaxations, and potentially lifting problems into simpler higher-dimensional forms. EAGO utilizes a variety of techniques to accomplish this goal. First, linear and quadratic constraints are stored in a sparse format during the solution routine. A list of variables appearing only in linear and quadratic terms are constructed. Subsequently, a DAG representation of every nonlinear objective and constraint function present in the model is generated. After construction, the DAG undergoes algebraic rearrangement to improve relaxation performance. These rearrangements simplify expressions and change terms with weak relaxations into equivalent terms with tighter relaxations (e.g., treating the subexpression $x\log(x)$ as the convex negative entropy function). This is similar to ANTIGONE's use of (2) and (7) for reducing the level

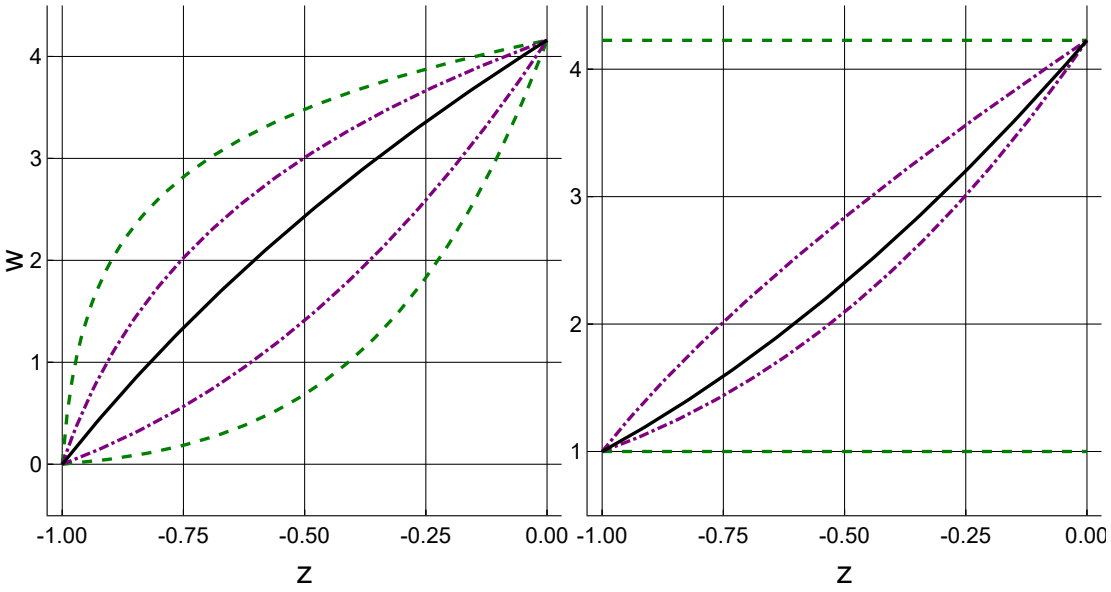

Figure 3. Suppose $z \in [-1, 0]$, $a = 2$, $g(z) = (z + 2)^3$ and the standard order of operations is used to evaluate expressions. **Left:** The function $w = a^{\log g(z)}$ (——) is plotted with convex/concave relaxations of $w = a^{\log g(z)}$ (- - -) and the rearrangement $w_r = g(z)^{\log a}$ (-·-·-). **Right:** The function $w = \log g(z)g(z)$ (——) is plotted with convex/concave relaxations of $w = \log g(z)g(z)$ (- - -) and the rearrangement $w_r = \log g(z) + \log g(z)$ (-·-·-).

at which auxiliary variables are introduced in standard factorable program [37] or using the rearrangements of Khajavirad and Sahinidis [32] to improve the inferences made via disciplined convex programming (DCP) [57].

EAGO's framework provides a simple syntax for implementing expression transformation by registering template DAGs corresponding to subexpressions. This is done by creating the appropriate `Template_Graph` objects then invoking the `register_substitution!`. Transformations to the Wengert list are made by locating subexpressions that match specified forms and substituting in equivalent subexpressions. The rearrangements given in Equations (2)-(8) are included in EAGO's optimizer and result in McCormick relaxations that may be significantly tighter than the original form.

$$\exp(x)\exp(y) = \exp(x + y) \tag{2}$$
$$\log(a^x) = x\log(a) \tag{3}$$
$$(a^x)^b = (a^b)^x \tag{4}$$
$$(x^a)^b = x^{(ab)} \tag{5}$$
$$a^{\log(x)} = x^{\log(a)} \tag{6}$$
$$\log(xy) = \log(x) + \log(y) \tag{7}$$
$$\log(x/y) = \log(x) - \log(y) \tag{8}$$

An illustration of the tightening effect produced by applying (6) and (7) is given in Figure 3. Note that (2)-(5) reduce the number of subexpressions that are relaxed. This improves computation speed as calculations of relaxations are significantly more expensive than floating-point calculations. These rearrangements do not tighten natural interval bounds.

3.4 *User-Defined Functions*

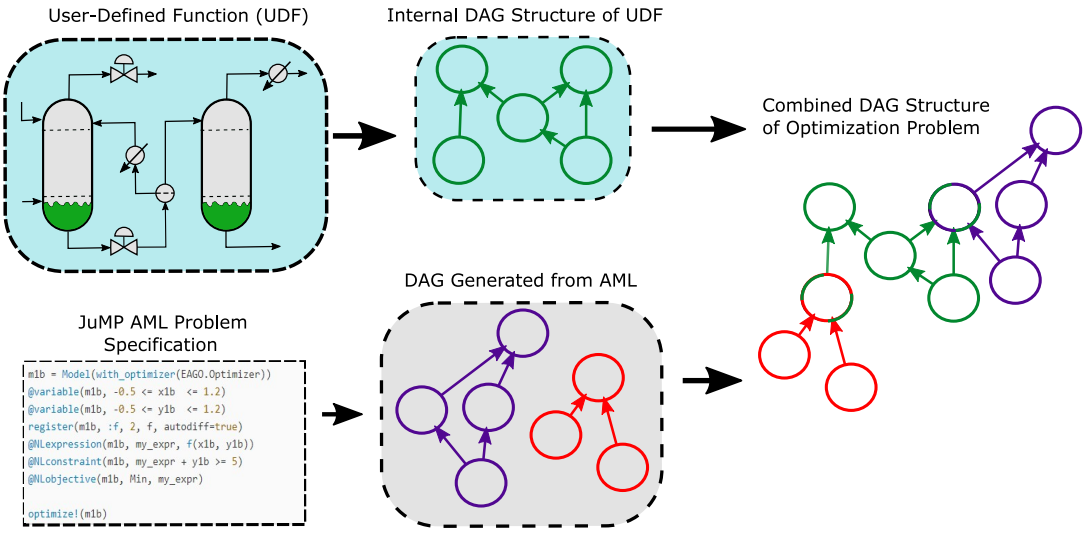

Figure 4. Given an optimization problem, EAGO generates a DAG representation for all user-defined functions, composes these into a shared DAG representation of all nonlinear expressions, further detects special structures, solves the optimization problem, and returns the model with respect to the original variables.

The development of relaxations of functions implemented in a script form rather than an AML form can be exceedingly difficult for a complete global optimizer to handle. While operator and method overloading techniques can be applied to a broad class of problems, the introduction of auxiliary variables for these functions can be challenging. Moreover, basic overloading approaches can be subject to a number of flaws that limit their practical use. In the case of AD methods, perturbation errors may result from nested overloading (e.g., required to form higher-order derivatives) [53]. Additionally, overloading approaches to tape generation require that all potential promotions are anticipated and accounted for. This limits the ability to integrate such tracing methods with other packages that may use structures internally to perform calculations. EAGO makes use of context-oriented programming tools introduced with Julia 0.7 to address forms inaccessible to method overloading or operator overloading approaches [52]. This allows for the generation of DAG representations from functions defined by Julia script; a core feature of EAGO.

In many technical application areas, components of a simulation may be represented by a series of scripts. For instance, a chemical process flowsheet model will typically embed a series of equations of state, mass and energy balances, and rules for solving the system operating under particular equipment specifications (e.g., adiabatic flash) for each block representing a unit operation. These simulations may be addressed using methods that embed sequential modular solves in these user-defined functions via a semi-explicit approach. EAGO's handling of user-defined functions allows for the construction of forward and reverse evaluation of blocks of these simulations and the determination of DAG structures that span from the process blocks to the entire model. Figure 4 illustrates how EAGO's framework allows it to reconcile process models with AML defined optimization problems. Additional user-defined functions may arise naturally as a description of dynamic behavior such as the chemical kinetic examples in [74] and Section 3.4.2.

AD is a powerful tool for addressing these formulations using local nonlinear optimizers by providing gradient information of the objective functions and constraints [26].

The direct optimization of problems with user-defined functions may also be desired in a global optimization context. To our knowledge, EAGO is the only deterministic global optimizer capable of handling user-defined factorable functions. User-defined functions may include type-assertions and calls to functions that aren't appropriate for direct overloading approaches as defined. Additionally, many implementations would require that the user change local storage objects. EAGO makes use of a context-oriented approach that overcomes each of these potential issues when generating the Wengert list for each function. In this manner, EAGO allows us to solve optimization problems that include partial derivatives of an arbitrary order as illustrated in Section 3.4.1.

3.4.1 Optimization of a Model With an Embedded Algorithm

Consider an aqueous n-butanol mixture undergoing a separation (by heating at constant pressure, P [bar], and temperature, T [K], in a fixed volume). Depending on the operating temperature, the system exhibits two immiscible liquid phases and potentially a minimum boiling azeotrope. We'll assume the existence of two immiscible phases is undesirable due to equipment limitations. This implies the following liquid-phase stability conditions:

$$\frac{d\ln(\gamma_1 x_1)}{dx_1} > 0, \quad \frac{d\ln(\gamma_2 x_2)}{dx_2} > 0 \tag{9}$$

where x_1 is the liquid-phase mole-fraction of n-butanol, x_2 is the liquid-phase mole-fraction of water, and a van Laar activity coefficient model is given by

$$\ln(\gamma_1) = \frac{1253/T}{(1 + 2.62(x_1/x_2))^2} \tag{10}$$

$$\ln(\gamma_2) = \frac{479/T}{(1 + 0.382(x_2/x_1))^2} \tag{11}$$

where γ_1 and γ_2, are the respective activity coefficients of n-butanol and water. A modified Raoult's Law and mass balance yield two other equality constraints:

$$\begin{aligned} 0 &= P - x_1\gamma_1 P_1^{vp} + x_2\gamma_2 P_2^{vp} \\ 0 &= x_1 + x_2 - 1 \end{aligned}$$

where P_1^{vp} and P_2^{vp} are, respectively, the vapor pressures of water and n-butanol, given by:

$$\begin{aligned} P_1^{vp} &= 1.33 \times \exp(11.83572 - 4169.84/(T - 17.665) \\ P_2^{vp} &= 1.33 \times \exp(11.33986 - 3724.52/(T - 69.854). \end{aligned}$$

An operating condition is sought with the minimum temperature such that the vapor phase mass fraction of n-butanol is at least 0.95 and no liquid-liquid phase split occurs. The vapor composition specification is then represented by the following constraint:

$$x_1\gamma_1 P_1^{vp}/P \geq 0.95.$$

We can then write the objective function as:

$$f(x1, x2, T) = T \tag{12}$$

where we restrict to the ranges of interest as follows $x_1 \in [0.01, 0.99]$, $x_2 \in [0.01, 0.99]$, and $T \in [363.15, 398.15]$. Within 10.1 seconds, EAGO is able to prove that no such operating condition exists, which is consistent with visual inspection of the Txy-diagrams [69].

Listing 1 Script used to set up and optimize the example given in Section 3.4.1.

```
using JuMP, EAGO, ForwardDiff

# Define the activity model
gamma1_x1(z) = z[1]*(1253/z[3])/(1 + 2.62*(z[1]/z[2]))^2
gamma2_x2(z) = z[2]*(479/z[3])/(1 + 0.382*(z[2]/z[1]))^2
cons_1ex(z...) = ForwardDiff.gradient(z -> log(gamma1_x1(z)), collect(z))[1]
cons_2ex(z...) = ForwardDiff.gradient(z -> log(gamma2_x2(z)), collect(z))[2]

# Define the JuMP model and solve
m = Model(EAGO.Optimizer)
register(m, :cons_1ex, 3, cons_1ex, autodiff = true)
register(m, :cons_2ex, 3, cons_2ex, autodiff = true)

@variable(m, 0.01 <= x[i=1:2] <= 0.99)
@variable(m, 363.15 <= T <= 398.15)
@constraint(m, x[1] + x[2] == 1.0)
@NLexpression(m, P1, 1.33*exp(11.83572 - 4169.84/(T - 17.665)))
@NLexpression(m, P2, 1.33*exp(11.33986 - 3724.523/(T - 69.854)))
@NLconstraint(m, cons_1ex(x[1],x[2],T)*P1 + cons_2ex(x[1],x[2],T)*P2 == 1.02)
@NLconstraint(m, cons_1ex(x[1],x[2],T)*P1/1.02 >= 0.95)
@NLconstraint(m, cons1, cons_1ex(x[1], x[2], T) >= 0.001)
@NLconstraint(m, cons2, cons_2ex(x[1], x[2], T) >= 0.001)
@NLobjective(m, Min, T)
optimize!(m)
```

3.4.2 Kinetic Parameter Estimation

Consider the kinetic parameter estimation problem [39], which was adapted from [59, 68]. The reaction mechanism can be modeled using the initial value problem:

$$\begin{aligned}
\frac{dx_A}{dt} &= k_1 x_Z x_Y - c_{O_2}(k_{2f} + k_{3f})x_A + \frac{k_{2f}}{K_2}x_D + \frac{k_{3f}}{K_3}x_B - k_5 x_A^2 \\
\frac{dx_B}{dt} &= c_{O_2}k_{3f}x_A - \left(\frac{k_{3f}}{K_3} + k_4\right)x_B, \quad \frac{dx_Z}{dt} = -k_1 x_Z x_Y \\
\frac{dx_D}{dt} &= c_{O_2}k_{2f}x_A - \frac{k_{2f}}{K_2}x_D, \quad \frac{dx_Y}{dt} = -k_{1s} x_Z x_Y \\
x_A(0) &= 0, x_B(0) = 0, x_D(0) = 0, x_Y(0) = 0.4, x_Z(0) = 140
\end{aligned}$$

where x_j is the concentration of species $j \in \{A, B, D, Y, Z\}$. The constants are given by $T = 273$, $K_2 = 46\exp(6500/T - 18)$, $K_3 = 2K_2$, $k_1 = 53$, $k_{1s} = k_1 \times 10^{-6}$, $k_5 = 1.2 \times 10^{-3}$, and $c_{O_2} = 2 \times 10^{-3}$. Intensity versus time data is available in [61] as well as a known dependency on concentration, $I = x_A + \frac{2}{21}x_B + \frac{2}{21}x_D$[58]. The reaction rate constants $k_{2f} \in [10, 1200]$, $k_{3f} \in [10, 1200]$, and $k_4 \in [0.001, 40]$ are unknown and form the parameter vector $\mathbf{p} = (k_{2f}, k_{3f}, k_4)$.

An implicit Euler discretization was constructed in [64, 74] and solved to global optimality by constructing relaxations of implicit functions using a fixed-point method. While EAGO can replicate this implicit approach, we'll consider the original case of the explicit Euler discretization of the problem [39] for simplicity's sake. In this example, a semi-explicit approach is used; the relaxations of intermediate factors $\mathbf{x}$ that arise during

the simulation of the ODEs are computed. These factors are subsequently propagated through to the objective function effectively eliminating the need to explicitly consider the $\mathbf{x}$ values in the problem formulation, as detailed in [13, 39]. A discretization consisting of 200 timesteps provides sufficiently high accuracy for this problem. The discretized model becomes:

$$
\begin{aligned}
x_A^{i+1} &= x_A^i + \Delta t \left(k_1 x_Y^i x_Z^i - c_{O_2}(k_{2f} + k_{3f}) x_A^i + \frac{k_{2f}}{K_2} x_D^i + \frac{k_{3f}}{K_3} x_B^i - k_5 (x_A^i)^2 \right) \\
x_B^{i+1} &= x_B^i + \Delta t \left(k_{3f} c_{O_2} x_A^i - \left(\frac{k_{3f}}{K_3} + k_4 \right) x_B^i \right) \\
x_D^{i+1} &= x_D^i + \Delta t \left(k_{2f} c_{O_2} x_A^i - \frac{k_{2f}}{K_2} x_D^i \right) \\
x_Y^{i+1} &= x_Y^i + \Delta t \left(-k_{1s} x_Y^i x_Z^i \right) \\
x_Z^{i+1} &= x_Z^i + \Delta t \left(-k_{1s} x_Y^i x_Z^i \right)
\end{aligned}
$$

where $i = 0, \ldots, 199$ and $\Delta t = 0.01$. While only three parameters are of interest in the original optimization problem, 1003 variables are required to specify this in an AML along with knowledge of reasonable variable box bounds. This is because the state vector:

$$
\mathbf{x} = \left(x_A^1, x_B^1, x_D^1, x_Y^1, x_Z^1, \cdots, x_A^{200}, x_B^{200}, x_D^{200}, x_Y^{200}, x_Z^{200} \right) \tag{13}
$$

must be accounted for as decision variables in typical formulations. Since EAGO computes composite relaxations, this problem formulation allows us to treat this state vector as a series of intermediate expressions to be evaluated; therefore, allowing us to define the problem with respect to the kinetic parameters of interest $\mathbf{p}$.

Listing 2 Explicit Euler integration scheme script used in the kinetic parameter estimation example (Ex. 3.4.2).

```
function explicit_euler_integration(p)

   x = zeros(typeof(p[1]), 1005);    # Data storage array
   x[4] = 0.4; x[5] = 140            # Sets initial condition
   T = 273; delT = 0.01; cO2 = 2e-3; k1 = 53; k1s = k1*1E-6;
   K2 = 46*exp(6500/T-18); K3 = 2*K2; h = delT; k5 = 1.2E-3

   for i=1:200                       # Offset by 1, initial condition is at x[1:5]
      term1 = k1*x[5i-1]*x[5i]-cO2*(p[1]+p[2])*x[5i-4]
      term2 = p[1]*x[5i-2]/K2+p[2]*x[5i-3]/K3-k5*x[5i-4]^2
      x[5i+1] = x[5i-4] + h*(term1 + term2)
      x[5i+2] = x[5i-3] + h*(p[2]*cO2*x[5i-4]-(p[2]/K3+p[3])*x[5i-3])
      x[5i+3] = x[5i-2] + h*(p[1]*cO2*x[5i-4]-p[1]*x[5i-2]/K2)
      x[5i+4] = x[5i-1] + h*(-k1s*x[5i-1]*x[5i])
      x[5i+5] = x[5i] + h*(-k1s*x[5i-1]*x[5i])
    end

    return x
end
```

The objective function for this problem can then be given by

$$
f(\mathbf{p}) = \sum_{i=1}^{n} \left(I_i^c(\mathbf{p}) - I_i^d \right)^2 \tag{14}
$$

where I_i^c are the calculated intensity values at time step i from the model and I_i^d are

Listing 4 Script to build the JuMP model and calculate the ϵ-global optimal solution for the Example given in Section 3.4.2.

```
# Create model and add variables
m = Model(EAGO.Optimizer)
@variable(m, pL[i] <= p[i=1:3] <= pU[i])

# Register objective, add objective function, and optimize
fobj(p...) = objective(p...)
JuMP.register(m, :fobj, 3, fobj, autodiff=true)
@NLobjective(m, Min, fobj(p...))
JuMP.optimize!(m)
```

the values corresponding to the experimental data. The EAGO optimizer converges to within 90% in just 2.6 seconds, and to within 95% in 8.2 seconds. This is comparable to the solution time presented in [39].

Listing 3 Script to load data and define the objective function for the Example given in Section 3.4.2.

```
using EAGO, JuMP, DataFrames, CSV

data = CSV.read("kinetic_intensity_data.csv")  # Loads data from csv file to DataFrame
pL = [10.0  10.0  0.001];   pU = [1200.0  1200.0  40.0]

I(xA,xB,xD) = xA + (2/21)*xB + (2/21)*xD       # Defines function for intensity
function objective(p...)                       # Integrates the ODEs and calculates SSE
    x = explicit_euler_integration(p)
    SSE = zero(typeof(p[1]))
    for i=1:200
        SSE += (I(x[5i-4],x[5i-3],x[5i-2]) - data[:intensity][i])^2
    end
    return SSE
end
```

3.5 *Domain Reduction*

Various techniques are commonly used to shrink each subdomain Y^l generated within the branch-and-bound algorithm. These techniques may significantly speed up the branch-and-bound algorithm by eliminating large regions of the search space. EAGO makes use of three main families of these routines: optimality-based bounds-tightening (Sec. 3.5.1), feasibility-based bounds-tightening (Sec. 3.5.3), and duality-based bounds-tightening (Sec. 3.5.2). EAGO executes the first two approaches during preprocessing while duality-based bounds-tightening is applied during postprocessing.

3.5.1 Optimization-Based Bounds-Tightening (OBBT)

In addition to the lower bound calculation, relaxations are also used to tighten bounds via optimization-based bounds-tightening (OBBT). For variables participating in a nonlinear term, problem (15) is solved to obtain potentially tighter lower and upper variable bounds.

EAGO implements OBBT using a greedy algorithm with filtering [25].

$$\begin{aligned} \min_{\mathbf{y}} \ & \pm y_k \\ \text{s.t. } & \mathbf{g}^{cv}(\mathbf{y}) \leq \mathbf{0} \\ & \mathbf{h}^{cv}(\mathbf{y}) \leq \mathbf{0} \\ & \mathbf{h}^{cc}(\mathbf{y}) \geq \mathbf{0} \\ & f^{cv}(\mathbf{y}) \leq \alpha_k \end{aligned} \tag{15}$$

OBBT entails solving a large number of optimization problems. As such, EAGO uses OBBT at the root node and then uses a heuristic to determine if it will be used at deeper nodes in the branch-and-bound tree. For a node of depth $d \leq k$ (default k: 4), OBBT is performed. For nodes of depth $d > k$, the OBBT performed with probability 2^{k-d} [6].

3.5.2 Duality-Based Bounds-Tightening (DBBT)

Duality-based bounds-tightening (DBBT) is performed following the solution of the relaxed problem. The dual multiplier λ_i of each variable y_i is queried along with the lower bound LBD; all positive dual multipliers are used to shrink the variable bounds [54]:

$$\begin{aligned} y_i &\geq y_i^U - \lambda_i^{-1}(\alpha_k - LBD) \\ y_i &\leq y_i^L + \lambda_i^{-1}(\alpha_k - LBD). \end{aligned}$$

3.5.3 Feasibility-Based Bounds-Tightening

Expression specific feasibility-based bounds-tightening is provided for linear constraints, univariate quadratic constraints [20], and bivariate quadratic constraints [71]. For linear constraints, the following relationships are used. The set of linear constraints $\sum_{j=1:n} a_{ij} y_j \leq b_i, i = 1, \ldots, m$. Each linear constraint i is then processed sequentially and the variable bounds are refined through application of the following relationship:

$$\begin{aligned} y_k^U &\leq \frac{1}{a_{ij}} \left(b_i - \sum_{j \neq k} \min(a_{ij} y_j^U, a_{ij} y_j^L) \right) && a_{ik} \geq 0 \\ y_k^L &\geq \frac{1}{a_{ij}} \left(b_i - \sum_{j \neq k} \min(a_{ij} y_j^U, a_{ij} y_j^L) \right) && a_{ik} < 0. \end{aligned}$$

For a full discussion of univariate and bivariate constraint bounds-tightening, the reader is encouraged to consult [20] and [71], respectively.

3.5.4 Constraint Propagation on the Directed Acyclic Graph

A two-stage constraint propagation scheme is used for nonlinear terms represented on the DAG. In this first stage, natural interval extensions along with McCormick relaxations (and associated subgradients) computed at a reference point $\bar{\mathbf{y}}$ of the nonlinear constraints and the objective are calculated via a forward pass on the DAG in topological order. The subgradients are used to improve the interval bounds if possible, according to Proposition 3.1 [44, 64]. The computed bounds of the constraints are then intersected

with constraint bounds. In the second stage, a reverse interval pass is then performed in reverse topological order [65]. This is repeated, inferring tighter variable values until either the variable bounds fail to tighten by a preset factor (default: 0.99) or a maximum number of repetitions is reached (default: 3)

PROPOSITION 3.1 *Let $v : Y \to V$ be a factor in the computation of McCormick relaxations such that $V = [v^L, v^U]$ are known interval bounds (e.g., by a natural interval extension) with convex/concave relaxations v^{cv}/v^{cc} of v on Y and their respective subgradients $\mathbf{s}_v^{cv}$, $\mathbf{s}_v^{cc}$ computed at $\mathbf{y} = \bar{\mathbf{y}} \in Y$. The functions $\omega, \mu : Y \to \mathbb{R}$ are the affine relaxations of the convex and concave relaxations of v on Y, respectively, and defined as:*

$$\omega(\mathbf{y}) \equiv v^{cv}(\bar{\mathbf{y}}) + \mathbf{s}_v^{cv}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}})$$
$$\mu(\mathbf{y}) \equiv v^{cc}(\bar{\mathbf{y}}) + \mathbf{s}_v^{cc}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}}).$$

The lower and upper bounds of these relaxations are themselves valid bounds of v on Y. Valid lower and upper bounds of the image set $v(Y)$ are given by:

$$v^{L,new} := \max(v^L,\ \omega^L(Y)) \tag{16}$$
$$v^{U,new} := \min(v^U,\ \mu^U(Y)). \tag{17}$$

By construction, the bounds furnished by (16) and (17) are at least as tight as the original interval bounds. EAGO uses the midpoint of Y as the reference point, $\bar{\mathbf{y}}$; this is repeated until either the variable bounds reduction falls below a preset threshold (default: 0.99) or until a maximum number of repetitions is reached (default: 5).

3.6 *Lower-Bounding Problem*

A lower bound on the optimal solution value is calculated by solving to global optimality the relaxation of (1), given as:

$$\begin{aligned} f^{LBD} = \quad & \min_{\mathbf{y}\in Y} f^{cv}(\mathbf{y}) \\ \text{s.t. } & \mathbf{g}^{cv}(\mathbf{y}) \le \mathbf{0} \\ & \mathbf{h}^{cv}(\mathbf{y}) \le \mathbf{0} \\ & \mathbf{h}^{cc}(\mathbf{y}) \ge \mathbf{0}. \end{aligned} \tag{18}$$

EAGO's default optimizer further relaxes this form via polyhedral outer approximation of the McCormick-based relaxations detailed in Section 2. For nonlinear expressions, an affine relaxation is generated via an affine approximation of the expression at the midpoint of the domain using subgradient information [39]. Objective function value cuts taken at the midpoint are also added using the current global upper bound, α_k:

$$\begin{aligned} f^{LBD} = \quad & \min_{\mathbf{y}\in Y} f^{cv}(\bar{\mathbf{y}}) + \mathbf{s}_f^{cv}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}}) \\ \text{s.t. } & \mathbf{g}^{cv}(\bar{\mathbf{y}}) + \mathbf{s}_g^{cv}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}}) \le \mathbf{0} \\ & \mathbf{h}^{cv}(\bar{\mathbf{y}}) + \mathbf{s}_h^{cv}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}}) \le \mathbf{0} \\ & \mathbf{h}^{cc}(\bar{\mathbf{y}}) + \mathbf{s}_h^{cc}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}}) \ge \mathbf{0} \\ & f^{cv}(\bar{\mathbf{y}}) + \mathbf{s}_f^{cv}(\bar{\mathbf{y}})^{\mathrm{T}}(\mathbf{y} - \bar{\mathbf{y}}) \le \alpha_k. \end{aligned}$$

This is done in part because the standard McCormick relaxations [39] are potentially nonsmooth and therefore may pose difficulty for gradient-based NLP optimizers. Additionally, the polyhedral relaxation can be significantly faster than a differentiable NLP relaxation for certain problems. The lower-bounding problem is then solved using the specified LP optimizer. Additional cutting planes are generated by adding constraints at new reference points based on the solution of prior relaxation (default: up to 3) and the objective is then updated with the new reference point. Alternatively, EAGO can supply an `Evaluator` structure to local NLP optimizers. This evaluator can be queried for function and subgradient values. If differentiable McCormick relaxations are selected a MathOptInterface-wrapped local NLP optimizer may be used to furnish lower bounds instead.

3.7 *Upper-Bounding Problem*

The use of tight upper bounds can accelerate the convergence of the branch-and-bound algorithm by increasing the rate at which nodes are fathomed. One popular choice is that of a feasible local solution. As solving an NLP to local optimality can be computationally expensive, EAGO makes use of a heuristic similar to that of Couenne [6] to limit the number of upper-bounding problems solved. That is, for a node of depth d less than a tolerance k, the local NLP is solved. For nodes of depth $d > k$, the local NLP is solved with probability 2^{k-d} [6]. By default, EAGO uses Ipopt [72], but any other `MathOptInterface.jl` compatible NLP optimizer can be specified passing keyword arguments to the EAGO optimizer.

4. Numerical Experiments

The data files and code for all examples are freely available in the EAGO Git repository, https://github.com/PSORLab/EAGO.jl. Additional special use cases in the Supplementary Materials further illustrate EAGO's flexibility. EAGO version 0.4.0 was used with an absolute tolerance of $\epsilon_a = 10^{-3}$ and a relative tolerance of $\epsilon_r = 10^{-3}$. All numerical experiments were run three times on a single thread of a 3.60GHz (4.00GHz turbo) Intel Xeon E3-1270 v5 processor with 32GB in Ubuntu 18.02LTS and Julia v1.4.2. The lower-bounding problem was solved using Gurobi 9.0.2 [27]. The upper-bounding problem was solved using Ipopt v3.12.13 [72]. Julia, Ipopt, and CPLEX are all compiled with Intel MKL 2019 (Update 3) versions of LAPACK/BLAS. EAGO makes use of the JuMP mixed-mode AD scheme for general problems [21] and a forward-mode AD scheme for user-defined functions [53]. MathOptInterface v0.9.13 and JuMP version 0.21.2 were used to formulate problems and provide interfaces to sub-solvers in a myriad of internal subroutines. The ValidatedNumerics.jl library was used for correctly-rounded interval calculations [8] using the `:accurate` rounding mode that is slightly more conservative than the IEEE Standard 1788-2015 [30], but is often significantly faster.

One of the primary advantages of EAGO is the relative speed of the Julia language. To illustrate this, we provide a comparison of the solution times between EAGO and three state-of-the-art deterministic global optimizers: BARON, ANTIGONE, and SCIP. Twenty problems were selected from the MINLP2 and GLOBAL library that contains only expressions supported by BARON, ANTIGONE, and SCIP. A list of the problems along with a brief summary of formulation traits is given in Table 1. A maximum of 1000 seconds are allowed for each numerical experiment. BARON v17.10.16, ANTIGONE v1.1, and SCIP v5.0 were used for these experiments. Optimizer performance is assessed using

the methods presented in Dolan and Moore [19]. A performance profile for this test set is provided in Figure 5 and accompanying data is provided in Tables 2 and 3 of the Supplementary Materials. The *performance* of optimizer s is the time in CPU seconds, $t_{p,s}$, required to solve problem p. The *performance ratio* on problem p by optimizer s is the ratio of the optimizer's performance to the best optimizer's performance in the set:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in S\}}.$$

The *performance profile* of optimizer s is the plot of the distribution function of the performance metric where $\rho_s(\tau)$ is the probability that a performance ratio $r_{p,s}$ is within a factor $\tau \in \mathbb{R}$ of the best possible ratio

$$\rho_s(\tau) = \frac{1}{n_p}\text{card}\{p \in \mathcal{P} : r_{p,s} \leq \tau\}$$

where card(S) denotes the cardinality of set S, $\mathcal{P}$ is the set of problems, $n_p = \text{card}(\mathcal{P})$. The percent relative gap remaining at time t for problem p is given by $g_t = 100 \times (UBD - LBD)/\max(|UBD|, |LBD|)$. As illustrated by Figure 5, EAGO solves many problems with times comparable to state-of-the-art global optimizers. Of the problems not solved within the time limit, EAGO typically yields a smaller percent relative gap than the other optimizers. This suggests EAGO can provide meaningful run time results when used to develop novel optimization routines. However, no claim of superiority is appropriate at this time. This is particularly evident for nonconvex quadratic programs in which SCIP and BARON both outperform EAGO. This is expected as EAGO makes use of the McCormick-based relaxation framework for relaxing quadratic constraints whereas other optimizers use specialized routines for relaxing quadratic constraints. We leave a more exhaustive benchmarking analysis for a later date.

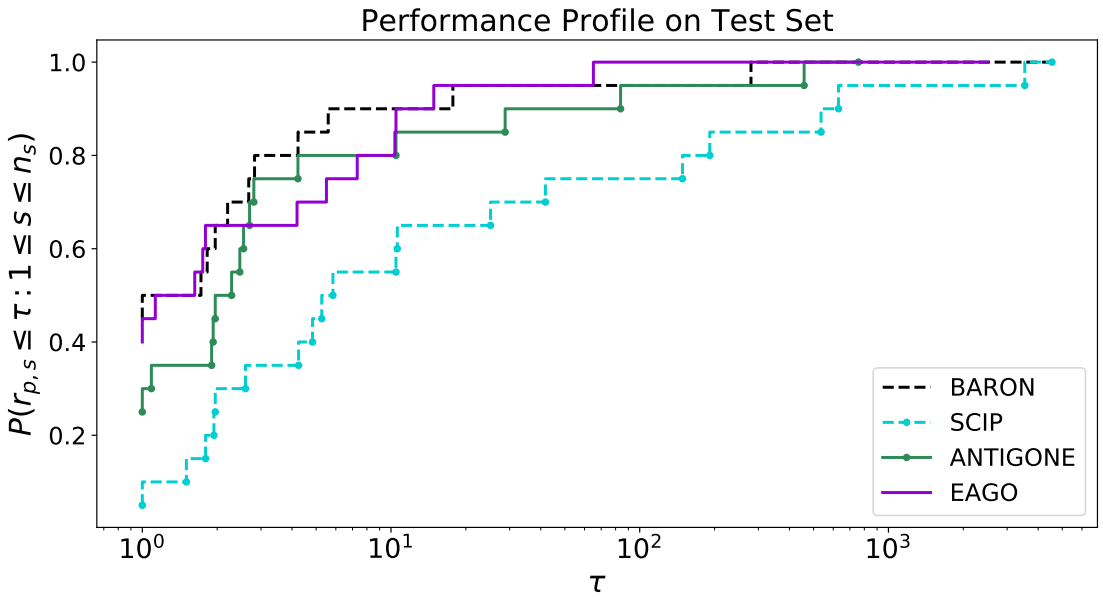

Figure 5. Performance profiles for the test set enumerated in Table 1.

5. Extensibility of EAGO

The EAGO framework has already been used to demonstrate multiple novel approaches to global optimization. The quasiconvexity of a hybrid-solar system was exploited allowing for the problems to be solved with a certificate of global optimality [62]. In subsequent works, novel affine relaxations [15] and relaxations of implicit functions [74], were implemented by customizing and extending EAGO, respectively. One illustration of the capability is provided in the EAGO package itself as functionality to solve nonconvex semi-infinite programs (SIPs) as illustrated in Section 5.1.

5.1 *Solving Semi-Infinite Programs*

EAGO implements the SIPres algorithm [38, 63] for solving general nonconvex SIPs that converges in a finite number of iterations under mild assumptions. For full implementation details, the reader is directed to the EAGO GitHub repository at https://github.com/PSORLab/EAGO.jl. Consider the standard-form SIP:

$$\begin{aligned} f^* = \quad & \min_{\mathbf{x}\in X} f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}(\mathbf{x},\mathbf{p}) \leq 0,\ \forall \mathbf{p}\in P,\ |P| \leq \infty \\ & X = \{\mathbf{x}\in\mathbb{R}^{n_z} : \mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U\} \\ & P = \{\mathbf{p}\in\mathbb{R}^{n_p} : \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\}. \end{aligned} \tag{19}$$

A design centering problem developed in [22] is presented here. A location consisting of coordinates x_1, x_2 is sought such that a disk with maximal radius, x_3, can be inscribed within a nonconvex container set. The SIP is stated formally as:

$$\begin{aligned} f^* = \quad & \max_{\mathbf{x}\in X} x_3 \\ \text{s.t.} \quad & g_1(\mathbf{x},p) = 0.3\sin(\pi z_1(\mathbf{x},p)) - z_2(\mathbf{x},p) \leq 0,\ \forall p\in P \\ & g_2(\mathbf{x},p) = z_1(\mathbf{x},p)^2 + 0.3 z_2(\mathbf{x},p)^2 - 1 \leq 0,\ \forall p\in P \\ & X = [-1.5,1.5]\times[-1,2]\times[0,1.5] \\ & P = [0,2\pi] \end{aligned} \tag{20}$$

where the $\mathbf{z}$ terms are determined by the expressions:

$$\begin{aligned} z_1(\mathbf{x},p) &= x_1 + x_3\cos(p) \\ z_2(\mathbf{x},p) &= x_2 + x_3\sin(p). \end{aligned}$$

This problem is constructed in the few lines of code contained in Code Listing 5. Using EAGO's implementation of the SIPres routine in combination with its global optimizer, we were able to solve (20) and obtain certification of global optimality with an absolute tolerance of $\epsilon_a = 10^{-3}$ in 1.97 seconds. All prior attempts to address this problem required approximations of the constraint set or the use of approximate methods as done in the original work of Floudas [22].

Listing 5 The code for solving the design centering problem given in Example 20 originally presented in [22]

```
using EAGO, Gurobi
```

```
f(x) = x[3]                                          # Define objective
z1(x,p) = x[1] + x[3]*cos(p[1])
z2(x,p) = x[2] + x[3]*sin(p[1])
gSIP1(x,p) = 0.3*sin(pi*z1(x,p)) - z2(x,p)           # Define SIP constraints
gSIP2(x,p) = z1(x,p)^2 + 0.3*z2(x,p)^2 - 1.0

xL = [-1.5, -1.0, 0.0];  xU = [1.5, 2.0, 1.5]        # Defines bounds
pL = [0.0];  pU = [2.0*pi]

result = explicit_sip_solve(xL, xU, pL, pU, f, [gSIP1, gSIP2], sip_sense = :max,
                            relaxed_optimizer = Gurobi.Optimizer())
```


6. Conclusions and Future Directions

We have presented the first openly-available McCormick relaxation-based global optimizer. This optimizer contains a number of features prevalent in modern nonconvex optimization software and each of these features can be readily included or omitted from user-developed routines. Among the features discussed were an interval constraint programming algorithm, an affine interval bounds-tightening algorithm, duality-based bounds-tightening algorithm, and a generic optimization-based bounds-tightening routine. Routines for constructing outer approximations of convex function envelopes were discussed.

The basic optimizer developed in EAGO performs impressively relative to state-of-the-art deterministic global optimizers on the selected examples shown. The number of algebraic expressions supported by EAGO is significantly larger than the existing complete global optimizers. In fact, the library size is comparable to that of modern AD software [3]. Most notably, we have illustrated the use of our deterministic global optimizer with native Julia code. As illustrated, support for script-defined functions serves a two-fold purpose. First, it allows non-experts to explore simple optimization problems without domain-specific knowledge. Second, it provides a framework that experts can use to construct specialized algorithms for computing global optima of problems with embedded simulations, already utilized in recent works [15, 62, 74].

Multiple avenues exist to further improve EAGO's branch-and-bound optimization framework and default optimizer. One potential avenue of interest is exploring the use of tighter interval arithmetic in conjunction with McCormick-based relaxations. The use of tighter interval bounds such as those generated by interval-Taylor arithmetic [10] may yield significantly tighter relaxations and speed solution time [43]. Another potential improvement to EAGO lies in further specializing it to handle various categories of models with embedded simulations. One such improvement may lie in selectively adding auxiliary variables to the formulation in the presolve step to tighten relaxations and improve domain reduction. Another improvement may lie in the automatic detection of implicit function reformulations [64, 74] and the application of specialized routines to address these. One major adaptation of the EAGO toolkit presently underway is the incorporation of a branch-and-cut framework for solving mixed-integer nonlinear problems [67].

Acknowledgment(s)

We would like to thank Huiyi Cao and Kamil Khan for their fruitful discussion on differentiable McCormick relaxations and EAGO functionality. We would also like to thank William Hale and Chenyu Wang for their preliminary feedback.

Funding

This material is based upon work supported by the National Science Foundation under Grant No. 1932723, as well as the University of Connecticut. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] B.M. Adams, W. Bohnhoff, K. Dalbey, J. Eddy, M. Eldred, D. Gay, K. Haskell, P.D. Hough, and L.P. Swiler, *DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 5.0 user's manual*, Sandia National Laboratories, Tech. Rep. SAND2010-2183 (2009).

[2] O. Andrzej and K. Stanislaw, *Evolutionary Algorithms for Global Optimization*, in *Global Optimization: Scientific and Engineering Case Studies*, chap. 12, Springer US, Boston, MA, 2006, pp. 267–300, Available at https://doi.org/10.1007/0-387-30927-6_12.

[3] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon, *Automatic differentiation of algorithms*, Journal of Computational and Applied Mathematics 124 (2000), pp. 171 – 190, Available at https://doi.org/10.1016/S0377-0427(00)00422-2.

[4] P.I. Barton and C. Pantelides, *gPROMS-a combined discrete/continuous modelling environment for chemical processing systems*, Simulation Series 25 (1993), pp. 25–25.

[5] P.I. Barton and C.C. Pantelides, *Modeling of combined discrete/continuous processes*, AIChE journal 40 (1994), pp. 966–979, Available at https://doi.org/10.1002/aic.690400608.

[6] P. Belotti, *CoUEnnE: a user's manual*, Tech. Rep., Technical report, Lehigh University, 2009.

[7] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter, *Branching and bounds tightening techniques for non-convex MINLP*, Optimization Methods & Software 24 (2009), pp. 597–634.

[8] L. Benet and D. Sanders, *ValidatedNumerics.jl* (2019), Available at https://www.github.com/JuliaIntervals/ValidatedNumerics.jl.

[9] B.W. Bequette, *Process control: modeling, design, and simulation*, Prentice Hall Professional, Upper Saddle River, NJ, USA, 2003.

[10] M. Berz and G. Hoffstätter, *Computation and application of Taylor polynomials with interval remainder bounds*, Reliable Computing 4 (1998), pp. 83–97, Available at https://doi.org/10.1023/A:1009958918582.

[11] J. Bezanson, A. Edelman, S. Karpinski, and V.B. Shah, *Julia: A fresh approach to numerical computing*, SIAM review 59 (2017), pp. 65–98.

[12] A. Bompadre and A. Mitsos, *Convergence rate of McCormick relaxations*, Journal of Global Optimization 52 (2012), pp. 1–28, Available at http://dx.doi.org/10.1007/s10898-011-9685-2.

[13] D. Bongartz and A. Mitsos, *Deterministic global optimization of process flowsheets in a reduced space using McCormick relaxations*, Journal of Global Optimization 69 (2017), pp. 761–796, Available at https://doi.org/10.1007/s10898-017-0547-4.

[14] D. Bongartz, J. Najman, S. Sass, and A. Mitsos, *MAiNGO: McCormick based algorithm for mixed integer nonlinear global optimization*, Tech. Rep., RWTH Aachen, 2018.

[15] H. Cao, Y. Song, and K.A. Khan, *Convergence of subtangent-based relaxations of nonlinear programs*, Processes 7 (2019), p. 221, Available at https://doi.org/10.3390/pr7040221.

[16] T. Coleman, M.A. Branch, and A. Grace, *Optimization toolbox*, For Use with MATLAB. User's Guide for MATLAB 5, Version 2, Relaese II (1999).

[17] I.I. CPLEX, *IBM ILOG CPLEX Optimizer*, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ (2020).

[18] L.H. De Figueiredo and J. Stolfi, *Affine arithmetic: concepts and applications*, Numerical Algorithms 37 (2004), pp. 147–158, Available at https://doi.org/10.1023/B:NUMA.0000049462.70970.b6.

[19] E.D. Dolan and J.J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical programming 91 (2002), pp. 201–213, Available at https://doi.org/10.1007/s101070100263.

[20] F. Domes and A. Neumaier, *Constraint propagation on quadratic constraints*, Constraints 15 (2010), pp. 404–429, Available at https://doi.org/10.1007/s10601-009-9076-1.

[21] I. Dunning, J. Huchette, and M. Lubin, *JuMP: A modeling language for mathematical optimization*, SIAM Review 59 (2017), pp. 295–320, Available at https://doi.org/10.1137/15M1020575.

[22] C.A. Floudas and O. Stein, *The adaptive convexification algorithm: A feasible point method for semi-infinite programming*, SIAM Journal on Optimization 18 (2008), pp. 1187–1208, Available at http://dx.doi.org/10.1137/060657741.

[23] R. Fourer, D.M. Gay, and B.W. Kernighan, *A modeling language for mathematical programming*, Management Science 36 (1990), pp. 519–554, Available at https://doi.org/10.1287/mnsc.36.5.519.

[24] GAMS Development Corporation, Washington, DC, USA, *GAMS - A User's Guide, GAMS Release 24.2.1* (2013), Available at http://www.gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf.

[25] A.M. Gleixner, T. Berthold, B. Müller, and S. Weltge, *Three enhancements for optimization-based bound tightening*, Journal of Global Optimization 67 (2017), pp. 731–757, Available at https://doi.org/10.1007/s10898-016-0450-4.

[26] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*, 2nd ed., SIAM, Philadelphia, PA, 2008, Available at https://doi.org/10.1137/1.9780898717761.

[27] Gurobi Optimization, LLC, *Gurobi optimizer reference manual* (2020), Available at http://www.gurobi.com.

[28] W.E. Hart, C.D. Laird, J.P. Watson, D.L. Woodruff, G.A. Hackebeil, B.L. Nicholson, and J.D. Siirola, *Pyomo-optimization modeling in Python*, Vol. 67, 2nd ed., Springer, Switzerland, 2012, Available at https://doi.org/10.1007/978-3-319-58821-6.

[29] R. Horst and H. Tuy, *Global Optimization: Deterministic Approaches*, Springer Berlin Heidelberg, 2013, Available at https://books.google.com/books?id=Pe_1CAAAQBAJ.

[30] IEEE, *IEEE Standard for Interval Arithmetic*, IEEE Std 1788-2015 (2015), pp. 1–97.

[31] R. Kannan and P.I. Barton, *The cluster problem in constrained global optimization*, Journal of Global Optimization 69 (2017), pp. 629–676, Available at https://doi.org/10.1007/s10898-017-0531-z.

[32] A. Khajavirad and N.V. Sahinidis, *A hybrid LP/NLP paradigm for global optimization relaxations*, Mathematical Programming Computation 10 (2018), pp. 383–421, Available at https://doi.org/10.1007/s12532-018-0138-5.

[33] K.A. Khan, H.A.J. Watson, and P.I. Barton, *Differentiable McCormick relaxations*, Journal of Global Optimization 67 (2017), pp. 687–729, Available at http://dx.doi.org/10.1007/s10898-016-0440-6.

[34] K.A. Khan, M. Wilhelm, M.D. Stuber, H. Cao, H.A.J. Watson, and P.I. Barton, *Corrections to: Differentiable McCormick relaxations*, Journal of Global Optimization 70 (2018), pp. 705–706, Available at https://doi.org/10.1007/s10898-017-0601-2.

[35] O. Kröger, C. Coffrin, H. Hijazi, and H. Nagarajan, *Juniper: An Open-Source Nonlinear Branch-and-Bound Solver in Julia*, in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, Cham, 2018, pp. 377–386.

[36] G.P. McCormick, *Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems*, Mathematical Programming 10 (1976), pp. 147–175, Available at http://dx.doi.org/10.1007/BF01580665.

[37] R. Misener and C.A. Floudas, *ANTIGONE: Algorithms for continuous/integer global optimization of nonlinear equations*, Journal of Global Optimization 59 (2014), pp. 503–526, Available at https://doi.org/10.1007/s10898-014-0166-2.

[38] A. Mitsos, *Global optimization of semi-infinite programs via restriction of the right-hand side*, Optimization 60 (2011), pp. 1291–1308, Available at http://dx.doi.org/10.1080/02331934.2010.527970.

[39] A. Mitsos, B. Chachuat, and P.I. Barton, *McCormick-based relaxations of algorithms*, SIAM Journal on Optimization 20 (2009), pp. 573–601, Available at https://doi.org/10.1137/080717341.

[40] R. Moore, *Methods and Applications of Interval Analysis*, Studies in Applied and Numerical Mathematics, Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1979, Available at https://books.google.com/books?id=bavU6VlOG4EC.

[41] R.E. Moore, *Interval analysis*, Vol. 4, Prentice-Hall Englewood Cliffs, 1966.

[42] J. Najman and A. Mitsos, *Convergence analysis of multivariate McCormick relaxations*, Journal of Global Optimization 66 (2016), pp. 1–32, Available at https://doi.org/10.1007/s10898-016-0408-6.

[43] J. Najman and A. Mitsos, *On tightness and anchoring of McCormick and other relaxations*, Journal of Global Optimization (2017), Available at https://doi.org/10.1007/s10898-017-0598-6.

[44] J. Najman and A. Mitsos, *Tighter McCormick relaxations through subgradient propagation*, Journal of Global Optimization 75 (2019), pp. 565–593, Available at https://doi.org/10.1007/s10898-019-00791-0.

[45] J. Najman, D. Bongartz, A. Tsoukalas, and A. Mitsos, *Erratum to: Multivariate McCormick relaxations*, Journal of Global Optimization 68 (2017), pp. 219–225, Available at https://doi.org/10.1007/s10898-016-0470-0.

[46] A. Neumaier, *Interval methods for systems of equations*, Vol. 37, Cambridge university press, Cam-

bridge, UK, 1990.

[47] A. Neumaier, *Second-order sufficient optimality conditions for local and global nonlinear programming*, Journal of Global Optimization 9 (1996), pp. 141–151, Available at https://doi.org/10.1007/BF00121660.

[48] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó, *A comparison of complete global optimization solvers*, Mathematical programming 103 (2005), pp. 335–356, Available at https://doi.org/10.1007/s10107-005-0585-4.

[49] Process Systems Enterprise, *gPROMS*, Available at http://www.psenterprise.com/products/gproms.

[50] Y. Puranik and N.V. Sahinidis, *Domain reduction techniques for global NLP and MINLP optimization*, Constraints 22 (2017), pp. 338–376, Available at https://doi.org/10.1007/s10601-016-9267-5.

[51] T.K. Ralphs and L. Ladanyi, *Symphony: A parallel framework for branch and cut* (1999), Available at https://www.coin-or.org/SYMPHONY/intro-2.7/symphony.html.

[52] J. Revels, *Cassette.jl*, https://github.com/jrevels/Cassette.jl (2018).

[53] J. Revels, M. Lubin, and T. Papamarkou, *Forward-mode automatic differentiation in Julia*, arXiv preprint arXiv:1607.07892 (2016).

[54] H.S. Ryoo and N.V. Sahinidis, *Global optimization of nonconvex NLPs and MINLPs with applications in process design*, Computers & Chemical Engineering 19 (1995), pp. 551–566, Available at https://doi.org/10.1016/0098-1354(94)00097-2.

[55] N.V. Sahinidis, *BARON: A general purpose global optimization software package*, Journal of Global Optimization 8 (1996), pp. 201–205, Available at http://dx.doi.org/10.1007/BF00138693.

[56] J.K. Scott, M.D. Stuber, and P.I. Barton, *Generalized McCormick relaxations*, Journal of Global Optimization 51 (2011), pp. 569–606, Available at https://doi.org/10.1007/s10898-011-9664-7.

[57] X. Shen, S. Diamond, Y. Gu, and S. Boyd, *Disciplined convex-concave programming*, in *Decision and Control (CDC), 2016 IEEE 55th Conference on*, IEEE, 2016, pp. 1009–1014.

[58] A.B. Singer, *Global dynamic optimization*, Ph.D. diss., Massachusetts Institute of Technology, 2004, Available at http://hdl.handle.net/1721.1/28662.

[59] A.B. Singer, J.W. Taylor, P.I. Barton, and W.H. Green, *Global dynamic optimization for parameter estimation in chemical kinetics*, The Journal of Physical Chemistry A 110 (2006), pp. 971–976, Available at https://doi.org/10.1021/jp0548873.

[60] E.M. Smith and C.C. Pantelides, *Global optimisation of nonconvex MINLPs*, Computers & Chemical Engineering 21 (1997), pp. S791 – S796, Available at http://dx.doi.org/10.1016/S0098-1354(97)87599-0.

[61] M.D. Stuber, *Evaluation of process systems operating envelopes*, PhD dissertation, Massachusetts Institute of Technology, 2013, Available at http://hdl.handle.net/1721.1/79143.

[62] M.D. Stuber, *A differentiable model for optimizing hybridization of industrial process heat systems with concentrating solar thermal power*, Processes 6 (2018), p. 76, Available at https://doi.org/10.3390/pr6070076.

[63] M.D. Stuber and P.I. Barton, *Semi-infinite optimization with implicit functions*, Industrial & Engineering Chemistry Research 54 (2015), pp. 307–317, Available at http://dx.doi.org/10.1021/ie5029123.

[64] M.D. Stuber, J.K. Scott, and P.I. Barton, *Convex and concave relaxations of implicit functions*, Optimization Methods and Software 30 (2015), pp. 424–460, Available at https://doi.org/10.1080/10556788.2014.924514.

[65] M.D. Stuber, A. Wechsung, A. Sundaramoorthy, and P.I. Barton, *Worst-case design of subsea production facilities using semi-infinite programming*, AIChE Journal 60 (2014), pp. 2513–2524, Available at http://dx.doi.org/10.1002/aic.14447.

[66] M. Tawarmalani and N. Sahinidis, *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, Nonconvex Optimization and Its Applications, Springer US, 2002, Available at https://books.google.com/books?id=MjueCVdGZfoC.

[67] M. Tawarmalani and N.V. Sahinidis, *A polyhedral branch-and-cut approach to global optimization*, Mathematical Programming 103 (2005), pp. 225–249, Available at http://dx.doi.org/10.1007/s10107-005-0581-8.

[68] J.W. Taylor, G. Ehlker, H.H. Carstensen, L. Ruslen, R.W. Field, and W.H. Green, *Direct measurement of the fast, reversible addition of oxygen to cyclohexadienyl radicals in nonpolar solvents*, The Journal of Physical Chemistry A 108 (2004), pp. 7193–7203, Available at https://doi.org/10.1021/jp0379547.

[69] J.W. Tester, M. Modell, *et al.*, *Thermodynamics and its Applications*, Prentice Hall PTR, Upper Saddle River, NJ, 1997.

[70] A. Tsoukalas and A. Mitsos, *Multivariate McCormick relaxations*, Journal of Global Optimization 59 (2014), pp. 633–662, Available at https://doi.org/10.1007/s10898-014-0176-0.

[71] S. Vigerske, *Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming*, Ph.D. diss., Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2013.

[72] A. Wächter and L.T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical Programming 106 (2006), pp. 25–57, Available at https://doi.org/10.1007/s10107-004-0559-y.

[73] A. Wechsung, J.K. Scott, H.A. Watson, and P.I. Barton, *Reverse propagation of McCormick relaxations*, Journal of Global Optimization 63 (2015), pp. 1–36, Available at https://doi.org/10.1007/s10898-015-0303-6.

[74] M.E. Wilhelm, A.V. Le, and M.D. Stuber, *Global optimization of stiff dynamical systems*, AIChE Journal 65 (2019), p. e16836, Available at https://doi.org/10.1002/aic.16836.

Benchmarking Problem Set Summary				
Name	Variables	Inequalities	Equalities	Nonlinear Terms
alkyl	15	0	7	$\times$, $(\cdot)^2$
BeckerLago	2	0	0	$(\cdot)^2$ $\sqrt{(\cdot)}$
ex2_1_8	24	0	10	$\times$
ex3_1_1	8	6	0	$\times$
ex4_1_9	2	2	0	$(\cdot)^2$, $(\cdot)^4$
ex5_4_3	16	13	0	$\times$, $(\cdot)/(\cdot)$, $(\cdot)^a$
ex6_2_10	6	0	3	$\times$, log, $(\cdot)/(\cdot)$
ex6_2_11	3	0	1	$\times$, log, $(\cdot)/(\cdot)$
ex6_2_13	6	0	3	$\times$, log, $(\cdot)/(\cdot)$
ex6_2_14	4	0	2	$\times$, log, $(\cdot)/(\cdot)$
ex7_2_1	7	14	0	$\times$, $(\cdot)/(\cdot)$, $(\cdot)^2$
ex7_2_3	8	6	0	$\times$, $(\cdot)/(\cdot)$
ex7_2_4	8	0	7	$\times$, $(\cdot)/(\cdot)$, $(\cdot)^a$
ex8_4_1	22	0	10	$(\cdot)^2$
ex8_4_2	24	0	10	$(\cdot)^2$
gold	2	0	0	$\times$, $(\cdot)^2$
hart6	6	0	0	exp$(\cdot)$, $\times$, $(\cdot)^2$
meanvar	8	0	2	$\times$
Model13	6	0	0	exp$(\cdot)$, $\times$, $(\cdot)^2$
process	10	0	7	$\times$, $(\cdot)/(\cdot)$, $(\cdot)^2$

Table 1. The selected benchmarking problems are summarized by their scale and complexity in terms of the number of variables, inequality and equality constraints, and types of nonlinear terms.

Supplementary Material for "EAGO.jl: Easy Advanced Global Optimization in Julia"

M. E. Wilhelm and M. D. Stuber*

Process Systems and Operations Research Laboratory, Dept. of Chemical and Biomolecular Engineering, University of Connecticut, 191 Auditorium Rd, Unit 3222, Storrs, CT, 06269-3222, USA

S1. Summary of McCormick Relaxation Theory

DEFINITION S1.1 (Convex and Concave Relaxations [6]) Given a convex set $Z \subset \mathbb{R}^n$ and a function $w : Z \to \mathbb{R}$, a convex function $w^{cv} : Z \to \mathbb{R}$ is a *convex relaxation* of w on Z if $w^{cv}(\mathbf{z}) \leq w(\mathbf{z})$ for every $\mathbf{z} \in Z$. A concave function $w^{cc} : Z \to \mathbb{R}$ is a *concave relaxation* of w on Z if $w^{cc}(\mathbf{z}) \geq w(\mathbf{z})$ for every $\mathbf{z} \in Z$.

The convex and concave relaxations of vector-valued and matrix-valued functions are defined by the componentwise application of the above inequalities.

DEFINITION S1.2 (Univariate Intrinsic Function [7]) The function $u : B \subset \mathbb{R} \to \mathbb{R}$ is a *univariate intrinsic function* if, for any $A \in \mathbb{I}B$, where $\mathbb{I}B = \{X \in \mathbb{IR} : X \subset B\}$, the following are known and can be evaluated computationally:

(1) an interval extension of u on A that is an inclusion function of u on A,
(2) a concave relaxation of u on A,
(3) a convex relaxation of u on A.

DEFINITION S1.3 (Factorable Function [7]) A function $w : Z \subset \mathbb{R}^n \to \mathbb{R}$ is *factorable* if it can be expressed in terms of a finite number of factors $v_1, \ldots, v_m$, such that given $\mathbf{z} \in Z$, $v_i = z_i$ for $i = 1, \ldots, n_z$, and v_k is defined for $n \leq k \leq m$ as either

(1) $v_k = v_i + v_j$, with, $i, j < k$,
(2) $v_k = v_i v_j$, with, $i, j < k$,
(3) $v_k = u_k(v_i)$, with, $i < k$, where $u_k : B_k \to R$ is a univariate intrinsic function

and $w(\mathbf{z}) = v_m(\mathbf{z})$. A vector-valued function is factorable if each of its components are factorable functions.

THEOREM S1.4 (Univariate McCormick Composition Rule [6]) *Let $Z \subset \mathbb{R}^n$ and $X \subset \mathbb{R}$ be nonempty convex sets. Consider the composite function $w = \phi \circ q$ where $w : Z \to \mathbb{R}$ is continuous, $\phi : X \to \mathbb{R}$, let $q(Z) \subset X$. Let $q^{cv} : Z \to \mathbb{R}$ and $q^{cc} : Z \to \mathbb{R}$ be convex and concave relaxations of q on Z, respectively. Let $\phi^{cv} : X \to \mathbb{R}$ and $\phi^{cc} : X \to \mathbb{R}$ be convex and concave relaxations of ϕ on X, respectively. Let $\xi^*_{min} \in X$ be a point at which ϕ^{cv} attains its infimum on X and let $\xi^*_{max} \in X$ be a point at which ϕ^{cc} attains its supremum*

*Corresponding author. Email: stuber@alum.mit.edu

on X. Then the convex and concave relaxations are, respectively, given by

$$w^{cv} : Z \to \mathbb{R} : \mathbf{z} \mapsto \phi^{cv}(\mathrm{mid}(q^{cv}(\mathbf{z}), q^{cc}(\mathbf{z}), \xi^*_{\min})) \tag{1}$$
$$w^{cc} : Z \to \mathbb{R} : \mathbf{z} \mapsto \phi^{cc}(\mathrm{mid}(q^{cv}(\mathbf{z}), q^{cc}(\mathbf{z}), \xi^*_{\max})). \tag{2}$$

In the above, the $\mathrm{mid}(\cdot,\cdot,\cdot)$ function takes the median value of its three arguments. Generally, the application of Theorem S1.4 requires the use of closed-form expressions to determine the values of $\xi^*_{\max}$ and $\xi^*_{\min}$ in conjunction with defined forms of the relaxations ϕ^{cv} and ϕ^{cc}.

PROPOSITION S1.5 (McCormick Addition Rule [6]) *Let $Z \subset \mathbb{R}^n$, and $w, q, r : Z \to \mathbb{R}$ such that $w(\mathbf{z}) = q(\mathbf{z}) + r(\mathbf{z})$. Let $q^{cv}, q^{cc} : Z \to \mathbb{R}$ be convex and concave relaxations of q on Z, respectively. Similarly, let $r^{cv}, r^{cc} : Z \to \mathbb{R}$ be convex and concave relaxations of r on Z, respectively. Then, $w^{cv}, w^{cc} : Z \to \mathbb{R}$, such that*

$$w^{cv}(\mathbf{z}) = q^{cv}(\mathbf{z}) + r^{cv}(\mathbf{z}), \qquad w^{cc}(\mathbf{z}) = q^{cc}(\mathbf{z}) + r^{cc}(\mathbf{z}), \tag{3}$$

are, respectively, a convex and concave relaxation of w on Z.

PROPOSITION S1.6 (McCormick Multiplication Rule [6]) *Let $Z \subset \mathbb{R}^n$ be a nonempty convex set. Let $w, q, r : Z \to \mathbb{R}$ such that $w(\mathbf{z}) = q(\mathbf{z})r(\mathbf{z})$. Let $q^{cv} : Z \to \mathbb{R}$ and $q^{cc} : Z \to \mathbb{R}$ be convex and concave relaxations of q on Z, respectively. Let $r^{cv} : X \to \mathbb{R}$ and $r^{cc} : X \to \mathbb{R}$ be convex and concave relaxations of r on Z, respectively. Further, let q^L, q^U, r^L, r^U be bounds on q, r such that*

$$q^L \le q(\mathbf{z}) \le q^U \qquad \text{and} \qquad r^L \le r(\mathbf{z}) \le r^U, \qquad \forall \mathbf{z} \in Z.$$

Let the following intermediate function $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2, \delta_1, \delta_2 : Z \to \mathbb{R}$ be defined as

$$\begin{aligned}
\alpha_1(\mathbf{z}) &= \min\{r^L q^{cv}(\mathbf{z}), r^L q^{cc}(\mathbf{z})\}, & \alpha_2(\mathbf{z}) &= \min\{q^L r^{cv}(\mathbf{z}), q^L r^{cc}(\mathbf{z})\},\\
\beta_1(\mathbf{z}) &= \min\{r^U q^{cv}(\mathbf{z}), r^U q^{cc}(\mathbf{z})\}, & \beta_2(\mathbf{z}) &= \min\{q^U r^{cv}(\mathbf{z}), q^U r^{cc}(\mathbf{z})\},\\
\gamma_1(\mathbf{z}) &= \max\{r^L q^{cv}(\mathbf{z}), r^L q^{cc}(\mathbf{z})\}, & \gamma_2(\mathbf{z}) &= \max\{q^U r^{cv}(\mathbf{z}), q^U r^{cc}(\mathbf{z})\},\\
\delta_1(\mathbf{z}) &= \max\{r^U q^{cv}(\mathbf{z}), r^U q^{cc}(\mathbf{z})\}, & \delta_2(\mathbf{z}) &= \max\{q^L r^{cv}(\mathbf{z}), q^L r^{cc}(\mathbf{z})\}.
\end{aligned}$$

Then, the convex and concave relaxations of r on Z are given by

$$\begin{aligned}
w^{cv} &: Z \to \mathbb{R} : \mathbf{z} \mapsto \max\{\alpha_1(\mathbf{z}) + \alpha_2(\mathbf{z}) - q^L r^L, \beta_1(\mathbf{z}) + \beta_2(\mathbf{z}) - q^U r^U\}\\
w^{cc} &: Z \to \mathbb{R} : \mathbf{z} \mapsto \min\{\gamma_1(\mathbf{z}) + \gamma_2(\mathbf{z}) - q^U r^L, \delta_1(\mathbf{z}) + \delta_2(\mathbf{z}) - q^L r^U\},
\end{aligned}$$

respectively.

DEFINITION S1.7 (Subgradients [8]) Let $Z \subset R^n$ be a nonempty convex set, $w^{cv} : Z \to \mathbb{R}$ be convex and $w^{cc} : Z \to \mathbb{R}$ be concave. A function $\mathbf{s}^{cv}_w : Z \to \mathbb{R}^n$ is a *subgradient* of w^{cv} on Z if for each $\bar{\mathbf{z}} \in Z, w^{cv}(\mathbf{z}) \ge w^{cv}(\bar{\mathbf{z}}) + \mathbf{s}^{cv}_w(\bar{\mathbf{z}})^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}}), \forall \mathbf{z} \in Z$. Similarly, a function $\mathbf{s}^{cc}_w : Z \to \mathbb{R}^n$ is a subgradient of w^{cc} on Z if for each $\bar{\mathbf{z}} \in Z, w^{cc}(\mathbf{z}) \le w^{cc}(\bar{\mathbf{z}}) + \mathbf{s}^{cc}_w(\bar{\mathbf{z}})^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}}), \forall \mathbf{z} \in Z$.

Note that subgradients of vector-valued functions and subgradients of matrix-valued functions will be defined analogously and will be matrix-valued functions and third-order tensor-valued functions, respectively.

S2. Application to EAGO to Exotic Optimization Formulations

S2.1 *Application to Trigonometric Functions*

As a natural extension of supporting user-defined functions, EAGO also supports a wide variety of trigonometric, hyperbolic, and other transcendental functions. This contrasts the much more limited capabilities of other state-of-the-art global optimizers, such as BARON [9], ANTIGONE [5], and SCIP [10]. As a result, EAGO can address problem formulations such as the power scheduling problem presented below in (4). This example originates from a model of two electrical generators connected to a network with three nodes. The constraints are chosen to balance the power flowing into and out of each node. In [3], a local minimum was located. Bounds on the decision variables are given in Table S1. A preliminary version of EAGO was used in [2] to solve this problem to global optimality. Using the first full release of EAGO, this problem can be solved to global optimality in approximately 29 seconds.

$$
\begin{aligned}
&\min_{\mathbf{y}\in Y} 3000y_1 + 1000y_1^3 + 2000y_2 + 666.667y_2^3 \qquad (4)\\
\text{s.t. }\; & 0.4 - y_1 + 2Cy_5^2 + y_5y_6(D\sin(-y_8) - C\cos(-y_8))\ldots\\
&\qquad\qquad \ldots + y_5y_7(D\sin(-y_9) - C\cos(-y_9)) = 0\\
& 0.4 - y_2 + 2Cy_6^2 + y_5y_6(D\sin(y_8) - C\cos(y_8))\ldots\\
&\qquad\qquad \ldots + y_5y_7(D\sin(y_8 - y_9) - C\cos(y_8 - y_9)) = 0\\
& 0.2 - y_3 + 2Dy_5^2 - y_5y_6(C\sin(-y_8) + D\cos(-y_8))\ldots\\
&\qquad\qquad \ldots - y_5y_7(C\sin(-y_9) + D\cos(-y_9)) = 0\\
& 0.2 - y_4 + 2Dy_6^2 - y_5y_6(C\sin(y_8) + D\cos(y_8))\ldots\\
&\qquad\qquad \ldots - y_5y_7(C\sin(y_8 - y_9) + D\cos(y_8 - y_9)) = 0\\
& 0.8 + 2Cy_7^2 + y_5y_7(D\sin(y_9) - C\cos(y_9))\ldots\\
&\qquad\qquad \ldots + y_6y_7(D\sin(y_9 - y_8) - C\cos(y_9 - y_8)) = 0\\
& -0.337 + 2Dy_7^2 - y_5y_7(C\sin(y_9) + D\cos(y_9))\ldots\\
&\qquad\qquad \ldots - y_6y_7(C\sin(y_9 - y_8) + D\cos(y_9 - y_8)) = 0\\
& C = (48.4/50.176)\sin(0.25)\\
& D = (48.4/50.176)\cos(0.25)\\
& Y = [\mathbf{y}^L, \mathbf{y}^U]
\end{aligned}
$$

Table S1. Interval box constraints Y on the decision variables in (4).

Component	Lower Bound	Upper Bound
y_1	0.5	1.2
y_2	0.5	1.2
y_3	0	0.3
y_4	0	0.3
y_5	0.90909	1.0909
y_6	0.90909	1.0909
y_7	0.90909	1.0909
y_8	-0.5	0.3
y_9	-0.5	0.3

S2.2 *Customizing EAGO's Solver: A Quasiconvex Problem*

In addition to supporting a wide variety of problem forms, EAGO can be readily adapted to suit the unique needs of users and application requirements. In this example, we'll modify EAGO's branch-and-bound solver to solve a quasiconvex problem using a bisection technique. The problem consists of minimizing a quasiconvex function $f : C \to \mathbb{R}$ over a convex feasible set. This can be done using a few simple lines of code. Consider the quasiconvex problem presented in [4]:

$$
\begin{aligned}
f^* = \quad & \min_{\mathbf{y}\in Y} f(\mathbf{y}) && (5)\\
\text{s.t.} \quad & \sum_{i=1}^{5} i \cdot y_i - 5 = 0 && (6)\\
& \sum_{i=1}^{5} y_i^2 - 0.5\pi \leq 0 && (7)\\
& -\left(\frac{1}{2}y_1^2 + \frac{1}{2}y_2^2 + y_3^2 + 2y_1y_2 + 4y_1y_3 + 2y_2y_3\right) \leq 0 && (8)\\
& -y_1^2 - 6y_1y_2 - 2y_2^2 + \cos(y_1) + \pi \leq 0 && (9)\\
& Y = [0,5]^5
\end{aligned}
$$

where

$$
f(\mathbf{y}) = -\frac{\ln((5+y_1)^2 + \sum_{i=1}^{5} y_i)}{1 + \sum_{i=1}^{5} y_i^2}. \tag{10}
$$

Interval analysis shows that $f^* \in F = [f^L, f^U] = [-5, 0]$. As such, we can introduce a new auxiliary variable $t \in T = F$ and formulate the equivalent problem below:

$$
\begin{aligned}
t^* = \quad & \min_{\mathbf{y}\in Y, t\in T} t\\
\text{s.t.} \quad & (6)\text{-}(9)\\
& f(\mathbf{y}) - t \leq 0\\
& Y = [0,5]^2, \;\; T = [-5,0].
\end{aligned}
$$

In order to solve this problem, we resort to a bisection algorithm with respect to the interval T. Let $\phi_\tau(\mathbf{y}) = f(\mathbf{y}) - \tau$ such that $\tau = (t^L + t^U)/2$, then we solve for $\mathbf{y}$ subject to constraints (6)-(9) such that $\phi_\tau(\mathbf{y}) \leq 0$. If this problem is feasible, $t^* \in [t^L, \tau]$. Otherwise, $t^* \in [\tau, t^U]$. The other interval is then discarded (fathomed) and this manner of bisection is repeated until an interval containing a feasible solution with a width of at most ϵ is located [1].

Then, we simply need to bisect solely in the t dimension. To implement this new solver using EAGO, three main modifications to the optimizer must be made. First, we'll short circuit the preprocessing, postprocessing, and upper-bounding steps.

Listing S1 Short circuit preprocessing, postprocessing, and upper-bounding problem for example in Section S2.2.

```
using MathOptInterface, EAGO, JuMP
import EAGO: Optimizer

struct QuasiConvex <: EAGO.ExtensionType end

import EAGO: preprocess!, upper_problem!, postprocess!
function EAGO.preprocess!(t::QuasiConvex, x::Optimizer)
    x._preprocess_feasibility = true
end
function EAGO.upper_problem!(t::QuasiConvex, x::Optimizer)
    x._upper_feasibility = true
end
function EAGO.postprocess!(t::QuasiConvex, x::Optimizer)
    x._postprocess_feasibility = true
end
```

Next, one specifies that the algorithm should terminate on convergence to an ϵ-optimal point and return status codes should indicate a feasible optimal solution was returned.

Listing S2 Modify convergence and termination criteria for example in Section S2.2.

```
import EAGO: convergence_check, termination_check, repeat_check
function EAGO.convergence_check(t::QuasiConvex, x::Optimizer)
    gap = (x._upper_objective_value - x._lower_objective_value)
    return (gap <= x._parameters.absolute_tolerance)
end
function EAGO.termination_check(t::QuasiConvex, x::Optimizer)
    flag = EAGO.convergence_check(t, x)
    if flag
        x._termination_status_code = MathOptInterface.OPTIMAL
        x._result_status_code = MathOptInterface.FEASIBLE_POINT
    end
    return flag
end
```

It is further specified that bisection should only occur in the t dimension and only the feasible node need be saved.

Listing S3 Specify dimensions for bisection for example in Section S2.2.

```
branch_variable = [i == 6 for i=1:6]
EAGO.repeat_check(t::QuasiConvex, x::Optimizer) = true
```

We then specify how the subproblem should be solved at each iteration. Here, the problem is solved at the midpoint value of t using the native EAGO function `upper_problem!`. Bounds are then contracted depending on the feasibility of the subproblem.

Listing S4 Modify lower-bounding problem to solve new subproblem for example in Section S2.2.

```
import EAGO: lower_problem!
function EAGO.lower_problem!(t::QuasiConvex, x::Optimizer)
  y = x._current_node
```

```
  lower = y.lower_variable_bounds[6]
  upper = y.upper_variable_bounds[6]
  midy = (lower + upper)/2.0
  y.lower_variable_bounds[6] = midy
  y.upper_variable_bounds[6] = midy
  EAGO.solve_local_nlp!(x)
  feas = x._upper_feasibility
  y.lower_variable_bounds[6] = feas ?  lower : midy
  y.upper_variable_bounds[6] = feas ?  midy : upper
  x._lower_objective_value = y.lower_variable_bounds[6]
  x._upper_objective_value = y.upper_variable_bounds[6]
  x._lower_feasibility = true
  return
end
```

We can now define the problem using a JuMP syntax and retrieve the solution. The two keyword options which don't reference previously-defined functions specify an absolute tolerance of $\epsilon_a = 10^{-8}$ should be used and the routine should not terminate when a feasible point is located (even though no objective is specified).

Listing S5 Construct and optimize the JuMP model for example in Section S2.2

```
m = Model(optimizer_with_attributes(EAGO.Optimizer,
          "absolute_tolerance" => 1E-8,
          "branch_variable" => branch_variable,
          "ext_type" => QuasiConvex()))

@variable(m, ((i<6) ? 0 : -5) <= y[i=1:6] <= ((i<6) ? 5 : 0))
@constraint(m, sum(i*y[i] for i=1:5) - 5 == 0)
@constraint(m, sum(y[i]^2 for i=1:5) - 0.5*pi^2 <= 0)
@expression(m, expr1, 2*y[1]*y[2] + 4*y[1]*y[3] + 2*y[2]*y[3])
@constraint(m, -(0.5*y[1]^2 + 0.5*y[2]^2 + y[3]^2 + expr1) <= 0)
@NLexpression(m, expr2, log((5 + y[1])^2 + sum(y[i] for i=1:5)))
@NLconstraint(m, -y[1]^2 -6*y[1]*y[2] -2*y[2]^2 +cos(y[1]) + pi <= 0)
@NLconstraint(m, -expr2/(1 + sum(y[i]^2 for i=1:5)) - y[6] <= 0)
@objective(m, Min, y[6])
JuMP.optimize!(m)

# retrieve solution info
solution = JuMP.value.(y[1:5])
global_obj_value = JuMP.value.(y[6])
```

This problem can then be solved to an absolute tolerance of $\epsilon_a = 10^{-8}$ in just 0.7 seconds.

References

[1] S. Boyd and L. Vandenberghe, *Convex optimization*, Cambridge University Press, 2004.

[2] H. Cao, Y. Song, and K.A. Khan, *Convergence of subtangent-based relaxations of nonlinear programs*, Processes 7 (2019), p. 221.

[3] W. Hock and K. Schittkowski, *Test examples for nonlinear programming codes*, Journal of Optimization Theory and Applications 30 (1980), pp. 127–129.

[4] C. Jansson, *Quasiconvex relaxations based on interval arithmetic*, Linear Algebra and its Applications 324 (2001), pp. 27–53.

[5] R. Misener and C.A. Floudas, *ANTIGONE: Algorithms for continuous/integer global optimization of nonlinear equations*, Journal of Global Optimization 59 (2014), pp. 503–526.

[6] A. Mitsos, B. Chachuat, and P.I. Barton, *McCormick-based relaxations of algorithms*, SIAM Journal on Optimization 20 (2009), pp. 573–601.

[7] J.K. Scott, M.D. Stuber, and P.I. Barton, *Generalized McCormick relaxations*, Journal of Global Optimization 51 (2011), pp. 569–606.

[8] M.D. Stuber, J.K. Scott, and P.I. Barton, *Convex and concave relaxations of implicit functions*, Optimization Methods and Software 30 (2015), pp. 424–460.

[9] M. Tawarmalani and N.V. Sahinidis, *A polyhedral branch-and-cut approach to global optimization*, Mathematical Programming 103 (2005), pp. 225–249, Available at http://dx.doi.org/10.1007/s10107-005-0581-8.

[10] S. Vigerske and A. Gleixner, *SCIP: global optimization of mixed-integer nonlinear programs in a branch-and-cut framework*, Optimization Methods and Software 33 (2018), pp. 563–593, Available at https://doi.org/10.1080/10556788.2017.1335312.

Table S2. The solution times (CPU seconds) of the benchmarking problems are reported for each of the solvers in the comparison study. The relative standard error (RSE) of the three trials was less than 5% for all instance with total run time less than 0.5 seconds and less than 2% in all other instances.

Mean Solution Time (CPU Seconds)				
Name	EAGO	SCIP	BARON	ANTIGONE
alkyl	0.07	0.742	0.296	0.16
BeckerLago	0.14	0.34	0.08	60.6
ex2_1_8	0.12	0.632	0.671	0.324
ex3_1_1	0.476	0.758	0.422	0.459
ex4_1_9	0.168	0.29	0.06	0.148
ex5_4_3	0.056	0.26	0.15	0.143
ex6_2_10	>1000	>1000	67.3	189
ex6_2_11	21.9	>1000	9	5.228
ex6_2_13	>1000	>1000	95.5	>1000
ex6_2_14	2.07	>1000	0.8	0.283
ex7_2_1	0.22	>1000	>1000	0.424
ex7_2_3	509	>1000	>1000	>1000
ex7_2_4	37.0	5.37	>1000	3.57
ex8_4_1	>1000	214.4	0.4	0.76
ex8_4_2	>1000	>1000	>1000	>1000
gold	0.69	4.03	1.52	316.17
hart6	5.21	2.01	0.08	0.338
meanvar	0.03	1.253	0.532	0.863
Model13	0.13	50.37	0.08	6.701
process	0.61	0.66	0.62	0.34

Table S3. The relative gap remaining for each solver and benchmarking problem pair that did not converge to the desired tolerance within 1000 CPU seconds.

Relative Gap at Termination				
Name	EAGO	SCIP	BARON	ANTIGONE
ex6_2_10	0.031	949.96	-	-
ex6_2_11	-	4.01E7	-	-
ex6_2_13	0.185	1783	-	0.2284
ex6_2_14	-	0.47	-	-
ex7_2_1	-	3.83	0.013	-
ex7_2_3	-	234.42	0.698	0.114
ex8_4_1	0.404	-	-	-
ex8_4_2	0.967	inf	0.7812	0.496

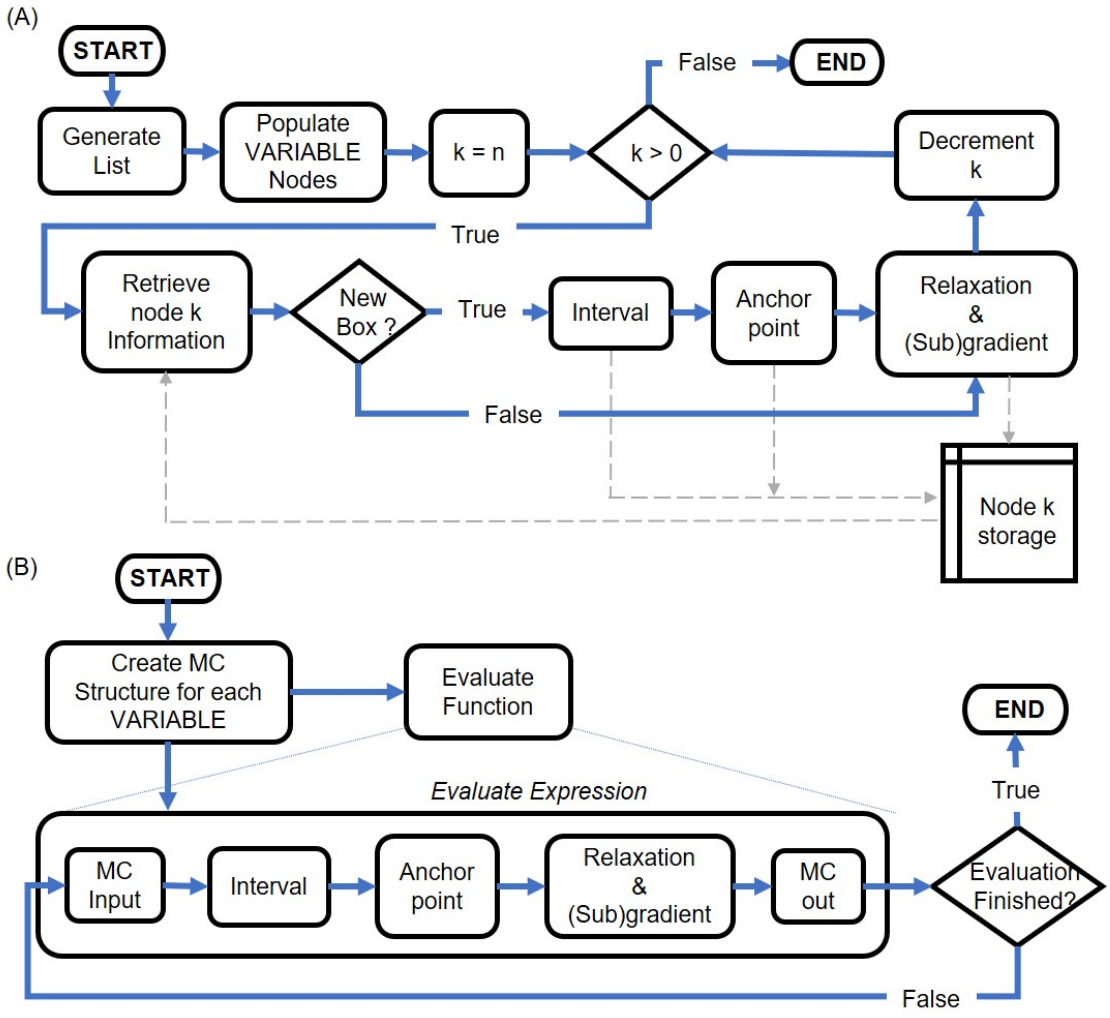

Figure S1. Two flowcharts are provided to contrast (A) EAGO's unique source code transformation-based computation of McCormick relaxations using a Wengert list (of length n) and (B) a standard operator/method overloading implementation approach to generating McCormick relaxations.