Solving MINLPs to global optimality with FICO® Xpress Global

Pietro Belotti[*] Timo Berthold[†‡] Tristan Gally[§]
Leona Gottwald[¶] Imre Pólik[‖]

July 18, 2025

Abstract

We present the architecture and central parts of the FICO Xpress Global optimization solver. In particular, we focus on how we built a global solver for the general class of mixed-integer nonlinear optimization problems by combining and extending two existing components of the FICO Xpress Solver, namely the mixed-integer linear optimization solver and the successive linear approximation-based local solver for nonlinear optimization. We give a detailed report on the performance impact of each component: presolve, cutting, branching, heuristics, and parallelization.

1 Introduction

Mathematical optimization is a field of Applied Mathematics that arguably has one of the largest direct business impacts. While in the second half of the 20th century Linear Programming (LP) became more and more widely used in industry, the time around the turn of the millennium saw mixed-integer linear programming become a technique to be used out of the box. In the past twenty years, nonlinear programming approaches have become more and more reliable, and today we might witness the beginning of an era of the practical usability of mixed-integer nonlinear programs (MINLPs).

In this paper, we consider MINLPs of the form

$$\begin{aligned} \min\ & f(x) \\ \text{s.t. } g_k(x) &\le 0, \quad \forall k = 1, \dots, m \\ x_i &\in \mathbb{Z}, \quad \forall i \in I, \end{aligned} \tag{1}$$

[*]DEIB, Politecnico di Milano, Italy

[†]Fair Isaac Deutschland GmbH, Germany

[‡]Technische Universität Berlin, Institut für Mathematik, Germany

[§]Fair Isaac Services Limited, United Kingdom

[¶]Hexaly, France. The contributions of this author were completed during her employment at Fair Isaac Deutschland GmbH.

[‖]Fair Isaac Corporation, USA, Corresponding author. `imre@polik.net`

where f is a nonlinear objective function, g_k are nonlinear constraint functions for $k = 1, 2, \ldots, m$, and $I \subseteq \{1, 2, \ldots, n\}$ is the set of integer variables[1]. We assume the functions f and g_k to be Lipschitz-continuous but not necessarily differentiable everywhere, and possibly nonconvex. We also assume f and all g_k to be *factorable*, i.e., each function can be formed by recursively combining variables and constants with operators from a finite set $\mathcal{O} = \{+, -, *, /, a^b, \sin, \cos, \log, \exp, \min, \max, \ldots\}$. In particular, functions can be nested inside each other to an arbitrary depth. As such, this framework in particular encompasses non-convex quadratically-constrained quadratic problems, which are included in the offering of the FICO® Xpress Optimizer[2], but also more general mixed-integer nonlinear problems including trigonometrics, arbitrary powers, or other nonlinearities.

The problem class (1) is among the most difficult in optimization. The difficulty stems both from the integrality of a subset of variables and the nonlinearity of the objective function and the constraints, thus generalizing two already tough classes of problems: mixed-integer linear optimization (MILP) and continuous nonlinear optimization (NLP) [34].

This article describes a solver for the MINLP class that extends the structure of an MILP solver of the branch-and-cut type, combining it with a local nonlinear optimization solver for heuristic solves and new techniques such as convexification cuts and spatial branching to provide a globally optimal solution to any MINLP.

Being able to solve Problem (1) to global optimality is important for several reasons:

- A local MINLP solver does not provide a dual bound or duality gap as a measure for the quality of the solution it found.
- A global solver can be used to validate the quality of the solutions found by a local solver or a quick heuristic.
- Only a global MINLP solver is able to certify the infeasibility of Problem (1), whereas a local (MI)NLP solver can only acknowledge that it was unable to find a feasible solution in a neighbourhood of the initial point without any guarantees about the existence of feasible solutions elsewhere, in particular in disconnected regions of the domain. We took advantage of this new capability by extending the infeasibility analysis tools in FICO Xpress to MINLPs. Most notably, FICO Xpress can find an Irreducible Infeasible Subsystem for Problem (1).

In addition to the FICO Xpress Global Solver, released in October 2022 and described in this paper, other software implementations of global optimization solvers exist. We list some of them here in chronological order of release: the

[1]FICO Xpress also provides special constructs for indicators, which can now be applied to nonlinear constraints, and specially ordered set (SOS) constraints.

[2]FICO is a registered trademark of Fair Isaac Corporation.

two commercial products BARON[3] [33], which implements spatial branch-and-bound on continuous variables and uses external MILP and local NLP solvers, and Lindo [23]; then the open-source software packages Couenne [5], SCIP [10], and shortly afterward ANTIGONE [30]. More recent commercial software packages include Octeract[4] [32], released in 2019, Hexaly[5], and Gurobi[6] [18], which was extended to mixed-integer nonlinear programs in December 2023. It is also worth mentioning other solver efforts, such as Bonmin [11], Minotaur [27], and SHOT [26], which target *convex* MINLPs, i.e., MINLPs whose continuous relaxation is convex.

The novelty of the Xpress Global Solver is the tight integration with both an MILP and a local NLP solver, both pre-existing within the FICO Xpress software offering, which allows us to measure how these different techniques interact with each other.

The outline of the paper is as follows: in Section 2 we broadly introduce the two existing solvers in the FICO Xpress suite that were extended to enable optimizing MINLPs to global optimality. In Section 3 we describe the computational setup that was used in testing all components of the global solver, whose features and computational impact are presented in detail in Section 4. Concluding remarks are given in Section 5.

2 Background

The FICO® Xpress Solver is a toolbox for mathematical optimization [14]. It features software tools to model and solve linear and nonlinear, continuous, integer, and mixed-integer optimization problems. It offers interfaces for a multitude of programming languages including Mosel, C/C++, Python, Java, .NET, R, Julia, and MATLAB[7]. Furthermore, users can integrate their own code into the solver through multiple kinds of callbacks.

2.1 FICO Xpress Optimizer

The main solver of this suite is the FICO Xpress Optimizer, a state-of-the-art LP and MILP solver that combines ease of use with speed and flexibility. It can read an optimization problem expressed in the following form:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \le b \\ & x_i \in \mathbb{Z} \quad \forall i \in I. \end{array} \tag{2}$$

If $I = \emptyset$, then the above is a linear programming (LP) problem, otherwise it is a general MILP. The FICO Xpress Optimizer solves LPs by either a primal or a

[3]BARON is a registered trademark of The Optimization Firm.
[4]Octeract is a registered trademark of Octeract Ltd.
[5]Hexaly is a registered trademark of Hexaly Inc.
[6]Gurobi is a registered trademark of Gurobi LLC.
[7]MATLAB is a registered trademark of The MathWorks Inc.

dual simplex algorithm, a barrier algorithm with crossover, a first-order hybrid gradient algorithm, or a concurrent parallel run of some of the previously mentioned solvers. For MILPs, the Optimizer employs a branch-and-cut algorithm, which combines a branch-and-bound tree search with the generation of cutting planes both on a global level (usually more extensively) and at local nodes of the search tree [21]. The branching step partitions a node subproblem by changing variable bounds and/or by adding constraints that exclude the subproblem's LP solution. A lower bound on the objective for the subproblem is obtained by solving its LP relaxation and improved through cut separators.

This basic framework is enhanced by a plethora of supporting subroutines, including primal heuristics, conflict analysis, specialized branching rules and cut generators, presolving, and domain propagation techniques. For a good introduction to the practices of computational MILP solving, see [24].

In addition to LPs and MILPs, the Optimizer solves optimization problems with convex quadratic constraints and objectives, second-order cone constraints, and their mixed-integer counterparts. Next to some specialized presolving algorithms, the foremost step for solving these problem classes is the ability to separate so-called *Outer Approximation* cuts [4].

2.2 FICO Xpress SLP

The FICO Xpress Solver also allows for solving general nonlinear, and, together with the Optimizer, integer nonlinear problems to local optimality. In addition to a nonlinear presolver, key parts of which will be detailed in Sections 4.4 and 4.5, it also includes a local NLP solver, which uses sequential linear programming (SLP), and has been part of the FICO® Xpress suite for more than 20 years.

Assuming the current working solution is x_0, SLP builds a linearization of (1) as

$$\begin{aligned} \min \quad & f(x_0) + \nabla f(x_0)^T y + p^T z \\ \text{s.t.} \quad & g_k(x_0) + \nabla g_k(x_0)^T y - z_k \leq 0, \quad \forall k = 1, \ldots, m \\ & y = x - x_0 \\ & -S \leq y \leq S \\ & z \geq 0, \end{aligned}$$

where y represents the change in the nonlinear variables, termed as delta variables, which are step bounded by trust region step sizes S, and z are infeasibility breakers allowing deviation from the linearization while being penalized in the objective. The step bounds are updated dynamically using an exponential trust-region scheme and SLP has mechanisms to fix the penalty breakers should the penalty terms become excessively large.

Convergence is measured in multiple ways. Ideally, the model converges in a mathematical sense, i.e., the change in the delta variables converges to zero. This is deemed as *strong convergence* since z converging to 0 implies infeasibility converging to 0 as well due to Lipschitz continuity. Apart from strong convergence, SLP applies several layers of increasingly weaker convergence criteria for stopping. In particular, it is possible to restrict convergence checks to variables in active constraints, and the convergence can be measured in the change of the linearization matrix or the activities of the linearization instead of the optimization variables themselves, for details, the reader is referred to the documentation of SLP. As these convergence criteria only consider the primal problem and provide no guidance on optimality, SLP checks the Karush-Kuhn-Tucker (KKT) conditions by solving a KKT problem where complementarity is approximated based on activities and infeasibility of the dual variables is minimized, once any of the convergence criteria has been satisfied.

3 Computational setup

In the following sections we will present computational results for individual components of the global solver. These results are based on a test set of 1308 instances, which includes:

- all solvable (by at least three different solvers according to the MINLPLib website [29]) MINLPLib [12] instances;
- an internal test set consisting of customer instances; and
- QPLIB [15] and MINLPLib instances that could be solved with previous versions of Xpress.

It is important for the test set to contain instances that demonstrate the effect of all the techniques in the paper. We consider three copies of every instance, each with a different permutation of variables and constraints[8], resulting in 3924 solves per run. Overall, the set is a representation of typical problems one would encounter in practice, including 19 instances which are provably globally infeasible. The solver in its default configuration solved 3036 cases.

All tests were run with FICO® Xpress 9.5.3 on Intel Xeon Gold 5218 2x16 core CPUs at 2.3GHz with 128GB of RAM, running a single job per CPU and using 16 threads in deterministic parallel mode. Hyperthreading was turned off and the solver job was bound to one of the 2 CPUs on the socket. The time limit was one hour of wall-clock time.

Unless indicated otherwise, the tables in this article show the change after *excluding* one feature of the solver, by showing both the difference in the number of solved instances and in the ratio of the 10s-shifted geometric mean[9] (SGM) of

[8]By *permutation* of a MINLP we mean an equivalent MINLP where the order of variables and constraints is changed. Runs with different permutations are carried out to mitigate the effects of solver performance variability, see, e.g., [25] or [20] for details.

[9]If $t_i, i = 1, \ldots, k$ are the solution times for a run on k instances, then the *shifted geometric mean* of the run is

$$\mathrm{SGM}(t) = \sqrt[k]{\prod_{i \in [k]} (t_i + \tau)} - \tau.$$

the solution time. We report the SGM ratio both for all instances (with unsolved instances counted with the time limit of 3600 seconds) and restricted to those instances that could be solved by both variants; these ratios are shown in the "Time ratio" groups under the columns "Overall" and "Solved Only", respectively. In particular, negative numbers in the *Solved* column indicate that we solve fewer instances (while a *Time ratio* larger than 1 indicates a slowdown on the instances we could solve) when turning *off* a technique. Note that "Solved" here means to proven global optimality or proven global infeasibility. As we will see, both of these measures are important when assessing the performance of the solver as different techniques contribute to one or the other. In addition to the runtime over all solvable instances in the whole testset under the "All" column, we also report runtimes over the subset of solved instances in which at least one of the compared solver versions took at least 100 seconds to avoid the dampening effect of trivial instances in the "$\geq$ 100s" column. In some tables we will also list the number of affected instances and the shifted geometric mean ratio over those instances to better quantify the impact of those techniques that only apply to specific subsets of instances.

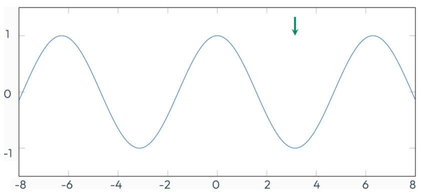
4 Solver structure

The FICO Xpress Global solver is, at its core, a branch-and-cut (B&C) solver that applies MILP and NLP techniques to find both valid bounds for the B&C enumeration and MINLP-feasible solutions. The solver applies *spatial branching* to the general branch-and-cut framework [19]. In this section we review the components and extensions to the FICO Xpress MILP and NLP solvers that allow for solving MINLPs to global optimality.

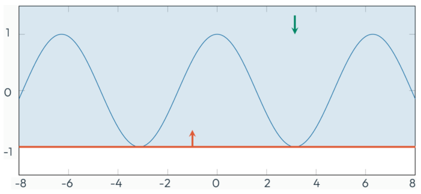
4.1 Overview

First, let us provide a brief overview of how a spatial branch-and-bound algorithm works to solve a nonconvex nonlinear optimization problem. For this example we will try to minimize the cosine function over the entire number line, i.e., we want to solve $\min\{\cos(x) : x \in \mathbb{R}\}$. After reformulation (Section 4.3), the problem becomes $\min\{y : y \geq \cos(x), x \in \mathbb{R}, y \in \mathbb{R}\}$. The steps of the algorithm can be followed in Figure 1. To minimize the function in Figure 1a we first add an initial set of convexification cuts (1b) to the problem. These are linear inequalities that constitute a polyhedral relaxation of the non-convex set $S = \{(x, y) \in \mathbb{R}^2 : y \geq \cos(x)\}$. At the beginning we can only add the trivial $y \geq -1$ constraint. We solve the resulting LP (1c) to obtain one of the many optimal solutions $(x, y) = (0, -1)$. This solution is not on the cosine function, i.e., it does not belong to S, and we cannot separate it from a polyhedral relaxation of S, so we need to branch. There are many ways to choose the branching point (see Section 4.7), but for this example we do this by splitting the interval

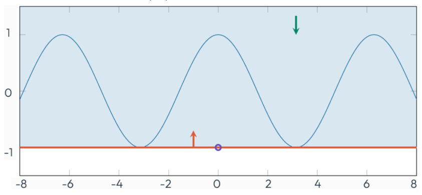
The shift parameter τ makes the measure less sensitive to the inherent uncertainty of time measurements, especially for very fast solves; we use $\tau = 10s$, which is a commonly used value.



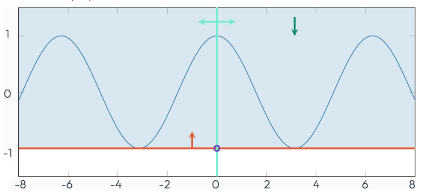
(a) $\min \cos x$



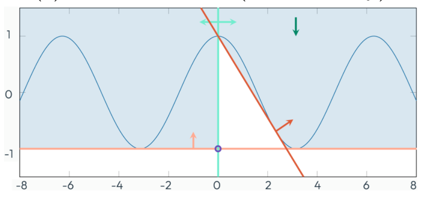
(b) Initial linear relaxation



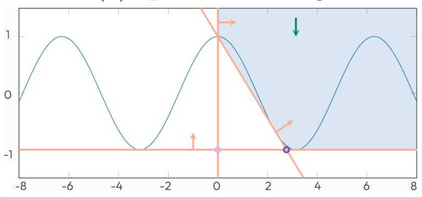
(c) Initial solution (one of many)



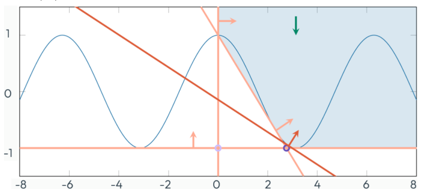
(d) Spatial branching



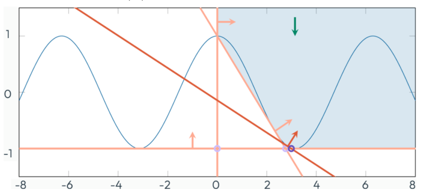
(e) Stronger cuts after branching



(f) New solution



(g) Cutting off the new solution



(h) Resolve, iterate

Figure 1: The main steps of an LP-based B&B nonlinear solver.

(1d) into two halves: $x \geq 0$ and $x \leq 0$. We process the right branch first (the other branch gets handled the same way), where the new lower bound introduced by branching now allows for creating a tighter polyhedral relaxation of $S' = \{(x, y) \in S : x \geq 0\}$; this tighter relaxation separates the solution from S' by adding (1e) a cut which is valid for this branch-and-bound node while it would have been invalid for the root as it would cut off valid solutions in the other branch. The LP relaxation now consists of two constraints. We resolve the relaxation to get a new solution (1f). This time, the LP solution can be separated from S' through an Outer Approximation cut (1g), resulting in a new solution (1h). Depending on the nonlinear feasibility tolerance, the LP-solution might already be nonlinear feasible w.r.t. the given tolerance. With stricter tolerances, more iterations of the cutting loop might be needed. Eventually, this procedure will converge to the point $(\pi, -1)$, which is a globally optimal solution.

In general, the cut loop might result in an infeasible (w.r.t. tolerances) problem after a few cuts have been added, in which case the node is pruned. If all nodes are pruned this way, the MINLP problem has been proven infeasible.

4.2 Problem representation

Under the assumption that the objective function and all constraints of Problem (1) are factorable, their expressions form trees whose leaf nodes are constants or variables, while non-leaf nodes are operators from the aforementioned set $\mathcal{O}$. For instance, the left-hand side of the constraint $x_2/x_3 + \sin(x_1 - 3) \leq 1$ is represented as the tree with thicker node contours in Figure 2. All constraints and the objective function of a MINLP share the variables as their leaf nodes. Their overlapping expression trees combine to form a *directed acyclic graph* (DAG). For example, the nonlinearities in the problem

$$\begin{aligned} \min\ & x_1 x_2 \\ \text{s.t.}\ & \frac{x_2}{x_3} + \sin(x_1 - 3) \leq 1 \\ & x_2 + x_3^3 \geq 2 \end{aligned} \tag{3}$$

can be represented by the DAG in Figure 2. DAG nodes are variables, con-

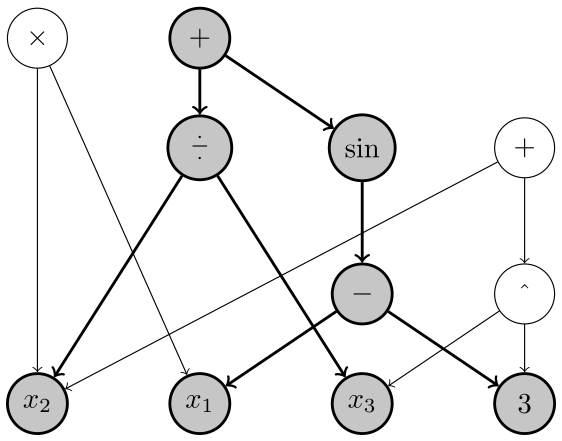


Figure 2: The DAG of Problem (3).

stants, and all mathematical operators used to construct the problem. The DAG therefore provides full information on the problem structure and is used by all components of the MINLP solver, as detailed below. It is created automatically inside the global solver from the expressions given by the user.

4.3 Reformulation and LP relaxation

Before constructing a convex relaxation, we *reformulate* the problem by introducing new *auxiliary variables*. Each auxiliary variable is associated with a node of the DAG, specifically with a single operator as used within the problem. The reformulation of Problem (3) in the example above is as follows[10]:

$$\begin{aligned}
\min\ & w_1 \\
\text{s.t. } & w_1 = x_1 x_2 \\
& w_2 = x_2/x_3 \\
& w_3 = x_1 - 3 \\
& w_4 = \sin w_3 \\
& w_5 = x_3^3 \\
w_2 + {}& w_4 \leq 1 \\
x_2 + {}& w_5 \geq 2.
\end{aligned} \tag{4}$$

This reformulation is performed by the solver from the general expressions provided by the user. While the nature of the problem itself has not changed, Problem (4) will form the basis for LP relaxations of (3).

4.4 Bound reduction

Bound reduction takes the initial bounds on all variables (some bounds may be infinite) and tightens them in such a way that at least one optimal solution is retained. Tight variable bounds are especially important for the convexification cuts described in Section 4.6, which provide an LP relaxation and hence a valid dual bound for the branch-and-cut algorithm.

FICO® Xpress Global uses a DAG-based bound reduction algorithm. Also known as Feasibility-Based Bound Tightening, or FBBT, it applies interval analysis techniques [31], already of common use in constraint programming and Artificial Intelligence [13], by exploring the DAG and using the initial bound interval on all variables, the constraints, and possible knowledge of a cutoff value for the objective function, in order to infer tighter bounds on variables. FBBT is known to be very quick and is therefore used throughout the branch-and-cut process, from the initial presolve steps to the analysis of branching candidates in the generic B&C code and as an initial step of the SLP-based heuristics that seek a feasible MINLP solution. Given its pervasive use in the solver, we do not report on its performance impact; it is also known from other solvers that FBBT is necessary to ensure reasonable convergence.

4.5 Presolve

It is well-established that presolving is a crucial part of a solver for any class of problems. By simplifying constraints and eliminating vast portions of the set of

[10]All auxiliaries are called w_i for clarity.

feasible solutions, the presolver can have a dramatic effect on efficiency, often reducing the computational effort from hours to mere minutes. FICO® Xpress Global adopts a two-pronged approach to presolving by first exploiting the nonlinearities of the original problem in a nonlinear presolver, which reduces and simplifies the problem by acting on all parts of the problem including nonlinearities, and then by reapplying the linear and mixed-integer parts of presolve[11] on the reformulated problem. Some of the specific operations carried out in nonlinear presolve include formula simplifications, nonlinear eliminations, and nonlinear probing.

The idea of *formula simplifications* is to avoid applying the full nonlinear machinery to formulas that happen to be constant, linear or could otherwise be simplified, for example by computing $\sin(\pi/2)$, replacing x^1 by x or $x^3 \cdot x^4$ by x^7. While all of these simplifications are trivial and should not be necessary for carefully modeled problems, they can easily appear in practice when models are combined with changing, automatically inserted data.

The process of *nonlinear eliminations* uses nonlinear equations such as $z = x^2 + y^2$ to replace z by its nonlinear representation in other formulas. This can reduce the total number of variables in the problem, which can be helpful for SLP, be it as a standalone solver or as a heuristic within global. Care needs to be taken, though, to not enlarge the non-zero pattern of the Jacobian of any constraints too much. In particular, introducing nonlinearities in otherwise linear constraints should usually be avoided.

Nonlinear probing is a simple procedure that uses bound tightening repeatedly on a set of candidate variables: a fictitious lower bound $x_i \geq \bar{\ell}_i$ is set for a variable x_i, then FBBT is applied to the resulting problem. If FBBT results in infeasibility, then $\bar{\ell}_i$ is a valid *upper* bound for x_i. Similar considerations lead to tighter variable lower bounds. Probing can be helpful in particular on unbounded variables, which can often cause numerical difficulty for the LP relaxation of a MINLP.

Technique	# Instances		Time ratio			
			Overall	Solved only		
	Affected	Solved		All	$\geq$ 100s	Affected
Presolve		-243	1.60	1.39	2.18	
Linear/quadratic		-96	1.13	1.07	1.28	
Nonlinear		-138	1.36	1.21	1.63	
Simplifications	1896	-82	1.13	1.04	1.12	1.08
Eliminations	151	+1	0.99	1.00	0.98	0.61
Probing	101	-1	1.00	1.00	1.00	1.06

Table 1: The impact of disabling presolve techniques. Disabling presolve affects all instances, hence the respective cells are left blank.

As we can see in Table 1, disabling presolve altogether leads to a significant degradation, losing about 8% of all solved instances with a 40% slowdown even

[11]See [1] for a good general overview on linear and integer presolve techniques.

on instances that can still be solved. Nonlinear presolve is the bigger contributor compared to linear presolve, although, not surprisingly, the overall slowdown of disabling presolve is larger than the sum of the two. Some techniques (such as linear bound tightening) are present in both linear and nonlinear presolve, hence skipping only one type of presolver would still get the benefit from applying the same technique in the remaining presolver; the extra losses are therefore due to some redundancy between the two classes of presolving techniques. Within nonlinear presolve, formula simplifications have by far the biggest effect, in particular on solvability, affecting about half of the instances with a 4% degradation and 82 instances lost when disabling them, while probing and eliminations affect far less instances, with eliminations even being a hindrance on some of the instances.

4.6 Convexification Cuts

At the core of FICO® Xpress Global is a procedure for separating *convexification cuts*, i.e., valid linear inequalities for the feasible set of the reformulated problem as described in Section 4.3. These cuts are based on the operator-auxiliary relationship outlined in Section 4.3. For each auxiliary-operator pair $(w_i, f(x))$, for example $(w_4, \sin(w_3))$, one or more linear inequalities are added that are valid for the set $\{(x, w_i) : w_i = f(x), x \in [\ell, u], w_i \in [\ell_i^w, u_i^w]\}$. Combining all such inequalities yields an LP relaxation of the MINLP and hence a lower bound on the optimal objective value. As mentioned above, the tightness of said LP relaxation strongly depends on the bounds on all variables, including auxiliaries.

In FICO Xpress Global we add standard convexification cuts for all supported function types [34]. In addition, we add *Reformulation-Linearization Technique* (RLT) cuts for bilinear expressions [9]. RLT cuts are separated by multiplying valid linear constraints with variables that are present in violated nonconvex bilinear expressions. The resulting constraints contain quadratic terms, which are subsequently linearized using the bilinear expressions they appear in to obtain valid linear inequalities. In case the bilinear expressions used for the linearization are violated, the so derived linear inequalities can be as well. In order to strengthen the LP relaxation in the presence of bounded bilinear (i.e., product) terms $w_h = x_i x_j$, we also add *Lifted Tangent Inequalities* (LTIs) [6]; these consist of tangent cuts $ax_i + bx_j \geq c$ to the set $\{(x_i, x_j) \in \mathbb{R}^2 : x_i x_j \geq \ell_h\}$ which are then strengthened by lifting to $ax_i + bx_j - dw_h \geq c$. While the McCormick inequalities [2, 28] provide the convex hull of the set $\{w_h = x_i x_j, x_i \in [\ell_i, u_i], x_j \in [\ell_j, u_j]\}$, the LTI cuts can provide an additional strengthening in the presence of non-trivial bounds on w_h that are not merely implied by the bounds on x_i and x_j.

The effect of RLT cuts and LTIs is summarized in Table 2, showing that RLT is essential for solving a significant number of instances, even though the time benefit on solvable instances is only 1%. Lifted Tangent Inequalities have a smaller impact, with the overall speedup being below 1%.

Note that we do not report numbers for disabling convexification cuts as a whole because they are necessary for solving MINLPs and, as such, disabling them would prevent reasonable convergence of the spatial branch-and-bound algorithm.

Cut class	# Instances		Time ratio			
			Overall	Solved only		
	Affected	Solved		All	$\geq$ 100s	Affected
RLT	2171	-34	1.04	1.01	1.03	1.01
LTI	1143	-2	1.00	1.00	1.01	1.01
Trigonometric	66	-1	1.00	1.01	1.03	1.84

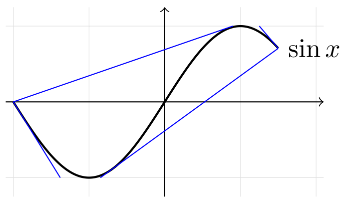
Table 2: Performance impact of disabling specific cutting planes.

Trigonometric functions sin and cos deserve special treatment as they are neither convex nor concave on a sufficiently large bound interval, thus simple tangents do not work in general. A trivial approach is to add cuts with slopes ± 1 at both endpoints of the interval. These are valid, because sin and cos have derivatives in $[-1, +1]$. A much stronger option is to add tight cuts at the endpoints of the interval and make sure they do not intersect the function. The slopes are calculated with a simple Newton iteration, making sure that the generated cut is always valid. These cuts are illustrated in Figure 3a. If the interval is longer than 2π, then these cuts are overall very weak, which we counter by using special branching rules for these functions. While their impact on the overall runtimes in Table 2 is rather small, because there are very few instances with trigonometric functions in the test set, on the small subset of trigonometric instances the impact is a speedup factor of 1.84.

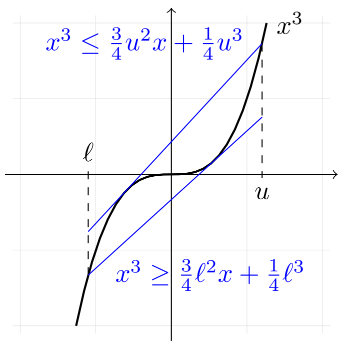
Another class of functions where special cuts are necessary are odd powers over an interval $[\ell, u]$ such that $\ell < 0 < u$. These functions are again neither convex nor concave, but there are simple closed form expressions for the tightest possible cuts [22]. These play a role only during the initial separation of convexification cuts, as we branch at 0 at the earliest opportunity, and the function will then have an easy outer envelope on the new intervals. These cuts are illustrated in Figure 3b.

Adding multiple cuts. We add several cuts in the initial step that builds an LP relaxation of Problem 1 and also during root node processing and later during the tree solve. The right number of cuts is critical: adding too many makes the relaxation better at the expense of increased relaxation solve time. On the other hand, having an efficient mechanism to deal with the cuts allows us to be quite aggressive here. We normally add up to 20 convexification cuts upfront for each node in the DAG, and update them as the bounds of the interval change.

For this to work efficiently it is crucial that the MIP solver in Xpress can deal with local cuts, i.e., cuts that are valid only on a part of the search tree.



(a) Trigonometrics



(b) Odd powers

Figure 3: Special cuts for trigonometric and odd-power functions.

Numerical stability. When adding cuts one needs to be careful about the numerical properties of the linear relaxation that needs to be solved. In general, we try to avoid adding cuts with very large coefficients or a large ratio among maximum and minimum coefficients in absolute value. The FICO® Xpress Optimizer also has routines to detect cuts that are nearly parallel to each other and it avoids inserting more than one of those cuts.

4.7 Branching

As we saw in Section 4.1, when solving general MINLPs to global optimality, it is no longer sufficient to only branch on integer variables: due to the non-convexity of the nonlinear functions involved, it is often necessary to also branch on continuous variables to advance the solution process by excluding the current sub-problem's LP solution with additional help from bound reduction and convexification cuts, as described in Sections 4.4 and 4.6. In general, any variable, original or auxiliary, appearing in an auxiliary-operator pair $(w_i, f(x))$ that is part of a violated constraint and for which the current LP solution $\tilde{w}_i, \tilde{x}$ violates w_i's definition, i.e., $\tilde{w}_i \neq f(\tilde{x})$, is a potential candidate for branching. In contrast to integer branching, where there is a natural variable split to exclude the fractional solution, for spatial branching there are two questions to be answered: which variable to branch on and where to split the bound interval of that variable.

The branching decision is carried out in two steps: first, a set of candidates is created, consisting of typically one variable x_{i_k} with a branching point τ_{i_k} to split the current node-problem S into two subproblems $\underline{S} := \{x \in S, x_{i_k} \leq \tau_{i_k}\}$ and $\overline{S} := \{x \in S, x_{i_k} \geq \tau_{i_k}\}$, although in general a branching candidate can include bound changes on multiple variables, which can be beneficial in specific cases, as we will see later. Each of those branching candidates is then evaluated in a second step, and the one with the best evaluation is chosen.

4.7.1 Branching point selection

We will start with the branching point selection, since this is the first decision to be made in the implementation. Given a variable x_i, there are two natural points to branch on: either the current LP-solution $\tilde{x}_i$, which generally maximizes the chance of the current solution becoming infeasible in both subproblems, see, e.g., [5], or the midpoint of the bound interval $\frac{1}{2}(\ell_i + u_i)$, which minimizes the worst-case number of branchings required to reach an interval length of ε, which, under Lipschitz assumptions, guarantees nonlinear feasibility within a given tolerance for any LP feasible solution. Similar to other solvers such as Couenne [5], by default FICO® Xpress Global branches on a convex combination $\alpha \cdot \frac{1}{2}(\ell_i + u_i) + (1-\alpha)\tilde{x}_i$ of these two, where $\alpha \in [0,1]$ depends on the specific operator, although most operators use $\alpha = 0.75$ by default[12]. Note that infinite or very large variable bounds are not taken into account this way: if the LP solution is too far from both endpoints of the interval, then the interval is ignored in choosing the branching point.

		Time ratio		
α	# Instances	Overall	Solved only	
	Solved		All	$\geq$ 100s
1 (midpoint)	-21	1.03	1.01	1.02
0.75	-19	1.02	1.00	0.96
0.5	-25	1.01	0.97	0.86
0.25	-23	1.01	0.98	0.93
0 (LP solution)	-145	1.23	1.14	1.37

Table 3: The impact of branching point selection.

As we can see in Table 3, using this convex combination solves at least 19 instances more than any fixed convex combination, even though moving further towards the LP solution may be faster if both options can solve it. This might not be overly surprising, given that the main advantage of choosing the midpoint is the better worst-case behavior, while branching closer to the LP solution tends to have a more immediate impact on the dual bound. This is a trade-off that the developers of any similar package have to deal with. Not using the midpoint at all is a significant loss in both instances and solve time, though.

While the convex combination tends to work well in general, for specific operators it can be beneficial to branch on special values, significantly enhancing the propagation possibilities or even making one or both of the resulting branches convex or concave. Examples for the former include (non-convex) quadratics where, for example, $x \cdot y \geq 4$ allows reductions on y if $x \geq 0$ or $x \leq 0$ but in general cannot be propagated if the sign of x is undecided. An example for the latter would be x^3 where branching on 0 will imply convexity on the up- and concavity on the down-branch. Similar ideas also apply to trigonometric

[12]This value depends heavily on other characteristics of the solver. Most notably, it was mentioned in [16] that the RAPOSa solver [17] uses a default value of 0.25.

functions, where branching on multiples of $\pi/2$ is beneficial.

In our implementation, for those operators where branching on 0 is preferred[13], we replace the original branching point with 0 if the two differ by less than 10% of the absolute value of the smaller of the two bounds for the cases of improved propagation and by 100% for the case of improved convexity. In the case of sin, cos and tan, we would round the branching point to the closest multiple of $\pi/2$, as long as that is still in the interior of the bound interval. Not only do these special branching rules help propagation, they also allow us to generate stronger cuts in later iterations. As can be observed from Table 4, these "roundings" have a small but positive effect on solve times, even though they are not really necessary to solve any particular instances.

Apart from these single-variable branchings, a common occurrence are complementarity constraints of the form $x_i \cdot x_j = 0$. Instead of separately branching on either x_i or x_j and using convexification cuts to enforce the nonlinear constraint, FICO® Xpress Global instead branches on the equivalent logical constraint $x_i = 0 \vee x_j = 0$, which is necessary to solve about 1% of the instances in the test set. Again, this is a technique that allows the solver to solve more instances at the expense of a small loss in the average speed.

Technique	# Instances		Overall	Time ratio		
				Solved only		
	Affected	Solved		All	$\geq$ 100s	Affected
Complementarity	1078	-30	1.03	0.99	0.97	0.97
Br. Point rounding	833	+3	1.00	1.01	1.01	1.04

Table 4: The impact of disabling special branching point selection rules.

4.7.2 Branching variable selection

Once a branching point τ_{i_k} for each potential branching candidate x_{i_k} has been computed, one of them must be chosen. There have been a number of techniques suggested for this in the literature, see, e.g., [5], but FICO Xpress Global follows the existing approach of the FICO Xpress Optimizer, which is mainly based on a combination of strong branching [3] in the initial stages of the tree search, i.e., at the root node and at the nodes with lower B&C tree depth, and pseudocosts deeper in the tree. Pseudocosts can naturally be extended to spatial branching by multiplying with the reduction of the bound interval instead of the fractionality to obtain an estimate on the current branching based on the historic unit cost [5].

Strong branching can also be extended to spatial branching in a straightforward way. However, in contrast to integer branching, the main effect of branching on the dual bound is not derived from the bound change alone but also from the bounds on other variables arising from propagation and the tighter

[13] These include powers of both even and odd degree, the absolute value function, inverse trigonometric functions and also the auxiliary variables of sin, cos and tan.

convexification cuts, compare Section 4.1. For this reason, strong branching in FICO Xpress Global applies bound propagation and convexification cuts[14] to each branching candidate prior to evaluating the change in the dual bound. This is a computationally expensive step in general, but it results in a more accurate evaluation of branching candidates and leads to much better performance, especially in the number of solved instances.

As we can see in Table 5, the cutting plane generation for branching variable evaluation is more important with a decrease of 583 in the number of solved instances when disabling it, while disabling both leads to 689 fewer instances solved.[15] Their effect on the solve times of already solved cases is in the 15-30% range, with larger numbers on harder instances. An essential ingredient for this performance boost is a highly efficient cut management framework, which the Xpress global solver inherits from the linear MIP solver.

Two general questions for MINLP solvers are whether to always branch on integers before branching on continuous variables, and whether to prefer original over auxiliary variables. In case of FICO® Xpress Global, there is currently no general preference for integer branching over spatial branching, but original variables are generally preferred over auxiliary variables, unless the latter led to many branching candidates at the root where one of the two children was infeasible.

Always considering both original and auxiliary variables when deciding on the branching variable leads to a slowdown of 11%, as shown in the last row of Table 5. It is also worth noting that branching only on original variables does not hurt the solver speed at all, but it reduces the number of instances solved significantly.

Technique	# Instances Solved	Time ratio Overall	Solved only All	Solved only $\geq$ 100s
Propagation	-96	1.15	1.09	1.35
Cuts	-583	2.40	1.22	1.68
Both propagation and cuts	-689	2.86	1.27	2.03
Only originals	-80	1.10	1.00	0.97
Originals and auxiliaries equally	-72	1.12	1.11	1.17

Table 5: The impact of techniques in branching variable selection. The first three rows are about disabling the techniques applied to branching candidates to get a better measure of their impact. The last two are about the balance between original and auxiliary variables.

[14]Strong branching (temporarily) changes the bounds, and these are then propagated. Stronger bounds then allow for tighter outer approximation cuts. Currently, we generate the cuts as if the candidate were chosen to be the actual branching variable. This is performed for all branching candidates and all but the winner's cuts are then discarded.

[15]Bound propagation would still be performed in the node presolver on the winning branching candidate, so the effect shown here is purely due to better variable selection because of the stronger bounds.

4.8 Heuristics

In general, any MILP heuristic can be applied to the MILP relaxation of a MINLP. However, since they are unaware of the nonlinearities beyond the current set of convexification cuts, these solutions tend to be integer feasible but nonlinear infeasible. Using a local NLP-solver such as SLP, we can resolve the nonlinear part of the problem: given an integer feasible solution, we fix all integer variables to their current values and then solve the remaining continuous problem using the local NLP solver.

To improve the solve, the underlying integer feasible solution can also be supplied as an initial point to the NLP solver for the remaining continuous variables. The same technique can also be applied to integer feasible but nonlinear infeasible node solutions, although care has to be taken with regards to the frequency of calling the nonlinear solver, in particular for continuous NLPs for which every node solution does not need to satisfy integrality constraints and can hence provide a starting point for the NLP solver. Given both the fixings

Technique	# Instances Solved	Time ratio Overall	Solved only All	$\geq$ 100s
SLP heuristic	-102	1.15	1.07	1.47

Table 6: The impact of disabling the SLP heuristic.

and the fact that a local NLP solver is used, there is of course no guarantee for the resulting NLP problem to be feasible or the returned solution to be globally optimal, but if the NLP solver can find a solution within the subproblem, then this solution is MINLP feasible for the original problem. While less powerful than dedicated MINLP heuristics, this approach benefits from the maturity of both MIP heuristics and local NLP solvers in the FICO® Xpress Optimizer. Table 6 shows that the speedup the heuristic provides can be substantial on hard models and it also helps to solve significantly more instances.

4.9 Parallelism

As evidenced by the experiments in [8], taking advantage of a multi-threaded environment is not a trivial task for a global optimization solver, with many solvers struggling to gain significant speedups from running with multiple threads. The branch-and-bound search in FICO Xpress is parallelized[16] (see [7]), which automatically makes the global solver a parallel branch-and-cut solver. The effect of threading is summarized[17] in Table 7, both for the whole test set and specifically for runs that required at least 100 nodes for all thread combinations to

[16]For completeness we have to mention that the initial linear relaxation is also solved with a parallel linear optimization solver, but since it is typically a very small fraction of the overall solution time we do not discuss it in more detail.

[17]Tests using fewer than 16 threads were still allocated 16 physical cores and the solver job was bound to some of them to prevent the operating system moving them around.

avoid cases that are solved before there exist enough nodes to distribute them among the different threads. As we can see, the relative speedup when doubling

		Time ratio			Speedup			
Threads	# Ins.	Overall	Solved only		Solved only		$\geq$ 100 nodes	
	Solved		All	$\geq$ 100n	to 1	double	to 1	double
1	-202	1.62	2.59	3.97	1.00		1.00	
2	-145	1.38	1.94	2.57	1.34	1.34	1.54	1.54
4	-72	1.22	1.51	1.80	1.72	1.28	2.21	1.43
8	-37	1.08	1.18	1.27	2.19	1.28	3.13	1.42
16	0	1.00	1.00	1.00	2.59	1.18	3.97	1.28
32	-8	0.99	1.00	0.98	2.59	1.00	4.05	1.02

Table 7: The impact of the number of threads in deterministic mode. The last four columns list the speedup relative to a single-threaded run, and the speedup when doubling the number of threads, respectively, once for all solvable instances and once for those that took at least 100 nodes for all thread numbers.

the number of threads gets smaller as the number of threads increases. This is not surprising, as only problems with a large B&B tree can take advantage of more than 16 threads. It is also worth pointing out that parallelism only allows for solving a larger number of subproblems simultaneously, so the extra solved instances are simply due to being able to process a larger tree. It is especially striking that going from 16 to 32 threads, the number of solved instances no longer increases but actually goes down slightly, and also the speedup is almost unmeasurable at this point. These results are similar to what we observe for our linear MIP solver for problems with a similar tree size.

There are two ways the parallel framework can work: deterministic and opportunistic. The deterministic mode is the default, and, as its name suggests, it guarantees that running the solver on the same problem with the same number of threads will terminate with the exact same result, regardless of what else is running in the background.[18] This is achieved by carefully synchronizing the parallel threads. Obviously, this comes with some overhead, as some threads will have to wait for other threads to finish. The impact of determinism is summarized in Table 8: for a given number of threads we measured how much faster the opportunistic variant would be. These results have to be taken with a grain of salt, as they are non-deterministic by their very nature, so they are not really reproducible. They serve as a rough guideline to assess the cost of determinism. Overall, the potential gain is not large enough to make up for the lack of determinism: the number of instances solved is roughly the same and the speedup is 2-9%. As expected, the speedup is larger as the number of threads is increasing, but not in a dramatic way.

[18]In fact, FICO Xpress guarantees an identical solution path even if the solver is running on a different operating system (Windows/Linux/Mac).

Threads	# Instances Solved	Time ratio Overall	Solved only All	Solved only $\geq$ 100n
2	+1	0.98	0.99	0.99
4	-6	0.98	1.00	0.98
8	-2	0.97	0.99	0.95
16	0	0.95	0.97	0.93
32	+3	0.93	0.97	0.91

Table 8: The advantage of opportunistic vs deterministic parallelism.

4.10 Unboundedness

For MILPs of the form (2), with rational coefficients and right-hand side, unboundedness of the convex relaxation, together with the existence of an integer feasible point, implies unboundedness of the MILP itself. This is not the case for MINLP, where an LP-based spatial branch-and-bound algorithm may produce an initial LP relaxation that is unbounded even though the MINLP itself is not.

In these (rare) cases, FICO® Xpress Global applies a bounding box: first the original variables are constrained to be within a given interval with the default being $[-10^6, 10^6]$. If the relaxation is still unbounded, then the same bounds are also applied to the auxiliaries. This ensures that the subsequent relaxations are all bounded and a spatial branch-and-bound is guaranteed to terminate, though of course Xpress will no longer claim global optimality, even though in practice the resulting solutions will often be globally optimal. Other solvers handle this situation in a similar way, although not all will distinguish this case from global optimality.

In the numerical experiments performed for this paper, the bounding box was applied in 73 out of 3924 solves and only in 12 of those 73 cases it needed to be applied to auxiliary variables as well.

5 Conclusions and future work

We have demonstrated how to extend a mixed-integer linear optimization solver with an existing local solver and new global-specific techniques for nonlinear optimization to solve MINLPs to global optimality. The new solver took advantage of the existing components of FICO® Xpress. All of the techniques discussed in this paper are part of the global solver in FICO Xpress 9.5.3.

There are several more advanced techniques that remain to be implemented, including more sophisticated optimization-based bound tightening, further specialized convexification cuts, more general expression types, convexity detection, symmetry detection, and special techniques and heuristics for some common problem structures.

Acknowledgments. The authors are indebted to their former colleague, Zsolt Csizmadia, for his early work on the global solver and his comments on a previous version of this paper. We also thank Michael Perregaard for his valuable comments.

References

[1] Tobias Achterberg, Robert E. Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[2] Faiz A. Al-Khayyal and James E. Falk. Jointly constrained biconvex programming. *Math. Oper. Res.*, 8:273–286, 1983.

[3] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The traveling salesman problem.* Princeton university press, 2006.

[4] Pietro Belotti, Timo Berthold, and Kellington Neves. Algorithms for discrete nonlinear optimization in FICO Xpress. In *2016 IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM)*, 2016.

[5] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software*, 24(4-5):597–634, 2009.

[6] Pietro Belotti, Andrew J. Miller, and Mahdi Namazifar. Valid inequalities and convex hulls for multilinear functions. *Electronic Notes in Discrete Mathematics*, 36:805–812, 2010.

[7] Timo Berthold, James Farmer, Stefan Heinz, and Michael Perregaard. Parallelization of the FICO Xpress-Optimizer. *Optimization Methods & Software*, 33(3):1–12, 2017.

[8] Ksenia Bestuzheva, Antonia Chmiela, Benjamin Müller, Felipe Serrano, Stefan Vigerske, and Fabian Wegscheider. Global optimization of mixed-integer nonlinear programs with SCIP 8. Technical Report 2301.00587, arXiv, 2023.

[9] Ksenia Bestuzheva, Ambros Gleixner, and Tobias Achterberg. Efficient separation of RLT cuts for implicit and explicit bilinear products. *arXiv preprint arXiv:2211.13545*, 2022.

[10] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and

Lixing Xu. The SCIP optimization suite 9.0. Technical Report 24-02-29, ZIB Report, 2024.

[11] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, Nicolas Sawaya, and Andreas Wächter. An algorithmic framework for convex mixed integer nonlinear programs. Technical Report RC23771, IBM Research Report, 2005.

[12] Michael R. Bussieck, Arne Stolbjerg Drud, and Alexander Meeraus. MINLPLib – a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.

[13] Ernest Davis. Constraint propagation with interval labels. *Artificial intelligence*, 32(3):281–331, 1987.

[14] The FICO-Xpress Optimizer. `https://www.fico.com/fico-xpress-optimization/docs/latest/overview.html`, 2024.

[15] Fabio Furini, Emiliano Traversi, Pietro Belotti, Antonio Frangioni, Ambros Gleixner, Nick Gould, Leo Liberti, Andrea Lodi, Ruth Misener, Hans Mittelmann, Nikolaos V. Sahinidis, Stefan Vigerske, and Angelika Wiegele. QPLIB: A library of quadratic programming instances. *Mathematical Programming Computation*, 2018.

[16] Julio González-Díaz. RAPOSa: A free global solver for mixed integer polynomial optimization problems. ISMP, Montréal, 2024.

[17] Brais González-Rodríguez, Joaquín Ossorio-Castillo, Julio González-Díaz, Ángel M. González-Rueda, David R. Penas, and Diego Rodríguez-Martínez. Computational advances in polynomial optimization: RAPOSa, a freely available global solver. *Journal of Global Optimization*, 85:541–568, 2023.

[18] Gurobi optimization. `https://www.gurobi.com/`, 2025.

[19] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[20] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, et al. MIPLIB 2010: mixed integer programming library version 5. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[21] Richard Laundy, Michael Perregaard, Gabriel Tavares, Horia Tipi, and Alkis Vazacopoulos. Solving hard mixed-integer programming problems with Xpress-MP: A MIPLIB 2003 case study. *INFORMS Journal on Computing*, 21(2):304–313, 2009.

[22] Leo Liberti and Constantinos C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25:157–168, 2003.

[23] Youdong Lin and Linus Schrage. The global solver in the LINDO API. *Optimization methods and software*, 24(4):657–668, 2009.

[24] Andrea Lodi. Mixed integer programming computation. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 619–645, 2010.

[25] Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. *INFORMS TutORials in Operations Research*, pages 1–12, 2013.

[26] Andreas Lundell, Jan Kronqvist, and Tapio Westerlund. The supporting hyperplane optimization toolkit for convex minlp. *Journal of Global Optimization*, 84(1):1–41, 2022.

[27] Ashutosh Mahajan, Sven Leyffer, Jeff Linderoth, James Luedtke, and Todd Munson. Minotaur: A mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation*, 13:301–338, 2021.

[28] Garth P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems. *Mathematical Programming*, 10:146–175, 1976.

[29] MINLPLib. `https://www.minlplib.org/instances.html`, 2025.

[30] Ruth Misener and Christodoulos A. Floudas. ANTIGONE: Algorithms for continuous / integer global optimization of nonlinear equations. *Journal of Global Optimization*, 59:503–526, 2014.

[31] Arnold Neumaier. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1991.

[32] The Octeract Engine. `https://www.octeract.com/octeract-engine`, 2025.

[33] Nikolaos V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.

[34] Mohit Tawarmalani and Nikolaos V. Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, volume 65. Springer Science & Business Media, 2013.