Transferring Information across Restarts in MIP

Timo Berthold[1][0000−0002−6320−8154], Gregor Hendel[1], and Domenico Salvagnin[2][0000−0002−0232−2244]

[1] Fair Isaac Germany GmbH, Stubenwald-Allee 19, 64625 Bensheim, Germany
{timoberthold,gregorhendel}@fico.com
[2] University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy
salvagni@dei.unipd.it

Abstract. Restarting a solver gives us the chance to learn from things that went good or bad in the search until the restart point. The benefits of restarts are often justified with being able to employ different, better strategies and explore different, more promising parts of the search space. In that light, it is an interesting question to evaluate whether carrying over detected structures and collected statistics across a restart benefits the subsequent search, or even counteracts the anticipated diversification from the previous, unsuccessful search.
In this paper, we will discuss four different types of global information that can potentially be re-used after a restart of a mixed-integer programming (MIP) solver, present technical details of how to carry them through a represolve after a restart, and show how such an information transfer can help to speed up the state-of-the-art commercial MIP solver FICO Xpress by 7% on the instances where a restart is performed.

Keywords: mixed integer programming · restart · global search

1 Introduction

Restarts have been used in SAT solvers for over 20 years [21]. It was quickly picked up by the MIP community in the form of root restarts [2, 3] and in other areas of optimization, like global optimization [17]. For an overview on restarting algorithms in the areas of CP, AI, and OR, see [23]. In this paper we consider the solution of *mixed-integer programs (MIPs)*, which are optimization problems of the form

$$x^\star \in \operatorname{argmin}\{c^T x \mid Ax \geq b, x_j \in \mathbb{Z} \text{ for all } j \in \mathcal{I}\},$$

with $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, and $\mathcal{I} \subseteq \{1, \ldots, n\}$.

For some time, restarts for MIP have primarily revolved around the rule of thumb of restarting the root node processing in the case enough variables have been globally fixed so that another round of full presolving seems beneficial [2]. More recently, restarts of the tree search have gained attention in the MIP community, see, e.g., [6], and most MIP solvers employ them nowadays [11, 13, 18]. Unlike root restarts, tree restarts are often based on an extrapolation of the

remaining time until the tree search is finished [16]. In contrast to SAT solving, aggressive periodic restarts of the tree search have not proven advantageous for MIP in the vast majority of cases for a few reasons:

1. Solving the initial root problem of a MIP is often orders of magnitude more expensive than solving a local node. This makes tree restarts in the early phase of the search way more expensive than their SAT counterpart.
2. While in SAT the set of learned conflicts completely describes the search space already explored, at least until conflict purging kicks in, the same does not apply to the MIP case, where the search strategy is hardly ever a pure DFS and conflict clauses are not always learnt. Thus, a tree restart inevitably leads to some redundant work being performed.
3. MIPs are only very rarely feasibility problems, where a tree restart might just be beneficial because it makes the algorithm find a feasible solution sooner that it would have done without. The optimality proof requires a certain amount of search to be performed no matter what, making restarts in a later phase of the search unattractive, if not harmful.

For those reasons, MIP solvers typically apply very few tree restarts (only one in most cases) and aim at addressing different types of problems with the solver behavior before and after said restarts. The solvers use settings aimed at easy problems in the first phase. Those will typically spare some expensive subroutines and make those problems solve faster. If a restart is conducted, the solver is aware that this problem is relatively hard to solve and will require a certain amount of tree search. Thus, it can adapt and employ more aggressive settings and expensive additional techniques right from the beginning of the subsequent searches. Furthermore, a restart might help in mitigating and even exploiting performance variability effects [19].

Given that one of the primary purposes of tree restarts is to make the search after the restart different from the one before, it is not obvious to which extent information from a given tree search should be carried over to be used in the next search second run, and which type of information is most suitable for such a transfer. The goal of this paper is to address these questions and evaluate the impact of global information transfer, taking the solver restart strategy as a given.

2 Global Information

A main difference between MIP and CP solvers is how constraints interact with each other. In CP solvers, the variables' domain is the central communication object, and each constraint individually contributes to tightening the domain store and guiding the search by propagation. Hence, information is processed very locally, on a constraint-by-constraint level. In MIP solvers, the central object is the LP relaxation, which is solved to optimality, considering all constraints simultaneously.

This is our role model for what we would like to call global information. The data structures that we consider for transfers across restarts have in common that they can connect pieces of information from multiple constraints into a single structure, and the information contained therein is globally valid for the problem at hand. Some of them are gained during the search, like *cutting planes* [25] and *pseudo-costs* [8], and they depend on algorithmic choices of the solver. Hence, rerunning the search with slightly different start conditions would lead to constructing a different object. Others, like the *implication graph* [24], are detected in presolving or during root node processing. These structures might only be partially created for large input problems as the involved detection procedures use certain work limits. Again, a slight change in start conditions would lead to a different result, and we cannot simply recollect the exact same information after a restart.

We will now briefly sketch each type of global information relevant to the present work. For an introduction to how global structures can be used for primal heuristics, see [14].

The conflict pool. MIP solvers attempt to create conflicts whenever pruning a node due to infeasibility or cutoff. They are generated by analyzing the sequence of branching decisions and propagations that led to the node being pruned, with the aim of extracting a small subset of those bound changes as an explanation for the infeasibility. From such a conflict, we can derive a conflict clause which is typically formulated as a disjunction of bounds [1]. A conflict clause is a globally valid, not necessarily linear, constraint that states that at least one of the bounds in the conflict must take a different value than in the infeasible subproblem. Note that it is equivalent to state the conflict as a conjunction of the bound changes that led to infeasibility. This is how Xpress stores conflict clauses. MIP solvers typically collect conflicts in a pool, see, e.g., [29] and only keep them for a certain number of nodes. The content of this pool highly depends on the search tree, and how much of it has already been traversed.

Cutting planes Throughout the search, and in particular at the root node, MIP solvers generate cutting planes to tighten the LP relaxation, improve the dual bound, and reduce the number of fractional variables. For an overview on cuts in MIP, see [20, 27, 30]. Globally valid cutting planes, as those separated at the root, can be either directly added to the current LP relaxation or added to a so-called *cut pool*, from which they can be separated again on demand, e.g., at subsequent nodes [25]. Transferring globally valid cuts after restarts can be effective in a twofold way: a) we can warmstart the root cutloop with a set of cuts which have proven useful at the previous root node and b) if we take them into account while represolving they can lead to further reductions.

The implication graph. An implication is a relation between two variables that state that a bound change in one variable leads to a bound change in another variable. E.g., for $y \leq ax + b$, it holds that a tightened upper bound on x implies an upper bound on y, which is potentially tighter than the current upper bound.

The set of all such (known) relations for a MIP can be represented as a directed graph, where each arc represents an implication and each node represents a bound change on a variable [24]. Implications might be directly deduced from single constraints but, much more interestingly, we might find them by *probing*, i.e., the process of tentatively changing the bound of a variable and propagating this bound change. Since probing is relatively expensive, it is typically not done on all variables but only on a few promising ones under tight working limits.

Pseudo-costs. On MIP problems with a nonzero objective function, branching decisions that raise the LP objective, and hence the node dual bound in the created children, are essential to prune subproblems and thus create small search trees quickly. The pseudo-costs [8] of a variable x_j summarize the average change of the LP objective observed after branching on x_j. Prioritizing variables based on their recorded pseudo-costs is still a state-of-the-art selection strategy because it provides a good selection quality that is computationally cheap to evaluate. For a recent overview on branching in MIP, see [12]. The main disadvantage of pseudo-costs is that they are uninitialized at the beginning of the search, when the most crucial top-level branching decisions have to be made. The typical remedy is to perform an explicit look-ahead called strong branching [22] for a number of times [5, 15], to make an informed branching selection and initialize the pseudo-costs of the tested candidates. In the context of tree restarts, pseudo-costs from the first run can be transferred to have branching statistics available right from the beginning of the subsequent search.

3 Implementation Details

This section highlights the algorithmic choices in FICO Xpress [11] regarding the transfer of global information across restarts. Trivially, we always transfer the primal and dual bounds from a previous run. Further, we preserve the incumbent solution, although it may not remain feasible for the remaining search problem due to additional presolving, in which variables might be fixed to solution values that differ from their value in the incumbent solution, e.g., by dual reductions [4] or even reduced cost fixing.

Mapping between presolve spaces. Whether in the tree or at the root, one major effect of a restart is that the problem gets (re)-presolved another time. To transfer a piece of information across a restart, we need to be able to map a mathematical object involving variables in the presolve space before the restart to the corresponding mathematical object (if any) involving variables in the presolve space after the restart.

We compute a mapping between the two presolve spaces in the following way. Independent of restarts, the solver maintains a stack with the presolve reductions applied so far, as this is needed, among other things, to map presolved solutions back to the original space. When represolving, new reductions are simply appended to the very same stack. To compute a mapping between

the two spaces, we initialize the mapping to the identity and then process the new presolve reductions, in the order in which they were added to the stack, and update the mapping accordingly, depending on the logic of each individual reduction. In particular, for each variable in the old presolve space, we distinguish the following cases:

1. the variable got fixed to some value and hence removed from the problem. In this case, we mark the variable as *fixed* in the mapping and store the value it was fixed to.
2. the variable was carried through unmodified but has potentially a different index in the new presolved space. We simply store the new index of the variable.
3. the variable changed meaning (e.g., it changed type) or was eliminated in such a way that its value can only be computed a-posteriori during the postsolve phase. In this case, we mark the variable as *unmappable*. Any object involving an unmappable variable will become unmappable itself and thus not transferred across a restart.

Another essential piece of information that we need for a correct transfer is scaling [26]. However, scaling itself is not considered a presolve reduction in FICO Xpress, and thus it does not contribute to the stack. Still, we need to keep track of the scaling factors (w.r.t. the original problem) before and after represolve to compute the *intermediate* scaling factors between the two presolve spaces. Note, however, that integer variables are never scaled.

Pseudo-costs. For each variable that is still mappable after a represolve, we condense the entire pseudo-costs from both strong branches and regular branches on a variable x from the old run into a single pseudo-cost record, in the spirit of [9]. As a result, we have initialized pseudo-costs available from the beginning of the search in the new run. They are not considered *reliable* [5] yet, such that also these variables will eventually be subject to further strong branching evaluations. When recording new branching information during the new run, the obtained, more recent pseudo-cost information eventually outweighs the transferred information.

The transfer of the variable branching history follows the intuition that the transferred information is still accurate when the problem does not change too much during a represolve. Following this logic, we keep track of the relative change in the number of rows, columns, and matrix nonzero elements after the restart. If one of these numbers changes by more than 10 % (5 % in the case of rows), we discard the collected pseudo-costs and start the search with all pseudo-costs uninitialized.

Conflicts. Conflicts stored in the conflict pool are processed one by one, using the mapping and scaling factors computed after represolve. Given a conflict in conjunctive form, each bound change is mapped to the new space individually, leading to one of the following cases:

1. the bound change is on an unmappable variable: the conflict will be dropped;
2. the bound change is on a fixed variable: depending on the fixing value, we can either drop the bound change from the conflict (because it is always satisfied) or drop the conflict itself (the condition is never satisfied and thus the conflict would be useless);
3. the bound change is on another variable: simply update to the new index.

If a conflict survives the mapping process, it is finally dealt with as follows:

1. if the conflict is now empty, the problem is infeasible, and we can stop immediately;
2. if the conflict has size one, we turn it into a globally valid bound change;
3. otherwise, the conflict is added to the new conflict pool.

Implication table. Processing of the implication table is by far and large similar in spirit to the processing of the conflict pool. Each implication in the table is mapped individually, and the outcome depends on a few cases:

1. if either the implying or the implied variable is unmappable, then the implication itself is discarded;
2. if either the implying or the implied variable is fixed, then we can turn the implication into a globally valid bound on the other variable. If both variables got fixed, the implication either directly proves infeasibility or is redundant and can be discarded.
3. if both variables survived, we map the implication by updating the indices of the involved variables and recomputing the coefficients using the intermediate scaling factors.

Cuts. While we could implement the mapping of globally valid cutting planes with the same logic that we used for conflicts and implications, the represolve of cuts follows a completely different logic for historical and technical reasons. In particular:

1. first, cuts are temporarily added to the problem before represolve, as if they were regular constraints in the model;
2. then, we execute the represolve;
3. finally, the (surviving) cuts are removed again from the model and added to the new global cut pool.

There are advantages and disadvantages to this strategy: on the one hand, the presence of cutting planes in the model might hinder some reductions, like column domination; on the other hand, their presence could also lead to a stronger presolved model, as bound tightening can in principle derive tighter bounds exploiting the additional constraints for propagation. In general, cutting planes prevent those reductions that would make the cuts themselves unmappable, which again can be argued both for and against. We did not experiment with alternative approaches, as our current strategy works well in practice: however, that is certainly an interesting direction for future research.

4 Computational Experiments

In this section, we evaluate the transfer of global information across represolves computationally. To this end, we perform two runs with the recently released FICO Xpress 8.13 solver, with the transfer of global information enabled and disabled, respectively. As benchmark set we use a mix of publicly available and customer MIP instances that also serves as one of the main test sets in our daily Xpress MIP development. In order to mitigate the effect of performance variability, we solve three permutations of each of these 619 MIP instances, resulting in a testbed of 1857 instances in total: the unpermuted model and two cyclic permutations [10] characterized by a different initial random seed used for perturbing the rows, columns, and integer variables of the instance.

Both runs are performed on a cluster of 64 identical machines, each equipped with 2 Intel(R) Xeon(R) CPUs E5-2640 v4 @ 2.40GHz and 64 GB of memory. We set a time limit of 4 hours, equaling 14400 seconds, and allow each job to use 20 parallel threads.

		no-transfer			transfer			relative	
Class	N	Solved	Time	Nodes	Solved	Time	Nodes	Time	Nodes
all	1857	1844	72.97	760	1846	71.03	694	0.97	0.91
[10,14400}	1802	1798	108.22	1039	1800	104.97	952	0.97	0.91
[100,14400}	685	681	409.56	4167	683	390.44	3615	0.95	0.87
affected instances									
all	670	662	113.77	8781	664	105.97	6841	0.93	0.78
[10,14400}	565	561	165.65	17191	563	153.08	13638	0.92	0.79
[100,14400}	338	334	434.21	44505	336	393.87	33397	0.91	0.75

Table 1. Computational results obtained when enabling or disabling the transfer of global information across restarts.

We summarize the results of this experiment in Table 1. The table shows solved instances, time, and nodes for both tested versions for different instance classes. The top part of the table shows the results for subsets of the entire instance bed. The bottom part of the table shows the results only for affected instances, i.e., instances on which the transferred global information affects the solver behavior after the restart. The instance class all in the first row shows the results for the entire testbed. For each class, the number of involved instances is shown in column N. We also present the results for two bracketed subsets [10,14400} and [100,14400}. A bracketed subset consists of all instances that could be solved by at least one of the two tested versions, and the slower of the two solves took at least 10 or 100 seconds or timed out, respectively. The bracketing convention is helpful to filter instances that are solved fast by both

versions. Time and node results use a shifted geometric mean [2] with shift values of 10 seconds and 100 nodes.

The overall results suggest that transferring global information is valuable for the search procedure across the considered subsets of instances, yielding an overall time improvement of 3 % and node improvement of 9 %. The table shows that the number of affected instances amounts to roughly one-third of the test set, which is almost all instances for which FICO Xpress actually performs a restart.

On the affected instances, the performance improvements are naturally more pronounced. Overall, we see a gain of 7 % time and 22 % nodes. The observed improvement amounts to 9 % time and 25 % nodes on the most challenging bracket. All time and node improvements are significant according to a Wilcoxon signed rank test [28] with a confidence level of 95 %.

Especially the reduction in the solving nodes is a clear indication that the transfer of global information helps the search to make better decisions in the second run. When comparing the relative reduction in time and nodes, we see that the transferred global information comes at a cost. Especially the transferred cutting planes increase the size of the matrix, which increases the time to solve the node LP relaxations in the second run.

When testing the transfer of individual pieces of global information, we found that transferring cutting planes across represolves leads to the most substantial improvements. Transferring cutting planes alone causes a change in the solution path on almost all of the affected instances in Table 1, and is responsible for a speed up of 3 %, across all of the considered brackets. Transferring pseudo-costs only also gives a significant speedup, albeit a bit smaller than the one from cutting planes. A reason for this is that fewer instances (194) are affected by the change: this is partially due to the fact that pseudo-cost transfer will only affect tree restarts, but not root restarts, and partially due to the employed limits on the relative change in the represolved problem. However, if we restrict to the set of instances affected by the change, transferring pseudo-costs is responsible for a speed up of 5 %. As for the two remaining types of global information, namely conflicts and implications, our experiments show a performance-neutral result or slightly detrimental performance impact when transferred across restarts. In particular, transferring the implication graph is performance-neutral to 1% improvement, depending on the bracket you consider. Our interpretation is that many of the implications would be rediscovered in presolving and probing in the second run anyway, so there is not much value. This was also a reason for not yet trying to transfer the clique table, which is similar in nature. Finally, for conflicts, we observed slightly detrimental behavior, which surprised us because conflicts carry global information that cannot be easily, and cheaply, recomputed. Our explanation for this is that the transfer of old conflicts together with the strict limits on the overall conflicts pool size prevents fresh conflicts from being learned. This clearly asks for a more careful re-tuning which is part of future work.

5 Conclusion & Outlook

In this paper we showed that transferring some pieces of global information across a restart can be quite beneficial for the performance of a state of the art MIP solver, both in terms of average runtime and size of the resulting enumeration tree.

As future work we plan to investigate why transferring some of the global structures, namely conflicts and implications, did not result in a measurable performance improvement, and extend the transfer to additional structures, most notably the cliques in the global clique table [7].

References

1. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization **4**(1), 4–20 (2007)
2. Achterberg, T.: Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin (2007)
3. Achterberg, T.: SCIP: Solving Constraint Integer Programs. Mathematical Programming Computation **1**(1), 1–41 (2009)
4. Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS Journal on Computing **32**(2), 473–506 (2020)
5. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters **33**(1), 42–54 (2005)
6. Anderson, D., Hendel, G., Le Bodic, P., Viernickel, M.: Clairvoyant restarts in branch-and-bound search using online tree-size estimation. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 1427–1434 (2019)
7. Atamtürk, A., Nemhauser, G.L., Savelsbergh, M.W.: Conflict graphs in solving integer programming problems. European Journal of Operational Research **121**(1), 40–55 (2000)
8. Bénichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribière, G., Vincent, O.: Experiments in mixed-integer programming. Mathematical Programming **1**, 76–94 (1971)
9. Berthold, T., Feydy, T., Stuckey, P.J.: Rapid learning for binary programs. In: Lodi, A., Milano, M., Toth, P. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010. LNCS, vol. 6140, pp. 51–55. Springer Berlin Heidelberg (2010)
10. Berthold, T., Hendel, G.: Learning to scale mixed-integer programs. Proceedings of the AAAI Conference on Artificial Intelligence **35**(5), 3661–3668 (2021)
11. FICO Xpress Optimization (2021), https://www.fico.com/en/products/fico-xpress-optimization
12. Gamrath, G.: Enhanced Predictions and Structure Exploitation in Branch-and-Bound. Ph.D. thesis, Technical University Berlin (2020)
13. Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., Hendel, G., et al.: The SCIP Optimization Suite 7.0. Tech. Rep. 20-10, ZIB (2020)

14. Gamrath, G., Berthold, T., Heinz, S., Winkler, M.: Structure-driven fix-and-propagate heuristics for mixed integer programming. Mathematical Programming Computation **11**(4), 675–702 (2019)
15. Hendel, G.: Enhancing MIP branching decisions by using the sample variance of pseudo costs. In: Integration of AI and OR Techniques in Constraint Programming. vol. 9075, pp. 199 – 214 (2015), in press
16. Hendel, G., Anderson, D., Le Bodic, P., Pfetsch, M.E.: Estimating the size of branch-and-bound trees. INFORMS Journal on Computing **0**(0) (2021)
17. Hu, X., Shonkwiler, R., Spruill, M.C.: Random restarts in global optimization. Tech. rep., Georgia Institute of Technology (2009)
18. ILOG CPLEX Optimization Studio 12.10.0 (2019), https://www.ibm.com/docs/en/icos/12.10.0?topic=v12100-changes-log
19. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. In: Theory Driven by Influential Applications, pp. 1–12. INFORMS (2013)
20. Marchand, H., Martin, A., Weismantel, R., Wolsey, L.A.: Cutting planes in integer and mixed integer programming. Discrete Applied Mathematics **123/124**, 391–440 (2002)
21. Moskewicz, M.H., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Annual Design Automation Conference. pp. 530–535. DAC '01, Association for Computing Machinery (2001)
22. Nemhauser, G.L., Wolsey, L.A.: Integer programming and combinatorial optimization. Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin **20**, 8–12 (1988)
23. Pokutta, S.: Restarting algorithms: Sometimes there is free lunch. In: Hebrard, E., Musliu, N. (eds.) International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 22–38. Springer (2020)
24. Savelsbergh, M.W.P.: Preprocessing and probing techniques for mixed integer programming problems. ORSA Journal on Computing **6**(4), 445–454 (1994)
25. Savelsbergh, M.W.P., Sigismondi, G.C., Nemhauser, G.L.: Functional description of MINTO, a mixed integer optimizer. Technische Universiteit Eindhoven (1991)
26. Tomlin, J.A.: On scaling linear programming problems. In: Computational practice in mathematical programming, pp. 146–166. Springer (1975)
27. Tramontani, A.: Enhanced mixed integer programming techniques and routing problems. Ph.D. thesis, Springer (2011)
28. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics bulletin pp. 80–83 (1945)
29. Witzig, J., Berthold, T., Heinz, S.: Experiments with conflict analysis in mixed integer programming. In: Salvagnin, D., Lombardi, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 14th International Conference, CPAIOR 2017. LNCS, vol. 10335, pp. 211–220. Springer Berlin Heidelberg (May 2017)
30. Wolter, K.: Implementation of Cutting Plane Separators for Mixed Integer Programs. Master's thesis, Technische Universität Berlin (2006)