Verifying Integer Programming Results

Kevin K. H. Cheung[1], Ambros Gleixner[2], and Daniel E. Steffy[3]

[1] School of Mathematics and Statistics, Carleton University, Ottawa, ON, Canada. kevin.cheung@carleton.ca

[2] Department Optimization, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany gleixner@zib.de

[3] Department of Mathematics and Statistics, Oakland University, Rochester, MI, USA steffy@oakland.edu

Abstract. Software for mixed-integer linear programming can return incorrect results for a number of reasons, one being the use of inexact floating-point arithmetic. Even solvers that employ exact arithmetic may suffer from programming or algorithmic errors, motivating the desire for a way to produce independently verifiable certificates of claimed results. Due to the complex nature of state-of-the-art MIP solution algorithms, the ideal form of such a certificate is not entirely clear. This paper proposes such a certificate format designed with simplicity in mind, which is composed of a list of statements that can be sequentially verified using a limited number of inference rules. We present a supplementary verification tool for compressing and checking these certificates independently of how they were created. We report computational results on a selection of MIP instances from the literature. To this end, we have extended the exact rational version of the MIP solver SCIP to produce such certificates.

Keywords: correctness, verification, proof, certificate, optimality, infeasibility, mixed-integer linear programming

1 Introduction

The performance of algorithms for solving mixed-integer linear programs to optimality has improved significantly over the last decades [3, 4]. As the complexity of the solvers increases, a question emerges: *How does one know if the computational results are correct?*

Although rarely, MIP solvers do occasionally return incorrect or dubious results [13]. Despite such errors, maintaining a skeptical attitude that borders on paranoia is arguably neither healthy nor practical. After all, machines do outperform humans on calculations by orders of magnitude and many tasks in life are now entrusted to automation. Hence, the motivation for asking how to verify correctness of computational results is not necessarily because of an inherent distrust of solvers. Rather, it is the desire to seek ways to identify and reduce errors and to improve confidence in the computed results. Previous research on computing accurate solutions for MIP has utilized various techniques including interval arithmetic [37], exact rational arithmetic [6, 13, 17], and safely derived cuts [12]. Nevertheless, as stated in [13], "even with a very careful implementation and extensive testing, a certain risk of an implementation error remains".

One way to satisfy skeptics is formal code verification as is sometimes found in software for medical applications and avionics. For global optimization, progress in this direction has been made very recently [36, 41]. For modern MIP solvers, which easily consist of several 100,000 lines of code, this may be an ambitious goal. An alternative is to build solvers that output extra information that facilitates independent checking. We shall use the word *certificate* to refer to such extra information for a given problem that has been solved. Ideally, the certificate should allow for checking the results using fewer resources than what are needed to solve the problem from scratch. Such a certificate could in principle be used in formal verification using a proof checker as done in the Flyspeck Project [19, 39, 42] for a formal proof of Kepler's Conjecture, or informal verification as done by Applegate *et al.* [5] for the Traveling Salesman Problem and by Carr *et al.* [11] in their unpublished work for MIP in general. Naturally, certificates should be as simple to verify as possible if they are to be convincing.

We highlight two specific applications where solution verification is desirable. First, Achterberg [1] presented MIP formulations for circuit design verification problems, for which solvers have been shown to return incorrect results [13]. Second, Pulaj [40] has recently used MIP to settle open questions related to Frankl's conjecture. Software developed in connection with this paper has been successfully used to generate and check certificates for MIP models coming from both of these applications.

For linear programming, duality theory tells us that an optimal primal solution and an optimal dual solution are sufficient to facilitate effective verification of optimality. In the case of checking infeasibility, a Farkas certificate will do. Therefore, verifying LP results, at least in the case when exact rational arithmetic is used, is rather straightforward. However, the situation with MIP is drastically different. From a theoretical perspective, even though some notions of duality for MIP have been formulated [24], small (i.e. polynomial size) certificates for infeasibility or optimality may not even exist. As a result, there are many forms that certificates could take: a branch-and-bound tree, a list of derived cutting planes, a superadditive dual function, or other possibilities for problems with special structures such as pure integer linear programming and binary programming [10, 16, 31, 33]. Which format would be preferred for certificate verification is not entirely clear, and in this paper we provide reasoning behind our choice.

From a software perspective, MIP result certification is also considerably more complicated than LP certification. Even though most solvers adopt the branch-and-cut paradigm, they typically do not make the computed branch-and-bound tree or generated cuts readily available, and they may also utilize many other techniques including constraint propagation, conflict analysis, or reduced cost fixing. Thus, even if a solver did print out all information used to derive its solution, a verifier capable of interpreting such information would itself be highly complex, contradicting our desire for a simple verifier. As a result, other than accepting the results of an exact solver such as [13], the best that many people can do today to "verify" the results of a solver on a MIP instance is to solve the instance by several different solvers and check if the results match or minimally check that a returned solution is indeed feasible and has the objective function value claimed, as is done by the solution checker in [32].

The main contribution of this paper is the development of a certificate format for the verification of mixed-integer linear programs. Compared to the previous work of Applegate *et al.* [5] for the Traveling Salesman Problem and the unpublished work of Carr *et al.* [11] for general MIP, our certificate format has a significantly simpler structure. It consists of a sequence of statements that can be verified one by one using simple inference rules, facilitating verification in a manner akin to natural deduction. The approach is similar to that for verification of unsatisfiability proofs for SAT formulas. (See for example [27, 43].) This simple certificate structure makes it easier for researchers to develop their own independent certificate verification programs, or check the code of existing verifiers, even without any expert knowledge of MIP solution algorithms.

To demonstrate the utility of the proposed certificate format, we have developed a reference checker in C++ and added the capability to produce such certificates to the exact version of the MIP solver SCIP [13, 21]. We used these tools to verify results reported in [13]. To the best of our knowledge, this work also represents the first software for general MIP certificate verification that has been made available to the mathematical optimization community.

Organization of the paper. Even though the proposed format for the certificate is straightforward, some of the details are nevertheless technical. Therefore, in this paper, we discuss the certificate format at a conceptual level. The full technical specification is found in the accompanying computer files.[4] We begin with the necessary ingredients for the simple case of LP in Section 2. In Section 3, the ideas for dealing with LP are extended to pure integer linear programming. The full conceptual description of the format of the certificate is then given in Section 4. Computational experiments are reported in Section 5, and concluding remarks are given in Section 6. Throughout this paper, we assume that problems are specified and solved with exact rational arithmetic.

2 Certificates for linear programming

A certificate of optimality for an LP is a dual feasible solution whose objective function value matches the optimal value. However, there is no need to specify the dual when one views the task of certification as an inference procedure, see, e.g., [28]. Suppose we are given the system of linear constraints

$$Ax \geq b, A'x \leq b', A''x = b'', \tag{S}$$

where x is a vector of variables, $A \in \mathbb{R}^{m \times n}$, $A' \in \mathbb{R}^{m' \times n}$, $A'' \in \mathbb{R}^{m'' \times n}$, $b \in \mathbb{R}^m$, $b' \in \mathbb{R}^{m'}$, and $b'' \in \mathbb{R}^{m''}$ for some nonnegative integers n, m, m', and m''.

[4] See `https://github.com/ambros-gleixner/VIPR`

We say that $c^\mathsf{T} x \geq v$ is obtained by taking a *suitable linear combination* of the constraints in (S) if

$$c^\mathsf{T} = d^\mathsf{T} A + {d'}^\mathsf{T} A' + {d''}^\mathsf{T} A'',\ v = d^\mathsf{T} b + {d'}^\mathsf{T} b' + {d''}^\mathsf{T} b''$$

for some $d \in \mathbb{R}^m$, $d' \in \mathbb{R}^{m'}$, and $d'' \in \mathbb{R}^{m''}$ with $d \geq 0$ and $d' \leq 0$. If x satisfies (S), then it necessarily satisfies $c^\mathsf{T} x \geq v$. We say that the inequality $c^\mathsf{T} x \geq v$ is *inferred* from (S). We will refer to this general inference procedure as *linear inequality inference.*

Remark 1. Together, d, d', d'' is simply a feasible solution to the linear programming dual of the linear program

$$\min\{c^\mathsf{T} x \mid Ax \geq b, A'x \leq b', A''x = b''\}. \tag{LP}$$

The inequality $c^\mathsf{T} x \geq v$ is sometimes called a *surrogate* of (S). (See [28].)

Suppose that an optimal solution to (LP) exists and the optimal value is v. Linear programming duality theory guarantees that $c^\mathsf{T} x \geq v$ can be inferred from (S). Therefore, linear inequality inference is sufficient to certify optimality for linear programming. Conceptually, the certificate that we propose is a listing of the constraints in (S) followed by the inequality $c^\mathsf{T} x \geq v$ with the associated multipliers used in the inference as illustrated in the following example.

Example 2. The following shows an LP problem and its associated certificate.

$$\begin{array}{l} \min 2x + y \\ \text{ s.t.} \\ C1 : 5x - y \geq 2 \\ C2 : 3x - 2y \leq 1. \end{array}$$

Given	
$C1 : 5x - y \geq 2$	
$C2 : 3x - 2y \leq 1$	
Derived	**Reason**
`obj` $: 2x + y \geq 1$	$\{1 \times C1 + (-1) \times C2\}$

Here, $C1$ and $C2$ are constraint labels. Taking the suitable linear combination $1 \times C1 + (-1) \times C2$ gives $2x + y \geq 1$, thus establishing that 1 is a lower bound for the optimal value.

Remark 3. This type of linear inference can also be used to derive $\leq$-inequalities or equality constraints. Assuming that all problem data is rational, rational multipliers are sufficient to certify infeasibility or optimality.

3 Handling Chvátal-Gomory cutting planes

Gomory [23] showed in theory that, for pure integer linear programming (IP), optimality or infeasibility can be established by a pure cutting-plane approach. Such an approach can also work in practice [8, 44]. In addition to linear inequality inference, a rounding operation is needed.

Suppose that $c^\mathsf{T} x \geq v$ can be inferred from (S) by taking a suitable linear combination of the constraints. If $c_i \in \mathbb{Z}$ for $i \in I$ for some $I \subseteq \{1, \ldots, n\}$ and

$c_i = 0$ for $i \notin I$, then any $x \in \mathbb{R}^n$ satisfying (S) with $x_i \in \mathbb{Z}$ for $i \in I$ must also satisfy $c^{\mathsf{T}}x \geq \lceil v \rceil$. We say that $c^{\mathsf{T}}x \geq \lceil v \rceil$ is obtained from $c^{\mathsf{T}}x \geq v$ by *rounding.* When $I = \{1, \ldots, n\}$, the inequality $c^{\mathsf{T}}x \geq \lceil v \rceil$ is known as a *Chvátal-Gomory cut* (CG-cut in short). It can then be added to the system and the process of obtaining another CG-cut can be repeated. Conceptually, a certificate for an IP instance solved using only CG-cuts can be given as a list of the original constraints followed by the derived constraints.

Example 4. The following shows an IP problem and its associated certificate.

$$\begin{aligned} &\min x + y \\ &\text{s.t.} \\ &C1 : 4x + y \geq 1 \\ &C2 : 4x - y \leq 2 \\ &x, y \in \mathbb{Z} \end{aligned}$$

Given	
$x, y \in \mathbb{Z}$	
$C1 : 4x + y \geq 1$	
$C2 : 4x - y \leq 2$	
Derived	**Reason**
$C3 : y \geq -\frac{1}{2}$	$\left\{\frac{1}{2} \times C1 + \left(-\frac{1}{2}\right) \times C2\right\}$
$C4 : y \geq 0$	{round up $C3$}
$C5 : x + y \geq \frac{1}{4}$	$\left\{\frac{1}{4} \times C1 + \frac{3}{4} \times C4\right\}$
$C6 : x + y \geq 1$	{round up $C5$}

Note that the derived constraints in the certificate can be processed in a sequential manner. In the next section, we see how to deal with branching without sacrificing sequential processing.

4 Branch-and-cut certificates

In practice, most MIP instances are not solved by cutting planes alone. Thus, certificates as described in the previous section are of limited utility. We now propose a type of certificate for optimality or infeasibility established by a branch-and-cut procedure in which the generated cuts at any node can be derived as split cuts and branching is performed on a disjunction of the form $a^{\mathsf{T}}x \leq \delta \vee a^{\mathsf{T}}x \geq \delta + 1$ where $\delta \in \mathbb{Z}$ and $a^{\mathsf{T}}x$ is integral for all feasible x.

The use of split disjunctions allows us to consider branching and cutting under one umbrella. Many of the well-known cuts generated by MIP solvers can be derived as split cuts [14] and they are effective in closing the integrality gap in practice [18]. Branching typically uses only simple split disjunctions (where the a above is a unit vector), although some studies have considered the computational performance of branching on general disjunctions [15, 20, 30, 38].

Recall that each branching splits the solution space into two subcases. At the end of a branch-and-bound (or branch-and-cut) procedure, each leaf of the branch-and-bound tree corresponds to one of the cases and the leaves together cover all the cases that need to be considered. Hence, if the branch-and-bound tree is valid, all one needs to look at are the LP results at the leaves.

Our proposal is to "flatten" the branch-and-bound tree into a list of statements that can be verified sequentially. Thus, our approach departs from the approaches in [5] and [11], which require explicit handling of the tree structure. The price we pay is that we can no longer simply examine the leaves of the tree.

Instead, we process the nodes in a bottom-up fashion and discharge assumptions as we move up towards the root. We illustrate the ideas with an example.

Example 5. It is known that the following has no solution.

$$\begin{aligned} C1 :&\ 2x_1 + 3x_2 \geq 1 \\ C2 :&\ 3x_1 - 4x_2 \leq 2 \\ C3 :&\ -x_1 + 6x_2 \leq 3 \\ &\ x_1, x_2 \in \mathbb{Z} \end{aligned}$$

Note that $(x_1, x_2) = (\frac{10}{17}, -\frac{1}{17})$ is an extreme point of the region defined by $C1$, $C2$, and $C3$. Branching on the integer variable x_1 leads to two cases:

- **Case 1.** $A1 :\ x_1 \leq 0$
  Note that $(x_1, x_2) = (0, \frac{1}{3})$ satisfies $C1, C2, C3, A1$. We branch on x_2:
  Case 1a. $A3 :\ x_2 \leq 0$
  Taking $C1 + (-2) \times A1 + (-3) \times A3$ gives the absurdity $C4 :\ 0 \geq 1$.
  Case 1b. $A4 :\ x_2 \geq 1$
  Taking $\left(-\frac{1}{3}\right) \times C3 + \left(-\frac{1}{3}\right) \times A1 + 2 \times A4$ gives the absurdity $C5 :\ 0 \geq 1$.
- **Case 2.** $A2 :\ x_1 \geq 1$
  Taking $\left(-\frac{1}{4}\right) \times C2 + \left(\frac{3}{4}\right) \times A2$ gives $C6 :\ x_2 \geq \frac{1}{4}$. Rounding gives $C7 :\ x_2 \geq 1$.
  Taking $\left(-\frac{1}{3}\right) \times C2 + (-1) \times C3 + \frac{14}{3} \times C7$ gives the absurdity $C8 :\ 0 \geq 1$.

As all cases lead to $0 \geq 1$, we conclude that there is no solution. To issue a certificate as a list of derived constraints, we need a way to specify the different cases. To this end, we allow the introduction of constraints as assumptions.

Figure 1 shows a conceptual certificate for the instance. Notice how the constraints $A1$, $A2$, $A3$, and $A4$ are introduced to the certificate as assumptions. Since we want to end with $0 \geq 1$ without additional assumptions attached, we get there by gradually undoing the case-splitting operations. We call the undoing operation *unsplitting*. For example, $C4$ and $C5$ are both the absurdity $0 \geq 1$ with a common assumption $A1$. Since $A3 \vee A4$ is true for all feasible x, we can infer the absurdity $C9 :\ 0 \geq 1$ assuming only $A1$ in addition to the original constraints. We say that $C9$ is obtained by *unsplitting* $C4, C5$ on $A3, A4$. Similarly, both $C8$ and $C9$ are the absurdity $0 \geq 1$ and $A2 \vee A1$ is true for all feasible x, we can therefore unsplit on $C8, C9$ on $A2, A1$ to obtain $C10 :\ 0 \geq 1$ without any assumption in addition to the original constraints.

In practice, the list of assumptions associated with each derived constraint needs not be specified explicitly in the certificate, but can be deduced on the fly by a checker. For example, when processing $C4$, we see that it uses $A1$ and $A3$, both of which are assumptions. Hence, we associate $C4$ with the list of assumptions $A1, A3$. As any linear inequality can be introduced as an assumption, branching can be performed on general disjunctions.

Remark 6. Our proposed certificate can also be used to represent split cuts. Split cuts are inequalities that are valid for the defining inequalities taken together

Given		
$x, y \in \mathbb{Z}$		
$C1: 2x_1 + 3x_2 \geq 1$		
$C2: 3x_1 - 4x_2 \leq 2$		
$C3: -x_1 + 6x_2 \leq 3$		
Derived	**Reason**	**Assumptions**
$A1: x_1 \leq 0$	$\{\text{assume}\}$	
$A2: x_1 \geq 1$	$\{\text{assume}\}$	
$A3: x_2 \leq 0$	$\{\text{assume}\}$	
$C4: 0 \geq 1$	$\{C1 + (-2) \times A1 + (-3) \times A3\}$	$A1, A3$
$A4: x_2 \geq 1$	$\{\text{assume}\}$	
$C5: 0 \geq 1$	$\left\{\left(-\frac{1}{3}\right) \times C3 + \left(-\frac{1}{3}\right) \times A1 + 2 \times A4\right\}$	$A1, A4$
$C6: x_2 \geq \frac{1}{4}$	$\left\{\left(-\frac{1}{4}\right) \times C2 + \left(\frac{3}{4}\right) \times A2\right\}$	$A2$
$C7: x_2 \geq 1$	$\{\text{round up } C6\}$	$A2$
$C8: 0 \geq 1$	$\left\{\left(-\frac{1}{3}\right) \times C2 + (-1) \times C3 + \frac{14}{3} \times C7\right\}$	$A2$
$C9: 0 \geq 1$	$\{\text{unsplit } C4, C5 \text{ on } A3, A4\}$	$A1$
$C10: 0 \geq 1$	$\{\text{unsplit } C8, C9 \text{ on } A2, A1\}$	

Fig. 1. Certificate for Example 5

with each one of the inequalities in a split disjunction, $a^\mathsf{T} x \leq \delta \vee a^\mathsf{T} x \geq \delta + 1$, where δ is an integer and a is an integer vector that is nonzero only in components corresponding to integer variables. To derive a proof of a split cut's validity, the inequalities in the split disjunction can each be introduced as assumptions, the cut can be derived for each side of the split disjunction using linear inequality inference, and then unsplitting can be applied to discharge the assumptions.

5 Computational experiments

In this section, we describe software developed to produce and check certificates for MIP results using the certificate format developed in this paper. It is freely available for download, along with a precise technical specification of the file format.[5] One of its features is that after each derived constraint an integer is printed to specify the largest index of any derived constraint that references it. This allows constraints to be freed from memory when they will no longer be needed. The following C++ programs are provided:

– `viprchk` verifies MIP results provided in our specified file format. All computations are performed in exact rational arithmetic using the GMP library [22].
– `viprttn` performs simple modifications to "tighten" certificates. It removes unnecessary derived constraints to reduce the file size. In order to decrease peak memory usage during checking, it reorders the remaining ones using a depth-first topological sort and for each derived constraint that remains, it computes the largest index over constraints that references it.

[5] See `https://github.com/ambros-gleixner/VIPR`

– `vipr2html` converts a certificate file to a "human-readable" HTML file.

We again emphasize that the format was designed with simplicity in mind; the certificate verification program we have provided is merely a reference and others should be able to write their own verifiers without much difficulty.

In addition, we created a modified version of the exact rational MIP solver described in [13] and used it to compute certificates for several MIP instances from the literature. The exact rational MIP solver is based on SCIP [21] and uses a hybrid of floating-point and exact rational arithmetic to efficiently compute exact solutions using a pure branch-and-bound algorithm. In our experiments, the rational MIP solver uses CPLEX 12.6.0.0 [29] as its underlying floating-point LP solver and a modified version of QSopt_ex 2.5.10 [7] as its underlying exact LP solver. The exact MIP solver supports several methods for computing valid dual bounds and our certificate printing functionality is currently supported by the *Project-and-shift* method (for dual solutions only) and the *Exact LP* method (for both dual solutions and Farkas proofs), for details on these methods see [13]. This developmental version is currently available from the authors by request. We note that the certificate is printed concurrently with the solution process which leads to certificates that have potential for reduction and simplification by `viprttn`, or other routines. For example, as each node is processed its derived dual bound is printed to the certificate even though it may become redundant if branching is performed and new dual bounds are computed at the child nodes; also, discovery of a new primal solution might allow pruning of a large subtree, rendering many bound derivations redundant.

The program `viprttn` processes the list of derived constraints in two passes. In the first pass, it builds the dependency graph with nodes representing the derived constraints and arcs uv such that the derived constraint represented by u is referenced by the reason for deriving the constraint represented by v. In the second pass, it performs a topological sort using depth-first search on the component that contains the final constraint and writes out the reordered list of derived constraints with updated constraint indices.

In the following, we report some computational results on the time and memory required to produce and verify certificates. We considered the *easy* and *numerically difficult* (referred to here as '*hard*') test sets from [13]; these test sets consist of instances from well known libraries including [2, 9, 32, 34, 35]. Experiments were conducted on a cluster of Intel(R) Xeon(R) CPU E5-2660 v3 at 2.60 GHz; jobs were run exclusively to ensure accurate time measurement. Table 1 reports a number of aggregate statistics on these experiments. The columns under the heading SCIP report results from tests using the exact version of SCIP, using its default dual bounding strategy. The columns under SCIP+C report on tests involving the version of exact SCIP that generates certificates as it solves instances; it uses only the dual bounding methods *Project-and-shift* and *Exact LP* that support certificate printing, contributing to its slower speed. Columns under the heading VIPR report time and memory usage for certificate checking.

For each of the easy and hard test sets, we report information aggregated into four categories: 'all' reports statistics over all instances; 'solved' reports over instances solved by both SCIP and SCIP+C within a 1 hour time limit and a 10 GB limit on certificate file size; 'memout' reports on instances where SCIP+C stopped because the certificate file size limit was reached; and 'timeout' reports on the remaining instances, where one of the solvers hit the time limit. All averages are reported as shifted geometric means with a shift of 10 sec. for time and 1 MB for memory. The column N represents the number of instances in each category; N_{sol} represents the number in each category that were solved to optimality (or infeasibility) by a given solver; t_{MIP} represents the time (sec.) used to solve the instance and, when applicable, output a certificate; t_{ttn} is the time (sec.) required by the `viprttn` routine to tighten the certificate file; t_{chk} is the time (sec.) required to for `viprchk` to check the certificate file – on instances in the memout and timeout rows this represents the time to verify the primal and dual bounds present in the intermediate certificate printed before the solver was halted. The final three columns list the size of the certificate (in MB), before tightening, after tightening and then after being compressed to a gzipped file. Timings and memory usage for individual instances are available in a document hosted together with the accompanying software.

Table 1. Aggregated computational results over 107 instances from [13].

		SCIP		SCIP+C		VIPR (viprttn)				
Test set	N	N_{sol}	t_{MIP}	N_{sol}	t_{MIP}	t_{ttn}	t_{chk}	size$_{\text{raw}}$	size$_{\text{ttn}}$	size$_{\text{gz}}$
easy-all	57	54	63.3	39	190.9	8.9	27.2	227	77	24
-solved	39	39	23.2	39	48.0	3.6	11.5	77	34	10
-memout	5	4	600.6	0	1760.4	47.8	138.3	10286	513	157
-timeout	13	11	338.3	0	3600.0	23.3	102.7	1309	434	129
hard-all	50	23	725.2	14	975.6	7.9	12.1	373	38	11
-solved	13	13	22.9	13	40.7	2.2	5.3	49	15	5
-memout	12	2	2476.4	0	1713.1	32.7	59.8	10266	235	67
-timeout	25	8	2052.1	1	3518.1	4.3	5.3	216	25	7

From this table, we can make a number of observations. First, there is a noticeable, but not prohibitive, cost to generate the certificates. The differences in t_{MIP} between SCIP and SCIP+C are due to both the difference in dual bounding strategies, and the overhead for writing the certificate files. In some additional experiments, we observed that on the 39 instances in the easy-solved category, the file I/O amounted to roughly 7% of the solution time, based on this we believe that future modifications to the code will allow us to solve and print certificates in times much closer to those in the SCIP column. Perhaps most importantly, we observe that the time to check the certificates is significantly less than the time to solve the instances.

Moreover, the certificate tightening program `viprttn` is able to make significant reductions in the certificate size, and the resulting certificate sizes are often surprisingly manageable. Most striking is the tightening in the memout categories, which significantly exceed the approximately 50% reduction that could be expected by removing the redundant linear inferences derived for internal nodes of the branch-and-bound tree. The most extreme tightening was achieved for the instance `markshare1_1` in 'easy-memout', from 10 GB to 8 kB. This is explained by the fact that the root dual bound is already zero and the tree search is only performed for finding the optimal solution. Hence, the certificate is highly redundant and the derived constraints for all but the root node can be removed.

The average reductions in the other categories are smaller, but also strictly above 50%. This shows that `viprttn` performs more than just a removal of internal nodes. These results also show two aspects in which SCIP's certificate printing can be improved: by avoiding printing dual bound derivations for internal nodes using a buffering scheme, and by not generating dual bound derivations for nodes that do not improve upon the bound of the parent node.

6 Conclusion

This paper presented a certificate format for verifying integer programming results. We have demonstrated the practical feasibility of generating and checking such certificates on well-known MIP instances. We see this as the first step of many in verifying the results of integer programming solvers. We now discuss some future directions made possible by this work.

Even in the context of floating-point arithmetic, our certificate format could serve a number of purposes. Using methods described by [12, 37], directed rounding and interval arithmetic may allow us to compute and represent valid certificates exclusively using floating-point data, allowing for faster computation and smaller certificate size. Additionally, generating approximate certificates with inexact data could be used for debugging solvers, or measuring the maximum or average numerical violation over all derivations. In a more rigorous direction, one could also convert our certificates to a form that could be formally verified by a proof assistant such as HOL Light [25].

Acknowledgements. We thank Kati Wolter for the exact version of SCIP [13] and Daniel Espinoza for QSopt_ex [7], which provided the basis for our experiments, and Gregor Hendel for his Ipet package [26], which was a big help in analyzing the experimental results. This work has been supported by the Research Campus MODAL *Mathematical Optimization and Data Analysis Laboratories* funded by the Federal Ministry of Education and Research (BMBF Grant 05M14ZAM). All responsibility for the content of this publication is assumed by the authors.

References

1. Achterberg, T.: Constraint Integer Programming. PhD Thesis, TU Berlin (2007)
2. Achterberg, T., Koch, T., Martin, A.: The mixed integer programming library: MIPLIB 2003. *Oper. Res. Lett.* **34(4)**, 361–372 (2006)

3. Achterberg, T. and Wunderling, R.: Mixed integer programming: Analyzing 12 years of progress. In Jünger, M. and Reinelt, G., editors, *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, 449–481 (2013)
4. Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R.: MIP: Theory and Practice – Closing the Gap. In Powell, M.J.D. and Scholtes, S., *System Modelling and Optimization*, 19–49 (2000)
5. Applegate, D.L., Bixby, R.E., Chvátal, V. Cook, W., Espinoza, D.G., Goycoolea, M., Helsgaun, K. Certification of an optimal TSP tour through 85,900 cities. *Oper. Res. Lett.* **37**, 11–15 (2009)
6. Applegate, D.L., Cook, W.J., Dash, S., Espinoza, D.G.: Exact solutions to linear programming problems. *Oper. Res. Lett.* **35**(6), 693-699 (2007)
7. Applegate, D.L., Cook, W.J., Dash, S., Espinoza, D.G.: QSopt_ex: `http://www.math.uwaterloo.ca/~bico/qsopt/ex/` Last accessed on November 13, 2016
8. Balas, E., Fischetti, M., Zanette, A.: A hard integer program made easy by lexicography. *Math. Program., Ser. A* **135**, 509–514 (2012)
9. Bixby, R.E., Ceria, S., McZeal, C.M., Savelsbergh, M.W.: An updated mixed integer programming library: MIPLIB 3.0. *Optima* **58**, 12–15 (1998)
10. Boland, N.L. and Eberhard, A.C. On the augmented Lagrangian dual for integer programming. *Math. Program., Ser. A* **150**(2), 491–509.(2015)
11. Carr, R., Greenberg, H., Parekh, O., Phillips, C.: Towards certificates for integer programming computations. *Presentation, 2011 DOE Applied Mathematics PI meeting*, October 2011. Slides `www.csm.ornl.gov/workshops/applmath11/documents/talks/Phillips_talk.pdf` Last accessed on November 13, 2016
12. Cook, W., Dash, S., Fukasawa, R., and Goycoolea, M.: Numerically safe Gomory mixed-integer cuts. *INFORMS J. Comput.*, **21**(4), 641-649 (2009)
13. Cook, W., Koch, T., Steffy, D., and Wolter, K.: A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Math. Program. Comput.*, **3**, 305–344 (2013)
14. Cornuéjols, G.: Valid inequalities for mixed integer linear programs. *Math. Program. Ser. B*, **112**, 3–44 (2008)
15. Cornuéjols, G., Liberti, L., Nannicini, G.: Improved strategies for branching on general disjunctions. *Math. Program. Ser. A*, **130**, 225–247 (2011)
16. De Loera, J.A., Lee, J., Malkin, P.N., Margulies, S.: Computing infeasibility certificates for combinatorial problems through Hilberts Nullstellensatz. *J. Symbolic Comp.*, **46**(11), 1260–1283 (2011)
17. Dhiflaoui, M., Funke, S., Kwappik, C., Mehlhorn, K., Seel, M., Schomer, E., Schulte, R., Weber, D.: Certifying and repairing solutions to large LPs, how good are LP-solvers? *In: SODA 2003,* 255-256. ACM/SIAM, New York (2003)
18. Fukasawa, R. and Goycoolea, M.: On the exact separation of mixed integer knapsack cuts. *Math. Program. Ser. A*, **128**, 19–41 (2008)
19. The Flyspeck Project. `https://code.google.com/archive/p/flyspeck/`. Last accessed on November 13, 2016.
20. Gamrath, G., Melchiori, A., Berthold, T., Gleixner, A.M., Salvagnin, D.: Branching on multi-aggregated variables. In *Integration of AI and OR Techniques in Constraint Programming*, volume 9075, 141–156 (2015)
21. Gamrath, G., et al.: The SCIP Optimization Suite 3.2. ZIB-Report (15-60) (2016)
22. GNU MP: The GNU Multiple Precision Arithmetic Library version 6.1.1. `http://gmplib.org`. Last accessed on November 16, 2016.
23. Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.* **64**, 275–278 (1958)

24. Guzelsoy, M. and Ralphs, T.K.: Duality for mixed-integer linear programs. *Int. J. Oper. Res.* **4**(3), 118–137 (2007)
25. Harrison, J.: HOL Light: A tutorial introduction. *Inter. Conf. on Formal Methods in Comp.-Aided Design.*, 265–269 (1996)
26. Hendel, G.: Empirical analysis of solving phases in mixed integer programming. Master's thesis, Technische Universität Berlin (2014). `http://nbn-resolving.de/urn:nbn:de:0297-zib-54270`
27. Heule, M.J.H., Hunt, Jr., W.A., Wetzler, N.: Verifying refutations with extended resolution. In *Conference on Automated Deductio (CADE), LNAI*, **7898**:345–359 (2013)
28. Hooker, J.N.: *Integrated Methods for Optimization (2nd ed.)*, Springer, New York (2012)
29. IBM ILOG. CPLEX. `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`. Last accessed on November 16, 2016.
30. Karamanov, M. and Cornuéjols, G.: Branching on general disjunctions. *Math. Program. Ser. A*, **128**, 403–436 (2011)
31. Klabjan, D.: Subadditive approaches in integer programming. *Eur. J. Oper. Res*, **183**, 525–545 (2007)
32. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D.E., Wolter, K.: MIPLIB 2010 *Math. Program. Comp.*, **3(2)**, 103–163 (2011)
33. Lasserre, J.B.: Generating functions and duality for integer programs. *Discrete Optim.* **1**(2), 167–187 (2004)
34. Lehigh University COR@L mixed integer programming collection. `http://coral.ie.lehigh.edu/wiki/doku.php/info:datasets:mip`. Last accessed on November 18, 2016.
35. Mittelmann, H.D.: Benchmarks for Optimization Software. `http://plato.asu.edu/bench.html` Last accessed on November 18, 2016.
36. Narkawicz, A. and Muñoz, C. A Formally Verified Generic Branching Algorithm for Global Optimization *Verified Software: Theories, Tools, Experiments (VSTTE), LNCS* **8164**, 326–343 (2014)
37. Neumaier, A., Shcherbina, O.: Safe bounds in linear and mixed-integer linear programming. *Math. Program.* **99**(2), 283–296 (2004)
38. Owen, J.H. and Mehrotra, S.: Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. *Comput. Optim. Appl.* **20**, 159–170 (2001)
39. Obua, S. and Nipkow, T.: Flyspeck II: the basic linear programs. *Ann. Math. Artif. Intell* **56**, 245–272 (2009)
40. Pulaj, J.: Cutting Planes for Families Implying Frankl's Conjecture. ZIB-Report (15-51), (2016)
41. Smith, A.P., Muñoz, C.A., Narkawicz, A.J. and Markevicius, M.: Kodiak: An Implementation Framework for Branch and Bound Algorithms *Technical report: NASA/TM-2015-218776*, (2015)
42. Solovyev, A. and Hales, T.: Efficient formal verification of bounds of linear programs. *Lecture Notes in Comp. Sci.* **6824**, 123–132 (2011)
43. Wetzler, N., Heule, M.J.H., Hunt, Jr., W.A.: DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing (SAT), LNCS* **8561**, 422–429 (2014)
44. Zanette, A., Fischetti, M., Balas, E.: Lexicography and degeneracy: can a pure cutting plane algorithm work? *Math. Program., Ser. A* **130**, 153–176 (2011)