The Supporting Hyperplane Optimization Toolkit

A Polyhedral Outer Approximation Based Convex MINLP Solver Utilizing a Single Branching Tree Approach

Andreas Lundell[a]*, Jan Kronqvist[b], and Tapio Westerlund[b]

[a]Mathematics and Statistics, Åbo Akademi University, Turku, Finland;

[b]Process Design and Systems Engineering, Åbo Akademi University, Turku, Finland;

June 27, 2018

In this paper, a recently released open-source solver for convex mixed-integer nonlinear programming (MINLP) is presented. The Supporting Hyperplane Optimization Toolkit (SHOT) solver combines a dual strategy based on polyhedral outer approximations (POA) with several primal heuristics. The outer approximation is achieved by expressing the nonlinear feasible set of the MINLP problem with linearizations obtained with the extended supporting hyperplane (ESH) and extended cutting plane (ECP) algorithms. The dual strategy is tightly integrated with the mixed-integer programming (MIP) subsolver in a so-called single-tree manner, *i.e.*, only a single MIP optimization problem, is solved, and the polyhedral linearizations are added as lazy constraints through callbacks in the MIP solver. This does not force the MIP solver to rebuild the branching tree in each iteration, as is the norm in most other POA-based methods. SHOT has been released as a COIN-OR open-source project, and it utilizes a flexible task-based structure which makes it easy to extend and modify. The main functionality and solution strategies available in SHOT are described, and their impact on the performance are illustrated through numerical benchmarks on all the 366 convex MINLP problems in the problem library MINLPLib. To show the effectiveness of the new solver, SHOT was also compared to other state-of-the-art solvers on the same benchmark set.

1 Introduction

The Supporting Hyperplane Optimization Toolkit (SHOT), is a open-source solver for convex mixed-integer nonlinear programming (MINLP). It is based on the SHOT algorithm described later on in this paper, which is a combination of a dual and a primal strategy that, when considering a minimization problem, gives lower and upper bounds on the objective function of an optimization problem

*Corresponding author. Email: andreas.lundell@abo.fi

respectively. The dual strategy is based on polyhedral outer approximation (POA) through supporting hyperplanes of the integer-relaxed feasible set defined by the nonlinear constraints in the problem. The primal strategies consists of several deterministic primal heuristics, *e.g.*, based on root searches. Here the term 'heuristic' does not indicate that the procedures are stochastic, only that they are not guaranteed to succeed in finding a new, better solution.

Convex MINLP is a subclass of the more general nonconvex MINLP problem class, which combines the nonlinearity aspect from nonlinear programming (NLP) with the discreteness of mixed-integer programming (MIP). This allows for much flexibility when used as a modeling tool, since nonlinear phenomena and relations can be expressed in the same framework as discrete choices. Due to its generality, the solution process for problems in the MINLP class is often computationally very demanding and even small problem instances may pose problems for solvers available today. This is one of the motivations behind considering the more specific convex MINLP problem type. The convexity assumption (either by manual determination or by automatic means) means certain properties of convex functions can be utilized to make the solution procedure more tractable. For example, the property that a linearization of a convex function in any point always give an underestimation of its value in all points, forms the theoretical foundation for many solution methods, *e.g.*, those utilizing POA, such as Outer Approximation (OA) [13], as well as the Extended Cutting Plane (ECP) [55] and Extended Supporting Hyperplane (ESH) [28] algorithms.

However, due to the discrete nature of the MINLP problems, convexity is not enough to guarantee an efficient solution process, as the noncontinuous search-space caused by the discrete variables provides a very daunting challenge. The discreteness are normally handled by considering a so-called branch-and-bound tree that in its nodes considers smaller and easier subproblems where the discrete possibilities are gradually reduced [22]. Handling such a tree is difficult in itself as the number of nodes required are often very large, so creating an efficient MINLP solver based on the branch-and-bound methodology almost certainly requires a lot of development effort solely for how to manage the tree structure.

MIP solvers such as CPLEX and Gurobi have also faced the problem of efficiently handling a large tree structure, and their performance has, through the years, improved steadily. The main idea behind SHOT, to utilize the MIP solvers as much as possible, makes it directly benefit from the progress made in the subsolvers. For example, without much extra effort by its developers, SHOT takes full advantage of the parallelism provided by modern computer processors; thus, since being able to perform several operations in parallel is nowadays mostly what drives the increase in computing performance, the performance of SHOT will continue to scale with that of the MIP solvers. SHOT also takes full advantage of preprocessing, heuristics, integer cuts, and other features provided by the MIP solver. The MIP solver integration is mainly accomplished through so-called callbacks, which allows a third party to modify and affect the behaviour of the MIP solution process. The nonlinearities in the MINLP problem are iteratively introduced into the MIP model through linear approximations with supporting hyperplanes through the means of lazy constraints added through specialized callbacks. By adding the constraints as lazy, the MIP solver does not need to invalidate the previously built branching tree, and can continue without rebuilding it. Also, this can efficiently be done in parallel on multiple threads.

There are many different convex MINLP solving software available today. Many of these are implemented, or integrated with one or more of the mathematical modeling systems such as GAMS [8], AIMMS [5], AMPL [15], Pyomo [23] or JuMP [12]. Convex MINLP solvers available include AlphaECP [53], BONMIN [6], AOA [25], DICOPT [21], Juniper [26], Minotaur [40], Muriqui [41], Pajarito [34] and SBB [18]. Nonconvex global MINLP solvers include ANTIGONE [43], BARON [47], Couenne [3], LINDO [33], and SCIP [50]. Of the mentioned MINLP solvers, AlphaECP, BARON, BONMIN, AOA, DICOPT, Minotaur, SBB and SCIP are compared to SHOT in a numerical bench-

mark given in Section 7.1. Recent reviews over methods and solvers for convex MINLP are provided in [27, 49]; in [27] an extensive benchmark between convex MINLP solvers is also provided. Older overviews over MINLP methods and software are also given in [11] and [20]. An older benchmark of MINLP solvers was performed in [31].

This paper is structured as follows: In Section 2, a short background about convex MINLP and some discussion about the terminology utilized in the paper is given. In Section 3, a discussion about the differences between single- and multi-tree polyhedral outer approximation is provided. The dual and primal strategies utilized in SHOT is described in Sections 4 and 5, respectively, and in Section 6 the main SHOT algorithm is discussed in detail, as are details about the solver implementation. To provide insight into how different strategies in SHOT affect the performance of the solver, as well as to compare it other state-of-the-art MINLP solvers, an extensive benchmark is given in Section 7. Finally, some conclusions and ideas for future work are provided in Section 8.

2 Background

SHOT is available as an open-source COIN-OR project[1]. A simplified version is also available for Wolfram Mathematica[2]; for more details see [36]. However, this version lacks many of the features described in this paper and its performance is not comparable to the main SHOT solver. In this paper we assume that we are dealing with a minimization problem to avoid confusion regarding the upper and lower objective bounds, even though SHOT, of course, solves both maximization and minimization problems. Therefore, the problem type considered is a convex MINLP problem of the form

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{x}), \\ \text{subject to} \quad & \mathrm{A}\mathbf{x} \leq \mathbf{a},\ \mathrm{B}\mathbf{x} = \mathbf{b}, \\ & g_k(\mathbf{x}) \leq 0 && \forall k \in K, \\ & \underline{x}_i \leq x_i \leq \overline{x}_i && \forall i \in I = \{1,2,\ldots,n\}, \\ & x_i \in \mathbf{Z} && \forall i \in I_Z, \\ & x_i \in \mathbf{R} && \forall i \in I \setminus I_Z. \end{aligned} \tag{1}$$

The objective function f can be either affine, quadratic or general nonlinear. It is furthermore assumed that the objective function and all nonlinear functions g_k (indexed by the set K) are convex and at least once differentiable. (Note that some NLP subsolvers may have other requirements.) It is assumed that the intersection of the linear constraints is a bounded polytope, *i.e.*, all variables (indexed by the set I) are bounded. In theory, SHOT work even if the functions are nonalgebraic but convex, and more generally as long as the implicit feasible region and the objective function is convex, however currently this is not implemented; therefore, in this paper, all functions are assumed to be compositions of standard mathematical functions expressible in the Optimization Services instance Language (OSiL) syntax as discussed in Section 6.2.

SHOT is based upon a technique for generating tight POAs closely integrated with the MIP solver, in combination with efficient primal heuristics. In [28] the extended supporting hyperplane (ESH) algorithm, forming the basis of the dual strategy in SHOT, was introduced to solve MIP problems in each iteration. This allows for a flexible and efficient way to utilize the performance gains made by the MIP solvers in recent years. In most solvers based on POA, a new branch-and-bound tree is created in each iteration where additional constraint in the form of hyperplane cuts are added. Information about

[1]Project website: `www.github.com/coin-or/SHOT`

[2]Project website: `www.github.com/andreaslundell/MathematicaSHOT`

previously found solutions can by communicated to the MIP solver between iterations, but a lot of unnecessary work is still performed. It is however possible to enhance the performance by tightening the integration by utilizing so-called lazy constraints in combination with MIP solver callbacks [35]. Callbacks are methods provided by the MIP solver that activate at certain parts of the solution process, *e.g.*, when a new integer feasible solution has been found or a new node is created in the branching tree. This provides great flexibility for the user to implement their own strategies that can impact the behaviour of the MIP solver in a way not possible through normal solver parameters.

Whenever a new integer-feasible solution is found by the MIP solver, the callback is activated and control is returned to SHOT. If this new solution fulfills also the nonlinear constraints in the original MINLP problem, and is better than the current best known solution, the so-called primal solution is updated. If the solution does not fulfill the nonlinear constraints, a (linear) lazy constraint is added that removes the invalid solution point from the MIP subproblem and tightens the POA. This cut can either be a supporting hyperplane [28] of the nonlinear feasible set or a cutting plane [55]. This procedure gives an increasing lower bound that, by its own, can solve the MINLP problem to a given tolerance on the nonlinear constraints. Convergence for the main SHOT algorithm follows naturally from the fact that its dual strategies ESH and ECP are convergent. Note however, that an integer-feasible solution is not guaranteed until the final ECP or ESH iteration; this can be circumvented by introducing an upper bound strategy based on primal heuristics. These primal strategies fulfill a very important role in SHOT, since they makes it possible to obtain integer-feasible solutions during the entire solution process. Thus, termination based on absolute and relative objective function gaps can be performed, significantly increasing the performance of the solver.

2.1 Primal and dual bounds

A *primal solution* is here considered to be a solution that fulfills all the constraints in the problem (1), including linear, nonlinear and integer constraints, to given user specified tolerances. The incumbent solution is the currently best know primal solution by the algorithm and its objective value is the current primal bound. Since we are considering minimization problems this is the one with the lowest objective value. Whenever a primal solution with a lower objective value is found, the incumbent is updated. In SHOT the primal solutions are found by a number of heuristics described in, Section 5.

A solution point not satisfying all constraints, *e.g.*, an integer-relaxed solution, but whose objective function value provides a valid lower bound of the optimum of problem (1), is here referred to as a *dual solution*. In SHOT, the dual solutions are obtained by utilizing a POA of the nonlinear constraints, and by solving either mixed-integer linear programming (MILP) or mixed-integer quadratic programming (MIQP) problems. In this paper, when mentioning MIP problems, we refer to either MILP or MIQP problems. The dual bounds are selected as the lowest objective bound returned by the MIP solver. Thus, all dual solutions provide a dual bound, but not all dual bounds give a dual solution. This ensures that a valid bound is obtained from the MIP solver even if the subproblem is not solved to optimality. Dual bounds can also be found by solving integer-relaxed problems, *i.e.*, linear programming (LP) or quadratic programming (QP) problems.

The primal and dual bounds can be utilized as quality measures of a solution and hence be used for determining when to terminate the solver. Since we always consider a minimization problem, the absolute and relative primal-dual objective gaps in SHOT are defined as

$$\mathrm{GAP}_{\mathrm{abs}} = \mathrm{PB} - \mathrm{DB} \qquad \text{and} \qquad \mathrm{GAP}_{\mathrm{rel}} = \frac{\mathrm{PB} - \mathrm{DB}}{|\mathrm{PB}| + 10^{-10}}, \tag{2}$$

where PB and DB are the current primal and dual bounds respectively. There are other alternative ways

to calculate the gaps, however this mimics the behaviour of the MIP solvers CPLEX and Gurobi, a necessity due to the tight integration with these solvers. Initially PB is defined as an infinitely large positive value and DB as an infinitely large negative number; the gaps are initialized to plus infinity.

3 Single- vs multi-tree approach

The standard strategy of implementing POA algorithms such as ECP, ESH and OA is to solve several separate MIP instances in a sequence. This provides a flexible and easily implemented strategy since no deep integration with the MIP solver is required, and a basic implementation can simply write a MIP subproblem to a file and call the subsolver on this problem. The solution from the optimization can then be read back into the MINLP solver and linearizations based on, *e.g.*, cutting planes or supporting hyperplanes, can be created and added to the MIP problem file before it is resolved. At a certain point some termination tolerance is met and the final solution to the MINLP problem has been found. This type of strategy is called multi-tree because a branch-and-bound tree is built in each call of the MIP solver. Methods that utilize this type of procedure are, *e.g.*, OA, and standard ECP and ESH algorithms.

In a so-called single-tree POA algorithm, only one master MIP problem is solved and lazy constraints are added through callbacks to cut away integer-feasible solutions that violate the nonlinear constraints in the MINLP problem. To understand how such an algorithm works, one can imagine that supporting hyperplanes have been generated in every point on the boundary of the feasible region described by the nonlinear constraints in the MINLP problem. The optimal solution to this MIP subproblem would then, of course, provide the optimal value to the original MINLP problem. However, it is neither possible nor required to add all of these constraints to obtain the optimal MINLP solution; when utilizing lazy constraints, the supporting hyperplanes are only added as needed to cut away MIP solutions not feasible in the nonlinear model. Thus, it can be imagined that the model has all the constraints required to find the optimal solution to the MINLP problem, but only those that are actually needed have been activated.

As soon as a new integer-feasible solution is found by the MIP solver, the callback is activated, and it is possible to determine whether a cut should be generated that remove the point or not. Since we are not required to restart the solution procedure as when adding normal constraints in multi-tree algorithms, the same branch-and-bound tree can be utilized.

Another solution method that utilizes a single-tree approach is the LP/NLP based branch-and-bound algorithm [46], which integrates OA and BB, to reduce the amount of work spent on solving MILP subproblems in OA. This technique is implemented in the MINLP solvers BONMIN [6], AOA [25], FilMINT [1] and Minotaur [40], and has previously been shown to significantly reduce the number of nodes required in the branching tree [32].

4 Dual strategy based on polyhedral outer approximation

In this section, we consider the dual strategies available in SHOT, *i.e.*, strategies for updating the POA, and techniques to efficiently integrate the strategies with the MIP solver. We describe how to efficiently implement the multi-tree approach by using early termination of the MIP solver and the MIP solution pool, as well as, how to construct an algorithm utilizing the single-tree approach with its MIP callbacks and lazy constraints.

In SHOT, a POA is created (by the means of the ECP and ESH algorithms) to represent the nonlinear feasible region of the MINLP problem. The approximation is updated iteratively to remove the

previous solution point(s) to the MIP problem, thus improving the POA. What differentiates ECP and ESH is how the linearizations are generated. In the ECP method, the solution point in the current (k-th) iteration $\mathbf{x}_k$ is directly used to generate the cutting plane

$$g_m(\mathbf{x}_k) - \nabla g_m(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) \leq 0.$$

Normally, g_m is the constraint function with the largest (positive) value in the point $\mathbf{x}_k$. In the ESH algorithm, a simple one-dimensional root search is utilized to approximately project $\mathbf{x}_k$ onto the integer-relaxed feasible set. This root search is performed on the line connecting $\mathbf{x}_k$ and an interior point $\mathbf{x}_{\text{int}}$ to find the point where the line intersects with the boundary of the integer-relaxed feasible set, *i.e.*, to find $\mathbf{x}$ such that

$$G(\mathbf{x}) := \max_j g_j(\mathbf{x}) = 0.$$

A supporting hyperplane is then generated to the integer-relaxed nonlinear feasible region in this point. The interior point must fulfill both the linear and nonlinear constraints in problem (1), and while it can fulfill also the integer requirements, this is not a necessity. Methods for obtaining an interior point are described in Section 4.2.

In the following sections, some enhancements to the standard dual strategies are described that help to improve the efficiency of the ESH and ECP algorithms. Many of these enhancements are not unique to these algorithms, but may be integrated into other methods based on POA such as the OA algorithm to enhance performance.

4.1 Handling problems with a nonlinear objective function

In addition to adding linearizations of the nonlinear constraints, a nonlinear objective must also be represented in the MIP problem. This can either be accomplished by rewriting problem (1) into the so-called epigraph form by introducing an auxiliary variable μ, *i.e.*, the variable vector is replaced as $\mathbf{x} := (\mathbf{x}, \mu)$, and setting the objective function $f(\mathbf{x}) := \mu$. The original nonlinear objective function, from now on denoted as $\tilde{f}$, can now be moved into an objective function constraint $\tilde{f}(\mathbf{x}) - \mu \leq 0$, which is included in the set of nonlinear constraints. This reformulated problem is now equivalent to the original one in the sense that they have the same solution in the original variables.

If all nonlinearities in the reformulated MINLP problem is ignored to get the initial dual MIP problem, the auxiliary variable μ is not present in any constraints. As the MIP solver will try to minimize the value of the μ variable that constitutes the objective function in the subproblems, initially a very low solution value for this variable is returned. However, as linearizations are performed for the added objective constraint function $\tilde{f}(\mathbf{x}) - \mu$ in the same manner as described above for the normal constraints, the solution value for μ will sooner or later increase. A constraint such as $\tilde{f}(\mathbf{x}) - \mu \leq 0$ has the property that if a cutting plane is generated for it, the cutting plane will automatically become a supporting hyperplane [52].

Although this strategy to treat the nonlinear objective constraint in the same manner as all other nonlinear constraints will work, there is a benefit to instead performing a one-dimensional root search in the μ-direction to obtain an interval $[\underline{\mu}, \overline{\mu}]$ that contains the value of μ that exactly contains the root $\tilde{f}(\mathbf{x}) - \mu = 0$. Then a supporting hyperplane is generated in this exterior point, while the interior point is a good primal candidate since it fulfills all constraints in the current MIP problem. This is described in more detail in Algorithm 4 and illustrated in Figure 1.

An objective linearization does not necessarily need to be added in each ESH or ECP iteration if not all the nonlinear constraints are fulfilled. However, it is often best to get as tight linear representation

of the objective function as possible, so therefore this procedure is performed in each iteration in SHOT, and on all available MIP solutions.

4.2 Obtaining an interior point in the ESH algorithm

There are currently two main strategies for finding one or more interior points for use in the ESH algorithm. The first is based on solving the following minimax reformulation of problem (1):

$$\begin{aligned} \text{minimize} \quad & \nu, \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{a},\ B\mathbf{x} = \mathbf{b} \\ & g_k(\mathbf{x}) - \nu \leq 0 \qquad \forall k \in K, \\ & \underline{x}_i \leq x_i \leq \overline{x}_i,\ \ x_i \in \mathbf{R} \qquad \forall i \in I, \\ & \nu \in \mathbf{R}. \end{aligned} \tag{3}$$

The problem formulation above, which is a NLP problem, assumes that the objective function in the original MINLP problem is affine. Problem (3) can either be solved utilizing an NLP solver, or our own tailor-made cutting plane based method described in Section 6.5. Note that the problem normally does not need to be solved to optimality, as long as a valid interior point is found. The cutting plane method is preferred, since it can easily be terminated as soon as a valid interior point has been found; also, some of the cuts generated while solving the minimax problem may also be utilized in the dual strategy in SHOT.

The second option for obtaining the interior point is to solve an integer-relaxed version of MINLP problem where the objective function is a constant (zero) with an interior point based solver such as IPOPT [51]. An interior point-based NLP solver will then return the analytic center of the nonlinear, integer-relaxed feasible set, as the solution point, and this is of course is a valid interior point as well. Another option is to utilize the center cut algorithm [29].

4.3 Generating multiple hyperplanes per iteration

In the standard ECP and ESH algorithms only one hyperplane cut is generated in each iteration of the algorithms. However, in the ECP algorithm additional cutting planes can be generated for more than one of the nonlinear constraints not fulfilled by the point $\mathbf{x}_k$; in theory, a cutting plane can be generated for all constraints, but in practice this may result in a too large increase in the size of the MIP subproblems. In the ESH algorithm the root search with respect to the max function G will often only give one active constraint in the found root, and thus only one supporting hyperplane will be obtained. However, it is also possible to do root searches on the individual constraint functions g_j instead of the max function. This is a feasible strategy as long as the function evaluations of g_j are computationally cheap (which almost always is the case whenever the functions are defined as compositions of standard mathematical functions). If function evaluations are expensive, or it is not possible to evaluate functions in certain points (*e.g.*, only at some discrete values), the ECP method is preferable to the ESH method as it only requires function and gradient evaluations in the MIP solution point $\mathbf{x}_k$.

As generally more integer-feasible solutions are found in the single-tree than in the multi-tree strategy, the cut generation pace should be more moderate in the former. This can be controlled, *e.g.*, by reducing the number of individual constraints the root search is performed against. It is also possible to allow the MIP solver to remove lazy constraints that it deems are unnecessary to reduce the number of constraints in the model.

4.4 Early termination of MIP problems

An important aspect for obtaining good performance in the multi-tree strategy is to initially solve the MIP subproblems to feasibility instead of optimality. In fact, only the final iteration need to be solved to optimality. In this way, we quickly obtain points to generate new hyperplanes in and in the process rapidly a tight initial polyhedral approximation. In SHOT this strategy is accomplished by utilizing the solution limit termination criterion of the MIP solvers. When the number of integer-feasible solutions are found, the MIP solver terminates, regardless of the MIP objective gap and other termination criteria.

The solution limit parameter is initially set to the value one, *i.e.*, the MIP solver terminates as soon as a solution fulfilling all linear and integer constraints in the MIP problem is found. If the solution status returned from the MIP solver is that the solution limit parameter has been reached and the first solution in the solution pool does not fulfill one or more of the nonlinear constraints (to a given tolerance), linearizations are generated by the ESH or ECP methods and the solver proceeds to the next iteration. However, if the best solution in the solution pool fulfills also the nonlinear constraints to a user set tolerance, the solution limit parameter is increased, and the MIP solver will continue without rebuilding the tree to find additional solutions. The rebuilding is avoided since no new hyperplane cuts are added to the MIP model in these iterations. (Cuts are still generated for the solution points in the solution pool that do not fulfill the nonlinear constraints, but these are added in the next iteration without solution limit update.) To help improve the efficiency, limits on how many iterations can be solved before forcing a solution limit update may also be introduced.

This strategy was previously described in [56] and implemented in the GAECP solver [54] in a similar manner. In the AlphaECP solver in GAMS [53], instead of the solution limit, the relative MIP gap termination is used to roughly accomplish the same thing. Here, the relative MIP gap termination is reduced in steps to finally give optimal solutions the MIP subproblems.

4.5 Solving quadratic subproblems

As described in [28] it is also possible to solve the subproblems of the type QP or MIQP if the original problem is quadratic and the MIP subsolver supports it. In this case, the strategies for handling a nonlinear objective function described in Section 4.1 are not needed as the objective function is by construction exact in the subproblems. Of the MIP solvers available in SHOT, CPLEX and Gurobi can solve MIQP problems, while Cbc does not directly provide this functionality.

CPLEX and Gurobi also support convex quadratic constraints, and SHOT has the functionality to directly pass these on to these subsolvers. In this case, no hyperplane cuts are generated for these constraints. However, numerical experiments have shown that it is often preferable to consider the nonlinear constraints as general nonlinear and handle them using the ECP or ESH algorithms. In this case MIQP problems are solved in SHOT if the objective function is quadratic, and MILP problems otherwise.

4.6 Solving integer-relaxed subproblems

Since integer-relaxed problems are often solved much faster than corresponding MIP problems, these can be exploited to enhance the performance of the algorithm. The first technique, based on solving LP or QP problems, is to be used as a presolve strategy and can be utilized in both the single- or multi-tree strategies, while the second and third strategy are for the multi-tree and single-tree strategies respectively.

It is possible to initially ignore the integer constraints of the problem and rapidly solve a sequence of LP or QP problems to obtain a tight POA as a presolve step. Since each subproblem is solved to optimality in this case, the solution always provides a dual bound that can be utilized by SHOT. As soon as a maximum number of LP or QP problems have been solved, or the progress of the dual bound has stagnated, *i.e.*, the integer-relaxed solution to the MINLP problem has been obtained, the integer constraints are activated. After this, MIP subproblems are solved that include all the hyperplane cuts generated when solving the integer-relaxed problems. If SHOT is tasked to solve an MINLP problem without integer variables, *i.e.*, an NLP problem, no additional iterations are required after this step.

Another relaxation strategy can be used whenever the same integer combination is obtained repeatedly in several subsequent iterations. Here, the integer variables are fixed to these values and a sequence of integer-relaxed LP or QP problems are solved. The idea in this strategy is similar to fixing the integer variables in the MINLP problem and instead solving NLP problems as described in Section 5.2. However, it can be more efficient to solve LP or QP problems instead of NLP problems, and we can also utilize the generated hyperplane cuts in the main MIP problem afterwards. Note that the solutions to these subproblems cannot be used to update the dual bound since it is not known whether the selected integer combination is the optimal one. Primal candidates found, are however, still valid for the main problem.

The previous strategy cannot be used in the single-tree strategy as it would require that the branching tree is rebuilt both before and after. However, an alternative method can be implemented in the single-tree strategy if the MIP solver also provides callbacks that activate on relaxed solutions. Then linearizations can be generated also in these points. Note that in both CPLEX and Gurobi, the generated constraints must be saved temporarily and added later when a new integer-solution is found, as it is not possible to add lazy constraints in all callback types. This strategy will most often give rise to a significant increase in the number of lazy constraints in the model, and should be used moderately.

5 Primal strategy based on heuristics

Any solution that fulfills all the constraints in the MINLP problem is a candidate to become the new incumbent, and thus it does not matter how a primal solution is obtained. This enables the usage of heuristic strategies for finding such solutions. Since these affect when the solution process is terminated, *cf.*, Eq. (2), they have a significant impact on the efficiency of the solver implementation as well as the quality of the solutions returned. In SHOT there are a number of such strategies implemented, and these are described in this section. Other primal heuristics currently not implemented, but planned, include the center-cut algorithm [29] and feasibility pumps [4, 7].

5.1 Utilizing the MIP solution pool

The simplest primal heuristic implemented in SHOT is to utilize valid alternative solutions to the MIP problem that also fulfill the nonlinear constraints. If the single-tree strategy is used, such solutions may be found during the MIP search and these can be utilized as primal candidates. In the multi-tree strategy, the so called solution pool containing nonoptimal integer-valid solutions found during the solution process, can be checked for solutions fulfilling the nonlinear constraints. Also, as described in Section 4.4, there is normally no need to solve all MIP problems to optimality, and in this case the MIP solver can terminate with MIP solutions fulfilling the nonlinear constraints.

The methods described above do not cost anything extra performance-wise for SHOT; however, some MIP solvers also provide a functionality to actively search for additional solutions after the

solution process has terminated, *i.e.*, to populate the solution pool. This will, of course, add an extra time penalty, so a trade-off is needed on how much time is spent on finding primal candidates. This functionality is currently not implemented in SHOT.

5.2 Solving fixed NLP problems

When an integer solution has been found by, *e.g.*, the MIP solver, it is possible to fix the integer variables to their corresponding values and to solve the corresponding NLP problem. If the solution to this problem is feasible for the MINLP problem, a new primal solution candidate has been obtained. If the NLP solution point does not fulfill the nonlinear constraints, this point can be used to either generate new cutting planes or, through root searches to interior points, supporting hyperplanes. If all the integer variables in the problem are binaries, also an integer cut can be added to filter out this integer combination in the future. Note that SHOT will behave similarly to an OA-type algorithm if fixed NLP problems are solved in each iteration.

5.3 Utilizing information from root searches

As mentioned in Section 4.1, if the objective function of the MINLP problem is nonlinear, a root search on the added nonlinear objective constraint can be utilized to find an interior point. This point fulfills all linear constraints in the problem and is integer-feasible, and is thus, a good candidate for a primal solution.

Generally, whenever a root search is performed in SHOT, a point on the interior of the integer-relaxed feasible set is obtained, and this point is tested to see if it may be a primal solution, *i.e.*, if it fulfills all constraints in the original problem. Most often, these points do not fulfill linear equality constraints. They may also not fulfill the integer requirements for one or more variables, which naturally invalidates them. One possibility, still not implemented in SHOT, is to utilize some rounding and projection heuristic on these partial primal solutions.

6 The SHOT solver implementation

The SHOT solver is based on the dual strategies discussed in Section 4 in combination with the primal heuristics discussed in Section 5. The basic steps in the integrated primal-dual method is presented as the SHOT algorithm in Algorithm 1. To make the algorithm easier to understand, some of the functionality, including the main steps in the single- and multi-tree algorithms are extracted into Algorithms 2–4.

SHOT is implemented in C++ 11 and released as an COIN-OR open-source project [37]. Its external dependencies include the COIN-OR projects Optimization Services (OS) [19] for problem file handling and function evaluations, Cbc [14] for solving MILP subproblems and IPOPT [51] for solving NLP subproblems. SHOT can also interface with commercial MIP and NLP solvers, as described later on in this section. In addition to these external dependencies, SHOT also utilizes functionality in the Boost libraries [48] for root and line search functionality as well as a few other auxiliary helper functions.

The SHOT solver is itself very modular and is based on a system where most functionality is contained in specific tasks. This makes it easy to change the behaviour of the solver by defining solution strategies that contain a list of specific tasks to perform in sequence. Since there are also tasks that perform in a similar manner to if- and goto-statements in normal programming languages, iterative algorithms can be easily implemented without modifying the underlying tasks. The tasks

Algorithm 1 Main steps of the SHOT algorithm.

Specify absolute and relative objective termination values ε_{rel} and ε_{abs}, iteration limits $K_{\max}$ and $K_{\max}^{\text{LP}}$ for the main iteration and initial relaxed steps respectively, and a time limit $T_{\max}$. If the multi-tree strategy is selected, specify the threshold ε_{SL} and iteration count $\text{SL}_{\max}$ for updating the solution limit. Initialize best known dual ($\text{DB} = -\infty$) and primal ($\text{PB} = +\infty$) bounds. Parse the problem.

1. *MIQP strategy*: If the objective function f is quadratic and no nonlinear constraints are present, solve the problem directly with a MIQP solver (if available) and set `MIPstatus` to the value `optimal`, `feasible`, `unbounded`, `timelimit` or `infeasible` depending on the return status of the subsolver. Set $\text{DB} = \text{DB}_{\text{MIQP}}$, *i.e.*, the dual bound obtained from the MIQP solver. If a solution $\mathbf{x}$ is found, set $\mathbf{x}^* = \mathbf{x}$, $\text{PB} = f(\mathbf{x}^*)$ and update GAP_{rel} and GAP_{abs} according to Eq. (2). Then go to Step 6.
2. *Interior point step*: If the ESH method is to be used for generating cuts, find an interior feasible point $\mathbf{x}_{\text{int}}$ to problem (1) using either the minimax strategy in Algorithm 5, or one of the other methods mentioned in Section 4.2. If no interior point is found, use the ECP method until a primal solution has been found.
3. *Initialize the MIP problem*: Set the time limit in the MIP solver to what time is remaining of $T_{\max}$ and its relative and absolute MIP gaps to ε_{rel} and ε_{abs}. Create a MIP problem based on the original MINLP problem containing all variables and constraints except for the nonlinear ones. If the objective is linear or quadratic, use it directly. Otherwise denote the original nonlinear objective function with $\tilde{f}(\mathbf{x})$, introduce an auxiliary variable μ, *i.e.*, $\mathbf{x} := (\mathbf{x}, \mu)$, and set the MIP objective to minimize this variable, *i.e.*, $f(\mathbf{x}) := \mu$. If the minimax solver in Algorithm 5 was used in Step 2, the cutting planes generated on the exterior of the integer-relaxed feasible set of problem (1), are added to the dual problem.
4. *Integer-relaxation step*: Set the iteration counter $k := 0$ and relax the integer-variables in the MIP problem, *i.e.*, we now have a LP or QP problem. While $k < K_{\max}^{\text{LP}}$ repeat:
   a) Solve the relaxed problem to obtain the solution $\mathbf{x}_k$ and dual bound DB_{MIP} from the subsolver. If no solution is obtained, set `MIPstatus:=infeasible` and go to Step 6.
   b) If $G(\mathbf{x}_k) < \varepsilon^{\text{LP}}$, terminate the integer-relaxation step and go to Step 5.
   If $G(\mathbf{x}_k) > \varepsilon^{\text{LP}}$ generate linearizations according to Algorithm 4 and increase the counter $k := k+1$.
5. *Main iteration*:Activate the integer constraints in the MIP problem. Select one of the following:
   a) *Single-tree strategy:* Enable a callback in the MIP solver performing the steps in Algorithm 2 whenever a new integer-feasible solution $\mathbf{x}_k$ is found. Solve the subproblem until the MIP solver terminates, *e.g.*, when its relative or absolute gaps have been met. Set `MIPstatus` to either `optimal`, `feasible`, `timelimit` or `infeasible` depending on the return status. Go to Step 6.
   b) *Multi-tree strategy*: Set the MIP solver's solution limit to $\text{SL} := 1$ and the iteration counter $k := 0$. Denote with HPS a list of not added hyperplane linearizations. Repeat while $k < K_{\max}$:
      i. Solve the current MIP problem and set `MIPstatus` to the termination status of the MIP solver.
      ii. If `MIPstatus` is `infeasible` or `unbounded` go to Step 6. Otherwise, denote the solution point with the best objective value in the solution pool as $\mathbf{x}_k$.
      iii. If `MIPstatus=solutionlimit`:
         A. Perform the steps in Algorithm 3 to generate the hyperplane cuts and add them to the list HPS instead of the MIP problem.
         B. If the solution limit has not been updated in $\text{SL}_{\max}$ iterations or if $G(\mathbf{x}_k) < \varepsilon_{\text{SL}}$, increase the limit $\text{SL} := \text{SL}+1$. Otherwise, add all linearizations in HPS to the MIP model.
      iv. If `MIPstatus` is `optimal`, perform the steps in Algorithm 3.
      v. Increase the iteration counter $k := k+1$.
6. *Termination*:

   If the objective function is nonlinear, remove the last component (the value of the objective variable) of the solution $\mathbf{x}^*$.

   - If $\text{GAP}_{\text{abs}} < \varepsilon_{\text{abs}}$ or $\text{GAP}_{\text{rel}} < \varepsilon_{\text{rel}}$, terminate with the status `optimal` and the solution $\mathbf{x}^*$.
   - If $\text{PB} < \infty$, *i.e.*, a primal solution is found but the algorithm has been unable to verify optimality to the given tolerances, return with the status `feasible` and the solution $\mathbf{x}^*$.
   - If `MIPStatus` is `timelimit`, `iterlimit`, `unbounded` or `infeasible`, return this status.

Algorithm 2 Steps performed in the callback activated by the MIP solver whenever a new integer-feasible solution is found.

Denote the MIP feasible solution found with $\mathbf{x}$, and the current dual bound provided by the MIP solver by DB_{MIP}.

1. *Update dual bound*: If $\text{DB}_{\text{MIP}} > \text{DB}$, set $\text{DB} := \text{DB}_{\text{MIP}}$; update GAP_{rel} and GAP_{abs}.
2. *Update incumbent*: If $G(\mathbf{x}) \leq 0$ and $f(\mathbf{x}) < \text{PB}$, set $\text{PB} := f(\mathbf{x})$ and $\mathbf{x}^* := \mathbf{x}$; update GAP_{rel} and GAP_{abs}.
3. *Check termination criteria*: If $\text{GAP}_{\text{abs}} < \varepsilon_{\text{abs}}$ or $\text{GAP}_{\text{rel}} < \varepsilon_{\text{rel}}$ exit the callback without excluding the current solution $\mathbf{x}$ with a lazy constraint.
4. *Update interior point*: If $G(\mathbf{x}) < 0$ and the ESH method is used but no point $\mathbf{x}_{\text{int}}$ is yet available, set $\mathbf{x}_{\text{int}} := \mathbf{x}$.
5. *Generate linearizations*: If $G(\mathbf{x}) > 0$ generate one or more linearizations according to Algorithm 4 and add these to the MIP model as lazy constraints.
6. *Primal search*: Perform one or more of the primal heuristics mentioned in Section 5 to reduce PB. If PB is updated pass it on to the MIP solver as the new incumbent and recalculate GAP_{rel} and GAP_{abs}.
7. Increase the iteration counter $k := k+1$. If $k = K_{\max}$ set `MIPstatus` to `iterlimit` and go to Step 6 in Algorithm 1.

Algorithm 3 Main iterative step in the multi-tree strategy.

Denote the j-th integer feasible solution in the current MIP solution pool with with $\mathbf{x}_j$, and the dual bound provided by the MIP solver by DB_{MIP}. It is assumed that the dual solutions in the solution pool are ordered in ascending order depending on the MIP objective value of the solutions.

1. *Update dual bound*: If $\text{DB}_{\text{MIP}} > \text{DB}$, set $\text{DB} := \text{DB}_{\text{MIP}}$; update GAP_{rel} and GAP_{abs}.
2. *Repeat for each point in the solution pool:*
   a) *Update incumbent*: If $G(\mathbf{x}^j) \leq 0$ and $f(\mathbf{x}_j) < \text{PB}$, set $\text{PB} := f(\mathbf{x}_j)$ and $\mathbf{x}^* := \mathbf{x}_j$; update GAP_{rel} and GAP_{abs}.
   b) *Update interior point*: If $G(\mathbf{x}_j) < 0$ and the ESH method is used but no interior point is yet known, set $\mathbf{x}_{\text{int}} := \mathbf{x}_j$.
   c) *Generate linearizations*: If $G(\mathbf{x}_j) > 0$ generate one or more linearizations according to Algorithm 4.
3. *Primal search*: Perform one or more of the primal heuristics mentioned in Section 5 to reduce PB. If PB is updated, recalculate GAP_{abs} and GAP_{rel} and pass the new primal solution on to the MIP solver.
4. If $\text{GAP}_{\text{abs}} < \varepsilon_{\text{abs}}$ or $\text{GAP}_{\text{rel}} < \varepsilon_{\text{rel}}$ or if $k = K_{\max}$ set `MIPstatus` to `optimal` or `iterlimit` respectively, and go to termination in Algorithm 1. Otherwise increase the iteration counter $k = k+1$.

Algorithm 4 Algorithm for generating cuts for unfulfilled nonlinear constraints.

Assume that a point $\mathbf{x}_{\text{ext}}$ is given that lies on the exterior of the integer-relaxed feasible region of problem (1).

1. If there are nonlinear constraints, the ESH dual strategy is used and an interior point $\mathbf{x}_{\text{int}}$ is known:
   a) Perform a root search on the line between $\mathbf{x}_{\text{int}}$ and $\mathbf{x}_{\text{ext}}$ with respect to the max function $G(\mathbf{x})$. This gives an interval $[\underline{t},\bar{t}] \subset \mathbf{R}$, so that for $\tilde{t} \in [\underline{t},\bar{t}]$, $G(\mathbf{p}(\tilde{t})) = 0$, and thus, $\mathbf{p}(\underline{t})$ and $\mathbf{p}(\bar{t})$ is on the exterior and interior of the integer-relaxed feasible set $G(\mathbf{x}) \leq 0$ respectively. It is also possible to perform root searches with respect to the individual constraints with $g_j(\mathbf{x}_{\text{ext}}) > 0$ instead of the max function G. In this case several points $\mathbf{p}_j(\underline{t})$ are obtained. The interior points found, *i.e.*, $\mathbf{p}(\bar{t})$ or $\mathbf{p}_j(\bar{t})$ can be used, *e.g.*, as the starting point in a primal heuristic.
   b) Generate a supporting hyperplane in $\mathbf{p}(\underline{t})$ for the constraint g_j with $j = \text{argmax}_j\, g_j(\mathbf{p}(\underline{t}))$:

$$g_j(\mathbf{p}(\underline{t})) + \nabla g_j(\mathbf{p}(\underline{t}))^T (\mathbf{x} - \mathbf{p}(\underline{t})) \leq 0$$

   or if individual root searches were performed for all constraints $g_j(\mathbf{p}_j(\underline{t})) > 0$:

$$g_j(\mathbf{p}_j(\underline{t})) + \nabla g_j(\mathbf{p}_j(\underline{t}))^T (\mathbf{x} - \mathbf{p}_j(\underline{t})) \leq 0.$$

2. If there are nonlinear constraints and the ECP dual strategy is used (or the ESH dual strategy is used, but no interior point is known): Generate cutting planes in $\mathbf{x}_{\text{ext}}$ for one or more of the violated constraints g_j, *e.g.*, at least for $j = \text{argmax}_j\, g_j(\mathbf{x}_{\text{ext}})$:

$$g_j(\mathbf{x}_{\text{ext}}) + \nabla g_j(\mathbf{x}_{\text{ext}})^T (\mathbf{x} - \mathbf{x}_{\text{ext}}) \leq 0.$$

3. If the objective function is considered as generally nonlinear, a root search is performed on the line connecting the points $\mathbf{x}_{\text{ext}}$ and $\mathbf{x}'_{\text{ext}}$ with respect to the auxiliary objective expression $\tilde{f}(\mathbf{x}) - \mu = 0$. Here $\mathbf{x}'_{\text{ext}}$ is a vector that fulfills the objective constraint $\tilde{f}(\mathbf{x}) - \mu < 0$, *e.g.*,

$$\left(x_1,\ x_2, \ldots,\ x_n,\ \tilde{f}(\mathbf{x}_{\text{ext}}) + \varepsilon\right), \quad \varepsilon > 0.$$

   Through the root search both an exterior point $\mathbf{p}(\underline{t})$ and an interior point $\mathbf{p}(\bar{t})$ are obtained. The interior point is a candidate for a new primal solution, and a supporting hyperplane for the nonlinear objective function is generated in the point $\mathbf{p}(\underline{t})$:

$$\tilde{f}(\mathbf{p}(\underline{t})) + \nabla \tilde{f}(\mathbf{p}(\underline{t}))^T (\mathbf{x} - \mathbf{p}(\underline{t})) \leq 0.$$

   This is illustrated in Figure 1 for a objective function of one variable. Note that, if the multi-tree strategy is used and the MIP solver has flagged the current solution as optimal, it will no longer be optimal when the value for the variable μ has been updated, since there is no guarantee (and most often is not the case) that the updated solution point is the optimal one for the current MIP problem.

are executed in a sequence defined by the solution strategy, but the order of execution can also be modified at runtime. Currently four main solution strategies are implemented: The single-tree and multi-tree strategies, a strategy for solving MIQP problems directly with the MIP solver, as well as one for solving problems without discrete variables, *i.e.*, NLP problems.

6.1 Solver options

Options are provided using either a scalar format based on the option file syntax in GAMS, or as an XML-file in the OSoL-format, which is part of the OS project. If no options file is specified, default options are used and options files `options.opt` and `options.xml` are created in the runtime directory. The options are organized in categories and subcategories, *e.g.*, `Termination.TimeLimit`, which is part of the termination category controls the time limit and can thus be given a double value in seconds. Other possible option types are strings, booleans and integer-valued options (sometimes also indicating logical choices). The settings are documented in the options file and in the solver documentation. A few selected options are mentioned in Appendix A.

6.2 Problem representation

SHOT can read problems in the XML-based OSiL-format [17] and AMPL-format [16] through the OS-interface. Through an interface to GAMS, it can also read problems in a scalar GAMS-format [10]. In all cases however, the model is currently translated into an object that uses the functionality from OS to obtain the problem structure, as well as calculate function values and gradients.

6.3 MIP solvers

SHOT currently interfaces with the commercial solvers CPLEX and Gurobi. This integration is done directly to the solvers, and not through the Osi-interface included in OS. The reason is that more functionality is utilized than provided by Osi, *e.g.*, the callbacks in the single-tree strategy. However, if Cbc is used as MIP solver, the interface is through Osi. Note that Cbc does not support quadratic objective functions so quadratic objectives are considered to be general nonlinear. The interface to Cbc, also does not support lazy callbacks so the single-tree strategy cannot be used with this solver. In general the Cbc interface in SHOT is not as mature as the CPLEX and Gurobi interfaces, and therefore usage one of the two commercial solvers is recommended.

By default, most MIP solvers work in parallel utilizing support for multiple concurrent threads on modern CPUs. This, however means that a fully deterministic algorithm is difficult, if not impossible, to achieve without hampering the performance too much since different solution paths lead to the same solution. Therefore, the solution times, and even *e.g.*, the amount of iterations, in SHOT may vary slightly between different runs. Some of the MIP solvers also have parameters that try to force the behaviour of the solver to be more deterministic, but total deterministic behaviour is not possible to achieve in most cases. The number of threads made available to the MIP solvers are controlled with the parameter `Dual.MIP.NumberOfThreads`, with the default 0 means that the choice is made by the MIP solver.

Some user specified options are passed on to the MIP solvers, including absolute and relative gap termination values and time limit. There are also a number of parameters that can be provided to the individual solvers; these parameters are documented in the solver manual.

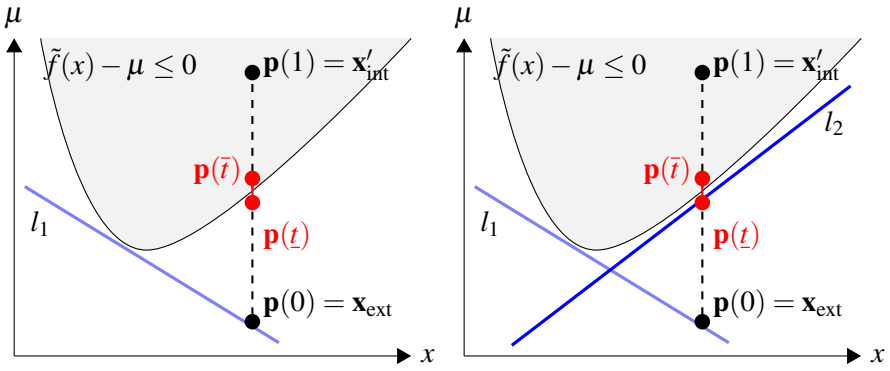

Figure 1: Illustration of the procedure to perform a root search on the objective function as described in Algorithm 4. The original objective function in the MINLP problem (1) is $\tilde{f}(x)$ and the variable vector $\mathbf{x} = (x, \mu)$. The thick boundary of the shaded region is where $\tilde{f}(x) - \mu = 0$, *i.e.*, the exact values of the original nonlinear objective function. The linearization l_1 of the objective function is the result of a previously generated objective linearization. In the figure to right, the objective function linearization l_2, generated in the point $\mathbf{p}(\underline{t})$, is illustrated. Note that, for the purpose of illustration, the interval between the two interval endpoints obtained through the root search is quite large, and therefore, the linearization in the figure is not an exact supporting hyperplane.

6.4 NLP solvers

SHOT currently directly supports only IPOPT as an NLP solver through the OS-interface. It is also recommended to utilize one of the HSL subsolvers with IPOPT [24] if available for maximum performance and stability. If GAMS is installed on the system, it is possible to utilize its NLP solvers, including IPOPT. Note that for GAMS NLP solvers to be used, the problem must be given in GAMS-syntax, *cf.*, Section 6.2. If available in GAMS, CONOPT is the recommended overall NLP solver because it has been found to be very stable in combination with SHOT.

6.5 Strategies for finding the interior point

If the ESH algorithm is to be used, SHOT will utilize one of the strategies mentioned in Section 4 to find an interior point, with the default being the tailor-made minimax solver described in Algorithm 5. If the selected method fails to find an interior point, the solution process continues by adding cutting planes until an interior point is found by either of the primal heuristics, and then switches back to generating supporting hyperplanes.

The cutting plane based strategy for obtaining the interior point has the good property that the cutting planes can be reused when solving the MIP subproblems. It also supports the functionality to terminate as soon as a valid interior point has been found, which is currently not possible when using IPOPT to solve the minimax problem. Therefore, the recommended (and default) strategy is to use the cutting plane minimax solver.

The interior point can be updated during the solution process, *e.g.*, it can be replaced with the current primal solution point; there is also an option to update the interior point an average between the current one and the current primal solution point. Furthermore, multiple interior points can be utilized to perform several root searches with different interior end points. This will decrease the impact of having a 'bad' interior point, however it will increase the number of supporting hyperplanes

Algorithm 5 Minimax algorithm for finding an interior point.

Specify a maximum number of iterations and a relative ($\varepsilon_{\text{rel}}^{\text{MM}}$) and absolute ($\varepsilon_{\text{abs}}^{\text{MM}}$) termination tolerance for the objective function. Set $\text{GAP}_{\text{rel}}^{\text{MM}} := \infty$ and $\text{GAP}_{\text{abs}}^{\text{MM}} := \infty$, and determine a maximal number of iterations $K_{\max}^{\text{MM}} \geq 2$. Set the iteration index $k := 1$.

1. Reformulate the original MINLP problem (1) into a minimax problem of the type in problem (3). Denote the function $G(\mathbf{x}) = \max_j g_j(\mathbf{x})$, *i.e.*, the maximum error of all nonlinear constraints functions in the original problem.
2. Create a copy of the minimax problem without the nonlinear constraints, *i.e.*, an LP problem.
3. Solve the LP problem. If the subproblem could not be solved to optimality, set `MinimaxStatus` to `failure` and go to Step 5; otherwise obtain the solution $(\mathbf{x}_1, v_1)$, and denote $\text{OBJ}_{\text{LP}} = v_1$.
4. Repeat while $k < K_{\max}^{\text{MM}}$.
   a) Increase the iteration counter, *i.e.*, set $k := k + 1$.
   b) Solve the LP problem. If the subproblem could not be solved to optimality, set `MinimaxStatus` to `failure` and go to Step 5; otherwise, obtain the solution $(\mathbf{x}_k, v_k)$ and set $\text{OBJ}_{\text{LP}} = v_k$.
   c) Solve the following one-dimensional minimization problem:
   $$\lambda_k = \text{argmin}_{\lambda \in [0,1]} G(\lambda \mathbf{x}_k + (1-\lambda)\mathbf{x}_{k-1})$$
   and calculate the solution point as $\mathbf{x}_{\text{LS}} = \lambda_k \mathbf{x}_k + (1-\lambda_k)\mathbf{x}_{k-1}$. Calculate $\text{OBJ}_{\text{LS}} = G(\mathbf{x}_{\text{LS}})$.
   d) Calculate the absolute and relative difference in objective values between the LP and line search:
   $$\text{GAP}_{\text{abs}}^{\text{MM}} = |\text{OBJ}_{\text{LP}} - \text{OBJ}_{\text{LS}}|, \qquad \text{and} \qquad \text{GAP}_{\text{rel}}^{\text{MM}} = \frac{|\text{OBJ}_{\text{LP}} - \text{OBJ}_{\text{LS}}|}{|\text{OBJ}_{\text{LP}}| + 10^{-10}}.$$
   If $\text{OBJ}_{\text{LP}} \leq 0$, and $\text{GAP}_{\text{rel}}^{\text{MM}} < \varepsilon_{\text{rel}}^{\text{MM}}$ or $\text{GAP}_{\text{abs}}^{\text{MM}} < \varepsilon_{\text{abs}}^{\text{MM}}$, set `MinimaxStatus` to `success` and go to Step 5.
   e) Select the constraint function with the largest error in the point $\mathbf{x}_{\text{LS}}$, *i.e.*, find
   $$j' = \text{argmax}_j g_j(\mathbf{x}_{\text{LS}})$$
   and add the following cutting plane to the LP problem
   $$g_{j'}(\mathbf{x}_{\text{LS}}) + \nabla g_{j'}(\mathbf{x}_{\text{LS}}))^T (\mathbf{x} - \mathbf{x}_{\text{LS}}) \leq 0.$$
   Note that it is also possible to generate cutting planes for more than one of the unfulfilled constraints.
5. *Termination:* If `MinimaxStatus` is `success`, return with the status `success` and the interior point $\mathbf{x}_{\text{LS}}$, otherwise with status `failure`.

generated, and thus, the complexity of the MIP subproblem(s).

6.6 Root and line search functionality

Since root searches are utilized extensively in SHOT, both for finding the point to generate supporting hyperplanes and in primal heuristics, an efficient and numerically stable root search implementation is required. The `roots` package in the Boost library [48], more specifically its implementation of the TOMS 748 algorithm [2], or a bisection-based method are provided. Functionality for minimizing a function along a line is also provided by Boost through the `minima` library. This functionality is used in the minimax solver for obtaining an interior point of the nonlinear feasible set described in Algorithm 5. There are several parameters controlling the root and line search functionality in SHOT. However, in most cases these can be left to their default values. Also, if a root search fails when generating a supporting hyperplane because of numerical issues, a cutting plane is always added, which significantly improves the stability in extreme cases.

Note that when performing a root search on a max function $G(\mathbf{x}) := \max_j g_j(\mathbf{x})$ (with convex component functions) on a line between an interior point $\mathbf{x}_{\text{int}}$ and an exterior point $\mathbf{x}_{\text{ext}}$, often not all constraint functions g_j need to be evaluated, but only those component functions that are positive in the exterior point. Also all functions that get a negative value during subsequent evaluations on the line connecting the interior and exterior point can be disregarded in the rest of the current root search iterations, as they will not be active in the final point due to the convexity of the functions g_j.

6.7 Termination

The SHOT solver is mainly terminated based on the relative and absolute objective gaps, *cf.*, equation 2. However, also time and iteration limits are considered. The iteration limit behaves differently depending on whether the single- or multi-tree methodology is used. In the multi-tree case, the iterations are counted as number of LP, QP, MILP or MIQP iterations performed, and thus correspond to the number of dual subproblems solved. In the single-tree case, the number of times a new integer-feasible solution has been found, *i.e.*, the times the lazy callback has been activated, is regarded as an iteration. Note that for the same problem, the number of iterations are often much higher in the single-tree case, but the solution time per iteration is often less. Solving NLP problems are not counted as main iterations.

In the multi-tree case, there is also the option to terminate whether the maximum constraint deviation in the MIP solution point $\mathbf{x}_k$ is fulfilled, *i.e.*, $G(\mathbf{x}_k) \leq \varepsilon_g$. In this case, the solution status of the current subproblem must be flagged optimal by the MIP solver. This termination criterion is the normal one in the standard ESH an ECP algorithms, motivating its inclusion in the SHOT solver. Note however, that a primal solution need not be found in this case, and therefore the solution status of the solver may indicate a failure.

6.8 Solver results and output

The obtained solution results and statistics are returned as an XML-file in the OSrL-format (by default in the directory where the problem file is located). Also, a GAMS trace file is created. The trace file does not contain the variable solution vector, only the dual and primal bounds on the objective value in addition to statistics about the solution process. The trace files are mainly provided for benchmarking purposes as these can be directly read by PAVER [9], the tool used in the benchmarks provided in the next section.

For debugging purposes the option switch `Output.Debug` will create files in a directory called `problem debug/problemname/` in the runtime directory. These include subproblems in text format, solution points, *etc*. The amount of output shown during the solution process, in the console and in the log file respectively, is controlled by the switches `Output.LogLevelConsole` and `Output.LogLevelFile`. There are also several other options to control how much information is shown to the user.

7 Numerical comparisons

In this section, SHOT is compared to some state-of-the-art MINLP solvers to illustrate its performance. To analyze a few of the different strategy choices in SHOT, some internal benchmarks are also presented, as is a comparison between using CPLEX and Gurobi as MIP subsolvers. The benchmark set is the 366 convex MINLP problems in MINLPLib [42], and these problems are listed in Appendix B. The performance of the solvers was analyzed using PAVER 2.0 [9], and the generated PAVER reports are available at `andreaslundell.github.io/minlpbenchmarks`.

All comparisons were performed on a computer with an Intel Xeon 3.6 GHz processor with four physical cores (with the possibility to process eight threads) and 32 GB memory. The MIP solver utilized in SHOT (version 0.9.2) was CPLEX (12.6.3) and Gurobi (8.0). The CPLEX version is not the latest one (at the time of writing version 12.8); the reason for using an older version is that CPLEX in combination with callbacks is not as stable in the newer versions. Since Cbc lacks the possibility to add lazy constraints and solve MIQP problems, it was not included as a subsolver in SHOT in the benchmarks. The NLP subsolver selected in SHOT was CONOPT 3.17G utilized through the GAMS interface. The termination criteria used was $\text{GAP}_{\text{rel}} = 0.1\%$, $\text{GAP}_{\text{abs}} = 0.001$ and a time limit of 900 seconds. The cutting plane minimax method mentioned in Section 6.5 was used to find the interior end points for the root searches in the ESH strategy.

7.1 Comparisons to other MINLP solvers

In [28], it was shown that an early version of SHOT was very competitive. However, the SHOT solver has been significantly updated since the publication of that paper, as have many of the other solvers. Therefore, a new, more extensive, comparison is conducted in this paper. The solvers included in the comparison are the GAMS solvers AlphaECP (version 2.10.06), BARON (version 17.10.16), BONMIN (version 1.8), DICOPT (version 2), SBB (version 25.0.3) and SCIP (version 5.0). The AIMMS solver AOA (version 4.53.1), and the solver Minotaur (release 05-21-2018) were also considered. If the solvers had a convex strategy, or recommended parameters for convex problems, these were used. For the solvers with multiple main strategies we selected one: in BONMIN the outer-approximation strategy, and for AOA and Minotaur their LP/NLP-based branch-and-bound strategies. The reason we selected these, is that they had the best performance (of the available methods in BONMIN, Minotaur and AOA) in the benchmark provided in [27]. The GAMS solvers utilized CPLEX 12.8 and CONOPT 3.17G as subsolvers. The solver AOA is implemented in AIMMS 4.53.1 and utilized CPLEX 12.8 and CONOPT 3.14V. Some additional parameters were used to prevent the solvers to terminate prematurely, and these parameters are listed in Appendix C.

In [27] an extended set of MINLP solvers is considered, and their performance is analyzed in more detail. In this paper, however, the main motivation behind the comparison, has been to see how SHOT compares to the state-of-the-art.

The comparisons are shown as a solution profile in Figure 2, and some statistics are provided in

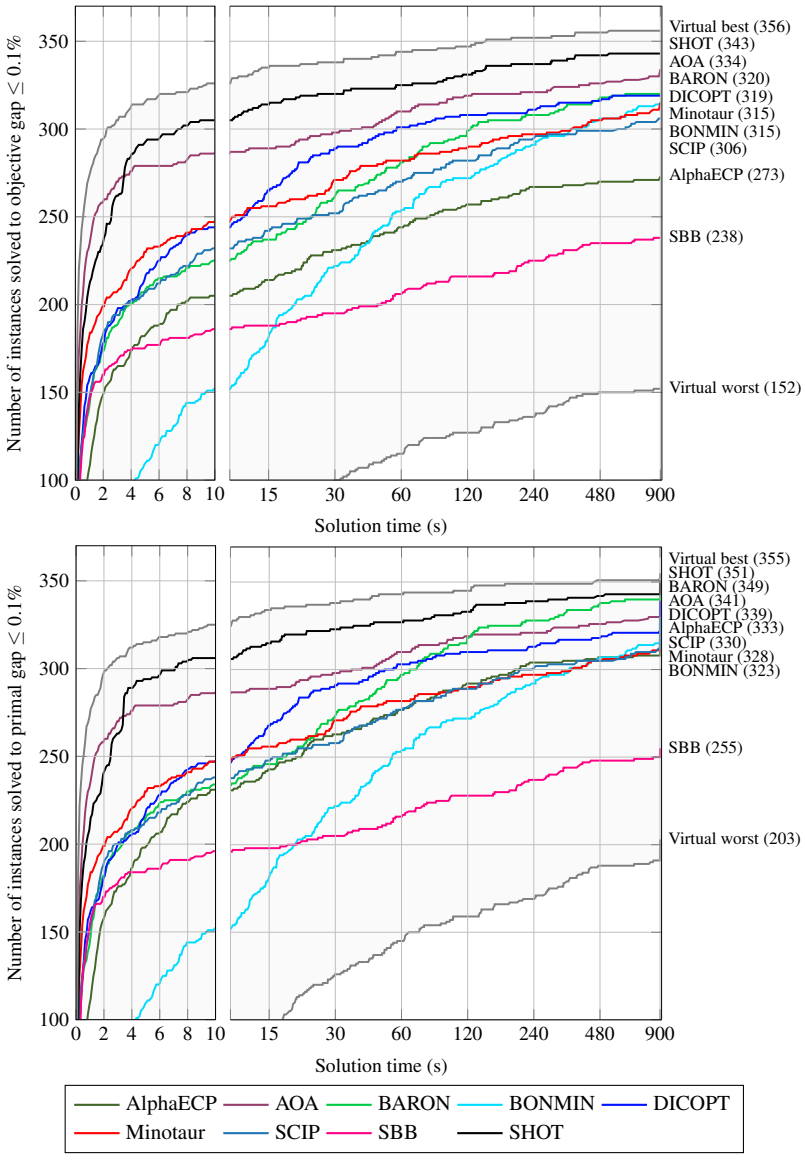

Figure 2: Solution profiles indicating the number of solved problem instances in the solver benchmark in Section 7.1 with a relative objective gap $\leq 0.1\%$ (*above*) and primal gap, *i.e.*, the relative difference to the best known solution in MINLPLib, $\leq 0.1\%$ (*below*). Most solvers have the same behaviour in both profiles, the exceptions are BARON and AlphaECP. BARON is very good at finding primal solutions, probably due to the primal heuristics utilized. AlphaECP is also quite good at finding primal solutions, but has difficulties tightening the objective gap.

Table 1: Some statistics from the comparison described in Section 7.1. The values indicate the number of solved instances with a certain property as reported by PAVER. The primal gap rows indicates the number of instances where the primal solution is within a certain percentage of the known optimal solution. The objective gap rows indicate the number of instances solved to a relative primal/dual objective tolerance. Full statistics are available in the PAVER report available at `andreaslundell.github.io/minlpbenchmarks/`.

	SHOT	AlphaECP	AOA	BARON	BONMIN	DICOPT	Minotaur	SBB
Primal gap								
$\leq 0.1\%$	350	333	341	349	323	339	328	255
$\leq 1.0\%$	351	337	342	352	323	339	334	258
$\leq 10\%$	355	342	347	355	325	344	341	261
Objective gap								
$\leq 0.1\%$	343	273	334	320	315	319	315	238
$\leq 1.0\%$	344	317	335	336	316	325	317	251
$\leq 10\%$	349	329	338	348	316	329	326	255
Termination status								
Normal[3]	346	313	341	343	322	323	323	259
Limit[4]	18	51	22	20	12	41	39	46
Error/other[5]	2	1	3	2	31	1	4	61
Capability[6]	0	1	0	1	1	1	0	0
Failed[7]	3	6	11	1	37	3	16	64

[3] Normal termination as determined by the solver. [4] Iteration or time limit reached by the solver.
[5] The solver returned an error or crashed.
[6] The solver does not handle all functions/variables in a problem.
[7] The reported solution by the solver contradicts the known solution or its bounds in MINLPLib.

Table 1. It is clear that SHOT is very competitive on the whole test set, as is AOA. AOA is an implementation in AIMMS of an LP/NLP based branch-and-bound strategy. Both SHOT and AOA utilize a single-tree strategy with tight MIP solver integration through lazy constraint callbacks, and this clearly illustrates the benefits of such an implementation. Minotaur is also based on the same main strategy as AOA, but it implements its own branch-and-bound tree and only uses CPLEX for solving LP problems in the nodes. The fact that AOA clearly performs better than Minotaur, strengthens the assumption that a single-tree method in combination with linearizations added through callbacks, forms a good foundation for an MINLP solver.

The AlphaECP implementation was not designed specifically for convex problems, but instead to be more of a heuristic solver for nonconvex problems. This explains that its performance is not on par with SHOT, even though the basic methods are similar (dual bounds by ECP and ESH, respectively, together with primal heuristics based on solving integer-fixed NLP problems). Also, AlphaECP does not currently support solving MIQP subproblems. When comparing the two profiles in Figure 2, it can be seen that AlphaECP is good at finding primal solutions, but it has troubles verifying them to optimality, *i.e.*, increasing the dual bound. BARON is a global solver, and can therefore solve

problems of the more general nonconvex MINLP class. However, as the results show, it is also very good at convex MINLP, especially at finding a good primal solution.

7.2 Impact of MIP solution strategies

In Figure 3, the multi-tree and single-tree strategies in SHOT are compared using both Gurobi and CPLEX as MIP solvers. The single-tree strategy performs somewhat better when considering how many instances were solved to an objective gap below 0.1%. However, the difference is quite small in favour of the single-tree strategy when considering the entire test set, even though the differences per problem can vary quite a lot between the two strategies.

This supports the assumption that a multi-tree strategy can also be very competitive if implemented efficiently. A multi-tree implementation like the one in SHOT communicates information to the MIP solver, such as the current primal solution providing a cut-off value for pruning the branching tree. Also the technique to only solve some subproblems to feasibility (*cf.*, Section 4.4) and utilizing the solution pool to add multiple linearizations per iteration (*cf.*, Section 4.3) impact the performance of the multi-tree implementation. Another important aspect why the multi-tree strategy is so efficient is the usage of the MIP presolver in each iteration in the subsolver: Due to the fact that many of the MIP subproblems are not solved to optimality when utilizing the MIP solution limit strategy, often no branching at all is needed, and the subproblems are solved during the presolving phase.

When comparing CPLEX and Gurobi, it seems that CPLEX is somewhat more effective when utilized in SHOT. The difference is, however, quite small and could probably be reduced further by tuning the MIP solver parameters. Note however, that no specific parameter tuning has been performed for CPLEX either. In general, both solvers are very stable in combination with SHOT.

In Section 4, certain methods to enhance the performance of the multi-tree strategy were described. These included solving MIP problems with early termination using the solution limit strategy, and generating multiple linearizations per iteration by utilizing the MIP solution pool. These are both enabled by default in the multi-tree strategy, and in Figure 4, the impact of disabling these are illustrated. Differences are noticeable, *e.g.*, disabling both the solution pool functionality and solving the problems to optimality doubles the time required to solve 280 instances. However, it is obvious that these features only constitute parts of what makes SHOT effective.

7.3 Impact of multi-threading

In Figure 5, the impact of utilizing several threads in the MIP solver is shown. The number of threads are set at eight in the cases with multiple threads (the maximum number supported simultaneously by the CPU). As can be expected, the number of threads have a significant influence on the solution times required to solve certain problem instances. For example, considering CPLEX it takes about 180 seconds to solve 330 problems with only one thread, while it takes only a third of that time to solve the same number with multiple threads. It is however clear that the performance does not grow linearly with the number of used MIP solver threads.

7.4 Utilizing quadratic functions in MIP subsolvers

Out of the 366 convex MINLP instances currently in MINLPLib, there are 67 pure MIQP problems (a quadratic objective function and no other nonlinearities). As stated in Section 4.5, such problems can be directly solved by CPLEX and Gurobi. This is normally much more efficient than utilizing iterative linearizations of the objective function to approximate its nonlinearity, since a quadratic objective can

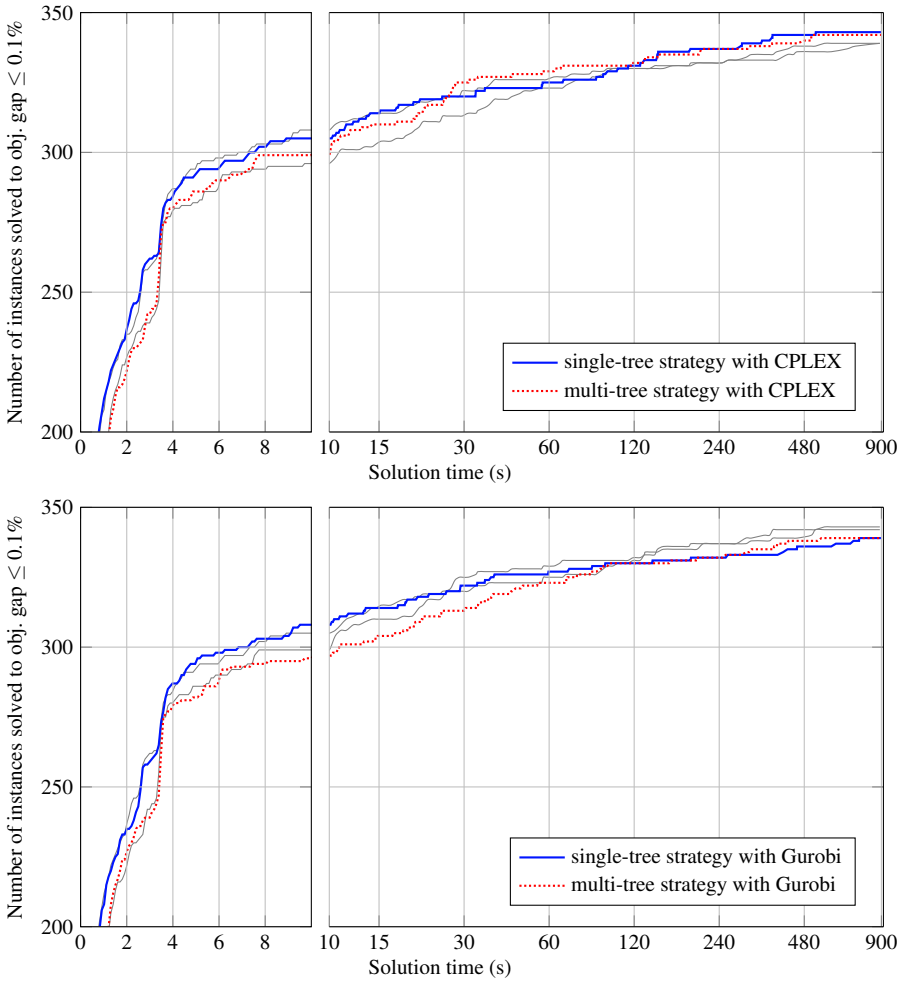

Figure 3: Impact of single- and multi-tree strategies when using CPLEX (above) and Gurobi (below) as MIP solvers. The single-tree strategy is more efficient for easier problem, while the multi-tree strategy catches up in the end when considering the number of solved problems. To aid the visual comparison between CPLEX and Gurobi, the thin grey lines corresponding to the results of the other solver are included in each graph.

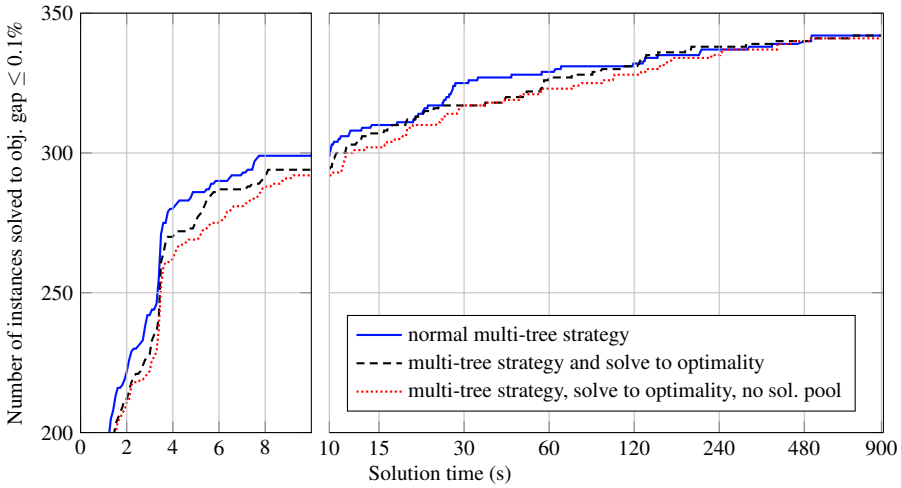

Figure 4: The impact of not utilizing the solution limit strategy described in Section 4.4 and the MIP solution pool described in Section 4.3.

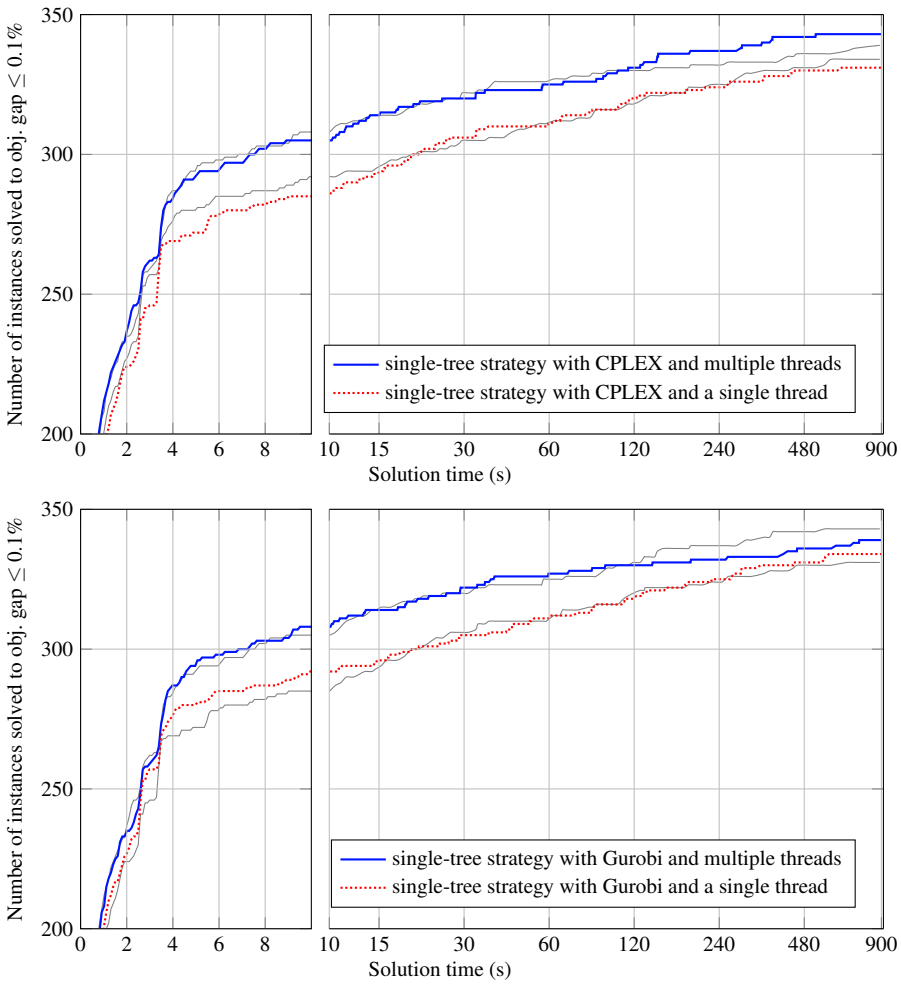

Figure 5: The impact of multi-threading on CPLEX (above) and Gurobi (below) with multi-tree strategy. The number of threads were set to eight in this case. It is clear that utilizing more threads is beneficial for more difficult problems. To aid the visual comparison between CPLEX and Gurobi, the thin grey lines corresponding to the results of the other solver are included in each graph.

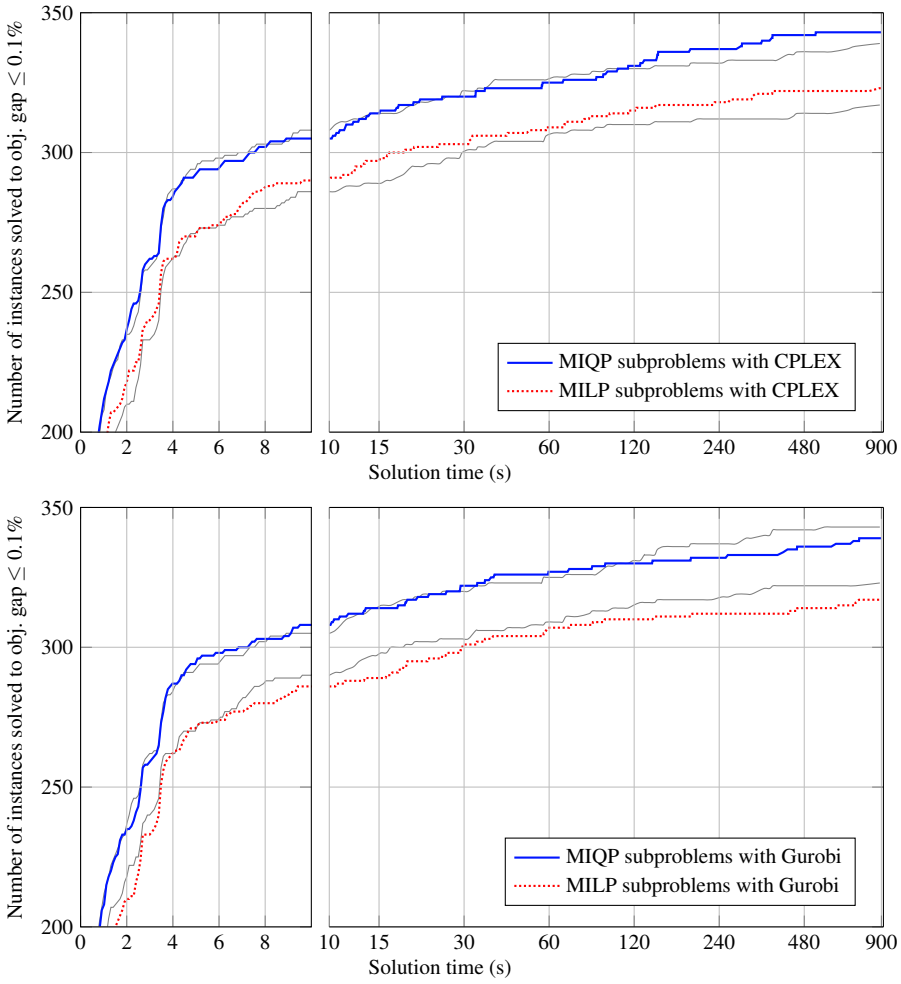

Figure 6: The impact of solving MIQP subproblems instead of MILP subproblems in SHOT as described in Section 4.5. To aid the visual comparison between CPLEX and Gurobi, the thin grey lines corresponding to the results of the other solver are included in each graph.

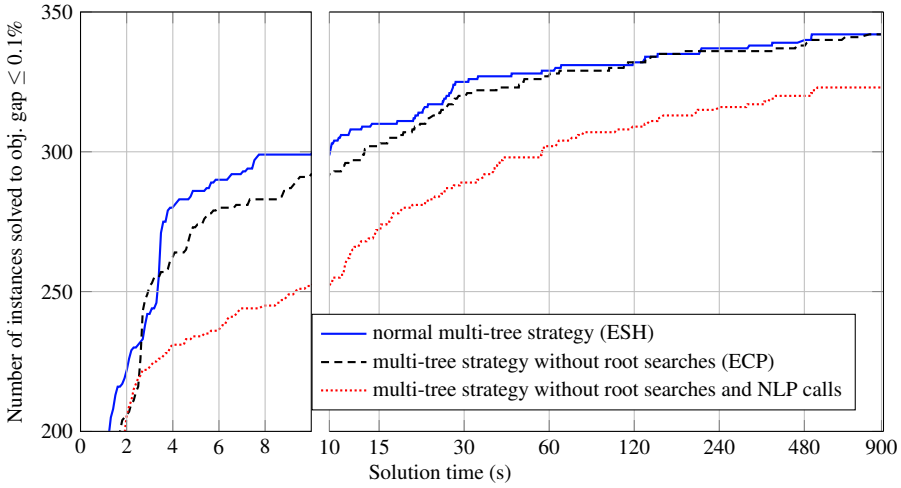

Figure 7: The differences in performance when utilizing the ECP method instead of the ESH method for creating linearizations, *i.e.*, not utilizing root searches. Also shown is the performance of SHOT without root searches for linearization generation and primal heuristics, and no fixed-integer primal NLP calls.

now be expressed exactly. There are also a few instances with a quadratic objective function, as well as nonlinear constraints. In this case, it is possible to handle the nonlinear constraints with the ESH or ECP methods, while passing the quadratic objective function on to the MIP solver. Thus, MIQP subproblems are solved instead of MILP subproblems. In Figure 6, the impact of utilizing a MIQP subsolver on the single-tree strategy has been illustrated, but the multi-tree strategy behaves similarly. As can be seen from the figure, the impact is quite significant, and it is quite obvious that CPLEX and Gurobi are a better choice for solving MIQP problems than SHOT with polyhedral outer approximation.

7.5 SHOT with or without root searches

In SHOT there are two main dual strategies, ESH and ECP, as described in Section 4. The main benefit of the ECP method for generating linearizations is that fewer function evaluations are needed since no root searches are performed. Thus, in case either the function evaluations are expensive or it is impossible to evaluate the functions at certain points, *e.g.*, non-integer points, the ECP method must be used. In Figure 7, SHOT utilizing ESH linearizations and root search based primal heuristics, and utilizing ECP linearizations and no root searches at all are compared. Included is also SHOT without root searches and where no fixed-integer primal NLP strategy is performed, keeping the number of function evaluations at a minimum. Out of the 366 test problems, there are no functions that can be classified as expensive. This comparison is, however, provided to illustrate the performance benefits of utilizing root searches and NLP-based heuristics in SHOT.

8 Conclusions and future work

In this paper, the SHOT solver was described in detail. The internal numerical benchmarks of the SHOT solver presented in Sections 7.2–7.5 indicate that the performance of SHOT is not due to a single feature, but a combination of many different strategies that together result in a well-performing

solver. Thus it can be summarized that SHOT builds on and extends 30 years worth of research on POA methods. One of the main features of POA-based solvers is that they rely heavily on the subsolvers utilized, and this is especially true for the single-tree strategy in SHOT. Although the numerical comparison between the single- and multi-tree strategies only showed a small gain when using the former, it should be mentioned, that a large part of the convex problems in MINLPLib are very easy to solve with both strategies as long as an efficient MIP solver is utilized. Also, the single-tree implementation is still not quite as mature as the multi-tree one, and there are still a lot of aspects to consider, such as the possibilities to generate hyperplane cuts also in relaxed solution points, and to only add cuts locally in the search tree.

For SHOT, one of the next major development steps are to include more primal heuristics, to utilize automated convexity preserving reformulations to enhance the performance especially for more demanding problems, and to make it available in more modeling systems. As was shown in [30], utilizing reformulations that split a convex nonlinear constraint of many variables into several constraints, but where each one has less variables, has a significant impact on POA-based solvers. Many of the problem instances in MINLPLib that were not solved by SHOT in the comparisons in this paper, are of the applicable type for the reformulations. Thus, automated reformulations would most certainly be very beneficial for not only solving these specific instances, but also for the overall performance of SHOT. Another aspect to be investigated is different presolve strategies. Currently the MIP solvers provide some basic presolving, however since they are not aware of the nonlinear constraints, there is still significant gains to be made with respect to bound tightening and range reduction. In the benchmark, the AOA solver utilizes preprocessing functionality available in AIMMS, which probably is one of the reasons behind its good performance.

In addition to being important in itself, convex MINLP can also be used as a tool for solving nonconvex problems by reformulating such problems into convex MINLP problems [38, 39, 44, 45]. In certain aspects, this mimics how MIP problems can be used in POA to solve convex MINLP problems. Of course, this is much more difficult in practice due to the fact that several convex MINLP problems are solved in sequence to find the optimal solution to a nonconvex MINLP problems. Also, the reformulated problems grow not only in the number of additional linear constraints as in POA, but also in the number of variables. However, a tight integration of these reformulations with SHOT will, hopefully, extend its applicability to nonconvex MINLP problems as well.

Acknowledgements

The authors want to thank both GAMS (especially S. Vigerske and M. Bussieck) and AIMMS (especially M. Hunting) for their support. A. Lundell also wants to express his gratitude for the financial support from the Magnus Ehrnrooth Foundation, as well as the Ruth and Nils-Erik Stenbäck Foundation.

References

[1] Abhishek, K., Leyffer, S., Linderoth, J., 2010. FilMINT: An outer approximation-based solver for convex mixed-integer nonlinear programs. INFORMS Journal on Computing 22 (4), 555–567.

[2] Alefeld, G., Potra, F. A., Shi, Y., 1995. Algorithm 748: Enclosing zeros of continuous functions. ACM Transactions on Mathematical Software 21 (3), 327–344.

[3] Belotti, P., 2009. Couenne: A user's manual. Tech. rep., Lehigh University.

[4] Berthold, T., Lodi, A., Salvagnin, D., 2017. Ten years of feasibility pump and counting. Tech. rep., Polytechnique Montréal, Département de Mathématiques e Génie Industriel.

[5] Bisschop, J., 2006. AIMMS optimization modeling. Lulu.com.

[6] Bonami, P., Biegler, L. T., Conn, A. R., Cornuéjols, G., Grossmann, I. E., Laird, C. D., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A., 2008. An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization 5 (2), 186–204.

[7] Bonami, P., Cornuéjols, G., Lodi, A., Margot, F., Jul 2009. A feasibility pump for mixed integer nonlinear programs. Mathematical Programming 119 (2), 331–352.

[8] Brook, A., Kendrick, D., Meeraus, A., 1988. GAMS, a user's guide. ACM Signum Newsletter 23 (3-4), 10–11.

[9] Bussieck, M., Dirkse, S., Vigerske, S., 2014. PAVER 2.0: An open source environment for automated performance analysis of benchmarking data. Journal of Global Optimization 59 (2-3), 259–275.

[10] Bussieck, M. R., Meeraus, A., 2004. General Algebraic Modeling System (GAMS). In: Kallrath, J. (Ed.), Modeling Languages in Mathematical Optimization. Springer US, Boston, MA, pp. 137–157.

[11] Bussieck, M. R., Vigerske, S., 2010. MINLP solver software. In: Wiley Encyclopedia of Operations Research and Management Science. Wiley Online Library, pp. 1–12.

[12] Dunning, I., Huchette, J., Lubin, M., 2017. JuMP: A modeling language for mathematical optimization. SIAM Review 59 (2), 295–320.

[13] Duran, M. A., Grossmann, I. E., 1986. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming 36 (3), 307–339.

[14] Forrest, J., Lougee-Heimer, R., 2005. CBC User Guide.
URL `http://www.coin-or.org/Cbc`

[15] Fourer, R., Gay, D., Kernighan, B., 1993. AMPL. Boyd & Fraser Danvers.

[16] Fourer, R., Gay, D. M., Kernighan, B. W., 1990. A modeling language for mathematical programming. Management Science 36 (5), 519–554.

[17] Fourer, R., Ma, J., Martin, K., Jan 2010. OSiL: An instance language for optimization. Computational Optimization and Applications 45 (1), 181–203.

[18] GAMS Development Corp., 2018. SBB user's manual.
URL `https://www.gams.com/latest/docs/S_SBB.html`

[19] Gassmann, H., Ma, J., Martin, K., Sheng, W., 2015. Optimization Services 2.10 User's Manual.
URL `http://projects.coin-or.org/svn/OS/trunk/OS/doc/osUsersManual.pdf`

[20] Grossmann, I. E., 2002. Review of nonlinear mixed-integer and disjunctive programming techniques. Optimization and Engineering 3 (3), 227–252.

[21] Grossmann, I. E., Viswanathan, J., Vecchietti, A., Raman, R., Kalvelagen, E., et al., 2002. GAMS/DICOPT: A discrete continuous optimization package. GAMS Corporation Inc.

[22] Gupta, O. K., Ravindran, A., 1985. Branch and bound experiments in convex nonlinear integer programming. Management Science 31 (12), 1533–1546.

[23] Hart, W. E., Laird, C. D., Watson, J.-P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., Siirola, J. D., 2012. Pyomo-optimization modeling in Python. Vol. 67. Springer.

[24] HSL, 2018. A collection of Fortran codes for large-scale scientific computation.
URL http://www.hsl.rl.ac.uk

[25] Hunting, M., 2011. The AIMMS outer approximation algorithm for MINLP. Tech. rep., AIMMS B.V.

[26] Kröger, O., Coffrin, C., Hijazi, H., Nagarajan, H., 2018. Juniper: An Open-Source Nonlinear Branch-and-Bound Solver in Julia. arXiv preprint: 1804.07332.

[27] Kronqvist, J., Bernal, D. E., Lundell, A., Grossmann, I. E., 2018. A review and comparison of solvers for convex MINLP. Preprint, Optimization Online.
URL http://www.optimization-online.org/DB_HTML/2018/06/6650.html

[28] Kronqvist, J., Lundell, A., Westerlund, T., 2015. The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. Journal of Global Optimization 64 (2), 249–272.

[29] Kronqvist, J., Lundell, A., Westerlund, T., 2017. A center-cut algorithm for solving convex mixed-integer nonlinear programming problems. In: Computer Aided Chemical Engineering. Vol. 40. Elsevier, pp. 2131–2136.

[30] Kronqvist, J., Lundell, A., Westerlund, T., 2018. Reformulations for utilizing separability when solving convex minlp problems. Journal of Global Optimization.

[31] Lastusilta, T., Bussieck, M. R., Westerlund, T., 2009. An experimental study of the GAMS/AlphaECP MINLP solver. Industrial & Engineering Chemistry Research 48 (15), 7337–7345.
URL https://doi.org/10.1021/ie801378n

[32] Leyffer, S., 1993. Deterministic methods for mixed integer nonlinear programming. Ph.D. thesis, University of Dundee.

[33] Lin, Y., Schrage, L., 2009. The global solver in the LINDO API. Optimization Methods & Software 24 (4-5), 657–668.

[34] Lubin, M., Yamangil, E., Bent, R., Vielma, J. P., 2016. Extended formulations in mixed-integer convex programming. In: Louveaux, Q., Skutella, M. (Eds.), Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016. Springer International Publishing, pp. 102–113.

[35] Lundell, A., Kronqvist, J., Westerlund, T., 2016. Improvements to the supporting hyperplane optimization toolkit solver for convex MINLP. In: XIII Global Optimization Workshop GOW'16. Vol. 16. pp. 101–104.

[36] Lundell, A., Kronqvist, J., Westerlund, T., 2017. SHOT – a global solver for convex MINLP in Wolfram Mathematica. In: Computer Aided Chemical Engineering. Vol. 40. Elsevier, pp. 2137–2142.

[37] Lundell, A., Kronqvist, J., Westerlund, T., 2018. The Supporting Hyperplane Optimization Toolkit.
URL http://www.github.com/coin-or/shot

[38] Lundell, A., Westerlund, J., Westerlund, T., 2009. Some transformation techniques with applications in global optimization. Journal of Global Optimization 43 (2), 391–405.

[39] Lundell, A., Westerlund, T., 2017. Solving global optimization problems using reformulations and signomial transformations. Computers & Chemical Engineering.

[40] Mahajan, A., Leyffer, S., Linderoth, J., Luedtke, J., Munson, T., 2017. Minotaur: A Mixed-Integer Nonlinear Optimization Toolkit. Preprint, Optimization Online.
URL http://www.optimization-online.org/DB_FILE/2017/10/6275.pdf

[41] Melo, W., Fampa, M., Raupp, F., 2018. An overview of MINLP algorithms and their implementation in Muriqui Optimizer. Annals of Operations Research, 1–25.

[42] MINLPLib, 2018. Mixed-integer nonlinear programming library. [Accessed 27-May-2018].
URL http://www.minlplib.org/

[43] Misener, R., Floudas, C. A., 2014. ANTIGONE: Algorithms for continuous/integer global optimization of nonlinear equations. Journal of Global Optimization 59 (2-3), 503–526.

[44] Nowak, I., Breitfeld, N., Hendrix, E. M., Njacheun-Njanzoua, G., 2018. Decomposition-based inner-and outer-refinement algorithms for global optimization. Journal of Global Optimization, 1–17.

[45] Pörn, R., Harjunkoski, I., Westerlund, T., 1999. Convexification of different classes of non-convex MINLP problems. Computers and Chemical Engineering 23, 439–448.

[46] Quesada, I., Grossmann, I. E., 1992. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. Computers & Chemical Engineering 16 (10-11), 937–947.

[47] Sahinidis, N. V., 1996. BARON: A general purpose global optimization software package. Journal of Global Optimization 8 (2), 201–205.

[48] Schäling, B., 2014. The Boost C++ Libraries (2nd edition). XML Press.

[49] Trespalacios, F., Grossmann, I. E., 2014. Review of mixed-integer nonlinear and generalized disjunctive programming methods. Chemie Ingenieur Technik 86 (7), 991–1012.

[50] Vigerske, S., Gleixner, A., 2018. SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods and Software 33 (3), 563–593.

[51] Wächter, A., Biegler, L. T., Mar 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106 (1), 25–57.

[52] Westerlund, T., Eronen, V.-P., Mäkelä, M. M., 2018. On solving generalized convex MINLP problems using supporting hyperplane techniques. Journal of Global Optimization.

[53] Westerlund, T., Lastusilta, T., 2008. AlphaECP GAMS user's manual.
URL `http://www.gams.com/latest/docs/S_ALPHAECP.html`

[54] Westerlund, T., Lundqvist, K., 2005. Alpha-ECP, an interactive minlp-solver based on the extended cutting plane method. Tech. rep., Åbo Akademi University.

[55] Westerlund, T., Pettersson, F., 1995. An extended cutting plane method for solving convex MINLP problems. Computers & Chemical Engineering 19 (0), 131–136.

[56] Westerlund, T., Pörn, R., 2002. Solving pseudo-convex mixed-integer problems by cutting plane techniques. Optimization and Engineering 3, 253–280.

Appendix A: Key solver options

Here a key selection of options in SHOT is given. A full list is available in the solver documentation available at `www.github.com/coin-or/shot`, and in the options files themselves.

Dual strategy parameters

`Dual.TreeStrategy` The main strategy (multi- or single-tree) to use. Default is multi-tree if the MIP solver supports it.

`Dual.CutStrategy` Selects if the ESH (default) or the ECP should be used in the dual strategy. Default is ESH.

`Dual.ESH.InteriorPoint.Solver` Selects the method for finding the interior point: the cutting plane method described in Section 6.5 (default), solving a minimax problem with IPOPT or solving an integer-relaxed problem with IPOPT.

`Dual.FixedInteger.Use` Solve a fixed LP problem if integer-values have not changes in several MIP iterations, *cf.*, 4.6. Default value is false.

`Dual.MIP.Solver` Selects the MIP solver to use in the dual strategy. Default value depends on which solvers are installed.

`Dual.MIP.SolutionLimit.Initial` The initial MIP solution limit to use in the multi-tree strategy as described in Section 4.4. Default value is 1.

`Dual.MIP.SolutionLimit.UpdateTolerance` The constraint tolerance to use when determining if the solution limit should be updated or not, *cf.*, ε_{SL} in Algorithm 1. Default value is 0.001.

`Dual.Relaxation.IterationLimit` The maximum number of integer-relaxed iterations to solve initially, *cf.*, K^{LP} in Algorithm 1. Default value is 200.

`Dual.Relaxation.TerminationTolerance` The constraint tolerance for when to stop the integer-relaxation step, *cf.*, ε^{LP} in Algorithm 1. Default value is 0.5.

`Dual.QuadraticStrategy` Determines how an quadratic objective function and quadratic constraints are treated, *i.e.*, what type of subproblems (MILP, MIQP or MIQCQP) are solved. Default is that quadratic objectives are kept, but quadratic constraints are treated as general nonlinear constraints. If Cbc is used as dual solver, the objective function and all constraints are treated as general nonlinear functions.

Primal strategy parameters

`Primal.FixedInteger.Use` Whether or not to solve fixed integer NLP problems to obtain primal solution candidates, as described in Section 5.2.

`Primal.FixedInteger.CallStrategy` Determines how often integer-fixed NLP calls are performed. The default is to solve integer-fixed NLP problems for all MINLP feasible solutions found, as well as when a specified time (`...Frequency.Time`) or iteration count (`...Frequency.Iteration`) since last call has passed. If the switch `...Frequency.Dynamic` is enabled, the time and iteration parameters are updated based on the success rate of the NLP calls.

`Primal.FixedInteger.Solver` The default NLP solver to use (for the fixed strategy). The default value depends on whether GAMS is installed and if the problem was provided in GAMS syntax, in which case the default GAMS NLP solver (or the one given by the parameter `Subsolver.GAMS.NLP.Solver`) is used. Otherwise IPOPT, called through OS, is the default NLP solver.

`Primal.Rootsearch.Use` Whether to use the root search primal heuristic described in Section 5.3. Default value is true.

Termination criteria

`Termination.IterationLimit` The max number of dual iterations in the multi-tree strategy or MIP incumbent callback activation in the single-tree strategy before SHOT is terminated.

`Termination.ObjectiveGap.Absolute` The absolute objective gap termination tolerance given in Eq. (2). Default value is 10^{-3}.

`Termination.ObjectiveGap.Relative` The relative objective gap termination tolerance given in Eq. (2). Default value is $10^{-3} = 0.1\%$.

`Termination.TimeLimit` The time limit in seconds (wall clock time) before termination. Default value is 10^6 seconds.

Appendix B: Included problems in the benchmark

The following 366 problems were used in the benchmark in Section 7. They are the instances that are identified as convex in MINLPLib, with at least one nonlinearity (in the objective function or constraints), and at least one discrete (binary or integer) variable. Of these, the 67 problems marked with an * have only a quadratic objective function and no further nonlinearities in the constraints.

`alan`*, `ball_mk2_10`, `ball_mk2_30`, `ball_mk3_10`, `ball_mk3_20`, `ball_mk3_30`, `ball_mk4_05`, `ball_mk4_10`, `ball_mk4_15`, `batch`, `batch0812`, `batchdes`, `batchs101006m`, `batchs121208m`, `batchs151208m`, `batchs201210m`, `clay0203h`, `clay0203m`, `clay0204h`, `clay0204m`,

clay0205h, clay0205m, clay0303h, clay0303m, clay0304h, clay0304m, clay0305h, clay0305m, cvxnonsep_normcon20, cvxnonsep_normcon20r, cvxnonsep_normcon30, cvxnonsep_normcon30r, cvxnonsep_normcon40, cvxnonsep_normcon40r, cvxnonsep_nsig20, cvxnonsep_nsig20r, cvxnonsep_nsig30, cvxnonsep_nsig30r, cvxnonsep_nsig40, cvxnonsep_nsig40r, cvxnonsep_pcon20, cvxnonsep_pcon20r, cvxnonsep_pcon30, cvxnonsep_pcon30r, cvxnonsep_pcon40, cvxnonsep_pcon40r, cvxnonsep_psig20, cvxnonsep_psig20r, cvxnonsep_psig30, cvxnonsep_psig30r, cvxnonsep_psig40, cvxnonsep_psig40r, du-opt*, du-opt5*, enpro48pb, enpro56pb, ex1223, ex1223a*, ex1223b, ex4*, fac1, fac2, fac3*, flay02h, flay02m, flay03h, flay03m, flay04h, flay04m, flay05h, flay05m, flay06h, flay06m, fo7, fo7_2, fo7_ar2_1, fo7_ar25_1, fo7_ar3_1, fo7_ar4_1, fo7_ar5_1, fo8, fo8_ar2_1, fo8_ar25_1, fo8_ar3_1, fo8_ar4_1, fo8_ar5_1, fo9, fo9_ar2_1, fo9_ar25_1, fo9_ar3_1, fo9_ar4_1, fo9_ar5_1, gams01, gbd, hybriddynamic_fixed*, ibs2, jit1, m3, m6, m7, m7_ar2_1, m7_ar25_1, m7_ar3_1, m7_ar4_1, m7_ar5_1, meanvarx*, meanvarxsc*, netmod_dol1*, netmod_dol2*, netmod_kar1*, netmod_kar2*, no7_ar2_1, no7_ar25_1, no7_ar3_1, no7_ar4_1, no7_ar5_1, nvs03*, nvs10*, nvs11*, nvs12*, nvs15*, o7, o7_2, o7_ar2_1, o7_ar25_1, o7_ar3_1, o7_ar4_1, o7_ar5_1, o8_ar4_1, o9_ar4_1, portfol_buyin, portfol_card, portfol_classical050_1, portfol_classical200_2, portfol_roundlot, procurement2mot, ravempb, risk2bpb, rsyn0805h, rsyn0805m, rsyn0805m02h, rsyn0805m02m, rsyn0805m03h, rsyn0805m03m, rsyn0805m04h, rsyn0805m04m, rsyn0810h, rsyn0810m, rsyn0810m02h, rsyn0810m02m, rsyn0810m03h, rsyn0810m03m, rsyn0810m04h, rsyn0810m04m, rsyn0815h, rsyn0815m, rsyn0815m02h, rsyn0815m02m, rsyn0815m03h, rsyn0815m03m, rsyn0815m04h, rsyn0815m04m, rsyn0820h, rsyn0820m, rsyn0820m02h, rsyn0820m02m, rsyn0820m03h, rsyn0820m03m, rsyn0820m04h, rsyn0820m04m, rsyn0830h, rsyn0830m, rsyn0830m02h, rsyn0830m02m, rsyn0830m03h, rsyn0830m03m, rsyn0830m04h, rsyn0830m04m, rsyn0840h, rsyn0840m, rsyn0840m02h, rsyn0840m02m, rsyn0840m03h, rsyn0840m03m, rsyn0840m04h, rsyn0840m04m, slay04h*, slay04m*, slay05h*, slay05m*, slay06h*, slay06m*, slay07h*, slay07m*, slay08h*, slay08m*, slay09h*, slay09m*, slay10h*, slay10m*, smallinvDAXr1b010-011, smallinvDAXr1b020-022, smallinvDAXr1b050-055, smallinvDAXr1b100-110, smallinvDAXr1b150-165, smallinvDAXr1b200-220, smallinvDAXr2b010-011, smallinvDAXr2b020-022, smallinvDAXr2b050-055, smallinvDAXr2b100-110, smallinvDAXr2b150-165, smallinvDAXr2b200-220, smallinvDAXr3b010-011, smallinvDAXr3b020-022, smallinvDAXr3b050-055, smallinvDAXr3b100-110, smallinvDAXr3b150-165, smallinvDAXr3b200-220, smallinvDAXr4b010-011, smallinvDAXr4b020-022, smallinvDAXr4b050-055, smallinvDAXr4b100-110, smallinvDAXr4b150-165, smallinvDAXr4b200-220, smallinvDAXr5b010-011, smallinvDAXr5b020-022, smallinvDAXr5b050-055, smallinvDAXr5b100-110, smallinvDAXr5b150-165, smallinvDAXr5b200-220, smallinvSNPr1b010-011, smallinvSNPr1b020-022, smallinvSNPr1b050-055, smallinvSNPr1b100-110, smallinvSNPr1b150-165, smallinvSNPr1b200-220, smallinvSNPr2b010-011, smallinvSNPr2b020-022, smallinvSNPr2b050-055, smallinvSNPr2b100-110, smallinvSNPr2b150-165, smallinvSNPr2b200-220, smallinvSNPr3b010-011, smallinvSNPr3b020-022, smallinvSNPr3b050-055, smallinvSNPr3b100-110, smallinvSNPr3b150-165, smallinvSNPr3b200-220, smallinvSNPr4b010-011, smallinvSNPr4b020-022, smallinvSNPr4b050-055, smallinvSNPr4b100-110, smallinvSNPr4b150-165, smallinvSNPr4b200-220, smallinvSNPr5b010-011, smallinvSNPr5b020-022, smallinvSNPr5b050-055,

smallinvSNPr5b100-110, smallinvSNPr5b150-165, smallinvSNPr5b200-220, squfl010-025*, squfl010-040*, squfl010-080*, squfl015-060*, squfl015-080*, squfl020-040*, squfl020-050*, squfl020-150*, squfl025-025*, squfl025-030*, squfl025-040*, squfl030-100*, squfl030-150*, squfl040-080*, sssd08-04, sssd12-05, sssd15-04, sssd15-06, sssd15-08, sssd16-07, sssd18-06, sssd18-08, sssd20-04, sssd20-08, sssd22-08, sssd25-04, sssd25-08, st_e14, st_miqp1*, st_miqp2*, st_miqp3*, st_miqp4*, st_miqp5*, st_test1*, st_test2*, st_test3*, st_test4*, st_test5*, st_test6*, st_test8*, st_testgr1*, st_testgr3*, st_testph4*, stockcycle, syn05h, syn05m, syn05m02h, syn05m02m, syn05m03h, syn05m03m, syn05m04h, syn05m04m, syn10h, syn10m, syn10m02h, syn10m02m, syn10m03h, syn10m03m, syn10m04h, syn10m04m, syn15h, syn15m, syn15m02h, syn15m02m, syn15m03h, syn15m03m, syn15m04h, syn15m04m, syn20h, syn20m, syn20m02h, syn20m02m, syn20m03h, syn20m03m, syn20m04h, syn20m04m, syn30h, syn30m, syn30m02h, syn30m02m, syn30m03h, syn30m03m, syn30m04h, syn30m04m, syn40h, syn40m, syn40m02h, syn40m02m, syn40m03h, syn40m03m, syn40m04h, syn40m04m, synthes1, synthes2, synthes3, tls12, tls2, tls4, tls5, tls6, tls7, unitcommit1*, watercontamination0202*, watercontamination0202r*, watercontamination0303*, watercontamination0303r*.

Appendix C: Used solver options in the benchmark

In the comparison in Section 7.1, the goal was to use the default solver options as much as possible. However, several of the solvers have certain default parameter values that makes them terminate prematurely, and these were then increased to avoid this behaviour. Also convex strategies are activated for solvers supporting this, and if the solver has recommended parameters for convex problems, these were used. The used parameters are the same as in [27], and are explained further there.

Name	Value
General GAMS	
MIP	CPLEX
threads	8
optcr	0.001
optca	0.001
nodlim	10^8
domlim	10^8
iterlim	10^8
reslim	900
AlphaECP	
ECPmaster	1
AOA	
IsConvex	1
IterationMax	10^7
RelativeOptimalityTolerance	0.1
TimeLimit	900
BONMIN	

bonmin.algorithm	B-OA
milp_solver	CPLEX
bonmin.time_limit	900
DICOPT	
convex	1
stop	1
maxcycles	10^8
infeasder	1
nlpoptfile	1
Minotaur	
bnb_time_limit	900
lp_engine	OsiCpx
obj_gap_percent	10^{-5}
threads	8
SBB	
memnodes	$5 \cdot 10^7$
rootsolver	CONOPT.1
SCIP	
constraints/nonlinear/assumeconvex	true
SHOT	
Dual.MIP.NumberOfThreads	8
Dual.MIP.Solver	CPLEX
Primal.FixedInteger.Solver	GAMS
Subsolver.GAMS.NLP.Solver	CONOPT
Termination.ObjectiveGap.Absolute	0.001
Termination.ObjectiveGap.Relative	0.001
Termination.TimeLimit	900
CONOPT (GAMS)	
rtmaxv	10^{30}