MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library

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 · Yuji Shinano*

July 12, 2019

Abstract We report on the selection process leading to the sixth version of the Mixed Integer Programming Library. Selected from an initial pool of over 5,000 instances, the new MIPLIB 2017 collection consists of 1,065 instances. A subset of 240 instances was specially selected for benchmarking solver performance. For the first time, the compilation of these sets was done using a data-driven selection process supported by the solution of a sequence of mixed integer optimization problems, which encoded requirements on diversity and balancedness with respect to instance features and performance data.

Keywords Mixed integer linear optimization · MIP · instance library · benchmarking · selection methodology

Mathematics Subject Classification 90C06 · 90C09 · 90C10 · 90C11

1 Introduction

Computational mixed integer (linear) optimization is an important sub-field of mathematical optimization. Hundreds of papers on the subject are published each year and a multitude of companies provide tools for modeling and solving mixed integer optimization problems (MIPs) based on the state of the art in research. Measuring performance on benchmark test instances has lain at the heart of computational research since the early days of mathematical optimization. Hoffman et al. [23] first reported on a computational experiment comparing implementations of three algorithms for linear optimization back in 1953. Their observation that "*[many] conjectures about the relative merits of the three methods by various criteria could only be verified by actual trial*" seems to hold even more for MIP algorithms today. The variety and complex interaction of different techniques regularly calls for empirical evaluation and motivates the collection and curation of relevant test instances.

Brought into existence in 1992 by Bixby, Boyd, and Indovina [7], the goal of the MIPLIB project has been to provide the research community with a curated set of challenging real-world instances from academic and industrial applications that are suitable

*Extended author information is available at the end of the paper.

for testing new algorithms and quantifying performance. It has been updated four times [1, 8, 3, 25] in order to reflect the increasing diversity and complexity of MIP models used in practice and the improving performance of available MIP solvers. In this article, we describe its sixth version, MIPLIB 2017, together with a new selection methodology developed during this process.

The exceptional algorithmic progress in solving real-world MIP instances over the last decades is recorded in various articles [6, 26, 27, 25]. It can also be observed on the previous version, MIPLIB 2010, both in terms of solvability and speed. By the end of 2018, the number of unsolved instances was reduced by nearly half. Of the 134 instances for which no solution with provable optimality guarantee was initially known, only 70 instances remain open. Comparable progress in the overall speed of solvers can be observed in the results of benchmark testing with different versions of available solvers. Since its release in April 2011, the subset of instances of MIPLIB 2010 that form the so-called "benchmark set", consisting of 87 problem instances, has been the accepted standard for evaluating solvers. Using this benchmark set, Hans Mittelmann has been evaluating a number of MIP solvers, including CPLEX [10], Gurobi [20], and Xpress [38]. When MIPLIB 2010 was released, the version numbers of these three commercial solvers were CPLEX 12.2.0.2, Gurobi 4.5.1, and Xpress 7.2. Aggregating the benchmark results of these three solvers at that time, we can construct results corresponding to a so-called "virtual best" solver and a so-called "virtual worst" solver. These are hypothetical solvers that, for each instance, produce run times that are equal to the best and the worst of the three, respectively. Doing this analysis yields shifted geometric mean runtimes of 36.3 and 113.0 seconds for the virtual best and virtual worst solver, respectively.[1] In December 2018, the solver versions were CPLEX 12.8.0, Gurobi 8.1.0, and Xpress 8.5.1. On the same hardware (with a newer operating system) the shifted geometric means of the runtimes had decreased to 13.5 seconds for the virtual best, and 31.3 seconds for the virtual worst solver. This corresponds to speed-up factor of 2.70 and 3.62, respectively, which amounts to roughly 16 % per year, just from improvements in the algorithms.

It was because of this development that the MIPLIB 2010 benchmark set was no longer considered to sufficiently reflect the frontier of new challenges in the field and the process of contructing a new MIPLIB was begun. In November 2016, a public call for contributions was launched and a group of 21 interested researchers, including representatives of the development teams of nine MIP solvers formed a committee in order to steer the process of compiling an updated library.[2] As with MIPLIB 2010, the overall goal was the compilation of two sets of instances. The MIPLIB 2017 *benchmark set* was to be suitable, to the extent possible, for performing a meaningful and fair comparison of the average performance of MIP solvers (and different versions of the same solver) across a wide range of instances with different properties, in a reasonable amount of computing time. The larger MIPLIB 2017 *collection* was to provide additional instances for a broader coverage without restrictions on the total runtime of the test set, including unsolved instances (as a challenge for future research) and instances not suitable for benchmarking due to problematic numerical properties, special constraint types (such as indicators), or exceptionally large memory requirements.

It should be emphasized that the benchmark set we have scientifically contructed is designed for the purpose of comparing the overall performance of general purpose solvers on a wide-ranging set of instances. Average performance on this set is not a suitable criterion to decide which MIP solver to use in a particular application scenario. For such

[1] The computations used 12 parallel threads. The corresponding log files can be found at [30]. The means were computed with a shift of 1 second (see Achterberg [2]).

[2] The members of the MIPLIB 2017 committee were Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Mary Felenon, Koichi Fujii, Gerald Gamrath, Ambros Gleixner, Gregor Hendel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans Mittelmann, Derya Ozyurt, Imre Pólik, Ted Ralphs, Domenico Salvagnin, Yuji Shinano, Franz Wesselmann, and Michael Winkler.

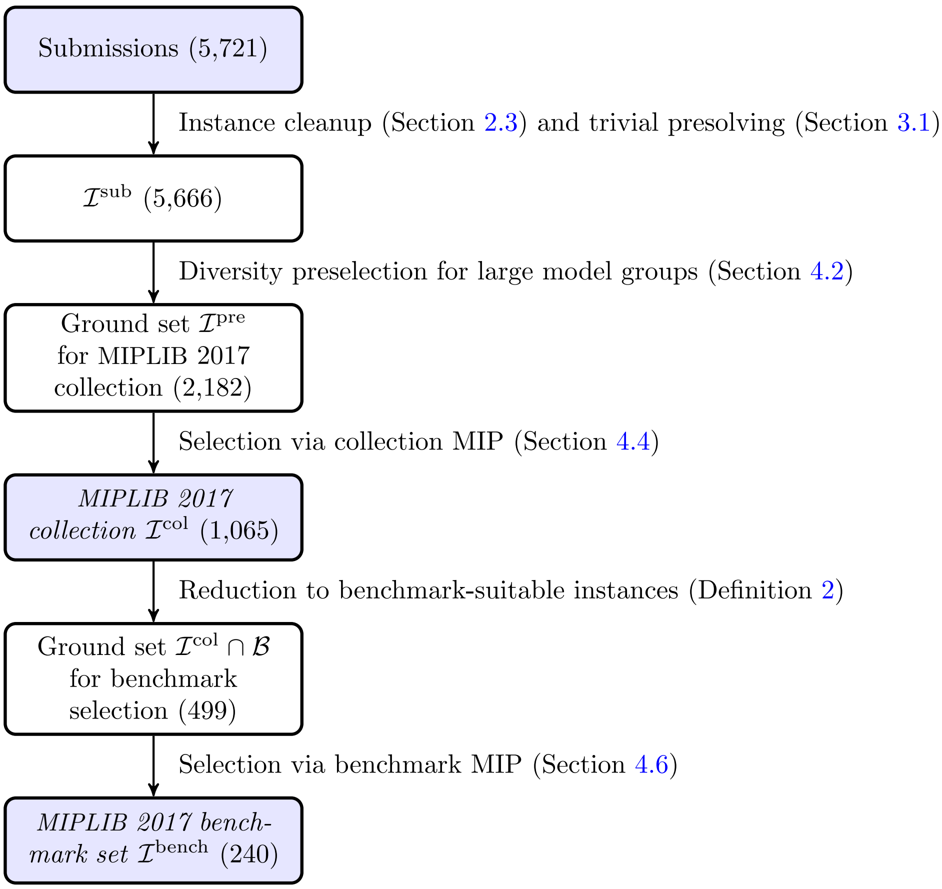


Figure 1: Outline of the steps involved in the selection of the MIPLIB 2017 collection and benchmark set. Number of instances remaining are given in parentheses.

decisions, it is important to consider what specific class(es) of instances are relevant, as well as what criteria beyond the raw speed and the ability to solve a wide range of problems are of interest. This is also underlined by the fact that each of the eight solvers that were used to collect performance data (see Section 3.6) proved to be the fastest solver on at least one instance.

Compiling a representative and meaningful instance library is a nontrivial endeavor. Compared to previous editions of MIPLIB, the increased number of submissions, the goals of compiling a significantly larger collection of instances and including a larger number of representatives of solvers posed new challenges to the selection process. In addition, MIPLIB 2017 is the first edition to provide supplementary data regarding the instances, such as the matrix structure and decomposability, as well as the underlying models from which the instances originated, where available. In order to produce a well-balanced library in a fair and transparent manner, we designed a new, heavily data-driven process. The steps applied between the initial submissions and the final MIPLIB 2017 are outlined in Figure 1. Driven by a diverse set of instance features, our methodology used multiple clusterings to populate a MIP model that was then solved to generate suitable candidates for the final library to be presented to the MIPLIB committee.

We consider this process of selecting instances from a large pool of submissions to be the main new feature of MIPLIB 2017. By contrast, the instances constituting previous versions of MIPLIB were manually selected by the members of the committee, depending heavily on their expertise in benchmarking to avoid common pitfalls like overrepresentation of certain problem classes. As one byproduct of this data-driven approach, we are now able to identify similar instances, which leads to sometimes surprising insights into connections between different, seemingly unrelated instances in the library. In addition to the raw feature data, we provide, for each instance, the five most similar instances in the collection on a newly designed web page (see Section 5.3).

The remainder of this article contains a detailed desciption of the selection process and its result. It is organized as follows. Section 2 describes the efforts to collect instances and meta data on their background and solvability. Section 3 details the collection of feature data for each instance that forms the basis for quantifying similarity of instances and balancedness of a given selection. In Section 4, we describe how this data was used as input in order to compute good candidates for the library by solving a MIP model. Section 5 summarizes the final result. We conclude with our final remarks and acknowledgements in Section 6.

2 The Collection Process

The first step in the creation of a new MIPLIB version is to collect a large set of candidate instances. This section elaborates on the submission process and the origin of the submitted instances, as well as preparatory steps for the subsequent selection process like instance cleanup and grouping.

2.1 The Submission Phase

The collection process started with a call for contributions in November 2016, which was advertised at many conferences, on numerous mailing lists, and in direct communication with partners from industry. Submissions were accepted until August 1, 2017. Overall, we received 128 data sets from 66 submitters with 54 different affiliations, 38 of them being academic and 16 industrial. Note that the affiliation of the submitter alone does not imply anything about the source or type of the model. Several of the submitters with academic affiliation submitted instances from industry projects and one submitter from industry submitted instances modeling a combinatorial game. Each submission was automatically committed as a new subdirectory to the public git repository `https://git.zib.de/miplib2017/submissions`. These subdirectories are structured as follows. In the root of the subdirectory there are two files: a bibtex file with references to papers related to the instances, if provided by the submitter, and a meta file containing information about the submitter, creator, and owner of the submitted instances, as well as a description of the instances and licensing information. For the first time, all contributions were required to be submitted under a license in order to explicitly grant the right of redistribution, among others. The default license was chosen to be the Creative Commons CC BY-SA 4.0[3] license, but it was possible to specify a different license if desired. In addition to these two files, there are up to three subdirectories. The `instances` subdirectory contains the instances themselves (.lp or .mps format, possibly compressed). The `models` subdirectory contains associated model files, which contributors were encouraged to include in order to provide researchers with richer information on the structure of the instances. Finally, additional information is provided in the `misc` subdirectory, ranging from extended descriptions to log files and MIP start files.

In total, 3,670 instances were newly submitted to MIPLIB 2017. Table 1 lists all submitters and the number of instances they submitted. We arrived at a total of 5,721 instances by adding 2,051 instances that were submitted for inclusion in MIPLIB 2010,

[3]`https://creativecommons.org/licenses/by-sa/4.0/`

Table 1: Submitters of new instances, and number of instances from their submission(s) after cleanup.

Submitter	#	Submitter	#
Andrea Arias	360	Yoshihiro Kanno	6
Dimitri Papageorgiou	300	Andrew Stamps	5
Pierre Le Bodic	121	Alexandra M Newman	4
Qie He	120	Austin Buchanan	4
Simon Felix	101	Rob Pratt	4
Gleb Belov (except MiniZinc challenge)	100	Sean MacDermant	4
Simon Bowly	92	Felix Cebulla	3
Gerald Gamrath	76	Irv Lustig	3
Hans Mittelmann	75	Jesus Rodriguez	3
Haroldo Gambini Santos	71	Jeff Linderoth (except NEOS server)	3
Shunji Umetani	60	Jonathan Eckstein	3
Matias Soerensen	42	Siwei Sun	3
Jordi Castro	34	Felix J L Willamowski	2
Toni Sorrell	34	Janos Hoener	2
Stephan Beyer	32	Joshua Friedman	2
Manuel Iori	30	Utz-Uwe Haus	2
Michael Winkler	28	Andreas Baermann	1
Cezar Augusto Nascimento e Silva	26	Balabhaskar Balasundaram	1
Pelin Damci-Kurt	26	Christian Liebchen	1
Domenico Salvagnin	22	Christopher Daniel Richards	1
Michael Bastubbe	19	Dan Neiman	1
George Fonseca	18	Daniel Bienstock	1
Sascha Kurz	17	Daniel Rehfeldt	1
Antonio Frangioni	15	Gavin Goodall	1
Daniel Heinlein	12	Gerald Lach	1
Marc Pfetsch	11	Hsiang-Yun Wu	1
Tamas Terlaky	11	Jesse Liu	1
Berk Ustun	10	Juan Javier Dominguez Moreno	1
Philipp Leise	9	Koichi Fujii	1
Salim Haddadi	9	Mark Husted	1
Dan Hiroshige, Koichi Fujii	8	Paula Carroll	1
Christopher Hojny	7	Sujayandra Vaddagiri	1
Laurent Sorber	6	Timo Berthold	1

keeping their original submission information intact. Those 2,051 comprised most of the submissions to MIPLIB 2010 except for a few duplicates already present in other MIPLIB 2017 submissions.

Table 2 reports the origin of the submitted instances. It shows that there are two blocks of new submissions with a high number of instances: the instances submitted to the NEOS server and MIP models of instances that were part of the MiniZinc Challenges[4] from 2012 to 2016. Even excluding those instances, the remaining new contributions comprise about as many instances as were submitted to MIPLIB 2010, half of which were collected from publicly available sources back then. Note also that nearly 50 instances originate from submissions to the user support of two commercial solvers.

2.2 Submissions from the NEOS Server

Before moving on, we briefly discuss how the instances originating from the NEOS Server were collected, since this was a separate procedure unto itself. The NEOS Server is a free internet-based service for solving numerical optimization problems hosted by the

[4] https://www.minizinc.org/challenge.html

Table 2: Origin of the submitted instances. Each instance is counted only once, in the first applicable row.

MIPLIB 3	65
MIPLIB 2003	28
MIPLIB 2010	335
MIPLIB 2010 submissions	1,623
new NEOS server submissions	785
MiniZinc challenge	988
new contributions	1,897

Wisconsin Institute for Discovery at the University of Wisconsin in Madison, with remote solving services provided by various sites, such as Arizona State University. In the calendar years 2014–2016, more than 1 million MIP instances were solved using NEOS. A subset of these instances was collected and submitted for inclusion into MIPLIB 2017.

To begin, the 14,571 MIP instances that were modeled through the AMPL [11] language and whose solution required more than five minutes of wall clock time to solve were collected. In discussions with the MIPLIB committee, it was decided that a representative, yet diverse, sample of around 700-800 instances would be an appropriate number of submissions from NEOS. The strategy to select the instances was based on clustering the instances with similar traits and then selecting instances from the clusters.

All 14,571 of these instances were re-run using CPLEX 12.6 to collect the optimal solution (if possible within 12 hours) and the solution to the root LP relaxation. For each instance, the following properties were collected:

- the number of variables, constraints, and nonzeros,
- the percentage of binary variables and of general integer variables,
- the best solution value found by CPLEX, the root gap, and the percentage of fractional variables in the root LP solution.

Instances whose number of nodes or constraints was more than 3 standard deviations away from the mean were excluded as outliers. In an attempt to eliminate duplicate instances, only one instance in a group that had the same number of constraints, same number of variables, and same optimal solution value, as well as the same associated email address, was retained. After this removal of outliers and suspected duplicates, 6,531 instances remained.

Each of these instances was assigned to a multi-labeled category using three independent calls to a k-means clustering algorithm [21] as follows. First, the instances were divided into four clusters according to properties associated with problem size: the number of constraints, number of variables, and number of nonzeros. Second, the instances were again divided into four clusters, this time according to the percentage of binary and general integer variables in the instance. Finally, the instances were divided into four more clusters based on two properties of the root LP relaxation: root gap and the percentage of integer variables whose root LP relaxation value was not integer-valued.

From these clusterings, each of the 6,531 instances was given a label in $\{0, 1, 2, 3\}^3$, all in all giving $4^3 = 64$ (possibly empty) clusters. Let S_i be the number of instances in the i-th cluster. The final selections were then made as follows:

- For each cluster with size $S_i \leq 2$, all instances of the cluster were selected;
- for each cluster with size $3 \leq S_i \leq 10$, $\lceil |S_i|/2 \rceil$ instances were selected at random;
- for each cluster with size $S_i > 10$, $\lceil |S_i|/10 \rceil$ instances were selected at random.

This strategy ensured a diverse sample, and resulted in a total of 710 instances to be considered for inclusion in MIPLIB 2017.

Separately, Hans Mittelmann also contributed 75 instances submitted to one of the ASU-hosted servers through the NEOS platform. These servers process requests for the SCIP, FEASPUMP, PROXY, and QSOPT_EX solvers. Using scripts, these user submissions were screened for promising instances with an emphasis on those that contain model information.

In combination, this resulted in a total of 785 new instances arising from NEOS. In addition, the instance pool for MIPLIB 2017 contains 391 NEOS instances that have been either included in previous versions of MIPLIB or were considered during past selection processes.

2.3 Instance Cleanup

Following the close of submissions, we performed several cleanup steps, as follows.

Instance renaming. While we tried to preserve original file names where possible, some submissions required renaming to create uniquely identifiable names. For instance, some submissions contained multiple instances with the same file name that originated from the same base model populated with different data. We also tried to make the names as short and expressive as possible, containing only lower-case characters, numbers and hyphens. After re-naming, the longest instance name was reduced from 151 to 31 characters. Note that submissions for MIPLIB 2010 were renamed to be consistent with the naming scheme of the final MIPLIB 2010 instances, but no submissions already present in MIPLIB 2010 were renamed.

Making NEOS instances identifiable. NEOS instances have traditionally been named *neos-xyz*, where *xyz* is a number with up to 7 digits, representing the unique NEOS instance ID of the submitted instance. In order to allow easier identification of the instances in papers and personal communications, we compiled a list of river names from all over the world and appended a river name to the original name, resulting in names of the form *neos-xyz-river*. As an example, NEOS instance `3075395` has been renamed `neos-3075395-nile` such that it can be colloquially called the "nile instance". We excluded river names such as "Amazon" to avoid ambiguous renaming. Note that we applied this renaming procedure only to the 785 NEOS instances newly submitted for MIPLIB 2017 (see Section 2.2), leaving all previously available NEOS instances under their old name to avoid confusion.

Format conversion and cleanup. All instances in the MIPLIB 2017 collection are provided in MPS format [24, 32]. Therefore, instances submitted in LP format were read into CPLEX and written out in MPS format. Given that different MIP solvers support different extensions of the MPS format, we ensured that all solvers could read all instances by restricting the format to a basic version. Maximization problems were turned into minimization ones by negating objective coefficients; lazy constraint markers were removed so that the constraints are treated as standard constraints; and coefficients stated as hexadecimal numbers were converted to their decimal equivalent. Additionally, we added a NAME section or changed the existing one to the updated instance name. For a small number of instances, it was necessary to change the MPS files because they contained ambiguous definitions or outright incorrect formatting. In those cases, we performed the minimally necessary changes to make the instance file adhere to the basic MPS standard.

Table 3: Model groups and counts for the different instance sources.

		Group Size $\in$			
Type	**Groups**	$\{1\}$	$\{2,\ldots,5\}$	$\{6,\ldots,10\}$	$\{11,\ldots,360\}$
MiniZinc	80	0	16	38	26
NEOS	110	8	24	34	44
MIPLIB (excl. NEOS)	241	172	30	16	23
Indiv. submissions	130	81	15	10	24

2.4 Model Groups

In most cases, the instances in a single submission are closely related in the sense that they originate from the same or a very similar MIP model. Some submissions, however, contain many instances out of which the instances are mostly unrelated, e.g., the submissions from the NEOS server or the MiniZinc challenge. Therefore, we introduced *model groups* to keep track of this form of meta information that may not be directly inferable from the submission ID or the numerical instance features described in Section 3. A model group represents a set of instances that is based on the same model or a very similar formulation and only uses different data. This grouping allowed us to avoid overrepresentation of a particular application or model class in the final library by limiting the number of instances with known similar model background during the selection process.

Each instance was assigned to one model group as follows. Initially, a submission of homogeneous instances was assigned to its own model group. If a submission contained multiple sets of instances, each implementing a different model for the same problem and data set, an individual group was created for each of the different model types. Publicly available instances with known application (including the MiniZinc instances) were grouped by hand by the authors.

Finally, NEOS instances were grouped in an automated way, since users often submit multiple, similar instances. In order to group the NEOS instances, a k-means clustering was computed with respect to the entire instance feature space (see Section 3). The parameter $k = 110$ was chosen manually to achieve a clustering with very similar instances in each NEOS model group. This clustering was applied to all 1,176 NEOS instances both from new submissions and previously available sources.

Table 3 summarizes the number of resulting model groups and the corresponding model sizes for the different sources of instances. The largest model group *cmflsp* comprises 360 instances of a capacitated multi-family lot-sizing problem.

3 Feature Computation

The ultimate goal of the selection process was to select a representative sample of all available instances with respect to problem structure and computational difficulty, while avoiding overrepresentation of similar models. In the spirit of data-driven decision making and in light of related work in the fields of algorithm selection and machine learning, we based this process both on performance data and on an extensive set of instance features. The first step in the selection process was simply to determine the features of interest and to compute the feature vectors associated with each instance. Although this may seem straightforward, it is important to note that the feature vector corresponding to an instance can be affected by seemingly irrelevant properties of its representation in MPS format. For instance, some of the raw MPS instances contained modeling language artifacts or artifical redundancies. For this reason, the instance features were computed only after applying some straightforward cleanup steps, which we refer to as *trivial pre-*

solving. We first describe this presolving process before describing what features of the instances were used and how their values were determined for each instance.

3.1 Trivial Presolving

As is traditional for MIPLIB, the submitted instances remain unchanged (except for MPS format corrections) and have not been replaced by a presolved version. This has several advantages. Primarily, it means that MIPLIB remains suitable for the benchmarking of presolving algorithms, which are actively developed and have a high impact on the performance of a solver (see [4, 16]). Secondarily, model generation procedures tend to introduce auxiliary variables and constraints or other redundancies that require appropriate handling by the solvers such that their efficiency in doing so should be part of the benchmarking as an important part of the overall solution process.

Despite the need to leave instances unchanged in the test set, all MIP solvers do apply some level of presolving procedures to the instance before the branch-and-bound search in order to tighten the problem formulation, remove redundant variables and constraints, and extract additional information about the instance. Consequently, statistics about the original instance may not correctly reflect the "true" properties of the instance that is solved after presolving was applied. This includes not only obvious properties, such as instance size, but less obvious ones, such as the type of constraints and variables present in the model. A model may, for example, have all variables integer with the exception of one continuous variable whose value is fixed to 1 and whose purpose is to model an objective offset. It would be unreasonable to consider such an instance to be an instance with both integer and continuous variables. At the other extreme, we may have an instance in which all binary and integer variables are implicitly fixed, leaving a purely continuous problem after presolving.

While it seems necessary to do some presolving before computing instance features, the full presolving done by solvers is itself a difficult computational balancing act and each solver does it differently. Too much presolving before feature computation would result in a presolved instance with features no more representative of the "true" ones than the completely unpresolved instance. As a compromise, all instance features introduced in Sections 3.3–3.5 were collected after applying a reduced set of the most obvious presolving techniques to the instance, but no more sophisticated techniques.

For this *trivial presolving*, we used SCIP 5.0, but disabled most presolving techniques, applying only simple ones, such as the removal of redundant constraints and fixed variables, activity-based bound tightening, and coefficient tightening. In contrast to standard SCIP presolving, which stops if the problem size could be reduced by only a small percentage during the last presolving round, we applied the simple presolving steps until a fixed point was reached. The complete set of SCIP parameters used to do the presolving is provided on the MIPLIB web page (see Section 5.3) as part of the feature extractor download.

For 55 of the 5,721 submitted instances, trivial presolving turns the instance into a pure LP or is even able to solve the instance by fixing all variables. These instances were not considered for inclusion and also serve to emphasize the importance of this preprocessing step. Overall, trivial presolving reduced the number of variables on 3,782 instances (66 % of the submission pool), sometimes by as much as 93 % (instance `a2864-99blp`). For 445 instances (8 %), more than 50 % of the variables were fixed. On average, trivial presolving reduced the number of variables by 15 %.

3.2 Canonical Form

Because the feature computation can not only be affected by presolving but by the exact form in which the instance is represented (equality constraints versus inequality, etc.), we transformed all presolved instances into the following canonical form, which is slightly more general than the usual one, prior to feature computation.

Definition 1. A *mixed integer optimization problem* P with input

- $m, n, n_b, n_i, n_c \in \mathbb{N}$, $n = n_b + n_i + n_c$,
- *coefficient matrix* $A \in \mathbb{Q}^{m \times n}$,
- *left-hand and right-hand side vectors* $L, U \in \mathbb{Q}^m_{\pm\infty}$,
- *lower and upper bound vectors* $\ell, u \in \mathbb{Q}^n_{\pm\infty}$, and
- *objective coefficient vector* $c \in \mathbb{Q}^n$

is defined to be an optimization problem of the form

$$\min \left\{ c^\top x : L \leq Ax \leq U, \ell \leq x \leq u, x \in \{0,1\}^{n_b} \times \mathbb{Z}^{n_i} \times \mathbb{Q}^{n_c} \right\}.$$

It is important to note that transformation to the canonical form of Definition 1 is not uniquely determined for each instance. There remain certain degrees of freedom to formulate equivalent instances by scaling continuous columns, the objective function $c^\top x$, or a constraint $L_i \leq a_i^\top x \leq U_i$ by a positive scalar $s > 0$. This may cause problems with the computation of some features. For example, some features of interest involve comparison of row coefficients, but this comparison is difficult if rows of the constraint matrix may have coefficients differing by several orders of magnitude, We address these issues by normalizing the objective coefficients c and every constraint $L_i \leq a_i^\top x \leq U_i$ by their maximum absolute coefficient $\|c\|_\infty$ and $\|a_i\|_\infty$, respectively, so that all objective and matrix coefficients lie in the interval $[-1, 1]$ before computing the feature matrix F (the downloadable instances are not altered).

It is based on this final presolved canonical representation that we define the $Q = 110$ features we consider. This results in a *feature matrix* $F \in \mathbb{R}^{N \times Q}$, where N is the total number of instances submitted. Table 4 lists these features, which were divided into $K = 11$ *feature groups* (also listed in the table). Feature groups were used for the selection process, during which instance clusters were computed for each feature group individually. Every feature group was chosen to represent a particular aspect of an instance in the form specified by Definition 1. The computation of features in most of the groups only requires information that can be extracted directly from the input of the (trivially presolved) problem. Two exceptions are the constraint classification and decomposition groups, which need to identify structures in the model. These are described in Sections 3.4 and 3.5.

3.3 Instance Features

Here, we describe the first nine feature groups in Table 4. We use the shorthand *vector statistics* to refer to five values summarizing the entries of a vector $v \in \mathbb{R}^d_{\pm\infty}$. Let $d' = |\{j : |v_j| < \infty\}|$ be the number of finite entries of v, which can be smaller than d in the case of, e.g., bound vectors, and let v' be the restriction of v to its finite entries. We assume without loss of generality that v' is sorted, $v'_1 \leq v'_2 \leq \cdots \leq v'_{d'}$. The five values are

- $\min : v \mapsto v'_1$,
- $\max : v \mapsto v'_{d'}$,

Table 4: Description of instance features used. Set notation is abbreviated, e.g., $\{A \neq 0\}$ denotes $\{(i,j) \in \{1,\dots,m\} \times \{1,\dots,n\} : a_{i,j} \neq 0\}$

Group	Features	Description	Scaling
Size	3	size m,n of matrix, nonzero entries $\lvert\{A \neq 0\}\rvert$	$\log_{10}(x)^2$
Variable types	3	Proportion of binary, integer, and continuous variables $\frac{n_b}{n}, \frac{n_i}{n}, \frac{n_c}{n}$	
Objective nonzero density	5	Nonzero density of objective function $\frac{\lvert\{c\neq 0\}\rvert}{n}$ both total and by variable type (bin., int., cont.), 0-1 indicator for feasibility problems without objective	
Objective coefficients	6	vector stats. and dynamism of c	c normalized by $\lVert c\rVert_\infty$
Variable bounds	12	Finite densities $\frac{\lvert\{\lvert\ell\rvert<\infty\}\rvert}{n}, \frac{\lvert\{\lvert u\rvert<\infty\}\rvert}{n}$ of bounds, vector stats. of upper bounds u and bound ranges $u-\ell$.	vector stats. scaled by $\mathrm{siglog}(x)$
Matrix nonzeros	6	vector stats. of nonzero entries $\lvert\{a_i \neq 0\}\rvert$ by row in A, nonzeros per column $\frac{\lvert\{A\neq 0\}\rvert}{n}$	$log_{10}(x)$ for nonzeros per column
Matrix coefficients	24	vector stats. of the coefficient vector stats. of each row of the matrix	every a_i normalized by $\lVert a_i\rVert_\infty$
Row dynamism	5	vector stats. of row dynamism $\frac{\lVert a_i\rVert_\infty}{\min_j\{\lvert a_{ij}\rvert \neq 0\}}$	$log_{10}(x)$
Sides	19	vector stats. of left- and right-hand sides L, U and concatenated $(L\|U)$, nonzero and finite densities of L, U	every a_i normalized by $\lVert a_i\rVert_\infty$
Constraint classification	17	Proportion of classes of special linear constraints: singleton, precedence, knapsack, mixed binary (see Section 3.4)	
Decomposition	10	Features describing decomposition D found by GCG with maximum area score (see Section 3.5): areascore(D), k, vector stats. (except std) of $\left(\frac{\lvert D_1^r\rvert}{m},\dots,\frac{\lvert D_k^r\rvert}{m}\right)^\top$ and $\left(\frac{\lvert D_1^c\rvert}{n},\dots,\frac{\lvert D_k^c\rvert}{n}\right)^\top$. Not available for all instances.	

– mean : $v \mapsto \frac{1}{d'} \sum_{j=1}^{d'} v'_j$,

– median : $v \mapsto \left(v'_{\lfloor \frac{d'+1}{2} \rfloor} + v'_{\lceil \frac{d'+1}{2} \rceil} \right) /2$, and

– std : $v \mapsto \sqrt{\frac{1}{d'} \sum_{j=1}^{d'} \left(v'_j - \text{mean}(v') \right)^2}$.

Note that infinite entries can only occur for the variable bound vectors ℓ and u and the left- and right-hand side vectors L, U. For a vector v that contains only infinite entries, i.e., for which $d' = 0$, the above vector summaries are not well-defined. If $d' = 0$, the corresponding statistics were set to 0 in the data. Note that even if the original formulation has infinite bounds on variables, trivial presolving may often infer finite bounds for those variables.

The *dynamism* of a vector with finite entries is the ratio of the largest and smallest absolute entries, i.e., $\|v\|_\infty / \min\{|v_j| : v_j \neq 0\}$. Note that the dynamism is always at least 1. If the dynamism in a single constraint exceeds 10^6, we see this as an indication of a numerically difficult formulation. Note that the dynamism is invariant to the normalization procedure. Combining the dynamism of each constraint yields an m-dimensional vector, which can be summarized using vector statistics.

For features such as the row or objective dynamism, which may differ by orders of magnitude between instances, we used a logarithmic scaling. While logarithmic scaling is fine for vectors with positive entries, it is not applicable to vectors with potentially negative entries such as the variable upper bounds u. In those cases, we apply a customized scaling

$$\text{siglog} : \mathbb{R} \to \mathbb{R}, x \mapsto \text{sig}(x) \log_{10}(|x| + 1)$$

to every entry of the corresponding column in the feature matrix F. The map siglog preserves the sign of each entry.

The collection of the instance features was performed with a small C++ application called *feature extractor*, which extends SCIP by the necessary functionality needed to report features after trivial presolving and optionally accepts a settings file to modify the default presolving explained in Section 3.1. The feature extractor is a modified version of a code used already by [17] and available for download on the MIPLIB 2017 web page (see Section 5.3).

3.4 Constraint Classification

Table 5 lists the constraint classification types used for the feature group CONSTRAINT CLASSIFICATION. A total of 17 types of linear constraints were identified, which often occur as a subset of the constraints of MIP instances. The table is sorted from most specific to most general. In case that a constraint belongs to multiple types, the classification always assigns the most specific, i.e., topmost type that applies. Note that even empty, free, and singleton constraints are listed. While these types are removed during trivial presolving, they may well be present in the original formulation.

There are several types of constraints supported by the MPS format [24, 32] that are not strictly linear as required by Definition 1. A well-known extension are *indicator* constraints, which are conditional, linear constraints that only need to be satisfied if a corresponding binary variable, the so-called *indicator variable*, is set to 1. It is possible to linearize such a constraint by employing a sufficiently large coefficient M for the indicator variable, in which case the reformulation is called a *big-M formulation*. In many practical applications, big-M formulations require a very large value of M, which is why they often lead to numerically difficult models. Directly expressing such constraints as indicator constraint allows the solver to handle them in a more algorithmically advantageous way.

Table 5: Classification of linear constraints, sorted from most specific to most general. A constraint is always assigned the first (topmost) type that applies.

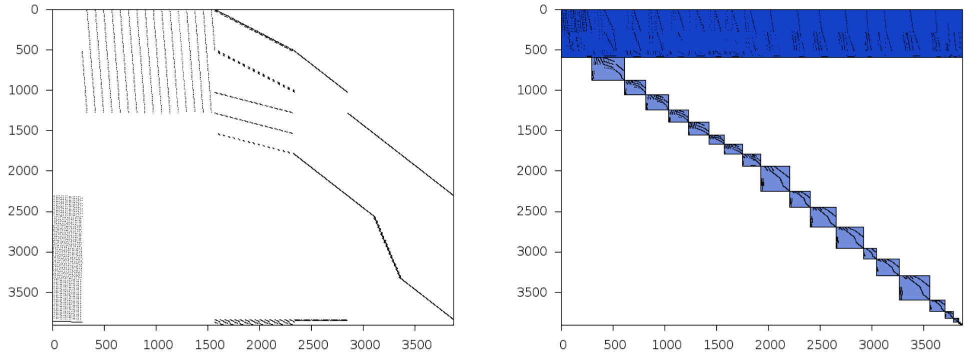
Type	Linear constraints ...
Empty	... with no variables
Free	... with no finite side
Singleton	... with a single variable
Aggregation	... of the type $ax + by = c$
Precedence	... of the type $ax - ay \leq b$ where x and y must have the same type
Variable Bound	... of the form $ax + by \leq c$, $x \in \{0, 1\}$
Set Partitioning	... of the form $\sum x_i = 1$, $x_i \in \{0, 1\} \, \forall i$
Set Packing	... of the form $\sum x_i \leq 1$, $x_i \in \{0, 1\} \, \forall i$
Set Covering	... of the form $\sum x_i \geq 1$, $x_i \in \{0, 1\} \, \forall i$
Cardinality	... of the form $\sum x_i = k$, $x_i \in \{0, 1\} \, \forall i$, $k \geq 2$
Invariant Knapsack	... of the form $\sum x_i \leq b$, $x_i \in \{0, 1\} \, \forall i$, $b \in \mathbb{N}, b \geq 2$
Equation Knapsack	... of the form $\sum a_i x_i = b$, $x_i \in \{0, 1\} \, \forall i$, $b \in \mathbb{N}, b \geq 2$
Binpacking	... of the form $\sum a_i x_i + ax \leq a$, $x, x_i \in \{0, 1\} \, \forall i$, $a \in \mathbb{N}, a \geq 2$
Knapsack	... of the form $\sum a_i x_i \leq b$, $x_i \in \{0, 1\} \, \forall i$, $b \in \mathbb{N}, b \geq 2$
Integer Knapsack	... of the form $\sum a_i x_i \leq b$, $x_i \in \mathbb{Z} \, \forall i$, $b \in \mathbb{N}$
Mixed Binary	... of the form $\sum a_i x_i + \sum p_j s_j \, \{\leq, =\} \, b$, $x_i \in \{0, 1\} \, \forall i$, $s_j \in \mathbb{R} \, \forall j$
General Linear	... with no special structure

Indicator constraints were allowed into the MIPLIB 2017 collection, but (as we describe later) were not allowed in the benchmark set. The only feature used that involves indicator constraints was their fraction among all constraints. This feature is also part of the feature group Constraint classification in Table 4. Other features regarding, e.g., the linear part of the indicator constraint, are not collected. In total, 437 of the submitted instances contain indicator constraints. Many of them appear in instances of the MiniZinc submission, which were additionally submitted with big-M formulation.

There are other special types of constraints allowed by MPS, such *special-ordered sets* (SOSs), *semicontinuous variables*, and piecewise linear constraints that not all solvers support. However, none of the instances submitted used such constraints.

3.5 Block-Structured Decomposition Features

Dantzig-Wolfe reformulation and Lagrangian relaxation are decomposition techniques that can exploit the presence of logical groupings of constraints and variables in the model. Classically, one identifies "complicating" or "linking" constraints that, when removed from the model, result in several subproblems that can be solved independently. Concretely, this occurs when the contraint matrix has so-called block angular structure (defined below). One challenge in applying these techniques within a general purpose solver is identifying such structure. Standard input formats do not allow information about these structures to be passed to the solver directly. For this reason, several decomposition-based solvers have been developed that accept auxiliary input files indicating this structure when it is present. These include GCG [15], DIP [12, 13, 14], and DECOMP (a decomposition-based solver that is part of SAS/OR and SAS Optimiza-



(a) original

(b) decomposition

Figure 2: Instance `b1c1s1` after trivial presolving; left the nonzero entries as they appear in the original constraint matrix; right with rows and columns re-arranged according to the decomposition produced by GCG.

tion [34]). All three can exploit the identified structure by reformulating the instance and applying a branch-and-price algorithm to solve it.

Although block structure may be "obvious" in the original model, this structure is often lost when the model is populated with data and transformed into a format such as MPS. When information about the block structure is not provided, it can still be derived algorithmically. This information is thus at a higher level of abstraction than the other instance features alone. Contributors to MIPLIB 2017 were invited to provide complementary material, such as model files in GAMS or AMPL format or information on block-structured decompositions in the .dec file format. While some 20 submitters accompanied their instances with model files to produce them, no block structure information was contributed. Nevertheless, GCG, DIP, and DECOMP are all able to derive information about the block structure automatically (see, e.g., [18, Sec. 5]).

Given the coefficient matrix A of an instance, we formally characterize a (block structured) decomposition by a partition of the rows $D^r = (D^r_1, \dots, D^r_k, L^r)$ of A and a partition of the columns $D^c = (D^c_1, \dots, D^c_k, L^c)$ of A, such that for every entry $a_{i,j} \neq 0$ of A with $i \in D^r_{\ell_1}$ and $j \in D^c_{\ell_2}$ it holds that $\ell_1 = \ell_2$. We say that such a decomposition has k blocks. The number k of blocks (normalized by the average number of blocks over all instances), the vector statistics (except the standard deviation) of the numbers of rows and columns per block, respectively, and the so-called *area score* give rise to ten features related to block-structured decompositions that were derived from the trivially presolved instances using the detection capabilities of GCG 3.0

The area score for a decomposition $D = (D^r, D^c)$ of A is defined as

$$\text{areascore}(D) = 1 - \frac{\sum_{b=1}^{k} |D^r_b||D^c_b| + n|L^r| + m|L^c| - |L^r||L^c|}{mn} ,$$

which intuitively measures the fraction of the matrix that is "white" in Figure 2b. A variant of this score is also used by GCG 3.0 to select a decomposition in the case that several are found (such structure is not uniquely determined). An area score closer to 1.0 is better. A very low area score indicates that the model does not contain any substructures that are easily identifiable by GCG 3.0. This does not imply that there *is no* such structure, but rather that it is not obvious.

As computing decompositions can be very time consuming, in particular for huge instances, GCG was run in a minimal detection configuration. In this configuration, GCG first groups the constraints of A according to (a) their type in SCIP, (b) their MIPLIB type (see Table 4), and (c) their number of nonzeros. For each respective grouping, the

Table 6: List of solvers used for performance evaluation.

Solver	Version	Threads	Ref.
CBC	2.9.8	4	[9]
IBM CPLEX	12.7.1	4	[10]
Gurobi	7.5.1	4	[20]
MATLAB	R2017b	1	[29]
MOSEK	8.1.0.30	4	[31]
SAS/OR	14.2	4	[34]
SCIP	4.0.0	1	[35]
FICO Xpress	8.2	4	[38]

smallest cardinality groups of constraints are joined until a specified (small) number of groups remains. From this collection, each subset is potentially selected as the set L^r of linking constraints of the decomposition. The so-called *connected finisher* assigns all remaining rows to as many blocks as possible. It is conceivable that this leads to only one block, in which case the area score of the resulting decomposition is 0.0. From the pool of decompositions that are constructed this way, one with maximum area score is selected for computing the decomposition features of the instance.

3.6 Acquisition of Performance Data

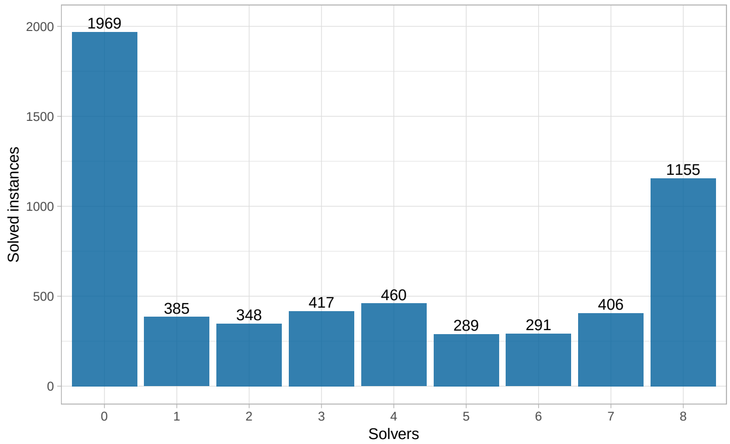
Every submitted instance was processed with each of the eight solvers listed in Table 6 to collect performance data. The experiments were performed on two Linux clusters. The first one consisted of 32 nodes, each equipped with two 3.20 GHz 4-core Intel Xeon X5672 CPUs and 48 GB RAM, the second consisted of 16 nodes, each equipped with two 2.50 GHz 10-core Intel Xeon E5-2670 v2 CPUs and 64 GB RAM. Two such jobs were run in parallel on the same cluster node, each job using a time limit of 4 hours and 4 threads, except for SCIP and MATLAB[5], which are both single-threaded. In total, this performance evaluation required almost 40 CPU years.

Figure 3a shows for every possible cardinality $k = 0, 1, \ldots, 8$ the number of instances solved by exactly k solvers. 1,969 of the 5,721 instances (34 %) were not solved by any solver within 4 hours (an instance was considered solved if there were no inconsistencies between solvers and the solution was verified to be feasible (see Section 3.7). There were 1,155 instances (20 %) that could be solved by all eight solvers.

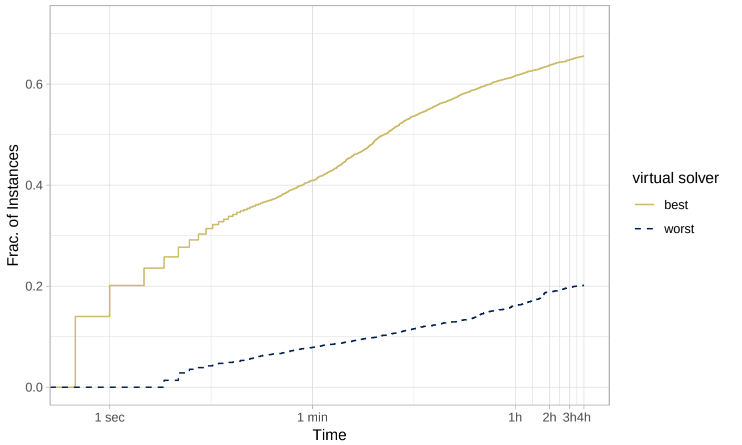
To summarize the results of the experiments, we report here the performance measures for *virtual solvers*, as described in the introduction. For each instance, a virtual solver is a summary of all tested solvers by means of an aggregation function such as min, max, and median, resulting in the best, worst, and median virtual solvers, respectively. The term "virtual" is used to distinguish the presentation from the best (fastest) or worst (slowest) actual solver over the complete set of instances. The performance measures collected are the time to optimality and the number of branch and bound nodes processed.

Figure 3b compares the fraction of instances solved by the virtual best and worst solvers. A large discrepancy between the curves can be observed. The virtual best solver finished on about 20 %, 40 %, and 60 % of the submissions within 1 sec., 1 minute, and 1 hour, respectively. The virtual worst solver required more than a second for any instance, and solved only 20.2 % of the instances within the time limit of 4 hours. The

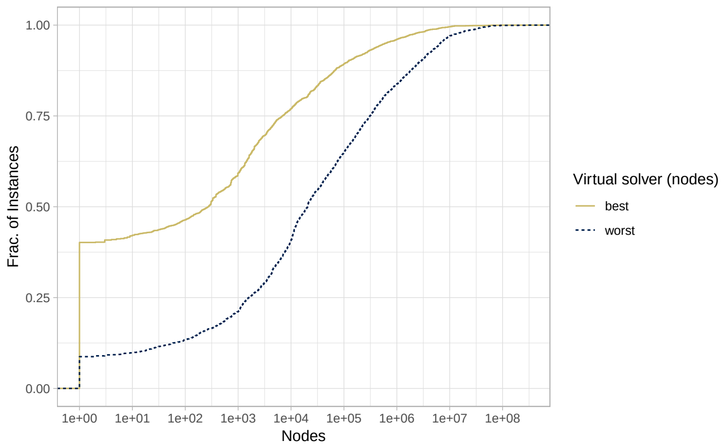
[5]MATLAB was run using the command `intlinprog`, which is part of the Optimization Toolbox (TM).



(a) Number of instances solved by a specific number of solvers.



(b) Fraction of instances solved by virtual best and worst solver for different time limits.



(c) Explored branch-and-bound nodes for *solved* instances.

Figure 3: Aggregated results of the performance evaluation on the entire submission.

virtual best solver solved more instances in 2 seconds than the virtual worst solver was able to solve in 4 hours. Note that all eight tested solvers contributed to the performance of the virtual best solver, i.e., each solver was the fastest on at least one instance.

Figure 3c summarizes data about the number of branch and bound nodes processed. As expected, the number of branch-and-bound nodes varies significantly between instances, but also between solvers on individual instances. In Figure 3c, the minimum and maximum number of explored nodes are shown. In this figure, we consider only runs that completed sucessfully. Note that there are differences in how solvers report the required number of branch-and-bound nodes. Concretely, the solution of an instance during presolving may be reported as 0 or 1 nodes depending on the solver used. We therefore normalize the node results so they are always at least 1, i.e., we consider presolving as part of the root node solution process. This is also justified because we only consider instances that could not be solved completely by trivial presolving. At the left end of the scale are the instances that could be solved within 1 node. This group amounts to 1,507 instances, which corresponds to 40 % of the instances that could be solved at all and 26 % overall. For more than 50 % of the solved instances, the solution process required less than 1,000 nodes. The maximum number of explored nodes is considerably larger. Less than 25 % of the considered instances were solved within 1,000 nodes by all solvers that finished within the time limit. Note that for all 385 records (see Figure 3a) for which only one solver finished within the time limit, the minimum and maximum number of explored nodes coincide.

3.7 Consistency Check of Solver Results

In order to identify numerically challenging instances and incorrect answers returned by the solvers, the results of the performance runs were independently verified in two different ways.

First, the feasibility of every primal solution reported at the end of a solution process was checked. To accomplish this step, a solution checker, whose purpose is to validate feasibility of the solutions computed by a given MIP solver against the original model has been bundled with the MIPLIB scripts since MIPLIB 2010. The checker tries to recognize incorrect results, while at the same time taking into account that most MIP solvers use floating-point arithmetic, and thus exact feasibility cannot be expected. The overall structure of the solution checker has been largely unchanged since MIPLIB 2010. It is important to emphasize that it only checks feasibility of a given solution, it cannot check optimality in general. The solution vector returned by the solver is verified against the original instance in MPS format, with all computations performed using the arbitrary precision arithmetic package GMP [19]. Feasibility of linear constraints, integrality and the objective value are all verified according to given tolerances. We refer to the MIPLIB 2010 paper [25] for more details on the solution checker design and limitations.

For MIPLIB 2017, we updated the solution checker such that it uses a more flexible (and forgiving) definition of violation of a linear constraint. The previous version of the checker used an absolute tolerance of $\varepsilon = 10^{-4}$, so that when given a linear constraint $a_i^\top x \leq U_i$ and a solution x^*, it would have considered the constraint satisfied if and only if

$$a_i^\top x^* - U_i \leq \varepsilon.$$

However, this turns out to be too strict if the linear constraint has coefficients with a large absolute value. At the same time, switching to a purely relative tolerance was not considered a viable option, as it can lead to a too small effective tolerance when tiny coefficients are involved. So, in the new version we introduced a different definition of violation, that tries to combine the strengths of absolute and relative tolerances, and also to possibly cope with cancellation effects when evaluating the variable part (activity) of the constraint. In particular, we split the value $a_i^\top x^*$ into its positive and negative parts as $a_i^\top x^* = (a_i^\top x^*)_+ - (a_i^\top x^*)_-$ and consider the linear constraint satisfied if and only if

$$a_i^\top x^* - U_i \leq \varepsilon \cdot \max\{(a_i^\top x^*)_+, (a_i^\top x^*)_-, |U_i|, 1\}.$$

Given the relaxed definition of violation, it was decided to use the stricter value $\varepsilon = 10^{-5}$ as tolerance. The very same logic is applied when checking variable bounds, and when checking the objective value c^* reported by the solver—we just treat it as the linear constraint $c^\top x^* = c^*$. Note that integrality is still checked with a purely absolute tolerance of 10^{-4}. Finally, the solution checker was extended to support indicator constraints.

Following the feasibility check, which can be done independently for each solver and solved instance, the results were compared between solvers to identify discrepancies in the optimal values reported or inconsistencies in primal and dual bounds. We used the publicly available tool IPET [22] to parse the solver log files and validate the results. We considered results inconsistent when solver A reported a verified, feasible solution with value c^*, while solver B timed out reporting a dual (lower) bound that was higher than c^*. This included the special case that an instance had been reported infeasible by solver B. For example, on the instance `bc1`, seven of eight solvers agreed on an optimal value of 3.338 after exploring search trees with 3k–20k nodes. The eighth solver, however, reported a solution of value 3.418 as optimal after 720 nodes. The eighth solver cut off the optimal solution. Note that while such a behavior can be caused by a bug in the solver, it is also possible that different optimal values can be "correctly" obtained when different tolerances are used. Since all MIP solvers rely on floating-point arithmetics and use feasibility tolerances, the definition of "the optimal objective value" for a problem instance is ambiguous. In particular for numerically challenging problems, a solver might return a different optimal objective value as a result of applying slightly stricter tolerances within the algorithm. Instances exhibiting such ambiguity are not suitable for benchmarking, since handling this numerical ambiguity can be done in different ways, requiring different amounts of computational effort. This leads to difficulties in comparison. Therefore, we disregarded all instances with such inconsistencies during the selection of the benchmark set (see Section 4.6), unless the inconsistency was obviously caused by a bug in one solver. 328 instances (5 %) were removed for this reason.

4 Selection Methodology

Due to the vast number of collected instances and the stark overrepresentation of some problem classes and instance types, it was crucial to reduce the submitted instances to a carefully chosen selection that provides both researchers and practitioners with a meaningful basis for experimental comparison. MIPLIB 2017 provides two main instance sets, namely the *benchmark set* and the *collection*. In the following, we discuss the actual selection process and the obtained result.

We approach this task in the reverse order by first selecting the larger MIPLIB 2017 collection from the submitted instances, and then choosing the MIPLIB 2017 benchmark set as a subset of the collection. Both the collection and benchmark set should provide a good coverage of the feature space given by all submissions and should also be balanced. Note that our aim is explicitly not to represent the composition of instance properties observed in the set of submitted instances, because this is typically highly unbalanced. We would like to choose the collection as large as possible in order to obtain a rich and diverse test set. As explained above, the benchmark selection step is more restrictive. Here the goal is not only to choose the largest set possible, but also to favor instances that are currently hard for all solvers.

It seems quite natural to formulate this task as an optimization problem. In fact, we approach the generation of MIPLIB 2017 with a sequence of optimization problems: a set of diversity preselection MIPs, the collection MIP, and finally, the benchmark MIP. After an initial cleanup, large model groups (see Section 2.4) are cut down to a handful of diverse instances by the application of the diversity preselection model described in Section 4.2. The main purpose of the first selection procedure is to avoid overrepresentation of instance types from large and very homogeneous submissions that do not add to the diversity of the instance library. Sections 4.3 and 4.5 introduce the clustering procedures to partition the instances based on instance features and performance data, respectively. Sections 4.4 and 4.6 describe the mixed integer optimization models used to compute the MIPLIB 2017 collection and benchmark sets.

Before the selection steps outlined here, the submission pool of 5,721 instances was already reduced to 5,666 instances by the removal of LPs (no discrete variables) and instances that are empty after a trivial presolving (see Section 3.1). Furthermore, we removed pairs of duplicate instances identified during the selection process.

4.1 Benchmark Suitability

The following definition characterizes the requirements for an instance to be in the benchmark set of MIPLIB 2017.

Definition 2 (Benchmark-suitable instance)**.** We call an instance $i \in \mathcal{I}$ *benchmark-suitable* if

($\mathcal{B}$.1) it can be solved by at least one considered solver within 4 hours;

($\mathcal{B}$.2) it requires at least 10 seconds with 50 % of the solvers;

($\mathcal{B}$.3) it has a constraint and objective dynamism of at most 10^6 (see Section 3.3);

($\mathcal{B}$.4) the absolute value of each matrix coefficient is smaller than 10^{10};

($\mathcal{B}$.5) the results of all solvers on i are consistent (see Section 3.7);

($\mathcal{B}$.6) it has no indicator constraints (see Section 3.4);

($\mathcal{B}$.7) it is bounded;

($\mathcal{B}$.8) the solution (objective) value of i is smaller than 10^{10};

($\mathcal{B}$.9) it has at most 10^6 nonzero entries.

The subset of benchmark-suitable instances from the ground set $\mathcal{I}^{\text{sub}}$ is denoted by $\mathcal{B}$.

($\mathcal{B}$.2) eliminates instances from the benchmark selection that are too easy. Conversely, ($\mathcal{B}$.1) ensures that benchmark instances can be solved by at least one solver, as already done for MIPLIB 2010. This avoids the situation of MIPLIB 2003, for which four instances still remain unsolved 15 years after the release of the test set. The criteria ($\mathcal{B}$.3), ($\mathcal{B}$.4), ($\mathcal{B}$.5), ($\mathcal{B}$.8) ensure that the benchmark set does not contain numerically difficult instances for which results may be ambiguous. Furthermore, benchmark instances should not contain special constructs that are not supported by all solvers. As noted in Section 3.4, the only special constraint type in the submissions are constraints of indicator type, which are excluded from the benchmark set via ($\mathcal{B}$.6). For two reasons, ($\mathcal{B}$.7) excludes unbounded instances from the benchmark set. First, a feasible, rational MIP is unbounded if and only if its LP relaxation is unbounded, rendering detection of unboundedness more a continuous than a discrete problem. Second, there is currently no clear consensus on the expected behavior and output of MIP solvers. Note that in contrast, infeasible instances are deliberately not excluded. Finally, ($\mathcal{B}$.9) reduces the hardware requirements to perform tests with the benchmark set.

Table 7 lists for each criterion the number of excluded instances. Note that an instance may be excluded for several reasons. In total, 3,407 instances were labeled as

Table 7: Number of instances considered not benchmark-suitable ($\mathcal{I}^{\text{sub}} \setminus \mathcal{B}$), as described in Section 4. The column "Crit." refers to the corresponding criterion in Definition 2. The ground set is the set of 5,666 instances available before preselection.

Crit.	Exclusion reason	Instances
($\mathcal{B}$.1)	Too hard: min. solver time > 4 hours	1,958
($\mathcal{B}$.2)	Too easy: median solver performance $\leq$ 10 seconds	741
($\mathcal{B}$.3)	Objective or constraint dynamism too large	552
($\mathcal{B}$.4)	Absolute matrix coefficients $> 10^{10}$	525
($\mathcal{B}$.6)	Presence of indicator constraints	437
($\mathcal{B}$.5)	Instances excluded for inconsistent results	334
($\mathcal{B}$.7)	Unbounded instances	87
($\mathcal{B}$.8)	Best known solution exceeds 10^{10}	80
($\mathcal{B}$.9)	Too many ($> 10^6$) nonzeros	40

not benchmark-suitable, the majority of them because no solver solved them within the time limit of four hours.

The larger MIPLIB 2017 *collection* covers a broader range of MIP instances. It includes at least one instance from each submitter, a constraint that cannot be enforced for the benchmark set due to runtime considerations. It may contain instances that are considered too easy or too hard for the benchmark set. It may contain instances with more dubious numerics suited for testing the robustness of solvers in general and techniques that explicitly aim at increasing numerical robustness. It may contain unbounded instances and instances with indicator constraints. It may contain up to five instances from each model group.

4.2 Diversity Preselection

As with previous editions of MIPLIB, the number of instances varies significantly between different submissions and, more importantly, also between the model groups described in Section 2.4. While some model groups contain a single MIP instance that represents an optimization problem on a specific data set, other model groups contain hundreds of instances using the same underlying model for different data. Hence, for larger model groups, we preselect a diverse subset of instances as follows.

Let $\mathcal{I} = \mathcal{I}^{\text{sub}}$ denote the index set of submitted instances and let $\mathcal{B} \subseteq \mathcal{I}^{\text{sub}}$ be the subset of benchmark-suitable instances according to Definition 2. The choice of a subset can be naturally encoded using binary variables x_i equal to one if and only if instance $i \in \mathcal{I}$ is selected. For two instances $i, j \in \mathcal{I}$, $d_{i,j}$ denotes the Euclidean distance of their feature vectors. Then for a given model group $\mathcal{G} \subset \mathcal{I}$ of instances and specified κ, we wish to choose κ instances maximally diverse in the sense that the minimum distance between two selected instances becomes maximal. If the model group contains benchmark-suitable instances, at least one of these should be included in the preselection. Such a preselection can be performed by solving the mixed binary optimization problem

$$\begin{aligned}
\max \quad & z && && \text{(1a)}\\
\text{s.t.} \quad & z \leq (d_{i,j} - D)x_i x_j + D && \text{for all } i, j \in \mathcal{G}, i \neq j && \text{(1b)}\\
& \sum_{i \in \mathcal{G}} x_i = \kappa && && \text{(1c)}\\
& \sum_{i \in \mathcal{G} \cap \mathcal{B}} x_i \geq 1 && \text{if } \mathcal{G} \cap \mathcal{B} \neq \emptyset && \text{(1d)}
\end{aligned}$$

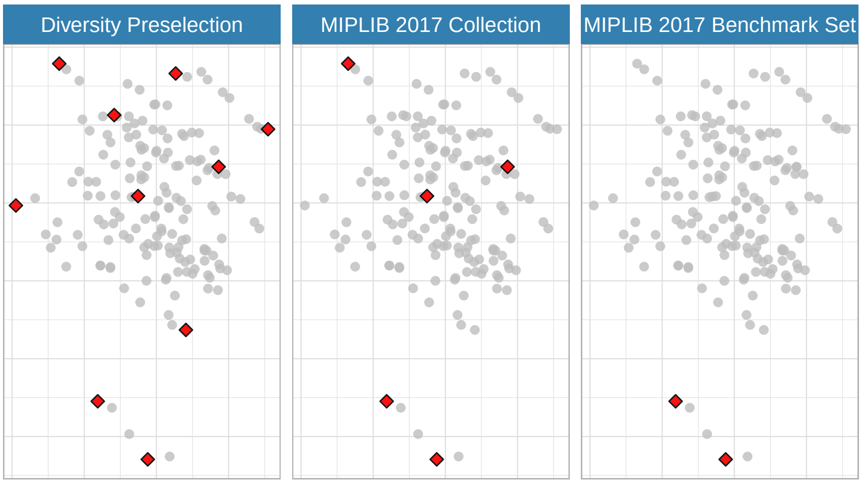


Figure 4: The results of diversity preselection for the model group “drayage”, which contains a total of 165 instances, in a multi-dimensional scaling plot. A red diamond indicates that the instance has been selected for the corresponding set.

$$x \in \{0,1\}^{\mathcal{G}}, z \in [0, D] \tag{1e}$$

An instance $i \in \mathcal{G}$ is preselected if and only if the corresponding binary variable x_i equals one. The value $D := \max\{d_{i,j} : i, j \in \mathcal{G}\}$ acts as big-M in Constraint (1b). In order to solve this optimization problem with a MIP solver, the bilinear product $x_i x_j$ in (1b) must be linearized by replacing it with an auxiliary binary variable $w_{i,j}$ under the additional constraints $w_{i,j} \leq x_i$, $w_{i,j} \leq x_j$, and $w_{i,j} \geq x_i + x_j - 1$ for all pairs $i \neq j \in \mathcal{G}$.

This preselection was performed for each model group exceeding five instances. The value of κ used was $\kappa_{\mathcal{G}} = 5$ for groups with $6 \leq |\mathcal{G}| \leq 10$ and $\kappa_{\mathcal{G}} = 10$ for larger groups. The number of variables and constraints of the preselection model depends on the size of the corresponding model group. For the largest model group “cmflsp”, which comprises 360 instances of a capacitated multi-family lot-sizing problem, the diversity preselection instance has 129,242 rows, 64,981 columns, and 323,485 nonzeros. Except for this largest model group, which required approx. 800 seconds, all preselection problems could be solved to optimality within a time limit of 500 seconds using Gurobi 7.5.1.

As an example, Figure 4 depicts the results of the preselection procedure for the model group “drayage”, which consists of 165 instances in total. The plot shows the instances from this group three times. The x and y-coordinates are computed using *multidimensional scaling* (MDS) [33, 28], which tries to approximately preserve distances between pairs of points in the original, unprojected feature space. The leftmost plot highlights the optimal solution of the corresponding diversity preselection instance. Visually, the selected solution for this group is scattered evenly across the feature space of this group. The middle and right plot show the instances from this group that are selected for the collection and benchmark set, respectively, for which stricter cardinality restrictions on model groups apply. While the five instances in the collection are also spread evenly across the depicted space, the two instances in the benchmark set, namely `drayage-100-23` and `drayage-25-23` are located relatively close together in the picture. However, a look at the performance results, which are taken into account for the selection of the benchmark set, reveal that the two selected instances vary substantially regarding solution time. The easier of the two instances, namely `drayage-100-23` could be solved by seven solvers with a median running time of 40 seconds. The harder instance `drayage-25-23` on the other hand could only be solved by three solvers and hence has a median running time of four hours. The three other instances lie in between. In total, diversity preselection reduces the instance set from 5,666 to 2,182 instances.

The preselected instances form a reduced index set $\mathcal{I}^{\text{pre}}$, which serves as input for the selection of the MIPLIB 2017 collection. Although the following selection procedure could have been applied to the entire set of submissions, we noticed several benefits of preselection empirically. It improves the results of the k-means clustering heuristic in the next Section 4.3, reduces the size and difficulty of the selection MIPs to be solved, and finally leads to a larger collection and benchmark set.

4.3 Preparing Multiple Clusterings

One major challenge in selecting a test set is how to navigate the trade-off of good coverage of all observed instance properties against a balanced selection that avoids overrepresentation. The first goal is certainly achieved best by simply selecting all suitable instances, while balancedness explicitly asks for removing instances from overrepresented problem classes.

A straightforward method would be to compute one clustering according to the entire feature matrix and pick instances uniformly from each cluster. When applied in a high-dimensional feature space, as in our setting, this naïve approach suffers from several problems well-known in data analysis, such as the curse of dimensionality [5] and the relative scaling of numerical features. The first term refers to the fact that with increasing dimensionality of the feature space, the difference of distances of one point to its nearest and to its farthest neighbor becomes smaller in relative terms. Hence, similar instances cannot be identified reliably. Conversely, depending on scaling, the distance with respect to one crucial feature may be dominated by less useful features, such that different instances cannot be distinguished reliably.

We counteract these effects by using multiple clusterings of the entire preselected instance set $\mathcal{I}^{\text{pre}}$ according to disjoint groups of features. Subsequently, we select instances such that they are balanced with respect to each of these clusterings. This selection process is more complex and cannot be achieved by simply picking uniformly from each cluster of each clustering. Instead, we formulate a mixed integer optimization problem with approximate balancing constraints for each of the multiple clusterings.

Formally, for a given index set $\mathcal{I}$ of instances we have K different clusterings, i.e.,

$$\mathcal{I} = \mathcal{C}_{k,1} \cup \cdots \cup \mathcal{C}_{k,L_k} \tag{2}$$

for $k \in \mathcal{K} = \{1, \dots, K\}$, with disjoint $\mathcal{C}_{k,1}, \dots, \mathcal{C}_{k,L_k}$ being a partition of the index set $\mathcal{I}$ for every k. The number of clusters L_k is allowed to vary, since different subsets of features may require a different number of clusters to achieve a high-quality clustering. We denote the index set of all clusters by $\mathfrak{C} := \{(k,\ell) : k = 1, \dots, K, \ell = 1, \dots, L_k\}$. Furthermore, the cluster sizes contain outliers, which need special treatment in order to avoid limiting the size of the resulting test set too much. Hence, we partition the set of clusters into small, medium (regular-sized), and large clusters and denote the respective index sets by $\mathfrak{S}$, $\mathfrak{M}$, and $\mathfrak{L} \subseteq \mathfrak{C}$, as follows. A cluster $\mathcal{C}_{k,l}$ is denoted small if its size is less than half the average size of $\mathcal{C}_{k,1}, \dots, \mathcal{C}_{k,L_k}$. On the other hand, a cluster is treated as large if it is displayed as an outlier in a typical boxplot. Concretely, $\mathcal{C}_{k,l}$ is considered large if its size exceeds the 75 % quantile among $\mathcal{C}_{k,1}, \dots, \mathcal{C}_{k,L_k}$ by more than 1.5 interquartiles.

For the selection of the MIPLIB 2017 collection, we use one clustering for each of the $K = 11$ groups of instance features listed in Table 4. The clusterings of all preselected instances (2,182 instances) are computed using a k-means heuristic [21], which yields a first family of clusterings denoted by $\mathcal{K}_1 = \{1, \dots, 11\}$.

Table 8: Clustering statistics for the collection MIP in Section 4.4.

Feature group	Quality [%]	$\mathbf{L_k}$	Small	Large	$\delta_{\mathbf{k},\mathbf{p}}$ Min.	$\delta_{\mathbf{k},\mathbf{p}}$ Max.
Variable bounds	91.28	23	7	1	6.57	22.03
Matrix coefficients	87.73	41	2	1	5.33	13.95
Matrix nonzeros	89.10	33	3	0	5.86	16.70
Decomposition	99.99	9	1	1	11.54	15.53
Row dynamism	91.70	17	3	3	9.85	18.12
Constraint classification	87.77	35	6	1	6.63	12.92
Objective nonzero density	93.17	9	0	1	9.95	14.01
Objective coefficients	96.51	43	2	3	4.39	23.05
Sides	98.33	35	10	0	1.00	18.77
Size	87.46	9	2	0	10.95	15.01
Variable types	95.26	7	0	1	8.89	13.54

Table 8 gives insight into the result of this clustering process. It shows the total number of clusters (value of L_k) for each feature group clustering. The individual value of L_k has been manually selected for every feature group. The quality column describes the percentage of the total feature group distance between clusters. More formally, for a clustering $k \in \{1, \dots, K\}$ of instances, the *quality* of this clustering is

$$\frac{\sum\limits_{i \neq j \in \mathcal{I}} d_{i,j} - \sum\limits_{\ell=1}^{L_k} \sum\limits_{i \neq j \in \mathcal{C}_{k,\ell}} d_{i,j}}{\sum\limits_{i \neq j \in \mathcal{I}} d_{i,j}}.$$

The quality of a clustering always lies in the interval $[0, 1]$. A clustering has a high quality if long distances between instances are between different clusters. Note that for the distance computation for the quality measure, only features contained in the corresponding feature group were considered. The table shows that the quality of the clustering was at least 87 % and often significantly above 90 %, yielding an average quality of 92 %. In addition to the value of L_k, the table presents the individual number of small and large parts, which are constrained less strictly than the normal parts, see Constraints (4a) and (4b). All parts that are neither small nor large are constrained with both an upper and lower limit on the selection. Finally, the minimum and maximum total dissimilarity per feature group, which are used for the right-hand side of the above constraints, are also shown in Table 8.

For the selection of the benchmark set, we additionally use performance data to handcraft three clusterings for each of the eight participating solvers, which yields additional clusterings $\mathcal{K}_2 = \{12, \dots, 35\}$, see Section 4.5 below.

4.4 Selection of the MIPLIB 2017 Collection

In the following, we describe linear formulations to enforce the requirements specified by the committee. At this stage, the instance set is limited to the instances $\mathcal{I} = \mathcal{I}^{\text{pre}}$ left after the diversity preselection procedure. The set of clusterings $\mathcal{K} = \mathcal{K}_1$ is the one determined using the instance feature groups from Table 4.

To express balancedness, consider one clustering $\mathcal{I} = \mathcal{C}_{k,1} \cup \dots \cup \mathcal{C}_{k,L_k}$. Naïvely, we would like to pick the same number of instances from each cluster, i.e., $\sum_{i \in \mathcal{C}_{k,\ell}} x_i \approx y_k$ for an auxiliary variable $y_k \geq 0$. However, enforcing this for all clusterings is highly restrictive. Furthermore, while the instances in each of the clusters $\mathcal{C}_{k,1}, \dots, \mathcal{C}_{k,L_k}$ should be homogeneous with respect to the features that were used to compute clustering k, they may be heterogeneous with respect to the entire feature vector. This interaction between different clusterings must be taken into account.

Therefore, for each cluster $(k,\ell) \in \mathfrak{C}$ a measure $\delta_{k,\ell} \geq 1$ for the *total dissimilarity* of its instances with respect to the entire feature space is used. Concretely, the vector of pairwise Euclidean distances $\{d_{i,j} : i < j \in \mathcal{C}_{k,\ell}\}$ available after trivial presolving is aggregated into $\delta_{k,\ell}$ using a shifted geometric mean, to which we add 1. The smallest possible value of $\delta_{k,\ell}$ is therefore 1 which only occurs for clusters containing exactly 1 element, or clusters that contain only instances which are indistinguishable in the feature space. Arguably, from clusters with higher total dissimilarity, more instances should be picked, i.e., $\sum_{i\in\mathcal{C}_{k,\ell}} x_i \approx \delta_{k,\ell} y_k$. Introducing a tolerance parameter ϵ, $0 < \epsilon < 1$, we arrive at the balancedness constraints

$$(1-\epsilon)\delta_{k,\ell} y_k \leq \sum_{i\in\mathcal{C}_{k,\ell}} x_i \leq (1+\epsilon)\delta_{k,\ell} y_k. \tag{3}$$

In practice, we discard the left inequality for small clusters and the right inequality for large clusters and use

$$\sum_{i\in\mathcal{C}_{k,\ell}} x_i \geq (1-\epsilon)\delta_{k,\ell} y_k \qquad \text{for all } (k,\ell) \in \mathfrak{C} \setminus \mathfrak{S}, \tag{4a}$$

$$\sum_{i\in\mathcal{C}_{k,\ell}} x_i \leq (1+\epsilon)\delta_{k,\ell} y_k \qquad \text{for all } (k,\ell) \in \mathfrak{C} \setminus \mathfrak{L}. \tag{4b}$$

Additionally, if two instances have identical feature vectors, then at most one of them should be chosen , i.e.,

$$x_i + x_j \leq 1 \qquad \text{for all } i,j \in \mathcal{I}\times\mathcal{I} \text{ with } i<j, d_{i,j}=0. \tag{4c}$$

At most five instances should be selected from each model group. If the model group contains benchmark-suitable instances, at least one of those should be included into the MIPLIB 2017 collection. Let $\mathcal{I} = \mathcal{G}_1 \cup \ldots \cup \mathcal{G}_P$ denote the partition of instances into different model groups, then this condition reads

$$\sum_{i\in\mathcal{G}_p} x_i \leq 5 \qquad \text{for all } p = 1,\ldots,P, \tag{4d}$$

$$\sum_{i\in\mathcal{G}_p\cap\mathcal{B}} x_i \geq 1 \qquad \text{for all } p = 1,\ldots,P \text{ with } \mathcal{G}_p \cap \mathcal{B} \neq \emptyset. \tag{4e}$$

Furthermore, from each submitter at least one instance should be selected, i.e.,

$$\sum_{i\in\mathcal{S}_s} x_i \geq 1 \qquad \text{for all } s = 1,\ldots,S, \tag{4f}$$

where $\mathcal{I} = \mathcal{S}_1 \cup \ldots \cup \mathcal{S}_S$ denotes the partition of instances with respect to S submitters.

Finally, we imposed relative limits on a small number of specific subsets of instances, $\mathcal{R}_r \subset \mathcal{I}$, by requiring

$$\sum_{i\in\mathcal{R}_r} x_i \leq \rho_r \sum_{i\in\mathcal{I}} x_i \qquad \text{for all } r \in \{\text{NEOS, MiniZinc, BP, small, medium}\}. \tag{4g}$$

The concrete values for ρ_r are given in Table 9. The numbers in parentheses show the size of respective ground sets $\mathcal{I}^{\text{pre}}$ and $\mathcal{I}^{\text{col}} \cap \mathcal{B}$, from which instances were selected. The amount of instances from the NEOS server was limited by the committee because of the lack of reliable information on their application and model background. Purely

Table 9: Instance sets for which relative limits on the selection apply, with limits shown for the collection and benchmark MIPs individually. The Size columns show the size of this set within the respective ground set the selection is based on. The parameter ρ_k is a relative limit on the allowed instances from this set in a solution as specified by Constraint (4g).

		Coll. MIP (2182)		**Bench. MIP (499)**	
k	**Sets**	**Size**	ρ_k	**Size**	ρ_k
1	MiniZinc instances	484	0.05	14	0.05
2	NEOS instances	696	0.33	182	1.00
3	small ($n \leq 2k$)	691	0.20	124	0.20
4	medium ($n \leq 10k$)	1,309	0.50	290	0.50
5	binary problems ($n = n_b$)	422	0.20	109	0.20

binary problems were limited because they often represent academic applications such as combinatorial puzzles, but less often occur in industrial, "real-world" instances. The limit ensures that enough actual mixed integer instances are selected for the collection and benchmark sets. For the groups in Table 9 that refer to instance features, the features are always evaluated after trivial presolving.

Subject to those constraints, our objective was to include as many instances as possible, preferring benchmark-suitable instances. Hence, we formulate the collection MIP as the mixed binary optimization problem

$$\max \Big\{ \sum_i \beta_i x_i : \text{(4a)} - \text{(4g)}, x \in \{0,1\}^{\mathcal{I}^{\text{pre}}}, y \in \mathbb{R}^{\mathcal{K}}_{\geq 0} \Big\}, \tag{5}$$

with objective cofficients β to prefer benchmark-suitable instances,

$$\beta_i = \begin{cases} 2, & \text{if } i \in \mathcal{B}, \\ 1, & \text{otherwise.} \end{cases}$$

We solved the collection MIP over the ground set $\mathcal{I}^{\text{pre}}$ of 2,182 instances (after diversity preselection). Despite our efforts to remove obvious duplicates, there remained 48 pairs of instances in $\mathcal{I}^{\text{pre}}$ with a feature distance of zero. From each such pair, at most one instance was selected for $\mathcal{I}^{\text{col}}$ because of Constraint (4c). We obtained the MIPLIB 2017 collection $\mathcal{I}^{\text{col}}$, which comprises 1,065 instances, 499 of which are benchmark-suitable.

4.5 Performance Clusterings

In addition to instance features that depend exclusively on instance data, computational difficulty is an important aspect to consider for the benchmark set. We assessed the computational difficulty of every instance empirically by considering the performance data of the eight tested solvers (see Section 3.6). To quantify performance, we considered a matrix of running times $t_{w,i} > 0$ for each solver $w \in \mathcal{W} := \{1, \ldots, W\}$ and instance $i \in \mathcal{I}$. If w could not solve the instance i, $t_{w,i}$ was set to the time limit of four hours. We denote by $\mathcal{I}_w \subseteq \mathcal{I}$ the set of instances that were solved by w within the time limit.

For each of the participating solvers, we created three different clusterings of the instances to capture different aspects of performance. The base set $\mathcal{I}$ for these clusterings are the 499 benchmark-suitable instances within the MIPLIB 2017 collection, i.e., $\mathcal{I} = \mathcal{I}^{\text{col}} \cap \mathcal{B}$. The overall goal was to avoid a biased selection of instances, i.e., to avoid that the absolute and relative performance of a solver on the benchmark set appears different than on $\mathcal{I}^{\text{col}}$. Each of the three clusterings avoids a different bias.

Absolute performance clustering. The first clustering uses an absolute performance ranking. For each solver w, we sorted the instances in $\mathcal{I}_w$ according to increasing running time $t_{w,i}$. For a fixed number of clusters B, which we set to $B = 11$, we grouped the instances in $\mathcal{I}_w$ into B equally-sized clusters w.r.t. increasing rank, i.e., we assigned the instances solved fastest to the first cluster, the next fastest set of instances to the second cluster, ..., and the slowest instances to the last cluster, so that each cluster contained approximately $|\mathcal{I}_w|/B$ instances.

The instances in $\mathcal{I} \setminus \mathcal{I}_w$ that could not be solved by solver w seem indistinguishable with respect to performance. However, it could be that the solution process was terminated only seconds prior to concluding the proof of optimality, or that the solution process would have continued unfinished for days or even months. We took this into account by inspecting the performance of the other solvers and formed two more clusters from those instances: instances that could be solved by exactly one solver and instances that could be solved by more than one solver. The case that instances could be solved by no other solver does not appear since such instances are not benchmark-suitable (Definition 2, Criterion ($\mathcal{B}$.1)).

Relative performance clustering. In contrast to the absolute performance clustering, the instances were ranked based on relative solver performance for a second clustering as follows. To this end, we defined the *relative performance* of solver w on instance i with respect to the other solvers as

$$t^{\mathrm{rel}}_{w,i} := \frac{t_{w,i} + \sigma}{\min\limits_{w' \neq w} t_{w',i} + \sigma}, \tag{6}$$

where $\sigma \in \mathbb{R}_{\geq 0}$ is a nonnegative shift as in the computation of shifted geometric means. Relative performance locates the individual solver performance relative to all other solvers on an instance, regardless of the absolute scale. The motivation is that solvers and solver improvements are traditionally measured by the shifted geometric mean instead of the arithmetic mean. For the instances that could be solved by this solver, we used this ranking to define B equally-sized clusters in the same fashion as with the absolute performance ranking. The timeout instances were again divided into two further clusters of instances that could be solved by exactly one and by more than one other solver, respectively.

Binned absolute performance clustering. The third clustering uses absolute solving time directly, partitioning possible solving times into $B' = 7$ intervals

$$[T_0 = 0, T_1), [T_1, T_2), \ldots, [T_{B'-1}, T_{B'}), \tag{7}$$

whose breakpoints are equal for all solvers. The concrete bin width used grows exponentially as follows.

$$T_j = 10^{-3.5+0.5j} \cdot 14400, \quad j = 1, \ldots, 7.$$

Hence, the righmost bin T_7 has the time limit of four hours as right breakpoint. Then for each solver $w \in \mathcal{W}$ we formed B' clusters $\{i \in \mathcal{I}_w : t_{w,i} \in [T_{j-1}, T_j)\}$, $j = 1, \ldots, B'$. Empty clusters were discarded. This is different from the absolute and relative performance clusterings in that it partitions the instances solved by a solver into clusters that differ in size. The instances in $\mathcal{I} \setminus \mathcal{I}_w$, which could not be solved by solver w, were again treated as two further clusters as above.

All in all, this led to 24 clusterings, $\mathcal{K}_2 = \{12, \ldots, 35\}$. The ranking-based clusterings yield approximately equal cluster sizes over $\mathcal{I}_w$, but these cluster sizes may differ to the ones on $\mathcal{I} \setminus \mathcal{I}_w$. The binned absolute performance clustering does not control cluster size and may yield very unequally sized clusters. In the following benchmark MIP we picked from the performance clusters according to their size, i.e., we use $\delta_{k,\ell} = |\mathcal{C}_{k,\ell}|$ for all $k \in \mathcal{K}_2$ in Constraint (4a) and (4b).

In the case of SCIP, as an example of an absolute performance clustering, each of the 11 parts contained between 20 and 22 instances, and the remaining 263 instances were split into 58 instances which could only be solved by one solver, and 205 instances solved by at least two solvers. Although the absolute and relative performance clusters were, by design, almost equal in size, we observed quite different partitions of the ground set. An example is the fastest relative performance cluster for SCIP, which shares 8 of its 21 instances with the fastest absolute cluster. The remaining 13 instances, for which SCIP was particularly fast compared to its competitors, were spread across 8 of 10 possible absolute clusters. This shows that to some extent, the two suggested clusterings exhibit an almost orthogonal instance partition.

4.6 Selecting the MIPLIB 2017 Benchmark Set

The benchmark set was selected from the ground set of benchmark-suitable instances in the MIPLIB 2017 collection, $\mathcal{I} = \mathcal{I}^{\mathrm{col}} \cap \mathcal{B}$. The balancedness constraints (4a) and (4b) are now defined using instance feature and performance clusterings, $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2$. The rationale is that the current performance of solvers in the collection should be reflected by the performance on the benchmark set in order to avoid any unintentional bias towards a solver during the selection of instances. The relative limit constraints (4g) are kept, but the restriction on instances from the same model group is reduced to one instance from NEOS groups and two instances, otherwise. The stricter limit on NEOS instances is imposed because little information is available for these anonymously submitted instances and the chance for duplicate instances in the same model group is deemed higher.

In addition, executing one benchmark run on this test set should be possible within a reasonable time frame on modern hardware, possibly a compute cluster. We specified a total time limit τ of 32 days for running the benchmark set with a hypothetical solver with median running times capped to a time limit of two hours, i.e., $\bar{t}_i := \min\{\mathrm{median}\{t_{w,i} : w \in \mathcal{W}\}, 7200\}$. The resulting *benchmark MIP* reads

$$
\begin{aligned}
\max\ & \sum_i \beta_i x_i && \text{(8a)}\\
\text{s.t.}\ & \text{(4a), (4b), (4g)}, &&\\
& \sum_{i \in \mathcal{G}_p} x_i \le \begin{cases} 1 & \text{for all } p = 1,\dots,P \text{ from NEOS},\\ 2 & \text{otherwise},\end{cases} && \text{(8b)}\\
& \sum_{i \in \mathcal{I}} \bar{t}_i x_i \le \tau, && \text{(8c)}\\
& x \in \{0,1\}^{\mathcal{I}}, && \text{(8d)}\\
& y \in \mathbb{R}^{\mathcal{K}}_{\ge 0}. && \text{(8e)}
\end{aligned}
$$

This approach also has a potential drawback. Representing current solver performance may overly favor instances that are tractable by current solver technology. Opposed to this, one main goal of the MIPLIB project is to provide a test bed that drives solver development forward. Hence, we used the objective coefficient

$$\beta_i := 1 + \frac{1}{20}\sqrt{\min_{w \in \mathcal{W}} t_{w,i}} \tag{9}$$

for instance $i \in \mathcal{I}$. This favors instances that are challenging even for the virtual best solver. The concrete choice of the square root was empirically motivated.

Using $\mathcal{I}^{\text{col}} \cap \mathcal{B}$ containing 499 instances as ground set, we solved the benchmark MIP that respects all feature group and performance clusterings. The solution to the benchmark MIP contained 240 instances and now constitutes the MIPLIB 2017 benchmark set $\mathcal{I}^{\text{bench}}$. We note that the imposed running time constraint (8c) was not tight on our performance data. A solver with median running times and a time limit of two hours would take about 14 days to process all benchmark instances sequentially.

We also note that for several reasons, the goal of representing computational difficulty in the reduced benchmark set cannot be achieved perfectly and is not even well-defined: The performance data is only a snapshot of current algorithms. It was gathered using a time limit, performance variability and parallel scalability were not captured, and correctness was only enforced approximately with respect to the tolerance parameter ϵ. Last but not least, the different performance clusterings may even contradict each other. However, we hope that it helps to avoid unintentionally and unconsciously introducing a bias both towards instances of a particular difficulty and towards any of the solvers.

5 The Final Collection and Benchmark Sets

The MIPLIB 2017 collection $\mathcal{I}^{\text{col}}$ has been compiled with a focus on a balanced and diverse representation/coverage of the feature space. The benchmark set $\mathcal{I}^{\text{bench}}$ incorporates similar requirements also for the performance data. This section discusses the feature and performance aspects of the compiled sets. We also assess the descriptive power of the feature space by (re-)detecting known model group associations.

5.1 Representation in Feature Space

A frequent question during the discussions about the MIPLIB 2017 compilation process was whether the choice of features and their scaling is able to distinguish instances in a useful way. Ideally, two instances based on the same model for the same application, but with different data, should be close to each other in the feature space, regardless of variations of, e.g., the size of the matrix. For MIPLIB 2017, we have two sources to assess similarity between instances, namely their distances in feature space and the model groups $\mathcal{G}$ from Section 2.4. In this paragraph, we evaluate the descriptive power of the feature space by comparing similarity in feature space and model group association of instances.

Let X denote the instances in the MIPLIB 2017 collection ($|X| = 1065$). The complete graph $K_X = (X, E)$ on the vertex set X has $|E| = \binom{|X|}{2} = 566580$ edges. Now consider two subsets of the edges. Let $E_{\mathcal{G}}$ denote edges between instances from the same model group. Furthermore, let for every $x \in X$, $S_x \subset X \setminus \{x\}$ denote the set of five most similar instances to x in the collection (most similar w.r.t. the distance in the scaled feature space). With the sets S_x, we define $E_{\mathcal{S}}$ to be the set of *similarity edges* as

$$E_{\mathcal{S}} := \{(x, y) \in E : \; x \in S_y \text{ or } y \in S_x\}.$$

Note that $x \in S_y$ does not necessarily imply the opposite containment $y \in S_x$, but holds in many cases. The actual cardinalities of the two sets are $|E_{\mathcal{G}}| = 1327$ and $|E_{\mathcal{S}}| = 3747$ and hence comprise less than 1 % of the total edge set. Indeed, computing the probability for an edge e that was selected uniformly at random from E to be a similarity edge is

$$\mathbb{P}(e \in E_{\mathcal{S}}) = \frac{|E_{\mathcal{S}}|}{|E|} \approx 0.007$$

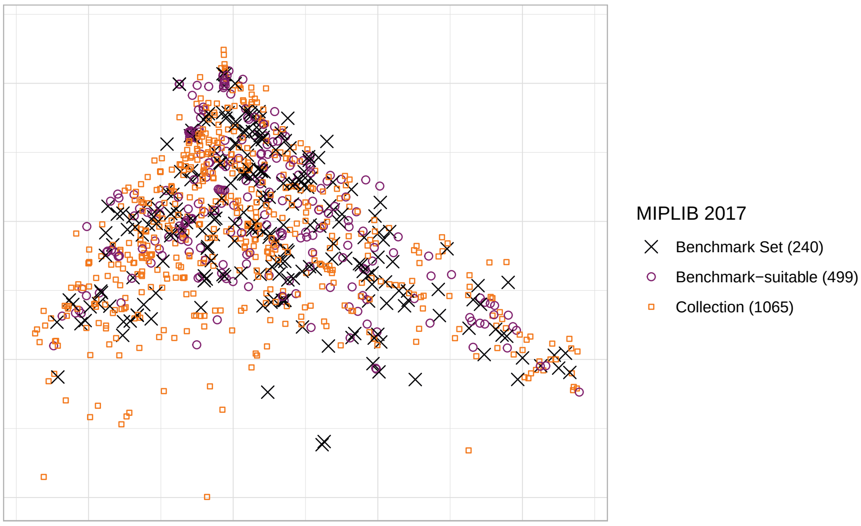


Figure 5: Location of instances in the feature space using multidimensional scaling.

Now what is the probability that a group edge is also a similarity edge? This question can be answered by computing the conditional probability

$$\mathbb{P}(e \in E_{\mathcal{S}} \,|\, e \in E_{\mathcal{G}}) = \frac{\mathbb{P}(e \in E_{\mathcal{S}} \cap E_{\mathcal{G}})}{\mathbb{P}(e \in E_{\mathcal{G}})} = \frac{|E_{\mathcal{S}} \cap E_{\mathcal{G}}|}{|E_{\mathcal{G}}|} = \frac{974}{1327} \approx 0.734.$$

The majority of group edges is contained in the similarity set, and the probability for a group edge to be a similarity edge is more than 100 times higher than for a randomly selected edge.

Recall from Section 2.4 that the model groups have been partially derived from the feature data. For submissions from the NEOS server, which have an unknown origin, clustering has been used to group similar NEOS instances into pseudogroups. If we omit all NEOS instances from the above computations (by considering the complete graph $K_{X \setminus X_{\text{NEOS}}}$ with 714 vertices), the probability for an edge to be a similarity edge is about the same, $\mathbb{P}(e \in E_{\mathcal{S}}) \approx 0.008$, whereas the conditional probability of a group edge to be a similarity edge is ≈ 0.815 and hence even higher than for K_X.

From this observation, we conclude that the feature space has been designed sufficiently well for the clustering approach. In fact, the used feature space recovers the model group data better than we expected. Even for an instance that does not belong to a dedicated model group or lacks bibliographical information, the similarity to other model groups can yield interesting hints at the type and application of this instance. Therefore, the web page of MIPLIB 2017 (see also Section 5.3) allows to browse the five most similar instances for every instance of the MIPLIB 2017 collection.

Figure 5 uses MDS [33, 28] to give a spatial impression of the locations of the MIPLIB 2017 benchmark set, the benchmark-suitable instances, and the collection, relative to each other in feature space. The distance computation is based on the feature vectors after they have been scaled over the entire set of submissions. Note that there is a subset relation for those sets, i.e.,

$$\text{Benchmark Set} \subsetneq \text{Benchmark-suitable} \subsetneq \text{Collection},$$

such that the plot only shows the innermost set membership for every instance. 27 collection instances lie outside of the plotted region to improve visibility, 5 of which are in the benchmark set, and 8 of which are benchmark-suitable. Figure 5 mainly

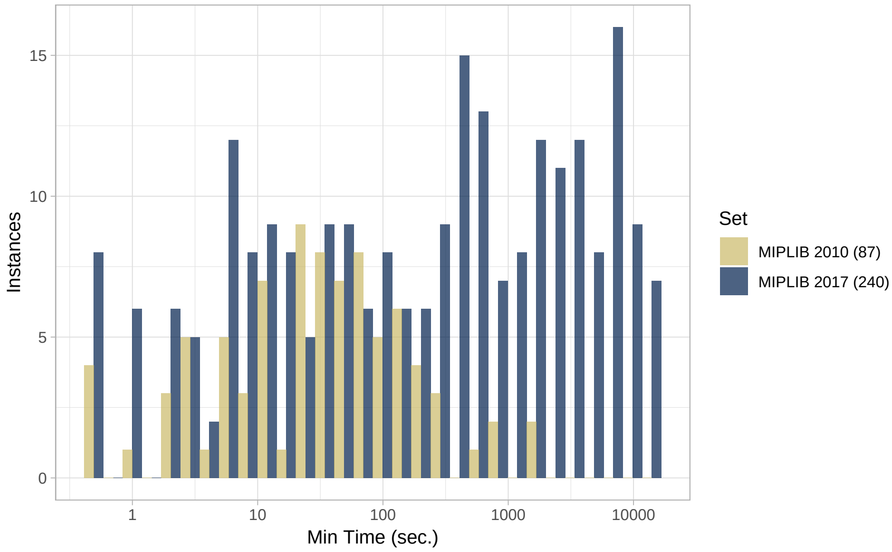


Figure 6: Minimum solving time in seconds of the benchmark sets of MIPLIB 2010 and MIPLIB 2017.

shows that benchmark-suitable instances cover the majority of the feature space that is comprised by the collection.

5.2 Solver Performance

One of the goals of MIPLIB has always been to provide a representative set to measure and compare solver performance. In this section, we analyze the solver performance on the collection, and in particular on the new benchmark set of 240 instances. For this analysis, we use the computational results obtained during 4-hour runs conducted for the selection process. Note, however, that in this article, solver performance is not reported directly for several reasons. One reason is that due to hardware restrictions, not all runs could be performed exclusively on the same hardware, and should hence not be reported in a way that could be confused for an actual benchmark. Again, the individual results are aggregated into the virtual best solver, i.e., a solver that always finishes as fast as the fastest actual solver for each individual problem instance.

In Figure 6, we compare the performance of this virtual best solver on the benchmark sets of MIPLIB 2010 and 2017. One of the motivations for the creation of MIPLIB 2017 was the demand for a harder benchmark set. As a consequence of Definition 2, the virtual best solver solves all instances within 4 hours (or 14400 seconds) as this is one of the criteria for benchmark suitability. The plot shows that the majority of the old benchmark set can be solved by the virtual best solver in less than 100 seconds, and that there is no instance left where the virtual best solver requires one hour or more. In contrast, the benchmark set of MIPLIB 2017 is much more demanding, as a significant portion of instances lies at the right end of the scale. Due to its increased size, the bars for the MIPLIB 2017 benchmark set lie almost consistently above the ones for the previous set. The MIPLIB 2017 benchmark set covers much more of the relevant performance scale than its predecessor covers nowadays. Note that the MIPLIB 2010 benchmark set appearing easy is an impressive result of 7 years of continuous solver improvements.

Table 10 shows the percentage of instances of the respective benchmark set (2010 or 2017) for which the virtual best solver takes longer than a certain time threshold, which we vary between 1 and 4 hours. As mentioned before, all instances of the 2010

	virtual median		virtual best	
	2010	2017	2010	2017
> 1 h	17.2 %	66.7 %	0 %	19.6 %
> 2 h	12.6 %	61.7 %	0 %	8.8 %
> 3 h	6.9 %	51.2 %	0 %	4.2 %
> 4 h	5.7 %	48.3 %	0 %	0.0 %

Table 10: Percentage of benchmark instances that could not be solved within 1–4 hours by the virtual best and median solvers. The percentages are relative to the individual benchmark sets (2010: 87 instances, 2017: 240 instances).

benchmark set could be solved within one hour by the virtual best solver. The table shows that among the new benchmark set, almost 20 % of the instances cannot be solved within one hour by the virtual best solver. Along with the virtual best solver, Table 10 also shows the performance of the virtual median solver, based on the median solution time over the eight involved solvers. Since the number of involved solvers is even, the median is computed by averaging the timing result of the two solvers ranking 4th and 5th for each instance individually. Therefore, the virtual median solver is faster than half of the solvers. On the MIPLIB 2010 benchmark set, 17.2 % of the instances are not solved within one hour by the virtual median solver. In contrast to the virtual best solver, the virtual median solver still times out after 4 hours on 5.7 % (5 of 87) instances even on the 2010 benchmark set. On the MIPLIB 2017 benchmark set, the virtual median solver needs more than one hour on two thirds of the instances, and still needs more than four hours of solving time on almost 50 % of the instances.

In Figure 7, the fraction of instances solved by the virtual median solver as a function of time is shown. The figure shows the corresponding curves for the MIPLIB 2017 collection of 1,065 instances, the set $\mathcal{I}^{\text{col}} \cap \mathcal{B}$ of 499 benchmark-suitable instances, and the MIPLIB 2017 benchmark set (240 instances). Recall that an instance is only a candidate for the benchmark set if it takes the virtual median solver at least 10 seconds to solve it, which is also visible from the plot. As minimum for any time measurement, 0.5 seconds were used, which is visible in the curve of the MIPLIB 2017 collection. The objective function for the benchmark set was designed to prefer harder instances. The effect of this design choice is visible in the figure, in which the percentage of solved instances of the benchmark set consistently lies below the curve for the 499 benchmark-suitable instances. The slope of the curve for the collection is approximately linear for about 90 % of the visible area. Due to the logarithmic scaling of the horizontal (time) axis, this suggests that to a certain extent, the solving behavior of the virtual median solver can be approximated by fitting a logarithmic function. Note that the clear change in behavior of the curves, which are otherwise almost logarithmic, towards the right end of the scale is an artifact from the median computation, which weighs in as soon as the 4th solver could still solve the instance, but the 5th solver couldn't. Their timings can even be very different. On the instance `blp-ic98`, the four best performing solvers finish within 1,700 seconds, while the fifth solver times out after four hours. The median solver therefore has a performance of 8,050 seconds. All individual curves of the actual solvers tested have a similar, almost logarithmic shape without the median artifact.

Statistically speaking, the curves in Figure 7 describe empirical cumulative density functions (CDF) of the random variable that represents median solving time for an instance. The Kolmogorov-Smirnov (KS) test is a statistical approach to measuring the similarity between a pair of CDF F_1, F_2. To this end, the KS test measures the maximum vertical distance[6] D between F_1 and F_2. With increasing D, the likelihood

[6]This distance D is the supremum norm $\sup_{x \in \mathbb{R}} |F_1(x) - F_2(x)|$.

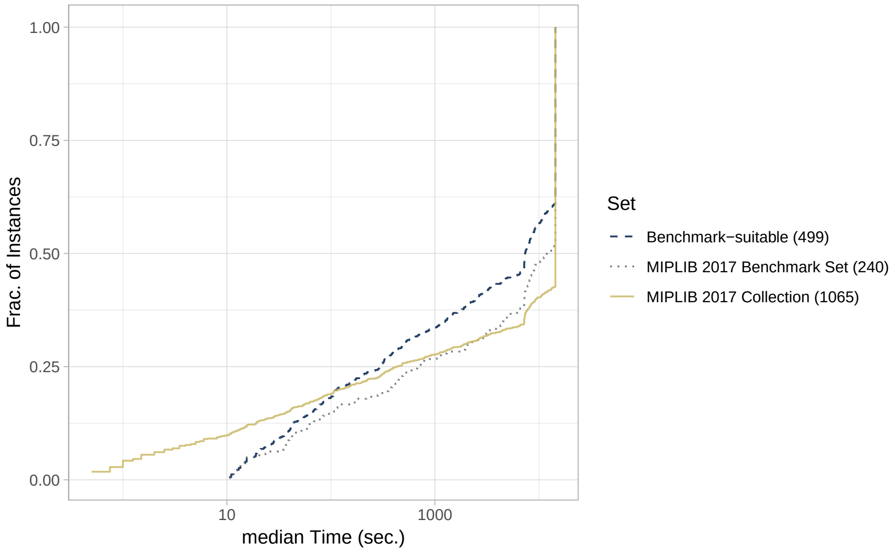


Figure 7: Virtual median solver performance on subsets of the MIPLIB 2017 collection.

decreases that F_1 and F_2 represent samples from the same distribution. In order to further quantify the hardness of the benchmark set, a KS test has been performed for every solver, comparing its individual CDF pair on the benchmark set and the set of benchmark-suitable instances. As alternative serves the hypothesis that the CDF of the benchmark set lies below the CDF of the larger set.

For the performance of the virtual median solver as depicted in Figure 7, the distance D is approximately 0.10, which results in a p-value of 0.04. A much smaller p-value of $6.513 \cdot 10^{-5}$ is obtained for the virtual best solver at a distance of $D = 0.17$. For four of the actual solvers, the maximum D is larger than 0.1, and the accompanying p-value is smaller than 1 %. This is significant evidence that for those solvers, the benchmark set has been selected as a particularly hard selection among all suitable instances. For the other four solvers, the value of D is smaller, which in turn results in larger p-values (each greater than 0.1). Note that even here, the curves of the CDF on the benchmark set tends to undercut the CDF of the set of suitable instances, but this discrepancy is not large enough to render the KS test significant. Hence, for those four solvers, the performance curve is more or less representative of the CDF over all suitable instances. The results show that the selection methodology has achieved both its conflicting goals, hardness and representability, with respect to the solver performance, equally well.

5.3 The MIPLIB 2017 Web Page

For the release of MIPLIB 2017, the web page `miplib.zib.de` has been written from scratch and received a modern design. The main page shows the current status of instances regarding the categories easy, hard, and open. The two main tables list the instances of the MIPLIB 2017 collection and benchmark set together with some key properties, their model group, and their optimal or best known objective value. All tables use tags on the instances to highlight certain properties that may be interesting for researchers, such as pure feasibility instances with no objective function, instances for which good decompositions are known, instances with critical numerics, the presence of indicator constraints, etc. It is possible to search and filter for instances by name, status, model group, or tag, or to sort the table by column.

The individual instance pages offer a short description of this instance and bibliographical information. Also, more information on the constraint mix for this instance before and after presolving is displayed, as well as decomposition information, if available. Finally, the optimal or best known solutions for every instance are displayed, as well as the five most similar instances as explained in Section 5.1. The web page also offers additional downloadable content, including

– the MIPLIB 2017 collection and benchmark sets,
– lists of instances with certain tags,
– available solutions,
– optimal/best known objective values for all instances,
– the collection and benchmark MIPs,
– the feature extractor, and
– bash scripts to run and validate solver performance experiments.

Some of the tags may change over time, such as the instances that fall into the categories easy, medium, and hard. Also, best known objective values are naturally changing or eventually proven to be optimal. Therefore, we provide versioned files for accurate referencing. The versioned files are periodically updated. All downloadable solutions are checked for feasibility with the solution checker from Section 3.7. Also, their exact solution values after fixing all integer variables are computed using SoPlex with iterative refinement. The actual collection and benchmark MIPs are also available for download. Obviously, these instances cannot be part of the actual collection and benchmark sets, respectively, since their presence would alter the feature space and hence their own formulation. Contributions in terms of updated bibliographic information or corrections to the instance descriptions are very welcome. In particular, we are constantly accepting and checking improving solutions to the open instances of the MIPLIB 2017 collection. In contrast to previous MIPLIB's, not only new optimal, but any improving solution will be considered for the periodic update of the page data. Improving solutions should be sent to the maintainers of the page, together with a description of how they have been obtained. Note that every submitted solution must adhere to the format accepted by the MIPLIB solution checker (Section 3.7), which is also available on the web page.

6 Conclusion

The sixth version of MIPLIB has, as was the case in previous updates, significantly increased in size compared to its predecessors. The distinction between a dedicated benchmark set and the entire collection, which was introduced with MIPLIB 2010, has been preserved. These sets now contain 240 and 1,065 instances, respectively. The process of how to reduce the initial submission pool of over 5000 instances to a balanced selection of this size, however, has been completely redesigned. Beyond the new MIPLIB 2017 itself, the development of this fully data-driven and optimization-based methodology is the main contribution of this paper.

We propose two related MIP models that have successfully provided decision support for the selection process to the MIPLIB committee. One key ingredient of this approach is the definition of a feature space covering more than a hundred different dimensions for characterizing a MIP instance. In order to ensure a balanced selection with respect to these features and, for the benchmark set, with respect to performance data, we advocate the use of multiple instance clusterings over partitions of the feature vector. A comparison with manually assigned model groups available from meta data of the submissions shows the high descriptive power of the used feature space. By approaching the selection problem as a MIP that encodes a balanced, simultaneous selection from each such clustering, the collection and benchmark MIPs provide the flexibility to incorporate both feature coverage and performance requirements as well as other restrictions from the comittee. Besides improving the final outcome, this formalization of the selection criteria has served to increase the transparency of the selection process.

While the selection methodology proposed here is not intended as a general blueprint for contruction of test sets, we hope that parts of the process of constructing the MIPLIB instance sets may apply to the curation of other test sets in the future. Certainly, many variations and adjustments of the approach are possible. Not only the chosen constants or heuristic clustering methods can be adapted, but also the role of objective function and constraints may be redefined. Furthermore, the interplay between the main selection models and the diversity preselection offers potential for variation. For example, a different approach may directly select and fix a number of instances with maximum diversity from each large model group for the collection and afterwards complete the collection with instances from smaller groups and instances with no known model group association. In light of these degrees of freedom and many ad hoc decisions that had to be made in advance, the final result is clearly only one of many possible and justified outcomes. However, we believe that the collection and benchmark sets presented in this paper are a profound attempt to provide the research community with test sets that represent the versatility of MIP.

One of the main characteristics of the benchmark set is that it provides a snapshot of current solver performance. Our hope is that the performance of solvers on this benchmark set will serve as a sort of bellwether for progress in the field of mixed integer optimization as a whole in the coming years. As future work, we propose to assess the performance representability of the benchmark set from hindsight, i.e., by comparing speed-ups for entire model groups as well as for the selected instances. Such data will finally allow to better compare different (pre-)selection models such as, e.g., the one presented here that favors diversity to a different one that selects nearest neighbors and maximizes representability.

Another benefit of our MIP-based selection process is the fact that the MIP models can be used to approach further questions beyond the initial creation of the test set. One example is the following case, in which it occurred that benchmark instances needed to be replaced over time as new computational data became available. Despite all the care that was taken to exclude numerically critical instances from the benchmark set, problematic numerical properties of the two instances `neos-5075914-elvire` and `neos-3754224-navua` remained undetected during the selection process.[7] A variant of the benchmark selection MIP was employed in order to compute a minimal update of the benchmark set that exchanges the discussed instances while preserving the balancedness requirements. An accordingly updated version of the benchmark set was published in June 2019.

Finally, by the time of this writing, the challenges of MIPLIB 2017 collection have already attracted a wide audience. In fact, we have received many new solutions to previously open instances. While some of those optimality or infeasibility proofs have been obtained by the use of massively parallel codes such as the Ubiquity Generator framework [36, 37], other instances inspired the development of customized cutting plane approaches, or even a rigid mathematical proof of infeasibility without any code in the case of the instance `fhnw-sq3`. In total, 30 originally open instances have already been solved.[8] We are looking forward to further contributions and many more years (and

[7] Both instances are at the border between feasibility and infeasibility, but at the time of collecting solution data no inconsistencies could be observed. For the first instance, two solvers agree on the optimal solution value although the instance should mathematically be infeasible. The second instance has only been declared infeasible by one solver during the selectiong process; we received a solution that is feasible within tolerances half a year after the original publication of the benchmark set.

[8] Compare the downloadable files `open-v1.test` and `open-v7.test`

versions) of MIPLIB to come.

Acknowledgments

The authors wish to thank all members of the MIPLIB 2017 committee, the creators of the previous editions of MIPLIB, all submitters to previous MIPLIB collections, and most importantly all contributors of new instances for their effort: Alexandra M. Newman, Andrea Arias, Andreas Bärmann, Andrew Stamps, Antonio Frangioni, Austin Buchanan, Balabhaskar Balasundaram, Berk Ustun, Cezar Augusto Nascimento e Silva, Christian Liebchen, Christopher Daniel Richards, Christopher Hojny, Dan Hiroshige, Koichi Fujii, Dan Neiman, Daniel Bienstock, Daniel Heinlein, Daniel Rehfeldt, Dimitri Papageorgiou, Domenico Salvagnin, Felix Cebulla, Felix J.L. Willamowski, Gavin Goodall, George Fonseca, Gerald Gamrath, Gerald Lach, Gleb Belov, Hans Mittelmann, Haroldo Gambini Santos, Hsiang-Yun Wu, Irv Lustig, Janos Hoener, Jeff Linderoth, Jesus Rodriguez, Jonathan Eckstein, Jordi Castro, Joshua Friedman, Juan Javier Dominguez Moreno, Koichi Fujii, Laurent Sorber, Manuel Iori, Marc Pfetsch, Mark Husted, Matias Soerensen, Michael Bastubbe, Michael Winkler, Paula Carroll, Pelin Damci-Kurt, Philipp Leise, Pierre Le Bodic, Qie He, Rob Pratt, Salim Haddadi, Sascha Kurz, Sean MacDermant, Shunji Umetani, Simon Bowly, Simon Felix, Siwei Sun, Stephan Beyer, Sujayandra Vaddagiri, Tamas Terlaky, Timo Berthold, Toni Sorrell, Utz-Uwe Haus, and Yoshihiro Kanno.

The work for this article, in particular by the first, second, third, ninth, and last author, has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM). All responsibility is assumed by the authors. The work of Hans D. Mittelmann was supported in part by Air Force Office of Scientific Research under grants FA9550-15-1-0351 and FA9550-19-1-0070.

References

[1] MIPLIB 2.0. URL http://miplib2010.zib.de/miplib2/miplib2.html.

[2] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, July 2007.

[3] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34 (4):361–372, 2006. doi: 10.1016/j.orl.2005.07.009.

[4] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. ZIB-Report 16-44, Zuse Institute Berlin, 2016.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In C. Beeri and P. Buneman, editors, *Database Theory — ICDT'99*, pages 217–235. Springer Berlin Heidelberg, 1999. doi: 10.1007/3-540-49257-7_15.

[6] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002. doi: 10.1287/opre.50.1.3.17780.

[7] R. E. Bixby, E. A. Boyd, and R. R. Indovina. MIPLIB: A test set of mixed integer programming problems. *SIAM News*, 25:16, 1992.

[8] R. E. Bixby, S. Ceria, C. McZeal, and M. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.

[9] CBC. COIN-OR branch-and-cut MIP solver, 2019. URL https://github.com/coin-or/Cbc.

[10] CPLEX. IBM ILOG CPLEX Optimizer, 2019. URL http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.

[11] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

[12] M. Galati, T. Ralphs, and J. Wang. Computational Experience with Generic Decomposition using the DIP Framework. In *Proceedings of RAMP 2012*, 2012. URL http://coral.ie.lehigh.edu/~ted/files/papers/RAMP12.pdf.

[13] M. Galati, T. Ralphs, and J. Wang. DIP/DipPy version 0.92. 2019. doi: 10.5281/zenodo.2656800. URL http://github.com/coin-or/Dip.

[14] M. V. Galati. *Decomposition in Integer Programming*. Phd, Lehigh University, 2009. URL http://coral.ie.lehigh.edu/{~}ted/files/papers/MatthewGalatiDissertation09.pdf.

[15] G. Gamrath and M. Lübbecke. Experiments with a generic Dantzig-Wolfe decomposition for integer programs. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lect. Notes in Comp. Sci.*, pages 239–252, Berlin, 2010. Springer-Verlag. doi: 10.1007/978-3-642-13193-6_21.

[16] G. Gamrath, T. Koch, A. Martin, M. Miltenberger, and D. Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, pages 1–32, 2015. doi: 10.1007/s12532-015-0083-5. URL http://dx.doi.org/10.1007/s12532-015-0083-5.

[17] A. Georges, A. Gleixner, G. Gojic, R. L. Gottwald, D. Haley, G. Hendel, and B. Matejczyk. Feature-based algorithm selection for mixed integer programming. ZIB-Report 18-17, Zuse Institute Berlin, 2018.

[18] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin, 2018.

[19] T. Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.2 edition, 2016. URL http://gmplib.org/.

[20] Gurobi. GUROBI Optimizer, 2019. URL http://www.gurobi.com/products/gurobi-optimizer/gurobi-overview.

[21] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979. doi: 10.2307/2346830.

[22] G. Hendel. IPET interactive performance evaluation tools. URL https://github.com/GregorCH/ipet.

[23] A. Hoffman, M. Mannos, D. Sokolowsky, and N. Wiegmann. Computational experience in solving linear programs. *Journal of the Society for Industrial and Applied Mathematics*, 1 (1):17–33, 1953.

[24] *Mathematical Programming System/360 Version 2, Linear and Separable Programming – User's Manual*. IBM Corporation, White Plains, N.Y., 3 edition, October 1969. Publication H20–0476–2.

[25] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3: 103–163, 2011. doi: 10.1007/s12532-011-0025-9.

[26] R. Laundy, M. Perregaard, G. Tavares, H. Tipi, and A. Vazacopoulos. Solving hard mixed integer programming problems with Xpress-MP: A MIPLIB 2003 case study. *INFORMS Journal on Computing*, 21:304–319, 2009.

[27] A. Lodi. MIP computation. In M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer, 2009.

[28] K. Mardia. Some properties of classical multi-dimensional scaling. *Communications in Statistics - Theory and Methods*, 7(13):1233–1241, 1978. doi: 10.1080/03610927808827707.

[29] MathWorks. MATLAB Optimization Toolbox, 2019. URL https://de.mathworks.com/products/optimization.html.

[30] H. Mittelmann. Benchmarks for optimization software. URL http://plato.asu.edu/bench.html.

[31] MOSEK ApS. MOSEK, 2019. URL https://www.mosek.com/.

[32] J. L. Nazareth. *Computer Solutions of Linear Programs*. Monographs on numerical analysis. Oxford University Press, 1987.

[33] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/.

[34] SAS. SAS/OR, 2019. URL https://www.sas.com/en_us/software/or.html.

[35] SCIP. Solving Constraint Integer Programs, 2019. URL http://scip.zib.de/.

[36] Y. Shinano. The Ubiquity Generator framework: 7 years of progress in parallelizing branch-and-bound. In N. Kliewer, J. F. Ehmke, and R. Borndörfer, editors, *Operations Research Proceedings 2017*, pages 143–149, Cham, 2018. Springer International Publishing. doi: 10.1007/978-3-319-89920-6_20.

[37] Y. Shinano, T. Achterberg, T. Berthold, S. Heinz, T. Koch, and M. Winkler. Solving Open MIP Instances with ParaSCIP on Supercomputers using up to 80,000 Cores. In *Proc. of 30th IEEE International Parallel & Distributed Processing Symposium*, 2016. doi: 10.1109/IPDPS.2016.56.

[38] Xpress. FICO Xpress-Optimizer, 2019. URL http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx.

Author Affiliations

Ambros Gleixner
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
E-mail: gleixner@zib.de
ORCID: 0000-0003-0391-5903

Gregor Hendel
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
E-mail: hendel@zib.de
ORCID: 0000-0001-7132-5142

Gerald Gamrath
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
E-mail: gamrath@zib.de
ORCID: 0000-0001-6141-5937

Tobias Achterberg
Gurobi Optimization GmbH, Foellerweg 37, 61352 Bad Homberg, Germany
E-mail: achterberg@gurobi.com

Michael Bastubbe
RWTH Aachen University, Lehrstuhl für Operations Research, Kackertstr. 7, 52072 Aachen
E-mail: bastubbe@or.rwth-aachen.de
ORCID: 0000-0002-4416-925X

Timo Berthold
Fair Isaac Germany GmbH, Stubenwald-Allee 19, 64625 Bensheim, Germany
E-mail: timoberthold@fico.com

Philipp M. Christophel
Operations Research R&D, Advanced Analytics Division, SAS Institute Inc.
E-mail: Philipp.Christophel@sas.com
ORCID: 0000-0002-3036-7239

Kati Jarck
MOSEK ApS, Fruebjergvej 3, Symbion Science Park, 2100 Copenhagen, Denmark
E-mail: kati.jarck@mosek.com

Thorsten Koch
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
E-mail: koch@zib.de
ORCID: 0000-0002-1967-0077

Jeff Linderoth
University of Wisconsin-Madison, Department of Industrial and Systems Engineering & Wisconsin Institute for Discovery
E-mail: linderoth@wisc.edu
ORCID: 0000-0003-4442-3059

Marco Lübbecke
RWTH Aachen University, Lehrstuhl für Operations Research, Kackertstr. 7, 52072 Aachen
E-mail: luebbecke@or.rwth-aachen.de
ORCID: 0000-0002-2635-0522

Hans D. Mittelmann
School of Mathematical and Statistical Sciences, Arizona State University, Tempe, AZ, USA
E-mail: mittelmann@asu.edu
ORCID: 0000-0003-1961-657X

Derya Ozyurt
The MathWorks, Inc., 1 Apple Hill Drive, Natick, MA 01760-2098, USA
E-mail: dozyurt@mathworks.com
ORCID: 0000-0002-2943-7762

Ted K. Ralphs
Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, 18015, USA
E-mail: ted@lehigh.edu
ORCID: 0000-0002-4306-9089

Domenico Salvagnin
Department of Information Engineering, University of Padova, Via Gradenigo 6/b, 35131 Padova
E-mail: domenico.salvagnin@unipd.it
ORCID: 0000-0002-0232-2244

Yuji Shinano
Zuse Institute Berlin, Department of Mathematical Optimization, Takustr. 7, 14195 Berlin
E-mail: shinano@zib.de
ORCID: 0000-0002-2902-882X