Zuse Institute Berlin

Takustr. 7
14195 Berlin
Germany

YUJI SHINANO[1], STEFAN VIGERSKE[2]

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

Yuji Shinano[1] Stefan Vigerske[2]

October 26, 2025

Abstract

Mixed-Integer Linear Programming (MIP) is 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 problems that currently cannot be solved. With the Smoothie solver we intend to solve extremely hard MIP problems by building on the many years that went into the development of several state-of-the-art MIP 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 MIP on a shared- or distributed-memory computing infrastructure, thereby employing several established MIP solvers simultaneously. For the first development phase, which is the topic of this report, both FICO Xpress and Gurobi are used in concurrent mode on a single machine, 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 MIP 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 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

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

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

in highly parallelized form on supercomputers [19] by means of the UG framework[2] and its incarnations ParaSCIP [18] and ParaXPRESS [22]. 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. 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]) [18, 20], FiberSCIP (ug[SCIP,C++ threads]) [23], and ParaXpress (ug[Xpress,MPI]) [21, 22], or the parallelization of SCIP-based solvers with little additional effort [24, 9, 15].

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[3] [16]. 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 [25].

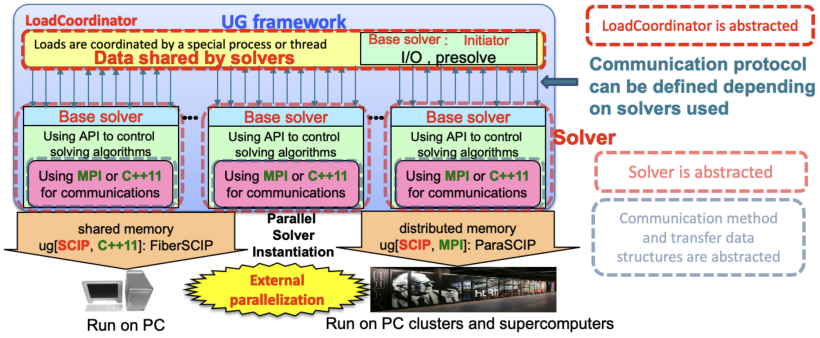

Figure 1: The UG Framework version 1.0

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

[2] https://ug.zib.de
[3] https://ug.zib.de/doc-1.0.0/html/CONCEPT.php

based solver is parallelized, but also additional MIP 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 MIP 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 FICO Xpress and Gurobi, both being among the strongest MIP solvers currently available. The corresponding design of Smoothie is illustrated in Figure 2.

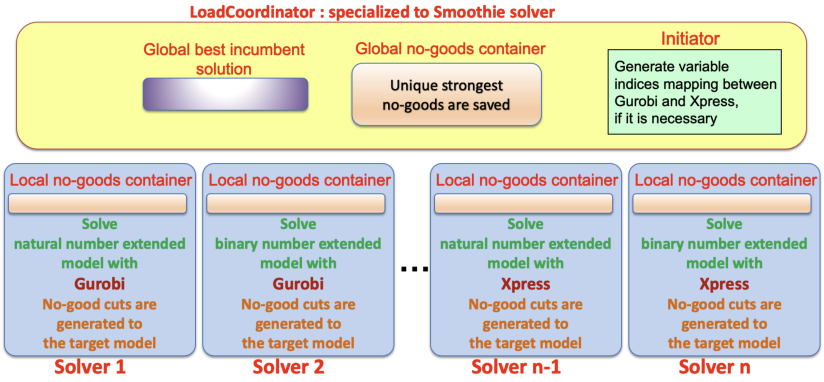

Figure 2: Smoothie Phase I design

3.1 No-Good Cut Formulation

Consider a MIP given in the following general form:

$$\min\{c^\top x \ : \ Ax \leq b, l \leq x \leq u, x_j \in \mathbb{Z}, \forall j \in I\}, \tag{1}$$

with matrix $A \in \mathbb{R}^{m \times n}$, vectors $b \in \mathbb{R}^m$ and $c, l, u \in \mathbb{R}^n$, and a subset $I \subseteq \{1, \ldots, n\}$ indicating the integer variables. 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 MIP is an LP-based branch-and-bound, 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, 17, 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.,

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

for some finite constant Δ. In the current implementation, $\Delta = 30$ is used. We then refer to

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

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

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

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

3.2 Racing Mode with No-Good Generation and Distribution

Smoothie runs several instances of solvers Gurobi and Xpress in parallel in different threads, each one using either formulation (4) or (5) and one of a number of different parameter settings. 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.

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 Gurobi and Xpress 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, Gurobi or Xpress on formulation (4) or (5) with the racing solvers settings) is copied and the fixings $x_j := \tilde{x}_j$, $j \in I \setminus F$ are applied. This *local branching MIP* is then solved using a limit on the solvers deterministic time (also called *work limits*). 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 $(\tilde{x}_j)_{j \in I \setminus F}$. This no-good cut is then stored in the solvers no-good container. If solving the local branching MIP 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 MIPs. If the local branching MIP 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.

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, the five solvers with the worst dual bound are stopped and replaced by new tasks when the incumbent solution is updated. 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.
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).

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 10 solvers with most inferior dual bound are considered for replacement. If all of them have not reached their minimal running time yet, the restart of solvers is postponed.

4 Preliminary Computational Results

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. 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. For now, 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[4]. The obtained solutions

[4]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.

were checked with MIPLIB's solution checker[5] 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.

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 [21, 22], 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 MIP 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 processed) to disk. When the solver fails, or has to be interrupted, UG can restart from

[5] `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.

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. Including the newly added no-goods into the checkpointing file should improve at this point, since these no-goods correspond to some already explored subtrees.

Finally, we also want to consider including additional state-of-the-art commercial or academic solvers, e.g., COPT, 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.

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

Acknowledgments

We like to thank Gregor Hendel and Michael Perregaard from FICO and Michael Winkler from Gurobi for a lot of valuable help on using the APIs of Xpress and Gurobi, respectively.

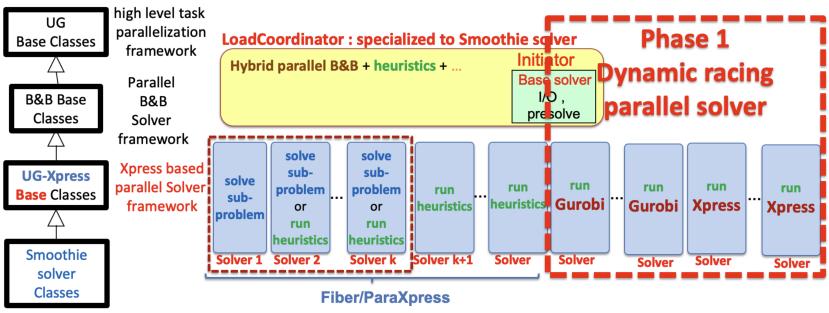

Figure 3: Smoothie target design

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] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.