Smoothie

Smoothie: Solver Mixing Object OrienTed Hybrid Integrated Executable

***Yuji Shinano, Stefan Vigerske**, Zuse Instiute Berlin*

In Short

- Development of the strongest parallel MIP solver in terms of solving previously unsolved instances to optimality.
- Sharing information dynamically among different solver implmentations.
- Generating a single binary to make it easy to run on any parallel computing environment.

We deal with solvers for NP-hard mixed-integer linear optimization problems in the form

$$\min\{c^\top x : Ax \leq b, l \leq x \leq u, x_j \in \mathbb{Z}, \text{ for all } j \in I\}, \tag{0.1}$$

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 $I \subseteq \{1, \ldots, n\}$. Currently, there exist several excellent commercial solvers, e.g., COPT[1], FICO XPRESS[2], Gurobi[3], and non-commercial solvers, e.g., HiGHS[4] and SCIP[5], all implementing mathematically highly sophisticated algorithms to tackle (0.1). Current state-of-the-art solvers use a branch-and-bound algorithm where a linear relaxation ((0.1) without $x_j \in \mathbb{Z}$, $j \in I$) is used for bounding, and branching and cutting plane generation are used to enforce the integrality restriction $x_j \in \mathbb{Z}$, $j \in I$. Additional algorithmic components like presolve and primal heuristics are used to accelerate the search for feasible solutions and proving optimality. Hereby, presolve refers to a collection of problem reductions that are typically applied in advance of the branch-and-bound procedure. Cutting plane generation is a method that tightens the linear relaxation by means of additional linear inequalities. Primal heuristics focus on the generation of feasible solution early and during the branch-and-bound algorithm.

Benchmarks have shown that solvers often excel on different instances, so that no solver's performance dominates another. The same can be observed when running a solver with different parameter settings. There is no single setting that is best for all instances. Therefore, a natural idea to solve hard instances is to run several solvers in parallel. By adding communication of feasible solutions and other useful information between different solvers, it is possible to obtain a new solver that potentially outperforms each base solver.

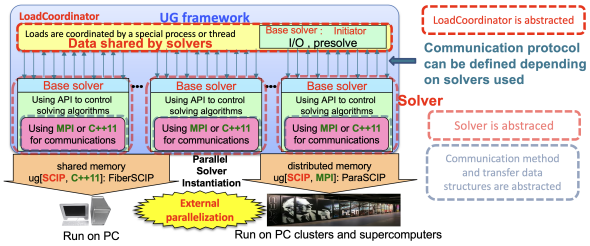

***Figure 1:** UG reached version 1.0: Controller object* `LoadCoordinator` *is also abstracted*

We had a series of NHR projects related to the Ubiquity Generator (UG) framework[6]. Originally, UG was developed to parallelize a state-of-the-art branch-and-bound based solver. This proposal is a follow up to the latest project Generalized UG (bem00052), which aims to develop a single unified framework to parallelize both branch-and-bound and non-branch-and-bound based solvers. The generalized UG framework reached version 1.0, in which not only the solver and communication parts, but also the controller part (the `LoadCoordinator`) were abstracted. This allows users of UG to design and implement specific controlling mechanisms, such as load balancing, for a specific parallel solver by defining its own message passing protocols; see also Figure 1.

There are a few successful results of previously unsolved instances being solved by using large scale computing environments. ParaSCIP and ParaXpress have solved 23 open MIP instances [1][2] and UG[SCIP-Jack, MPI] has solved five open Steiner tree problem instances [3][4]. These successful results were possible by using the parallelization software framework UG [5], which parallelizes existing state-of-the-art branch-and-bound based solvers. Basic concept of the UG framework is to exploit the performance of state-of-the-art "base solvers", such as SCIP, XPRESS, etc., without the need for parallelization within the base solver. The base solvers and communication libraries are then abstracted within UG.

This project aims to develop the strongest parallel MIP solver in terms of solving previously un-

[1] https://www.copt.de/
[2] https://www.fico.com/en/products/fico-xpress-optimization
[3] https://www.gurobi.com/
[4] https://highs.dev/
[5] https://scipopt.org/
[6] https://ug.zib.de

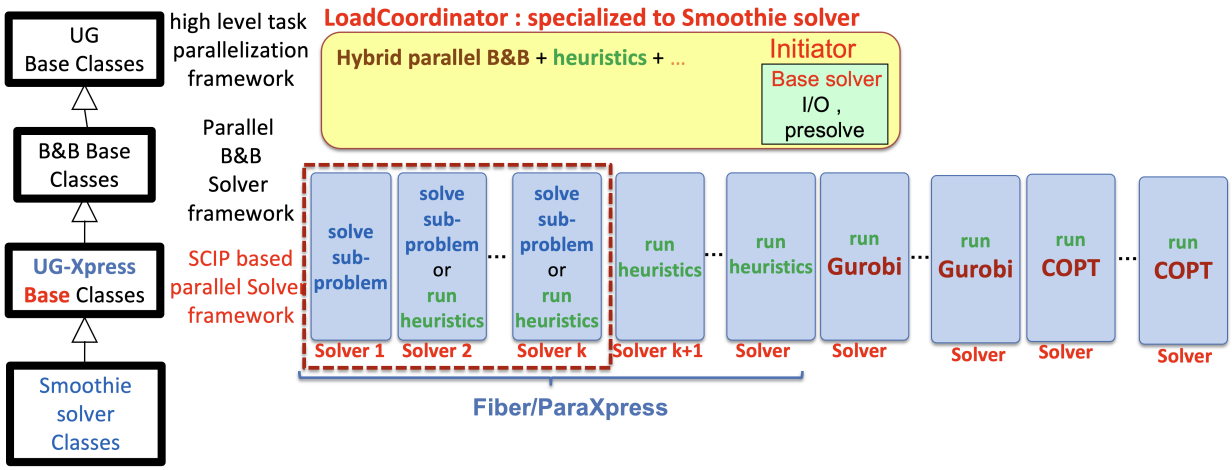

Figure 2: *Final configuration of Smoothie*

solved instances to optimality. To achieve this purpose, we consolidate the knowledge that has been akkumulated so far to develop the software framework *Smoothie (Solver Mixing Object OrienTed Hybrid Integrated Executable)*. Figure 2 shows the final configuration of Smoothie, which is an extension of Fiber/ParaXpress with, initially, Gurobi and COPT. We have decided to use these solvers as they frequently appear at the top of MIP solver benchmarks. Since we aim to solve previously unsolvable instances, we need to use the best performing solvers as base solvers. At a later stage, we plan to utilize also existing UG-adapters for the non-commercial solvers SCIP and HiGHS. The generalized UG, that is, the UG version 1.0, allows to develop a specialized load coordinator for Smoothie, in which next to the current best feasible solution also additional information can be shared among solvers. This concept has already been tested in the POEM project (bem00078) and should be developed further for Smoothie.

Currently, the next iteration of the MIP instance library [6], MIPLIB 2024, is developed by the community of MIP solver developers[7]. We believe that Smoothie can help to obtain optimal or nearly-optimal solutions for many of the new challenging instances that will be part of MIPLIB 2024.

We plan to develop Smoothie in two phases. In the first phase, we develop *Racing-Smoothie*, a solver for a large scale racing mode, in which XPRESS, Gurobi, and COPT are run in parallel with varying settings and random seeds, while communicating feasible solutions and other information. In this phase, we can test the effectiveness of information sharing. In the second phase, Racing-Smoothie will be extended to *Smoothie* by using ParaXpress for parallel branch-and-bound.

[7] https://miplibsubmissions.zib.de/

WWW

http://www.zib.de

More Information

[1] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler, *Proc. IPDPS* (2016). doi:10.1109/IPDPS.2016.56

[2] Y. Shinano, T. Berthold, and S. Heinz, *Optimization Methods and Software* **33**, no. 3, 530–539, (2018). doi:10.1080/10556788.2018.1428602

[3] Y. Shinano, D. Rehfeldt, and T. Gally, *Proc. IPDPSW* (2019). doi:10.1109/IPDPSW.2019.00095

[4] Y. Shinano, D. Rehfeldt, and T. Koch, *Proc. CPAIOR* (2019). doi:10.1007/978-3-030-19212-9_35

[5] Y. Shinano *Oper. Res. Proc.(2017)* (2018). doi:10.1007/978-3-319-89920-6_20

[6] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano *Math. Program. Comput.* (2021). doi:10.1007/s12532-020-00194-3

Project Partners

GAMS

Funding

Forschungscampus Modal

DFG Subject Area

409-02