Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

YUJI SHINANO[1], STEFAN VIGERSKE[2]

Smoothie: Mixing the strongest MIQCQP solvers to solve hard MIP and MIQCQP instances on supercomputers - Phase I development

[1] 0000-0002-2902-882X
[2] 0009-0001-2262-0601

ZIB Report 25-18 (October 2025)

Zuse Institute Berlin
Takustr. 7
14195 Berlin
Germany

Telephone: +49 30 84185-0
Telefax: +49 30 84185-125

E-mail: bibliothek@zib.de
URL: http://www.zib.de

ZIB-Report (Print) ISSN 1438-0064
ZIB-Report (Internet) ISSN 2192-7782

Smoothie: Mixing the strongest MIQCQP solvers to solve hard MIP and MIQCQP instances on supercomputers - Phase I development

Yuji Shinano[1] Stefan Vigerske[2]

November 2, 2026

Abstract

Mixed-Integer Linear Programming (MIP) and Mixed-Integer Quadratically Constraint Quadratic Programming (MIQCQP) are applicable to such a wide range of real-world decision problems that the competition for the best code to solve such problems has lead to tremendous progress over the last decades. While current solvers can solve some of the problems that seemed completely out-of-reach just 10 years ago, there are always relevant MIP and MIQCQP problems that currently cannot be solved. With the Smoothie solver we intend to solve extremely hard MIP and MIQCQP problems, such as open instances from MIPLIB and challenging instances from other benchmark libraries like the Quantum Optimization Benchmarking Library QOBLIB, by building on the many years that went into the development of several state-of-the-art MIQCQP solvers and by utilizing some of the largest computing resources available. The high-level task parallelization framework UG (Ubiquity Generator) is used and extended by Smoothie to build a solver that uses large-scale parallelization to distribute the solution of a single MIQCQP on a shared- or distributed-memory computing infrastructure, thereby employing several established MIQCQP solvers simultaneously. For the first development phase, which is the topic of this report, FICO Xpress, Gurobi, and, as additional solvers, IBM ILOG CPLEX and COPT are used in concurrent mode, either with threads on a single machine or with MPI processes, while information on incumbent solutions and explored branch-and-bound subtrees is exchanged. A dynamic restarting mechanism ensures that solver configurations are selected that promise most suitable for the instance to be solved. We report on initial findings using this early version of Smoothie on unsolved problems from MIPLIB 2017.

1 Introduction

MIPLIB[1], a library of mixed-integer linear programming problems, is the standard test set used to compare the performance of MIP (Mixed Integer Programming) solvers. Since its first release in 1992 [5], it has received

[1] 0000-0002-2902-882X
[2] 0009-0001-2262-0601

[1] `https://miplib.zib.de`

five major updates [6, 1, 14, 10]. Currently (October 2025), developers of many state-of-the-art academic and commercial MIP solvers are in the process of preparing the seventh edition of the library, MIPLIB 2024. It will include again new challenging MIP problems, which are likely going to drive the development of solver software to tackle these problems.

To find new improving or even proven optimal solutions for MIPLIB problems, state-of-the-art academic or commercial solvers have been run in highly parallelized form on supercomputers [20] by means of the UG framework[2] and its incarnations ParaSCIP [19] and ParaXPRESS [23]. In its first runs, UG-based parallel solvers solved 23 open instances from MIPLIB. With MIPLIB 2024 looming, we plan to explore again the flexibility of the UG framework to develop a new large scale parallel solver to tackle hard open MIP instances: Smoothie. By joining some of the strongest MIP solvers available nowadays and utilizing vast computing resources like supercomputers, we expect to solve MIP instances that single solvers on single machines cannot solve. Smoothie is, however, not restricted to MIPLIB. Benchmark libraries are only meaningful if the best known solutions and bounds for their instances are reliable and as strong as possible, since these serve as reference values against which new algorithms are measured. This is particularly relevant for newly established libraries such as QOBLIB[3], the Quantum Optimization Benchmarking Library [15], which was released only in 2025. It comprises ten problem classes, among them market split, low autocorrelation binary sequences, Steiner tree packing, sports tournament scheduling, portfolio optimization, and vehicle routing, whose instances have been selected to be challenging for established classical methods while having sizes that are within reach of current quantum hardware. QOBLIB provides instances as MIP as well as QUBO (Quadratic Unconstrained Binary Optimization) formulations and collects submitted solutions, both from classical and from quantum solvers. Since the library is still young, optimality has not yet been proven for many of its instances, and their best known solutions are likely to be improved further. To make a fair comparison with emerging quantum optimization methods possible, it is essential to know what classical state-of-the-art solvers can achieve on these instances when given massive computing resources. Smoothie can be applied to the MIP and QUBO formulations of QOBLIB as well as to instances of other libraries, for instance QPLIB[4], to compute improved primal solutions and, ideally, proofs of optimality that establish reference values for these benchmarks.

In this document, we briefly introduce Smoothie and its first development steps.

2 Ubiquity Generator (UG) Framework

The UG framework was originally developed to exploit the performance of state-of-the-art branch-and-bound based solvers in a parallel setting. In the original version, the *base solver* that is run in parallel and the *parallelization library* that is used to realize communication are abstracted. Popular instantiations of ug[base solver, parallelization library] include ParaSCIP (ug[SCIP,MPI]) [19, 21], FiberSCIP (ug[SCIP,C++ threads]) [24], and ParaXpress (ug[Xpress,MPI]) [22, 23], or the parallelization of SCIP-based

[2] `https://ug.zib.de`
[3] `https://github.com/ZIB-AOPT/QOBLIB`
[4] `https://qplib.zib.de`

solvers with little additional effort [25, 9, 16].

With version 1.0, UG has evolved into a high-level task parallelization framework, where not only branch-and-bound-based solvers can be parallelized, but any kind of parallelizable algorithm where tasks are defined, collected, and redistributed among workers, following a supervisor-worker paradigm[5] [17]. To achieve this generalization in UG, the controller process/thread, named LoadCoordinator, has been abstracted, see also Figure 1. This allows to design flexible parallel algorithms depending on the solver used by adding additional message passing, e.g., a solver for the shortest vector problem that does not rely on branch-and-bound [26].

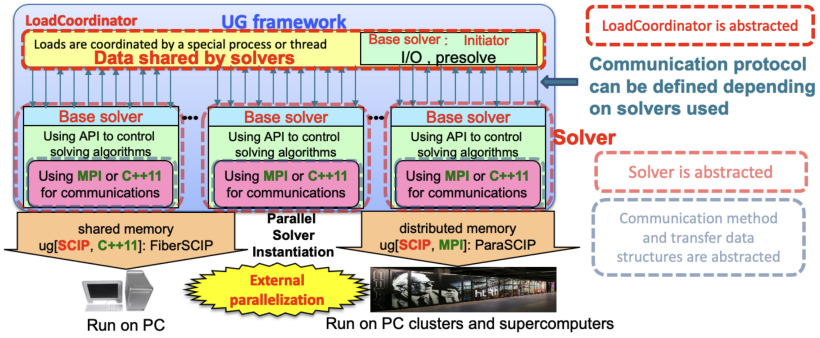

Figure 1: The UG Framework version 1.0

3 Smoothie Phase I

The abstraction of the LoadCoordinator in UG allows for the development of Smoothie, where not only the tree search of one branch-and-bound based solver is parallelized, but also additional solvers are run in parallel, thereby communicating information that goes beyond the incumbent solution. For the first development phase, which is the focus of this report, the information that can be shared among the solvers and the method of sharing are investigated. For this purpose, a simple parallelization of MIQCQP solvers, so-called racing or concurrent mode, where each solver solves the whole problem with different parameter settings, is used. A natural idea is to communicate incumbent solutions among the solvers during racing. When using solvers that provide detailed access to the solving process, also globally valid information like bound tightenings, cutting planes, or conflicts could be shared [12]. In the following, the basic idea of generating and sharing so-called *no-good cuts* among solvers is explored and a dynamic racing mode is introduced. The racing solvers used for this phase are primarily FICO Xpress and Gurobi, both being among the strongest MIQCQP solvers currently available. In addition, IBM ILOG CPLEX and COPT have been integrated as further racing solvers, see Section 3.4. The corresponding design of Smoothie, with Xpress and Gurobi as racing solvers, is illustrated in Figure 2. While Smoothie was initially developed as a shared-memory application, a distributed-memory version is available by now, see Section 3.6.

[5] `https://ug.zib.de/doc-1.0.0/html/CONCEPT.php`

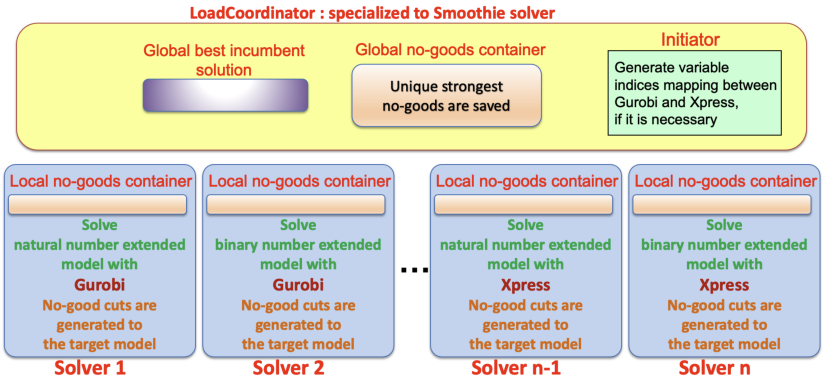

Figure 2: Smoothie Phase I design

3.1 No-Good Cut Formulation

Consider a mixed-integer quadratically constrained quadratic program (MIQCQP) given in the following general form:

$$\min\{f(x) \ : \ g(x) \leq b, l \leq x \leq u, x_j \in \mathbb{Z}, \forall j \in I\}, \tag{1}$$

where $f(x) := c^\top x + \frac{1}{2} x^\top Q x$ and $g(x) := (g_1(x), \ldots, g_m(x))^\top$ with $g_k(x) := a_k^\top x + \frac{1}{2} x^\top Q_k x$, $k = 1, \ldots, m$, with vectors $a_k \in \mathbb{R}^n$, symmetric matrices $Q, Q_k \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^m$, $c, l, u \in \mathbb{R}^n$, and a subset $I \subseteq \{1, \ldots, n\}$ indicating the integer variables. If $Q = 0$ and $Q_k = 0$ for all k, then (1) is a mixed-integer linear program (MIP), with $g(x) = Ax$ for the matrix A with rows $a_k^\top$. If only $Q \neq 0$, it is a mixed-integer quadratic program (MIQP). Let $B := \{j \in I : l_j = 0, u_j = 1\}$ be the index set of binary variables. For any $K \subseteq \{1, \ldots, n\}$, we denote by x_K the vector $(x_k)_{k \in K}$. The standard algorithm used to solve MIQCQP is branch-and-bound based on (in the linear case, LP) relaxations, which implicitly enumerates the whole solution space to find an optimal solution x^*.

Given an assignment x'_I to the integer variables, a no-good cut is an inequality that is violated by $x_I = x'_I$, but valid for any other value of x_I [2]. The use of no-good cuts has been explored in many algorithms [7, 11, 8, 18, 13]. Generally, a no-good cut can be formulated as $\sum_{i \in I} (x_i - x'_i)^2 \geq 1$, but in the context of MIP solvers, a linear formulation is usually preferred or even necessary. If $I = B$, such a formulation is easily found to be

$$\sum_{i \in I : x'_i = 0} x_i + \sum_{i \in I : x'_i = 1} (1 - x_i) \geq 1 \tag{2}$$

For general integer variables, however, a linear formulation requires the introduction of auxiliary variables. While one could introduce new auxiliary continuous and binary variables dynamically whenever the disjunction $x_i = x'_i \vee x_i \neq x'_i$ needs to expressed for a fixed x'_i, $i \in I$ [3], for the use in Smoothie a number of auxiliary variables is introduced a priori. This has the advantage that no new variables need to be added when adding a no-good cut during the solving process, something that solvers like Gurobi and Xpress do not support.

Two reformulations that make (1) amenable to the addition of no-good cuts are used by Smoothie. Let $J \subseteq I$ be the indices of integer variables

which domain is not too wide, i.e.,

$$J := \{j \in I : u_j - l_j < \Delta\}, \tag{3}$$

for some finite constant Δ. In the current implementation, Δ is a parameter with default value 30. We then refer to

$$\min \left\{ f(x) \; : \; \begin{array}{ll} g(x) \le b, & \\ l \le x \le u, & \\ x_j \in \mathbb{Z}, & \forall j \in I, \\ x_j = \sum_{i=l_j}^{u_j} i\, y_{ij}, & \forall j \in J \setminus B, \\ \sum_{i=l_j}^{u_j} y_{ij} = 1, & \forall j \in J \setminus B, \\ y_{ij} \in \{0,1\}, & \forall i \in \{l_j, \ldots, u_j\}, j \in J \setminus B, \end{array} \right\} \tag{4}$$

as *natural number based extension* of (1). In addition, the formulation

$$\min \left\{ f(x) \; : \; \begin{array}{ll} g(x) \le b, & \\ l \le x \le u, & \\ x_j \in \mathbb{Z}, & \forall j \in I, \\ x_j = l_j + v_j, & \forall j \in J \setminus B, \\ v_j = \sum_{i=0}^{k_j} 2^i\, y_{ij}, & \forall j \in J \setminus B, \\ v_j \le u_j - l_j, & \forall j \in J \setminus B, \\ y_{ij} \in \{0,1\}, & \forall i \in \{0, \ldots, k_j\}, j \in J \setminus B, \end{array} \right\} \tag{5}$$

where $k_j := \min\{k \in \mathbb{N} : u_j - l_j \le 2^k\}$, $j \in J \setminus B$, is referred to as *binary number based extension*.

Racing solvers may also work on the original formulation (1). On this formulation, a no-good can be expressed as a linear inequality without auxiliary variables if each of its variables is fixed at one of its bounds. That is, consider a no-good that forbids the assignment $x_S = x'_S$ for some $S \subseteq I$, and let $S_l := \{i \in S : x'_i = l_i\}$ and $S_u := \{i \in S : x'_i = u_i\}$. If $S = S_l \cup S_u$, then

$$\sum_{i \in S_l} (x_i - l_i) + \sum_{i \in S_u} (u_i - x_i) \ge 1 \tag{6}$$

is a valid no-good cut, which generalizes (2). Since (6) involves only original variables, it can be added to all three formulations and is used whenever applicable, also by solvers on formulations (4) and (5). No-goods that fix some variable to a value in the interior of its domain are stored and distributed as well, but can only be applied by solvers on formulation (4) or (5).

If $J \setminus B = \emptyset$, i.e., all integer variables with a small domain are binary, then formulations (4) and (5) coincide with (1). In this case, which includes the frequent case that all integer variables of the problem are binary, no auxiliary variables are added and all racing solvers work on the original formulation (1). Every no-good is then of the form (2) and can be applied

by every racing solver. Whether extended formulations are created is decided once, when the instance is read: the number of binary variables and the number of integer variables with a small domain are compared, and only if the latter is larger, formulations (4) and (5) are set up in the racing solvers.

3.2 Racing Mode with No-Good Generation and Distribution

Smoothie runs several instances of solvers Gurobi and Xpress in parallel in different threads. The configuration of a racing solver consists of the solver to be used, the formulation to work on, and a parameter setting for the solver. Regarding the formulation, four *model types* are distinguished:

natural number based The racing solver works on formulation (4).

binary number based The racing solver works on formulation (5).

original The racing solver works on the original formulation (1). It generates no-goods in the same way as a racing solver on an extended formulation, but it can apply only those no-goods that can be written as (6), since the auxiliary variables that are required to express the remaining ones are not available.

pure original The racing solver works on the original formulation (1), but neither generates no-goods by local branching nor applies any no-good cut. It only exchanges incumbent solutions and dual bounds with the other solvers, and therefore shows the performance of the underlying solver on the instance, without any influence of the additional Smoothie machinery. This is of interest since local branching and the addition of no-goods have been observed to slow down some solvers, in particular Gurobi, on some instances.

The model types are assigned to the racing solvers as follows. The first racing solver of each MIQCQP solver is of type pure original, which can be disabled by a parameter. The remaining racing solvers of this MIQCQP solver cycle through the types original, natural number based, and binary number based. If no extended formulations are created for the instance, cf. Section 3.1, then all racing solvers except the first one are of type original. Independently of the model type, the racing solvers of each MIQCQP solver cycle through a list of parameter settings. For Xpress, 25 parameter settings are used (see also Section 4). For Gurobi, seven parameter settings are used: default settings, and settings that change the parameters `NoRelHeurWork`, `ImproveStartWork` together with `ZeroObjNodes`, `MIPFocus`, `Presolve`, `Cuts`, or `Presolve` together with `Cuts`. Whenever a solver finds a new incumbent solution, it is transferred to all other solvers, and when one solver has proven optimality, a termination signal is send to all solvers. Since a solution that is found during presolve or in the root node may not be reported by a callback, each racing solver reports the incumbent of its search also after its search has ended.

To avoid that some branch-and-bound subtrees have to be explored over and over again, no-good cuts are generated for subsets of J, stored and filtered in the racing solvers and LoadCoordinator, and distributed to other solvers. A no-good cut is generated in the node callbacks of the MIQCQP solvers when the number of integer variables that take a fractional solution in the current LP relaxation or have a wide domain is small. That is, let $\tilde{x}$ be the solution of the LP relaxation in the node

where the node callback is called, and $F := \{j \in J : \tilde{x}_j \notin \mathbb{Z}\} \cup (I \setminus J)$ be the set of fractional or wide-domain integer variables. If $\phi \leq |F| \leq \Phi$ for given thresholds ϕ and Φ, then the solver instance from which the node callback is called (that is, the MIQCQP solver on the formulation and with the parameter settings of the racing solver) is copied and the fixings $x_j := \tilde{x}_j$, $j \in I \setminus F$ are applied. This *local branching MIQCQP* is then solved using a limit on the solvers deterministic time (also called *work limits*). In addition, the local branching MIQCQP is subject to the time remaining until the time limit of Smoothie. To limit the overhead of the no-good generation, a racing solver solves local branching MIQCQPs only after it has spent a certain amount of effort on its own search, and only in nodes where the dual bound of the node is still below the value of the incumbent solution, since only such nodes can contain an improving solution. Further, a local branching MIQCQP is not solved again for the same set of fixings as in the previous attempt. If a new incumbent solution is found, then it is stored and transferred to all solvers. Otherwise, if optimality or infeasibility was proven but no improving solution was found, then a no-good cut is generated that forbids the values $(\tilde{x}_j)_{j \in I \setminus F}$. To ensure that no assignment is forbidden that has not been examined, the local branching MIQCQP is solved with zero gap tolerance and, if it was solved to optimality, the no-good is generated only if the dual bound of the local branching MIQCQP reaches the cutoff value. This no-good cut is then stored in the solvers no-good container. If solving the local branching MIQCQP required branching operations, that is, a non-trivial subtree had to be explored, then the no-good is also transferred to the LoadCoordinator. Otherwise, the bound Φ is increased to potentially increase the difficulty of future local branching MIQCQPs. If the local branching MIQCQP could not be solved within the deterministic time limit, then the bound Φ is decreased.

The no-good containers in the LoadCoordinator and the solvers take care of removing duplicates and dominated no-goods. Solvers add the no-goods from the LoadCoordinator to their local no-good storage regularly and apply new no-goods while running. Each no-good is added as a cut at most once per call of the MIQCQP solver. If the number of no-goods in the container of a racing solver exceeds a limit, then the solver is interrupted, the limit is increased, and the search is restarted with all no-goods applied to the model.

Which cut is used to apply a no-good depends on the values that the no-good fixes and on the model type of the racing solver that applies it. If every variable of the no-good is fixed to one of its bounds in (1), then cut (6) is used. Since this cut uses original variables only, it can be applied by every racing solver, independent of the formulation it works on. In the special case that all variables of the no-good are binary, the cut reduces to (2). If, however, some variable of the no-good is fixed to a value in the interior of its domain, then the no-good can be applied only by a racing solver that works on formulation (4) or (5), where it is expressed by means of the auxiliary variables of the formulation. Whether a fixed value is a bound is always decided with respect to the bounds of the original problem (1) and not with respect to the bounds that a solver has tightened locally in the node where the no-good was generated. As this information is not transferred together with a no-good, a racing solver that receives a no-good from the LoadCoordinator determines it again before applying the no-good.

For solvers which dual bound is currently best among all solver, a

different no-good strategy is applied. Since such solvers are most promising in solving the problem to optimality, they are not interrupted for the generation of no-goods. In addition, currently no no-goods are applied by such solvers. This is due to an empirical observation where the addition of no-goods has been slowing down Gurobi considerably.

3.3 Dynamic Racing

With the racing mode as described so far, all solvers start without a known feasible solution and an empty set of no-goods. However, having a good primal bound and a number of strong no-goods already available during presolve may lead to considerable problem reductions which could go beyond the tightening of variables bounds that can be applied during solving, unless the solver decides to restart by itself. Further, since solvers typically search for feasible solutions more aggressively in the beginning of the solving process, having a good primal solution available at the start may be more helpful than updating the incumbent later in the search [4].

To explore the possible benefits from having a good primal solution and no-good cuts available during presolve and root node processing, solvers with a worst dual bound are stopped and replaced by new tasks. The LoadCoordinator checks for such a replacement regularly (by default, at most every 50 seconds) when it receives a status message from a racing solver. If the incumbent solution has been updated since the last check, then five solvers with worst dual bound are replaced. The configuration for the new tasks are chosen to be:

1. The solver, formulation ((4) or (5)), and parameter settings of a racing solver that currently has the best dual bound among all racing solvers that work on formulation (4) or (5).
2. Gurobi with formulation (4) and default parameter settings.
3. Gurobi with formulation (5) and default parameter settings.
4. Xpress with formulation (4) and default parameter settings.
5. Xpress with formulation (5) and default parameter settings.

All new solvers are given the new incumbent solution and the latest no-goods before presolve. In addition, if, instead of a new incumbent, a new no-good is added to the container in the LoadCoordinator, a solver with worst dual bound is replaced by one with the configuration of a leading solver (item 1 above). Further, if a racing solver finishes its search without Smoothie being terminated, it is immediately restarted with the configuration of a leading solver.

Racing solvers of model type original or pure original are never replaced, so that they continue to run as a reference of the performance of the underlying solver. For the same reason, the configuration of such a racing solver is never copied to another one. Also, a replacement that would remove the last Gurobi or the last Xpress racing solver is not carried out. To avoid that a solver that has just been restarted will be terminated again immediately, solvers that did not run at least a specific amount of time are not replaced. Instead, up to $\min\{10, \lfloor p/2 \rfloor\}$ solvers with most inferior dual bound are considered for replacement, where p is the number of racing solvers. If not all planned replacements can be carried out, the remaining ones are postponed to the next check. However, the minimal running time is ignored if the incumbent has been updated since the last replacement, because a new incumbent, and thus a better cutoff value, is regarded as

valuable information for every solver. Finally, no new tasks are started when the remaining time is less than half of the minimal running time.

3.4 Additional MIQCQP Solvers

In addition to Gurobi and Xpress, IBM ILOG CPLEX and COPT can be used as racing solvers (CPLEX only if all quadratic constraints are convex or can be reformulated as second-order cone constraint). The number of racing solvers for CPLEX and COPT can be specified by the user, whereby at least one Gurobi and one Xpress racing solver are always kept. The remaining racing solvers are divided equally between Gurobi and Xpress. The racing solvers of CPLEX and COPT are configured in the same way as those of Gurobi and Xpress, that is, the same model types are used and the racing solvers cycle through a list of parameter settings. For CPLEX, 25 parameter settings that mirror the Xpress settings are used, and for COPT, seven parameter settings that correspond to the Gurobi settings are used.

The racing solvers of CPLEX and COPT solve local branching MIQCQPs and generate, share, and apply no-goods in the same way as those of Gurobi and Xpress. Since CPLEX accepts user cuts only in the context of a solved LP relaxation of a node and COPT does not accept user cuts in its node callback, no-goods are generated and applied in the relaxation callbacks of these solvers. Their local branching MIQCQP is created from a copy of the original formulation (1). For CPLEX, the work limit for local branching MIQCQPs is translated into a limit on CPLEX's deterministic time, while for COPT, which does not offer a comparable limit, a time limit is used instead. Further, since neither solver reports the effort spent so far in a measure that is comparable to the one used by Gurobi and Xpress, these solvers start to solve local branching MIQCQPs after they have run for 100 seconds.

Like Gurobi and Xpress racing solvers, CPLEX and COPT racing solvers prune their search with the objective value of the best known incumbent solution and send new incumbent solutions and their dual bounds to the LoadCoordinator. Since CPLEX cannot tighten its cutoff value during a search, the search of a CPLEX racing solver is restarted when another solver has found a better solution. COPT, which does not provide a cutoff parameter, enforces the cutoff value by a constraint on the objective function, which is reinstalled between its solves. For a problem with a quadratic objective function, however, such a constraint cannot be created. COPT racing solvers also receive incumbent solutions found by other solvers. If CPLEX or COPT cannot read a given instance, the corresponding racing solvers are disabled automatically. The number of threads that a racing solver uses for its search and for its local branching MIQCQPs can be set separately for each of the four MIQCP solvers; by default, every racing solver runs with a single thread.

3.5 Further Features

Initial solution and cutoff value. A feasible solution can be passed to Smoothie. The LoadCoordinator checks the solution for feasibility and, if it is feasible, uses it as the initial incumbent. Each racing solver receives the solution as a starting point, where the values of the auxiliary variables of formulations (4) and (5) are completed according to the formulation,

and uses its objective value as cutoff. Alternatively, only a cutoff value for the objective function can be specified.

Checkpointing and restart. Like other UG-based solvers, Smoothie can write checkpoints, so that a run that is interrupted or reaches its time limit can be continued later. Since every racing solver works on the whole problem and the LoadCoordinator does not hold any branch-and-bound nodes, a checkpoint of Smoothie consists of the global no-good container of the LoadCoordinator and the incumbent solution. The no-goods are written to a compressed file periodically (by default every hour), once more shortly before the time limit (by default 10 minutes before), and when Smoothie terminates, while the incumbent is written whenever it is improved. When Smoothie is restarted from a checkpoint, the stored no-goods are sent to every racing solver in the same way as no-goods generated by other solvers, and the stored solution is checked and used as initial incumbent, so that all racing solvers start with its objective value as cutoff. Since a no-good from a different instance could cut off feasible solutions, the checkpoint files record the number of variables of the instance, and checkpoint data that does not match the instance to be solved is rejected.

Quadratic objective functions and constraints. Problems with quadratic terms in (1), i.e., MIQPs and MIQCQPs, need some special treatment. The quadratic objective term is taken into account when the LoadCoordinator evaluates solutions, and optimality criteria that rely on an integral objective function (see below) are disabled for problems with $Q \neq 0$. Since only the linear part of a quadratic constraint is available to the LoadCoordinator, such constraints are not checked when a solution is validated, but the racing solver that reported the solution is relied upon. Linear constraints, variable bounds, and integrality are still checked. Further, on an MIQCQP, CPLEX racing solvers report only their final solution, since a point that is integer feasible for the linear relaxation that CPLEX works on may violate a quadratic constraint.

Safeguards for primal and dual bounds. Since Smoothie combines information from different MIQCQP solvers that work on different formulations and use different numerical tolerances, several safeguards have been added to ensure that the reported primal and dual bounds are consistent. Every solution that a racing solver reports is checked by the LoadCoordinator: the values of integer variables are rounded, the objective value is recomputed, variable bounds and linear constraints are checked with a relative tolerance of 10^{-6}, and the solution is rejected if its objective value is below the proven dual bound. The dual bound that a racing solver reports is capped by the best objective value of a feasible solution known to this solver. The global dual bound maintained by the LoadCoordinator is monotonically nondecreasing. When a racing solver claims optimality, this is accepted by the LoadCoordinator only if the relative gap between the incumbent value and the dual bound is at most 10^{-5} and, if the objective function can only take integral values, the absolute gap is below one. A search that was stopped by the solver without having been completed is not regarded as a proof of optimality. Otherwise, racing continues.

3.6 Distributed-Memory Version

Besides the shared-memory version, which uses C++ threads for parallelization, Smoothie is now also available as a distributed-memory version that uses MPI. In this version, the process with rank 0 reads and presolves the instance and runs the LoadCoordinator, and each other process runs one racing solver. Instead of broadcasting the whole problem, only some problem data is sent to the racing solvers, which read the instance from a shared file system themselves. As in the shared-memory version, incumbent solutions, no-goods, dual bounds, and tasks for dynamic racing are communicated via the LoadCoordinator. No-goods for a racing solver are collected by the LoadCoordinator and sent together with its reply to a status message of the racing solver. To avoid that the LoadCoordinator and racing solvers block each other when sending messages to each other at the same time, solutions and notifications about the leading solver are sent in non-blocking mode. Further, a monitoring thread on rank 0 wakes up the LoadCoordinator when the time limit is approaching. In both versions, the LoadCoordinator does not wait indefinitely for racing solvers that do not respond to a termination request, e.g., because they are busy in a routine that cannot be interrupted. Both versions share the same implementation of the LoadCoordinator and racing solvers.

4 Preliminary Computational Results

4.1 October 2025

For initial tests, Smoothie has been build for a Linux/amd64 architecture with C++ threads as parallelization library, Gurobi 12.0.1, and FICO Xpress 45.01.02. The results reported first in this section were obtained in October 2025 with an earlier version of Smoothie that used only Gurobi and Xpress as racing solvers. In particular, the original formulation (1) and no-goods of the form (6), the refinements of dynamic racing, and the features described in Sections 3.4–3.6 were not available yet. After ensuring that the code runs sufficiently stable on a testset of over 3000 small to medium-size MIPs, we aimed to find improving solutions for the 221 problems in category `open` of MIPLIB 2017.

Smoothie was run on a cluster at Zuse Institute Berlin, which machines differ in their hardware configuration (CPUs: AMD Epyc 7542, Intel Xeon Gold 6246, Intel Xeon Gold 6338, Intel Xeon Gold 6342, Intel Xeon E5-2680, Intel Xeon E5-2697, Intel Xeon E5-4640, Intel Xeon E7-4830v3; RAM: 512GB or more). We ran Smoothie with 25 threads, each run using a compute node exclusively, and being limited to 506GB of RAM and a time limit of 12 hours. If Smoothie did not terminate within twice the time limit or reached the memory limit, the run was canceled by the cluster scheduler. A best known solution was not given to Smoothie.

Initially, for half the threads, Gurobi was run in single-threaded mode, each instance of Gurobi using a different seed for the random number generator. For the other half, Xpress was run, also in single-threaded mode, and using a different seed for the random number generator for each instance. In addition, the following parameters varied between different Xpress solvers: `HEURSEARCHROOTSELECT`, `HEURSEARCHTREESELECT`, `HEURSEARCHEFFORT`, `CUTFREQ`, `CUTFACTOR`, `TREECOVERCUTS`, `GOMCUTS`, `TREEGOMCUTS`, `VARSELECTION`, `PREPROBING`, `SBEFFORT`, and `SBESTIMATE`. For the limits on the size of the local-branching MIP (number of fractional or

wide-domain integer variables, cf. Section 3.2), the lower bound was set to $\phi = 350$ when solving with Gurobi and $\phi = 335$ when using Xpress. The upper bound Φ was initially set to 3000. The deterministic time limit for the local-branching MIP has been set to 700. If it could not be solved within this limit, then Φ is decreased by 10%. If it could be solved without branching, then Φ is increased by 5%. The minimum time a solver has to run before it can be replaced in dynamic racing, cf. Section 3.3, has been set to 1200 seconds.

Table 1 summarizes the problems from MIPLIB 2017 for which Smoothie was able to improve the best known solution[6]. The obtained solutions were checked with MIPLIB's solution checker[7] and the maximal absolute violation of integrality and constraints is reported. Even in this early state, Smoothie has already improved the best known solution for 11 problems.

problem	best known	improvement	violation
`bharat`	4170993	4039609	3.8e-7
`bmocbd3`	-373554910	-373568021	5.3e-9
`cdc7-4-3-2`	-296	-307	0.0
`ds-big`	195.43983	194.58675	0.0
`ivu59`	931	927.89475	3e-11
`mining`	-833555203	-833589149	0.0
`neos-3682128-sandon`	34666770	34666767	1.2e-6
`neos-4264598-oueme`	6038454	6006962	5.7e-6
`neos-5151569-mologa`	686759699	686748326	4.4e-8
`sct5`	-228.11949	-228.12919	4e-15
`supportcase35`	-501.23489	-34925.7	0.0

Table 1: Open MIPLIB 2017 problems for which an improving solution was found by Smoothie. Column "best known" shows the objective value of the incumbent as given by `miplib.zib.de` (version 35 of the solution file). Columns "improvement" and "violation" report the objective value and the maximal violation of integrality and constraints of a better solution found by Smoothie.

Table 2 summarizes instances that Smoothie could solve or almost solve to optimality. Due to the early state of development, and since we cannot verify the computed dual bounds independently, these results should be taken with caution. We were especially surprised by the performance for open problem `supportcase35`: a Gurobi instance that was spawned by Smoothie found the new solution and proved optimality in just 17 seconds.

4.2 October 2026

In October 2026, a version of Smoothie (githash `621423125f`) that includes all features from this report has been run again on the open instances from MIPLIB 2017. Smoothie was run with COPT 8.0.6, CPLEX 22.1.2.0, Gurobi 13.0.1 and FICO Xpress 46.01.02 on a set of identical machines with Intel Xeon Gold 6342 CPU (48 cores) and a memory limit of 483GB.

[6]The results for `ds-big`, `neos-3682128-sandon`, and `sct5` were produced from an earlier run with initial $\Phi = 2000$ (instead of 3000), the deterministic time limit for the local-branching MIP set to 50 (instead of 700), the minimum time a solver has to run before being replaced set to 60 seconds (instead of 1200), and the time limit for Smoothie itself being 6 hours only.

[7]`https://github.com/scipopt/scip/tree/v924/check/solchecker`

problem	primal bound	dual bound	gap
adult-regularized	7022.9535	7022.9535	**0.0000%**
bmocbd3	-373568021	-373603472	0.0095%
dlr1	2708118	2707853	0.0098%
gmut-76-40	-14169191	-14169529	0.0024%
gmut-76-50	-14170602	-14171898	0.0091%
mining	-833589149	-833599983	0.0013%
minutedispatchstrategy	3109.9026	3109.8537	0.0016%
neos-2974461-ibar	468906175	468728796	0.0378%
neos-3740487-motru	164.46917	164.46917	**0.0000%**
neos-4290317-perth	3017237	3017035	0.0067%
neos-4647032-veleka	27214.48	27213.98	0.0018%
neos-5044663-wairoa	4654500	4654200	0.0064%
neos-5151569-mologa	686748326	686484941	0.0384%
sct5	-228.12919	-228.1468	0.0077%
shs1014	22671.197	22668.356	0.0013%
shs1042	11070.702	11067.84	0.0258%
snp-06-004-052	1869533045	1869529031	0.0002%
snp-10-004-052	5906645422	5906645422	**0.0000%**
supportcase31	-3720093	-3721150	0.0284%
supportcase35	-34925.7	-34925.7	**0.0000%**
supportcase39	-1085058	-1085232	0.016%
uccase10	39133.657	39123.873	0.025%

Table 2: Open MIPLIB 2017 problems which optimality gap could be closed or almost closed. Columns "primal bound" and "dual bound" report the bounds computed by Smoothie, and column "gap" the corresponding relative gap.

Each subsolver was run in 12 variants as racing solvers. Smoothie was given the best known solutions as starting point.

In the first run, all 212 open MIPLIB 2017 instances were run with a timelimit of 6 hours and with the checkpointing mechanism (see Section 3.5) enabled. Smoothie solved 4 instances to optimality and computed improving solutions for 13 other instances. For 4 instances, wrong results were returned (primal or dual bounds contradicting the reference information in version 37 of the MIPLIB 2017 solution file, or a solution not being feasible). For 6 instances, the run was terminated when reaching the memory limit, and for 4 more instances Smoothie aborted due to an invalid memory access (segmentation fault).

Next, we selected all instances that did not produce wrong results, did not fail or abort, were not solved, and which terminated with a gap smaller than 10%, had improved the primal bound, or have no known solution. We started another run on these 93 instances using the same Smoothie version, hardware, and memory limit, but now with a time limit of 16 hours and restarting from the checkpoints of the first run. With this run, Smoothie solved 2 instances to optimality and computed improving solutions for 16 instances. Wrong results were obtained for 2 instances and 19 instances reached the memory limit.

Based on this result, we selected 19 instances on which progress was made with respect to primal or dual bound. On this third test set, we run Smoothie again, restarting with the checkpoints from the previous run, but the time using a timelimit of 60 hours. Smoothie solved one instance

to optimality, found 9 improving solutions, and reached the memory limit for 7 instances.

Tables 3 and 4 list the solutions for which an improvement in the primal bound was obtained and optimality was proven, respectively.

problem	best known	improvement	violation
bharat	4032573	4027050	6e-10
bmocbd3	-373568021	-373582391	2e-10
gmut-76-40	-14169478	-14169497	5e-10
ivu06-big	140.74	140.323	0
ivu59	912.7	911.698	0
lr1dr12vc10v70b-t360	1000791	1000378	0
mining	-833589149	-833592953	0
neos-5151569-mologa	686735235	686726972	4e-8
neos-5266653-tugela	65257.574	65144.089	0
nj1	382.0606	381.7715	0
nj2	382.0606	381.7715	0
nj3	382.0606	381.7715	0
ns1905797	13.98258	13.76558	3e-14
r4l4-02-tree-bounds-50	499061405	498998439	6e-9
rwth-timetable	772347	772259	7e-7
scpm1	540	537	0
seqsolve3short4288excess3	-40035	-40051	1e-12
snp-10-052-052	6059329715	6059209274	3e-8

Table 3: Open MIPLIB 2017 problems for which an improving solution was found by Smoothie in October 2026. Column "best known" shows the objective value of the incumbent as given by `miplib.zib.de` (version 37 of the solution file). Columns "improvement" and "violation" report the objective value and the maximal violation of integrality and constraints of a better solution found by Smoothie.

5 Future Extensions

The targeted design of Smoothie is depicted in Figure 3. Smoothie can be seen as an extension of UG[Xpress,*]. While the latter provides a racing ramp-up phase and a large-scale distributed parallelized tree search [22, 23], Smoothie extends this algorithm by additional solvers that run primal heuristics only and solvers that run in racing mode and generate no-goods, as described in the previous section. In the final version, Smoothie is expected to adapt the tasks to be executed dynamically. For example, if the number of open branch-and-bound nodes stored in the LoadCoordinator is not sufficient to keep all solvers busy, more solvers may be used for the dynamic racing mode, or solvers may be targeted to search for primal solutions only. It may also be possible to distribute a primal heuristic over several solvers.

When solving a single MIQCQP using a large-scale parallelized tree search approach, it is imperative to take precautions against failing compute nodes, reaching resource limits (time, memory) in solvers, or even an outage of the LoadCoordinator. For this reason, UG contains a checkpointing mechanism that periodically writes a state of the search (e.g., the incumbent solution and branch-and-bound nodes currently explored or waiting to be

problem	optimal value	violation	time
allcolor58	42	0	30s
bley_xs1	3855895.86	0	10s
gmut-76-40	-14169497	5e-10	8h1662s
nb10tb	13796452423	8e-7	2h2286s
s82	-33.7970678	1e-11	13h864s
sing5	18778435.6	1e-12	63h1396s
supportcase35	-448327.527	1e-16	7s

Table 4: Open MIPLIB 2017 problems for which Smoothie has proven optimality in October 2026. Column "optimal value" shows the obtained optimal value, "violation" reports the maximal violation of integrality and constraints, and "time" the time to prove optimality. If several runs were used to prove optimality (using checkpoint+restart), then the time of all runs has been summed up.

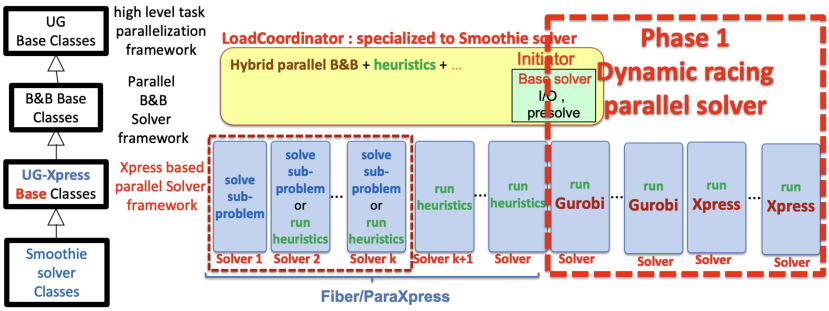

Figure 3: Smoothie target design

processed) to disk. When the solver fails, or has to be interrupted, UG can restart from the checkpoint instead of resuming the search completely from scratch. However, since a lot of nodes may be open at any time, only a set of nodes that cover all open nodes as children are actually included in the checkpoint. This has the disadvantage, that the information about some subtrees being explored already is lost. In its current racing-based version, Smoothie already writes its no-goods together with the incumbent solution to a checkpoint and can restart from it, see Section 3.5. Including the no-goods also in the checkpoints of the extended design with a distributed tree search should improve at this point, since these no-goods correspond to some already explored subtrees.

With CPLEX and COPT, two further MIQCQP solvers have already been included as racing solvers that generate and apply no-goods, see Section 3.4. The effect of these additional solvers on the performance of Smoothie remains to be evaluated computationally. Further, we also want to consider including additional state-of-the-art commercial or academic solvers, e.g., HiGHS or SCIP. While the APIs of COPT, Gurobi, and HiGHS are currently not sufficient to parallelize their tree search via UG, these solvers could still be used for racing, no-good generation, or primal heuristics.

The distributed-memory version of Smoothie, see Section 3.6, has so far been tested with a moderate number of processes only. It remains to evaluate it on large-scale computing resources and to use it for the

extended design depicted in Figure 3.

Eventually, we aim to use Smoothie to solve extremely hard MIQCQP instances to optimality by utilizing several state-of-the-art solvers and over a million CPU cores for the solution of a single MIQCQP.

6 Concluding Remarks

The UG-based parallel solvers ParaSCIP and ParaXpress have solved a number of open MIPLIB instances by utilizing large-scale parallel computing environments, i.e., supercomputers with up to 80,000 cores [19, 20, 21, 22, 23]. Even with such massive computing resources, however, at most four open instances were solved per year.

Smoothie, in contrast, is still in its first development phase and has so far been run with only 48 threads on a single compute node per instance and a time limit of up to 60 hours, see Section 4. Nevertheless, in the first single series of runs, it improved the best known solution for 11 open instances of MIPLIB 2017 and, with the caveats discussed in Section 4, solved four open instances to optimality. Therefore, even with the Phase I version of Smoothie, we can expect more optimal solutions to be found by computations at a much smaller scale than those carried out with ParaSCIP and ParaXpress.

Acknowledgments

We like to thank Timo Berthold, Gregor Hendel, and Michael Perregaard from FICO, Tobias Achterberg, Stefan Heinz, and Michael Winkler from Gurobi, and Gerald Gamrath from Cardinal Operations for a lot of valuable help on using Xpress, Gurobi, and COPT and their APIs, respectively. This work has been conducted within the *Research Campus MODAL* funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM).

References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006.

[2] Egon Balas and Robert Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

[3] David E. Bernal, Stefan Vigerske, Francisco Trespalacios, and Ignacio E. Grossmann. Improving the performance of DICOPT in convex MINLP problems using a feasibility pump. *Optimization Methods and Software*, 35(1):171–190, 2020.

[4] Timo Berthold, Thorsten Koch, and Yuji Shinano. MILP. Try. Repeat. *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021*, 2, 2021.

[5] Robert E. Bixby, E. Andrew Boyd, and Ronni R. Indovina. MIPLIB: A test set of mixed integer programming problems. *SIAM News*, 25(2):16, 1992.

[6] Robert E. Bixby, Sebastián Ceria, Cassandra M. McZeal, and Martin W.P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[7] Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, 1986.

[8] Daniel Frost and Rina Dechter. Dead-end driven learning. In *AAAI*, volume 94, pages 294–300, 1994.

[9] Gerald Gamrath, Thorsten Koch, Stephen J. Maher, Daniel Rehfeldt, and Yuji Shinano. SCIP-Jack - a solver for STP and variants with parallelization extensions. *Mathematical Programming Computation*, 9(2):231–296, 2017.

[10] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

[11] Fred Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[12] Robert Lion Gottwald, Stephen J. Maher, and Yuji Shinano. Distributed domain propagation. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics*, pages 6:1–6:11, Dagstuhl, Germany, 2017.

[13] John N. Hooker and Greger Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[14] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.

[15] Thorsten Koch, David E. Bernal Neira, Ying Chen, Giorgio Cortiana, Daniel J. Egger, Raoul Heese, Narendra N. Hegade, Alejandro Gomez Cadavid, Rhea Huang, Toshinari Itoko, Thomas Kleinert, Pedro Maciel Xavier, Naeimeh Mohseni, Jhon A. Montanez-Barrera, Koji Nakano, Giacomo Nannicini, Corey O'Meara, Justin Pauckert, Manuel Proissl, Anurag Ramesh, Maximilian Schicker, Noriaki Shimada, Mitsuharu Takeori, Victor Valls, David Van Bulck, Stefan Woerner, and Christa Zoufal. Quantum Optimization Benchmarking Library – The Intractable Decathlon. Technical Report 2504.03832, arXiv, 2025.

[16] Gioni Mexi, Dominik Kamp, Yuji Shinano, Shanwen Pu, Alexander Hoen, Ksenia Bestuzheva, Christopher Hojny, Matthias Walter, Marc E. Pfetsch, Sebastian Pokutta, and Thorsten Koch. State-of-the-art methods for pseudo-boolean solving with SCIP. Technical Report 2501.03390, arXiv, 2025.

[17] Ted K. Ralphs, Yuji Shinano, Timo Berthold, and Thorsten Koch. Parallel solvers for mixed integer linear optimization. In Youssef Hamadi and Lakhdar Sais, editors, *Handbook of Parallel Constraint Reasoning*, pages 283–336. Springer International Publishing, 2018.

[18] E Thomas Richards and Barry Richards. Nogood learning for constraint satisfaction. In *Proceedings CP-96 Workshop on Constraint Programming Applications*, page 47, 1996.

[19] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, and Thorsten Koch. ParaSCIP: a parallel extension of SCIP. In Christian Bischof, Heinz-Gerd Hegering, Wolfgang Nagel, and Gabriel Wittum, editors, *Competence in High Performance Computing 2010*, pages 135–148, 2012.

[20] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, Thorsten Koch, and Michael Winkler. Solving hard MIPLIB2003 problems with ParaSCIP on supercomputers: An update. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1552–1561, 2014.

[21] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, Thorsten Koch, and Michael Winkler. Solving open MIP instances with ParaSCIP on supercomputers using up to 80,000 cores. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 770–779, Los Alamitos, CA, USA, 2016.

[22] Yuji Shinano, Timo Berthold, and Stefan Heinz. A first implementation of ParaXpress: Combining internal and external parallelization to solve MIPs on supercomputers. In Gert-Martin Greuel, Thorsten Koch, Peter Paule, and Andrew Sommese, editors, *Mathematical Software – ICMS 2016*, pages 308–316, Cham, 2016. Springer International Publishing.

[23] Yuji Shinano, Timo Berthold, and Stefan Heinz. ParaXpress: An experimental extension of the FICO Xpress-Optimizer to solve hard MIPs on supercomputers. *Optimization Methods and Software*, 33(3):530–539, 2018.

[24] Yuji Shinano, Stefan Heinz, Stefan Vigerske, and Michael Winkler. FiberSCIP - a shared memory parallelization of SCIP. *INFORMS Journal on Computing*, 30(1):11–30, 2018.

[25] Yuji Shinano, Daniel Rehfeldt, and Tristan Gally. An easy way to build parallel state-of-the-art combinatorial optimization problem solvers: A computational study on solving Steiner tree problems and mixed integer semidefinite programs by using ug[SCIP-*,*]-libraries. In *Proceedings of the 9th IEEE Workshop Parallel / Distributed Combinatorics and Optimization*, pages 530–541, 2019.

[26] Nariaki Tateiwa, Yuji Shinano, Keiichiro Yamamura, Akihiro Yoshida, Shizuo Kaji, Masaya Yasuda, and Katsuki Fujisawa. CMAP-LAP: Configurable massively parallel solver for lattice problems. In *HiPC 2021 proceedings*, 2021.