Exploring the SMT-LIB Benchmark Library*

Hans-Jörg Schurr[1], François Bobot[2], Mathias Preiner[3], Aina Niemetz[3], Clark Barrett[3], Pascal Fontaine[4], and Cesare Tinelli[1]

[1] The University of Iowa, Iowa, USA
[2] Université Paris-Saclay, CEA, List, FR
[3] Stanford University, Stanford, USA
[4] Université de Liège, Liege, BE

Abstract. The SMT-LIB benchmark collection is a large set of problems for SMT solvers. It has been continuously maintained and expanded since its creation in the early 2000s by the SMT-LIB initiative. It has been used since 2005 by the annual SMT solver competition to compare the performance of SMT solvers, and by researchers to study novel solving techniques. Effective use of the collection often requires access to benchmark metadata (e.g., date, source, satisfiability status, theory symbol count, and so on). Furthermore, this metadata and the past competition results contain a wealth of historical information about the development of SMT solving. In this paper, we report on our efforts to collect and curate all metadata from the SMT-LIB benchmarks together with the results of all past SMT-COMP competitions in a single SQLite database. We also present tools to explore this database and extract relevant insights. Since APIs for SQLite databases are available for all major programming languages, the database makes it easy to add features using SMT benchmark metadata to SMT development tools. To illustrate the structure of the collected data we perform multiple case studies. In particular, we present a comparison of SMT solvers that is independent of the changing hardware and benchmarks used by the competition. The database is released annually on Zenodo, and serves as an archive of the state of SMT-LIB and, by extension, of the state of the art in SMT.

Keywords: SMT · SMT-LIB benchmarks · data integration · SQLite · database · automated reasoning

1 Introduction

The SMT-LIB [5] initiative is an ongoing international effort started in 2003 whose goal is to facilitate research and development in Satisfiability Modulo Theories (SMT). Part of this initiative is the development of the SMT-LIB language [6] for specifying SMT problems in a textual format and the collection

* This work was supported in part by the Stanford Center for Automated Reasoning, the Stanford Center for Blockchain Research, Defense Advanced Research Projects Agency (DARPA) contract FA875024-2-1001, National Science Foundation (NSF) grant number 2303489, and a gift from Amazon Web Services.

and maintenance of a large library of benchmark problems written in that format. The benchmark library, currently maintained by the authors of this paper, is curated and continually extended with contributions from the SMT community and published online in yearly releases. The 2025 release of the library, the latest one to date, contains a total of 495,180 benchmarks divided into two categories: *non-incremental* benchmarks (450,472), which consists of problems with a single satisfiability query, and *incremental* benchmarks (44,708), which contain multiple satisfiability queries.

The library is used to evaluate and improve SMT solving techniques on problems relevant to users or developers of SMT solvers. A large-scale evaluation is done annually by SMT solver competition SMT-COMP [21], which started in 2005. Overall, the benchmarks and the competition results provide a wealth of information on the structure of SMT problems (the majority of which come from real-world applications) and the development of SMT solving over time.

Data about SMT benchmarks is useful to SMT solver users and developers who often rely on advanced metadata for their work. For instance, when evaluating the performance of a procedure or solver configuration that targets a specific fragment of a theory, it may be desirable to only include benchmarks that contain only symbols of that fragment. Other examples are identifying benchmarks that are uniquely solved by a solver configuration, or not solved by any known configuration—both corner cases that may serve as important starting points for developing new solving techniques. Metadata can also be used for solving itself, e.g., to guide automatic selection of solving strategies [26].

However, until now this metadata was not easily accessible to SMT developers and users. Competition results were stored in different formats and different archives. The results of the competitions from 2007 to 2012 were lost due to a server migration. The benchmark metadata was commonly collected on demand, via ad-hoc scripts. This was not only error-prone but also inconvenient.

To address these issues, we have collected and integrated benchmark metadata, results from all past SMT competitions, and hand-curated data. The result is now publicly available as a single SQLite database together with the annual release of the benchmark library [25]. The database available under the terms of the CC-BY 4.0 license.

Since software libraries for interacting with SQLite databases exist for all major programming languages, the database can be integrated with existing benchmarking and testing systems easily. Furthermore, since every SQLite database is stored as a single file, the benchmark database is also easy to share.

As we discuss in Section 5, we have used the database to perform multiple case studies and to visualize the historical development of SMT solving. We first focus on simple investigations, such as the growth of the library over time. One difficulty when working with historical competition results is that the competition hardware changed over time. Hence, runtimes are not always directly comparable. Furthermore, the competitions used different subsets of the benchmarks. We illustrate how this can be addressed by comparing the relative performance of SMT solvers.

The database also serves an archival purpose: each annual release on Zenodo is a snapshot of past competition results and benchmark metadata. In the past, results from historical SMT competitions (2007-2012) became unavailable. Furthermore, the StarExec service [27], which partially served as an archive of the various releases of the library, has been recently discontinued. Our database helps to fill these gaps. To simplify the annual update of the database, we implemented the data integration pipeline through a collection of flexible Python scripts. For benchmark metadata extraction, we developed an optimized standalone tool called *Klammerhammer*. Klammerhammer and the Python pipeline are available under the terms of the 3-Clause BSD license.

Section 2 of this paper discusses related work. Then, Section 3 describes how benchmarks are represented in the database and gives a few example scenarios of its usage. In Section 4, we describe our data integration pipeline. There are two sources of data: the benchmark metadata (Section 4.1) and the outcome of SMT evaluations (Section 4.2). In Section 5, we discuss multiple studies we did on the collected data. Finally, we suggest future directions of research in Section 6.

2 Related Work

Benchmark libraries are common in the automated reasoning and theorem proving communities. Among those, the TPTP library [28] is close to SMT-LIB in spirit, but it precedes it and targets theorem provers instead of SMT solvers. TPTP problems store metadata in their header, including syntactic features such as symbol counts. The metadata header also contains a difficulty rating that inspired our rating (see Section 4.2). Instead of a standalone database, TPTP organizes benchmarks into folders and subfolders, and provides tools to search the library for benchmarks with specific characteristics. This is possible because the TPTP policy is to carefully curate the benchmark set and admit fewer benchmarks (the 9.2.1 release has 26,264 benchmarks).

The Global Benchmark Database [15] (GBD) catalogs benchmarks for propositional satisfiability (SAT). GBD is focused on a flexible data model where benchmarks are associated with different contexts, and contexts are associated with data fields. For example, the `cnf` context represents benchmarks in conjunctive normal form, and contains fields like the number of clauses and variables. To support this model, GBD's intended interface is a custom tool and query language. The GBD tool can be connected to different data sources using file formats such as SQLite and CSV. Hence, our database could be connected to GBD. However, our tables do not correspond to GBD contexts, and it would be necessary to present a denormalized schema. For example, each symbol count would be a dedicated field, instead of a relation between queries and the symbols list. GBD uses the concept of an *identification function* that maps benchmarks to identifiers. Such a function would help with the benchmark identification problems discussed in Section 4.2. However, we cannot use this approach, since we do not have archives of all benchmarks for every competition, and finding an iden-

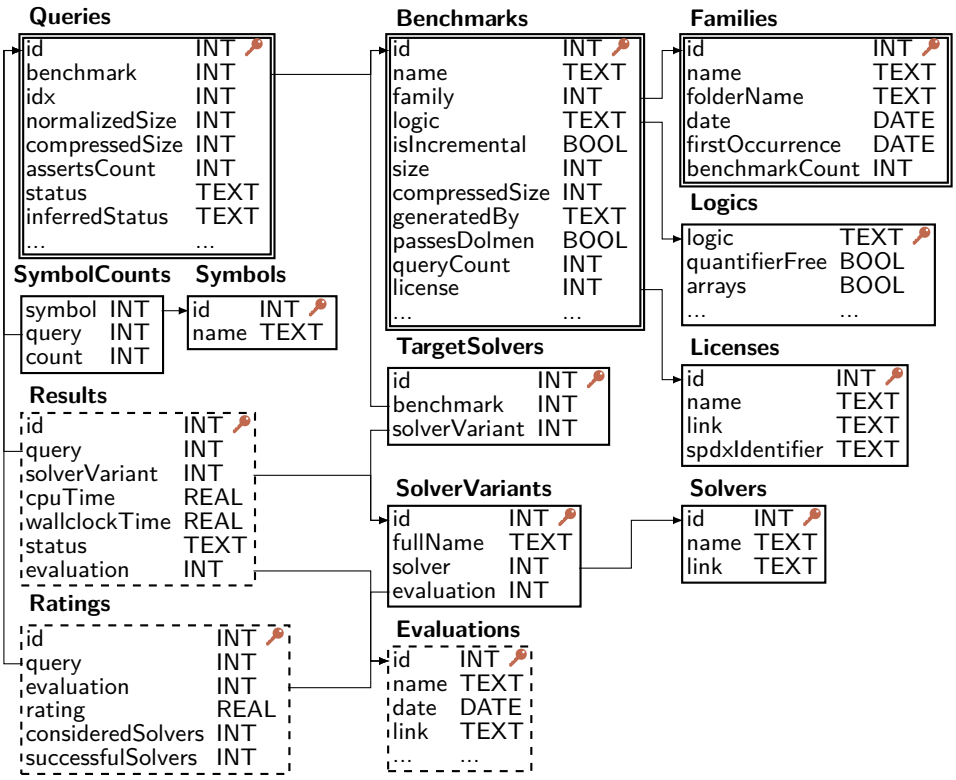


Fig. 1. Database schema of the catalog with some fields omitted.

tification function that is not affected by common benchmark editing operations is challenging due to the syntactic complexity of SMT problems.

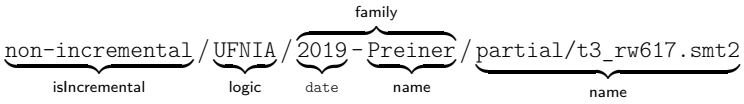
SMTQuery [16] is a SMT benchmark analysis tool focused on the theory of strings. It uses a custom, SQL-like, language. The focus on strings and the custom implementation allows SMTQuery to support complex queries on the metadata. For example, it is possible to search for benchmarks with word equations that have a specific shape. Metadata like symbol frequency has been used for machine learning-based SMT solver selection [26].

A few retrospectives of the SMT competition have been published over the years. This includes the first six years of SMT-COMP [8] and the competitions between 2015 and 2018 [31]. Most notably, in 2013 a large scale evaluation was performed instead of an SMT competition [13]. This included an evaluation using all benchmarks on all solvers from prior years.

3 The Structure of the Database

Figure 1 shows the schema of the SQLite database that stores the collected data. The database tables of the schema fall into three categories, distinguished visually by their border. The three tables highlighted with ▣ form the core of the database and list the benchmarks and the queries they contain; the tables marked by □ store static metadata, such as symbol frequencies; and the tables shown as ⁚⁚ boxes store the results of large scale evaluations.

Every row of the **Benchmarks** table represents one SMT-LIB benchmark, and each benchmark belongs to exactly one *family*, stored in the **Families** table. This classification follows the folder structure of the benchmark library. There, each benchmark is uniquely identified by its file path, for example:



The topmost folder indicates whether the benchmark is incremental or not. The next subfolder is named after the SMT-LIB *logic* used in the benchmark. A logic indicates the SMT theories referred to by the benchmark and the language fragment its queries belongs to (e.g., quantifier-free, linear, ...). A benchmark's logic is stored in the field logic of the **Benchmarks** table. The third component of the file path is the name of the benchmark's *family*. A family usually collects benchmarks with some property in common. For instance, they originate from the same application, or are generated by the same tool. Note that a family may contain benchmarks from several SMT-LIB logics. Hence, every benchmark refers to an entry in the **Families** table. The value of the field date (either a full date, or just a year) is chosen by the person who contributed the benchmarks, and is usually the date on which the benchmark family was generated.[5] The field firstOccurence records the date of the first competition that used a benchmark from the family. Overall, there are currently 287 families. Finally, the benchmark name is the path fragment following the family. It is stored in the name field of **Benchmarks**.

Incremental benchmarks contain more than one satisfiability query expressed with a `check-sat` command. Each `check-sat` command corresponds to a row in the **Queries** table. The command instructs the solver to determine the satisfiability of a set of formulas previously asserted with one or more `assert` commands. Asserted formulas are stored on a stack, which can be manipulated using the `push` and `pop` commands. The idx field of the **Queries** table is the index of the query in the benchmark. For example, if idx is 3, the query corresponds to the third `check-sat` call. Overall, there are 34,614,311 queries. Some benchmarks individually contain thousands of queries. The benchmark currently with the highest number of queries has 2,630,828 of them.

[5] Some benchmark families are not associated with a date for historical reasons.

```
(set-info :smt-lib-version 2.6)
(set-logic UFNIA)
(set-info :source |
  Generated by: Mathias Preiner
  Generated on: 2019-03-22
  Application: Verifying bit-vector rewrite rule candidates.
  Target solver: CVC4, Z3, Vampire |)
(set-info :license "https://creativecommons.org/licenses/by/4.0/")
(set-info :category "crafted")
(assert [...]) [...]
(set-info :status unknown)
(check-sat) (exit)
```

Listing 1.1. Abridged content of the SMT-LIB file from Section 3.

Exploring the Database The SMT-LIB benchmark library is released on the open-access repository Zenodo [22, 23]. Starting 2025, the metadata database is released as an additional Zenodo artifact "SMT-LIB Catalog" [25] consisting of a compressed archive containing the SQLite database proper as well as a number of helper files. Since this archive is large (currently, around 1.5 GiB, 5.4 GiB uncompressed), we expect users to download the database and perform queries locally. To reduce the file size, the database has no query indexes. However, the archive contains a script for generating default indexes.

The most basic way to explore the database is to use the SQLite command line tool to perform queries. Alternatively, one can use language bindings, such as Python's `sqlite3` module, and graphical tools, such as the DB Browser for SQLite. For example, the following query returns the number of non-incremental benchmarks containing at least 100 `bvxor` calls (currently, 7,130).

```
SELECT COUNT(Benchmarks.id) FROM Benchmarks
  JOIN Queries        ON Queries.benchmark = Benchmarks.id
  JOIN SymbolCounts   ON SymbolCounts.query = Queries.id
  JOIN Symbols        ON Symbols.id = SymbolCounts.symbol
 WHERE isIncremental = False AND Symbols.name = 'bvxor'
   AND SymbolCounts.count > 100;
```

The following query returns the number of benchmarks (currently 6,525) solved by the solver SONOLAR but not by Abziz at SMT-COMP 2014 (with id 10).

```
WITH Eval AS (
  SELECT Queries.id, Solvers.name AS sn FROM Queries
  JOIN Results        ON Results.query = Queries.id
  JOIN SolverVariants ON SolverVariants.id = Results.solverVariant
  JOIN Solvers        ON Solvers.id = SolverVariants.solver
  WHERE Results.evaluation = 10 AND Results.status != 'unknown' )
SELECT COUNT(DISTINCT ev.id) FROM Eval AS ev
  WHERE (sn == 'SONOLAR') AND
  NOT EXISTS (SELECT * FROM Eval WHERE sn == "Abziz" AND ev.id == id);
```

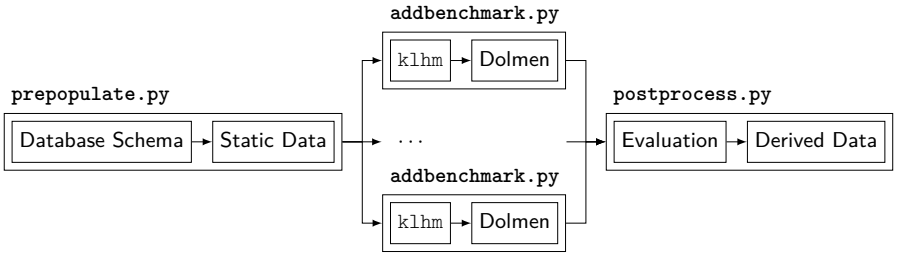


Fig. 2. The data integration pipeline.

Website. We also provide webpages generated from the database at explore.smt-lib.org, which allow one to quickly explore the SMT-LIB benchmarks. Users can browse benchmarks by logic and family. Once a benchmark is selected, a dedicated page shows the related metadata. Beyond the header data, and symbol counts, the page also lists all competitions where the benchmark was used, and the response by the participating solver variants. Every benchmark has a static URL based on its id. For example, the benchmark sketched in Listing 1.1 is available at explore.smt-lib.org/benchmark/111937.html. The website also shows data visualizations like those we present in Section 5 for all logics.

4 Data Integration

We collect metadata from two main sources: the individual benchmark files and the SMT competition data. The data collection from these sources and its integration into the database is implemented as a modular and easy-to-extend pipeline, written in Python.[6] Figure 2 depicts the workflow of the pipeline. It is divided into three stages, each implemented as a Python script. In the first stage, the script `prepopulate.py` creates the database scheme, and initializes the database with static data, such as the list of licenses and logics used in SMT-LIB benchmarks, the names of SMT solvers that participated in SMT-COMP, and so on. In the second stage, the script `addbenchmark.py` parses the individual benchmark files to extract the benchmark metadata. Since this stage is the most time consuming, to speed-up the data collection the user can run a provided auxiliary script to launch multiple copies of `addbenchmark.py` in parallel on different parts of the benchmark library. As a final step, the script `postprocess.py` integrates the SMT-COMP results data into the database and computes and stores additional data derived from these results.

4.1 Integrating Benchmark Metadata

The metadata tables (☐) store data associated with benchmarks or queries. Each benchmark includes a header section that stores metadata (see Listing 1.1).

[6] Available at github.com/SMT-LIB/SMT-LIB-db.

The header uses the `set-info` command to declare metadata fields. This command can specify a `:source` field, which contains sub-fields that give information about the source of the benchmark. Most of these fields relate to the entire benchmark. Hence, most metadata fields are mapped directly to a corresponding field in the **Benchmarks** table. The entry for the example benchmark would store `Mathias Preiner` in the generatedBy field. The `:status` field relates to a specific *query*. It indicates whether the next query is known to be satisfiable or unsatisfiable. This is stored in the status field in the **Queries** table.

The license field associates one license to the benchmark, currently among a list of eleven, in the manually curated **Licenses** table. The license is usually identified by a short code, such as `GPL`, or by a link. However, some benchmarks (e.g., the *CPAchecker_kInduction-SoSy_Lab* family) contain the entire license text. We shorten this to a license code ("CMU SoSy Lab" in this case). The `Target solver` entry lists the solvers targeted by the benchmark creator. We store this list in the **TargetSolvers** table that maps solver variants to benchmarks. Solver variants are also used for SMT competition results (Section 4.2).

A key data point about an SMT query is which theory symbols and SMT-LIB features are used, and how often they occur in a benchmark. While these counts are not directly available in the benchmark header, they can be computed by scanning the benchmark. There are two different categories of SMT-LIB constructs that can be counted. On the one hand, there are commands such as `assert` which asserts formulas. On the other hand, there are predefined theory symbols that appear in formulas. The first category is small and fixed, while the second category is large and grows as new theories are added to SMT-LIB. Counts from the first category are therefore fields of the **Queries** table, e.g., assertsCount gives the number of `assert` commands used by a query. For the second category, we use the **Symbols** and **SymbolCounts** tables. The former lists all theory symbols we consider. We extracted this table from the SMT-LIB parser of the *cvc5* SMT solver [3]. The **SymbolCounts** has one entry for each symbol that appears at least once in a query. To simplify the scanner, we do not distinguish theory symbols from user declared symbols. Hence, the **SymbolCounts** table may contain entries for symbols that are not part of the benchmark logic.

The normalizedSize field of the **Queries** table is the logical *size* of a query in bytes. A query is identified with each `check-sat` command and encompasses all the commands that assert in the stack information relevant to that `check-sat` command. We handle the `push` and `pop` commands to ensure that we do not take the size of inactive assertions into account. The compressedSize field is the size of the query after compression with the *zstd* algorithm. This field is intended to measure the problem size independent of factors such as the length of symbol names. The size and compressedSize fields of the **Benchmarks** table are the logical sizes of the entire benchmark.

Finally, the passesDolmen field of the **Benchmark** table records whether Dolmen, the reference parser and type checker for SMT-LIB [11], reports no error for the benchmark.

Klammerhammer. To extract the metadata quickly we developed *Klammerhammer*, a standalone tool that performs a simple scan of the benchmark. It stores symbol counts on a stack. SMT-LIB `push` commands push a copy of the counts onto the stack, while `pop` commands remove the topmost entry. Whenever a `check-sat` command is encountered, the tool prints the current counts as JSON data. After scanning the entire benchmark, the tool prints the metadata fields for the entries in the **Benchmarks** table. To compute the query size we also store the byte offset of `push` and `pop` calls on the stack.

The tool is implemented in the low level programming language Zig, and uses the zstd library to compute the compressed sizes. It can be used independently of our data integration pipeline to extract benchmark metadata on demand, for instance, from non-public benchmarks or to implement strategy selection tools.

4.2 Integrating SMT-COMP Results

The database not only stores metadata on individual benchmarks in the SMT-LIB library, but also combines it with the historical data from all SMT competitions. This allows users to get answers for questions like "Did solver X solve benchmark Y in the past?" or "How difficult is this benchmark?". Normally, to answer these questions one must evaluate SMT solvers on the benchmarks. Instead of performing their own evaluations, users of our database can rely on historical results of the yearly SMT competition [21].

SMT-COMP participants can compete in multiple tracks. For instance the *single query* (resp. *incremental*) track tasks solvers with solving non-incremental (resp. *incremental*) benchmarks. Integrating multiple competition years also allows us to cover more benchmarks, since recent competitions use only a random subset of the benchmarks from the SMT-LIB library due it its increasingly large size. In recent years, the competition organizers publish sanitized results for the non-incremental track. Unfortunately, they only provided summary results for the incremental track, without the solvers answers for each individual query. The 2025 release of the database contains results for the incremental track of SMT-COMP 2024 generated from raw logs. We plan to include results for the years 2019 to 2023 in the 2026 release based on backups of raw competition logs.

Each competition year is a row in the **Evaluations** table. In the future, we can also use this table to store the results of other evaluations. This row stores some basic data about the competition, such as a link to its website. We also store the *hardware generation* used by the competition. This number is increased whenever the competition changed computation hardware (see Table 1). The solvers that participate in an evaluation are collected in the **Solvers** and **SolverVariants** tables. A solver variant is a concrete version of a solver that participated in an evaluation (or is mentioned as a target solver in a benchmark). Since solvers have different versioning schemes, we do not attempt to record solver *versions*. Instead, the different variant can represent solver versions, but also the different names used to refer to the same solver (e.g., `cvc5` and `CVC5`). Both the **Solvers** and the **SolverVariants** tables are manually curated. Overall, we record 82 solvers and 484 variants.

For each evaluation, the **Results** table connects queries to solvers. The status field is `sat`, `unsat`, or `unknown`, depending on the answer of the solver. Furthermore, we record both the wall-clock time and the CPU time when available from historical data.[7] Note that, due to the changing competition hardware, runtimes cannot be compared naively between arbitrary years.

A major challenge is that the structure of the SMT-LIB benchmark collection has also changed over the years. For example, the logic field of misclassified benchmarks was updated. Benchmarks were also removed if they were found not to comply with the SMT-LIB standard. To address this, when extracting information from the raw data of a particular edition of SMT-COMP we search for benchmarks from the SMT-LIB library heuristically in multiple steps. First we do a selection based only on the benchmark's name field since that seldom changes. If that returns a single benchmark, we used that benchmark. Otherwise, we narrow down the search by adding also the family, and finally the logic. If we were unable to uniquely determine the benchmark using this method, we discard the result. Our goal is not to record the entire evaluation, but to collect the results related to benchmarks in the current release. The missing benchmarks often correspond to a cleanup of the benchmark library. For example, from 2014 to 2016 the *AProVE* family contained duplicate benchmarks, and, in 2018, the missing benchmarks are in the `QF_SLIA` logic that was experimental that year.

Table 1 lists competition years and the missing results. The second column shows the benchmark file format. The first two competitions are published as HTML websites. From 2007 until 2012 the competition used SMTExec [4]. Since this platform is no longer online, these results are not publicly available. We used an archived backup of the SMTExec database to add those years and are currently working on restoring the public results. The SMT-EVAL in 2013 [13] and the competitions between 2014 and 2017 use very similar CSV formats, with different column names. Since 2018, all results are available as JSON files.

We compute derived fields from the evaluation results. The firstOccurence field of the **Families** table is the date of the first evaluation where any benchmark of the family was used. This is useful for benchmarks without metadata header or date in the file path. The inferredStatus field is a status (`sat` or `unsat`), if at a single evaluation two distinct solvers agreed on that status, and there was no disagreement by a third solver. Hence, this field allows users to know the likely status of a query if no status is explicitly provided in the benchmark itself.

Finally, we compute a difficulty *rating* for each query at each evaluation. This rating is the fraction of solvers that solved a benchmark over the solvers that attempted it: successfulSolvers/consideredSolvers. We *consider* a solver if any of its variants responded to any benchmark in the same logic. This excludes solvers that do not support the benchmark logic. A solver is *successful* if any of its variants gave a `sat`/`unsat` answer that did not contradict the status or inferredStatus value. This rating is inspired by the TPTP library. Our calculation, however, is slightly different. TPTP ignores solvers that solve only a strict subset of queries solved by another solver. We keep these solvers, because a superseded

[7] Not all competitions recorded both.

			Results			Benchmarks		
Year	Format	Generation	Missing	Total	Percent	Missing	Total	Percent
2005	HTML	1	10	3,299	0.30%	1	355	0.28%
2006	↓	2	158	7,067	2.24%	58	1,127	5.15%
2007	SQL	3	684	12,370	5.53%	149	2,297	6.49%
2008	↓	↓	933	16,110	5.79%	253	2,993	8.45%
2009	↓	4	471	14,948	3.15%	232	3,711	6.25%
2010	↓	↓	684	12,898	5.30%	208	3,731	5.57%
2011	↓	↓	380	18,588	2.04%	88	3,779	2.33%
2012	↓	↓	496	8,020	6.18%	90	1,557	5.78%
2013	CSV_1	5	1,370	1,663,472	0.08%	87	95,491	0.09%
2014	CSV_2	↓	38,160	347,147	10.99%	9,097	67,426	13.49%
2015	↓	↓	68,675	980,235	7.01%	9,255	154,238	6.00%
2016	↓	↓	68,757	1,003,075	6.85%	9,274	154,424	6.01%
2017	↓	↓	435	1,186,056	0.04%	117	238,758	0.05%
2018	JSON	↓	29,789	1,388,191	2.15%	29,475	333,241	8.84%
2019	↓	↓	91	730,685	0.01%	13	64,154	0.02%
2020	↓	↓	878	563,052	0.16%	175	89,910	0.19%
2021	↓	↓	7	772,681	0.00%	1	99,254	0.00%
2022	↓	↓	0	658,873	0.00%	0	93,791	0.00%
2023	↓	↓	9	740,591	0.00%	1	111,285	0.00%
2024	↓	6	0	491,221	0.00%	0	123,486	0.00%
2025	↓	↓	0	823,169	0.00%	0	129,361	0.00%

Table 1. Evaluations: data format, hardware generation, and missed benchmarks.

solver is nonetheless typically the result of a serious research effort. Its inability to solve a query provides evidence of the difficulty of that query. One motivation for ignoring superseded solvers in TPTP is that the weaker solver is often a specialized variant of a stronger solver. We sidestep this consideration in our computation by grouping variants of the same solver into a single virtual solver.

5 Case Studies

In this section, we explain how we have used the collected data to better understand the development of the benchmark collection and SMT solvers over time. A key challenge in analyzing the date comes from the heterogeneity of SMT solving. Many solvers support only some theories, and research interests in different theories varies over time. Hence, we will focus on the big picture provided by the data. As mentioned in Section 3, the project webpage hosts graphs for all logics.

In Section 5.1 we visualize key data points from the database. Our goal is to understand how the library evolved over time. In Section 5.2 we study the development of solvers over time. Since the competition hardware and bench-

mark sets changed, we cannot compute a performance ranking. However, we can determine how *similar* their performance is.

5.1 Evolution of the Library

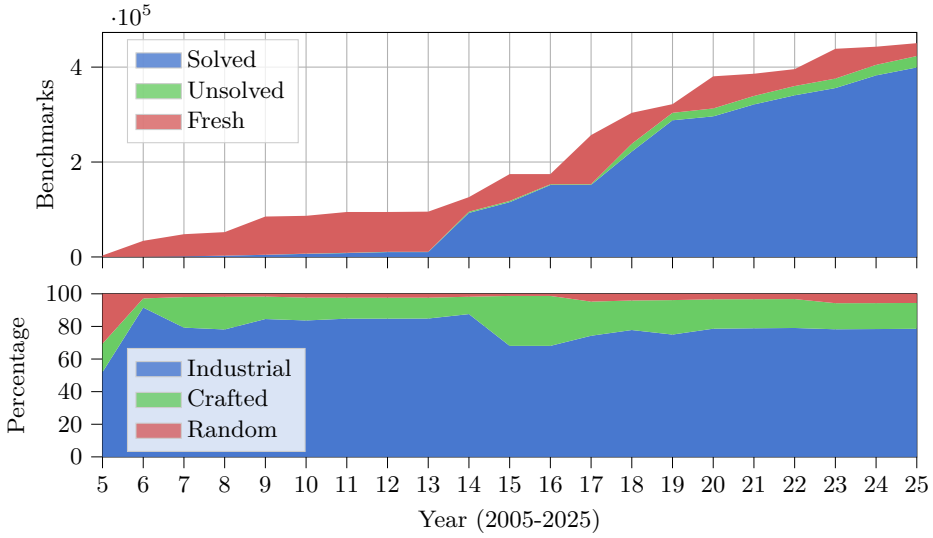


Fig. 3. Benchmarks and categories over time.

Figure 3 shows the growth of the benchmark collection over time. To compute this timeline we use the firstOccurence field of the benchmark families. Since this field uses competition results, and we only have limited competition data for incremental benchmarks, we restrict ourselves to non-incremental benchmarks.

The first graph shows how many benchmarks are added each year and how many benchmarks get solved. Benchmarks are *fresh* if they have never been used at a competition, benchmarks are *solved* if at least one solver gave a response not contradicting the benchmark status, and benchmarks are *unsolved* otherwise. Initially, the competitions used only a small subset of the available benchmarks every year. The 2013 SMT evaluation then used all existing benchmarks. Subsequent competitions always used all or large subsets of the benchmarks. It is remarkable that there are relatively few unsolved benchmarks. We suspect that benchmark contributors might be hesitant to submit difficult benchmarks, because they cannot identify whether the benchmark is simply too large, or whether it has any interesting properties.

The second graph shows the indicated benchmark categories over time. Most benchmarks are classified as *industrial*, which in SMT-LIB means that they were generated by client applications of SMT solvers, such as software verification

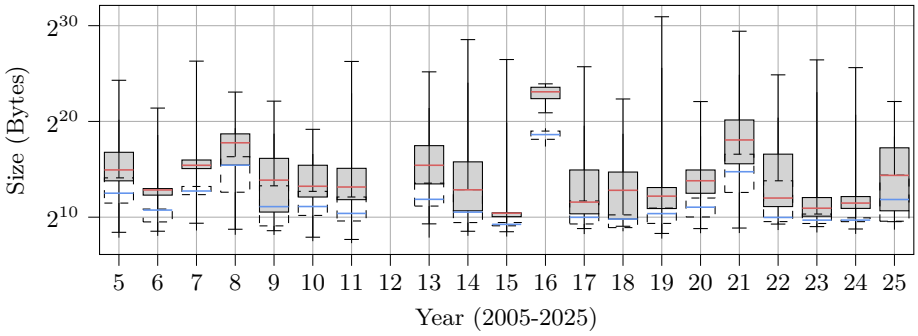


Fig. 4. Size of new benchmarks.

tools. It is not surprising that this category is dominating, since it is easy to generate benchmarks from a client application once it is implemented. *Crafted* benchmarks are benchmarks explicitly created to evaluate specific algorithms or to verify solver compliance to the SMT-LIB standard. One benefit of crafted benchmarks is that their difficulty is typically easier to determine in advance. Hence, they can be a source of interesting and/or challenging benchmarks.

An interesting question is whether benchmarks are getting larger. Figure 4 indicates that they are not. This figure is a box-plot of the size in bytes of benchmarks added to the benchmark collection. The plot is semi-logarithmic. The center line represents the mean, and the boxes extend from the first quartile to the third quartile. Differently from most box-plots, the caps here represent the smallest and largest benchmark. The dashed box shows the sizes after compression. Note that no new benchmarks were added in 2012, and only two were added in 2016. While the size of new benchmarks varies significantly from year to year, there is no clear trend towards larger benchmarks.

When benchmarks are sampled uniformly from all benchmarks of a logic, larger benchmark families are overrepresented. This can be problematic, since benchmarks from the same family share characteristics. In Figure 5 we explore the size of the benchmark families. This diagram shows histograms for all logics with at least five families (semi-logarithmic scale). While family sizes are generally well distributed, most logics also have some huge families.

Figure 6 shows the number of solvers that participated with at least one variant at the competition year by year. This also includes non-competing solvers, for instance from previous editions, that were included by the organizers for comparison purposes. In 2024, this practice changed and fewer solvers from older competitions were included.

5.2 Similarities Between Solvers

The database includes a wealth of information about how the solvers fare on the benchmarks. This information can be visualized effectively using cactus plots,

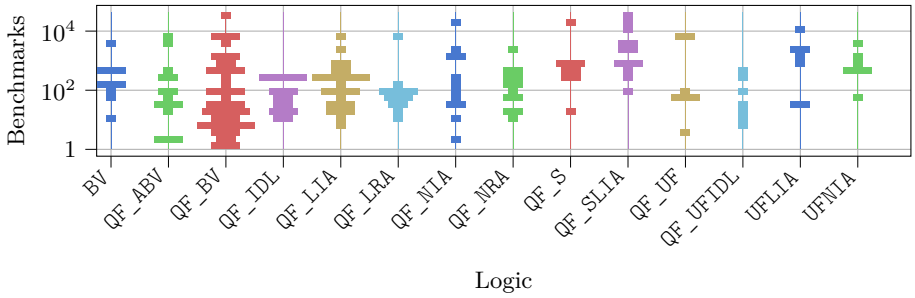


Fig. 5. Histogram of the number of benchmarks in the families.

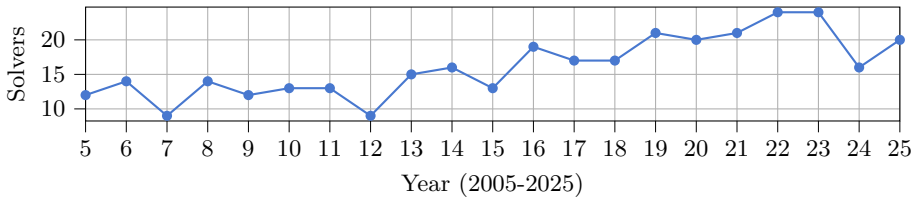


Fig. 6. Solvers over time.

even when many solvers are used [9]. However, we want further insights beyond observing the number of solved instances.

The database is also helpful to answer community-wide questions that are highly relevant for users and their choice of a solver for a particular application: Do solvers handle the same problems? Are solvers complementary? The usual representation using scatter plots would be impractical due to the large number of solvers involved. We rather use data analysis techniques that are naturally able to represents similarities and dissimilarities. Different algorithms exist that map a high dimension feature space with a distance metric into a lower dimension feature space using the Euclidean distance while preserving the initial distance as much as possible. Many choices for the selected features, the distance metric, and the algorithm are possible. We choose the established isomap algorithm as implemented in *scikit-learn* [1, 29]. Since the challenge is to aggregate results from two decades of competitions that considered different benchmark sets and used different computer hardware, the other choices need careful consideration.

As features, we choose just the *wall-clock* time taken by a solver on each benchmark because that information has been available since the first competition. The status *solved*/*unsolved* alone would be too limited to measure the difficulty of a benchmark. Still, in order to differentiate a solver that solved a goal in one second, say, from a solver that gave up in one second by answering `unknown`, we set the wall-clock time for `unknown` answers to 40 min as done for the PAR-2 score considered by the competition (last year's timeout was 20 min).

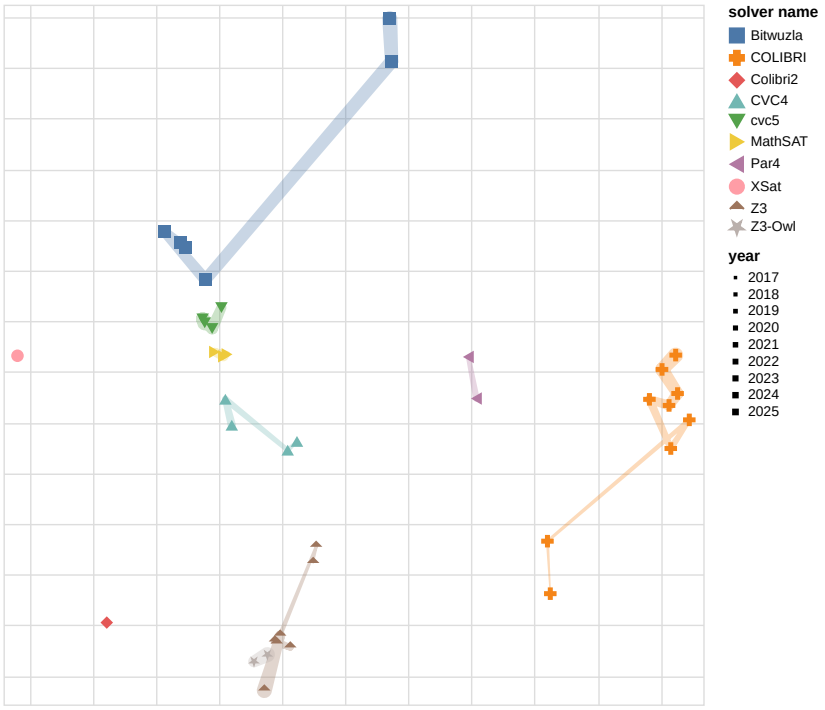


Fig. 7. Isomap for `QF_FP`: a version of a solver is a mark. The versions of the same solver are linked line growing with the competition years they appeared in.

To compute the similarity between two solvers we use the *cosine distance*, the cosine of the angle between two feature vectors (the distance between x and y is $1 - \frac{x\,y}{[\![x]\!][\![y]\!]}$). The cosine distance allows us to reduce the impact of the changing competition hardware on the measure. For example, since the cosine distance between a vector and one that is twice as long is zero. This means that, if the hardware of a competition is twice as fast as the hardware of an earlier competition, all cosine distances remain the same across the two competitions. We point out that the Manhattan distance (L1) used by Biere et.al. [9] for a similar purpose in SAT solving does not have this property. Moreover, it does not differentiate well between many small differences and a few large ones.

Missing feature values, in our case for benchmarks that were not selected in a competition or were added later to SMT-LIB, are usually completed using an *imputer*. Imputer can consist of a first data analysis phase using a distance that handles missing values, such as the *NaN-Euclidean* distances of scikit-learn [2]. To keep the analysis straightforward, we directly compute the distance as the cosine distance on the common benchmarks and do not use an imputer.

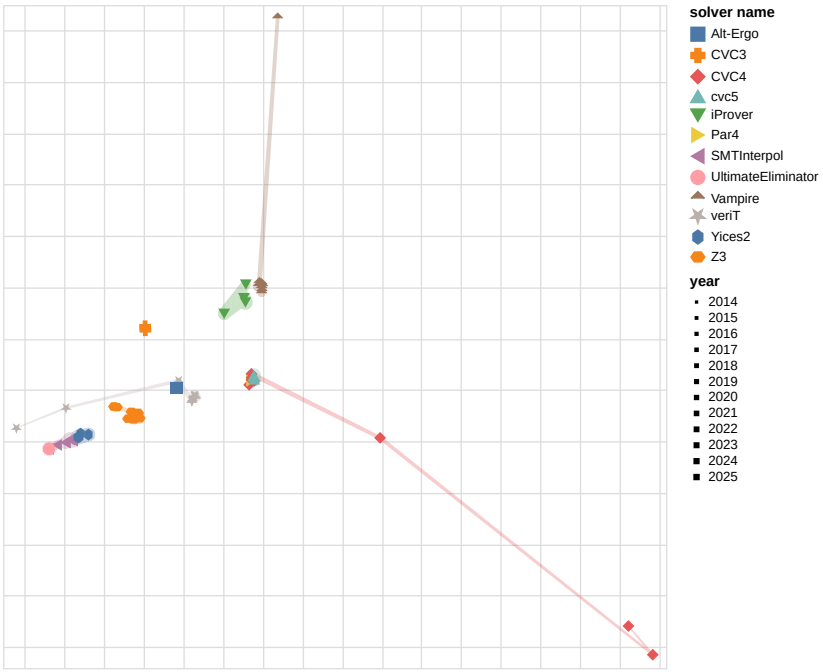


Fig. 8. Isomap for UF.

Figure 7 and Figure 8 show the isomap analysis for the logics QF_FP[8] and UF[9]. The trails show the evolution of each solver over the years.

The graphs have some easily identifiable characteristics. Solvers that implement similar techniques are, in general, close to each other. In QF_FP, solvers that are based on word-blasting, i.e., a reduction of floating-point arithmetic to bit-vectors, are gathered and aligned (Bitwuzla [19], CVC4 [7], cvc5 [3], MathSAT [12], Z3 [18]). Solvers that implement techniques different from word-blasting (COLIBRI [17], Colibri2 [10], and XSat [14]), on the other hand, are farther removed from each other and the word-blasting solvers. The portfolio-based solver Par4 [30], which includes configurations of COLIBRI, CVC4 and Z3, lies between COLIBRI and the word-blasting solvers. We observe that, in the graph, Bitwuzla and COLIBRI make a very distinct jump from one year to the other. For Bitwuzla, this can be attributed to a new solving procedure for bit-vectors based on abstraction-refinement [20], which has a beneficial impact on performance for floating-point arithmetic. For COLIBRI, according to its main developer Bruno Marre,[10] this can be explained by various major improvements

[8] Interactive chart: explore.smt-lib.org/isomap/QF_FP.html
[9] Interactive chart: explore.smt-lib.org/isomap/UF.html
[10] Private communication.

to the system, which is reflected in the graph. We want to emphasize, though, that while we can recognize these patterns in the isomaps for most logics, our assessment is only preliminary and should be extended in the future.

We can also see a stark contrast in the isomap of different logics. In `QF_FP` the solvers spread all over the map, suggesting that different solvers implement different techniques. The `QF_FP` ecosystem is quite active and solvers are typically complementary in terms of solved instances. In contrast, the solvers for `UF` are more clustered together. In addition, later versions of solvers do not move away much. This shows that the various solvers implement similar techniques, or that the benchmarks cannot distinguish among them (they are all solved).

We claim that studies performing data analysis like those above can help the community focus on logics in need of new benchmarks or new research.

6 Conclusion and Future Work

We have presented a new resource for the SMT community consisting of a database that seamlessly integrates data on SMT-LIB benchmarks from different sources, including historical data from 20 years of SMT competitions. We believe the new database and the tooling around it will be useful to both SMT solver users and developers. Our initial case studies also show that this data is useful to compare and understand SMT solvers, their performance and their evolution over time. We consider these studies just a first step towards a more systematic exploration of the collected data.

The first version of the database was released on Zenodo [25] earlier this year. We plan to release annual updates to the database in sync with the annual release of the SMT-LIB benchmark library. New releases will also provide an opportunity to add metadata requested by the community and integrate new data sources. An expected major change will be the transition to the upcoming Version 3 of the SMT-LIB language. Since Version 3 is substantially different from the current version, this will require updates to the data integration pipeline, and possible changes to the metadata fields, notably for the logic identifier.

We are also considering using the database to improve the consistency of the benchmark library and identify other errors in it. For example, using the `inferredStatus` field to update the status in the benchmark would make it easier for solver developers to detect classification errors.

Data-Availability Statement The experiments conducted for this paper used the 2025 release of the SMT-LIB benchmark library available on Zenodo [22,23]. The code to perform the experiments is available on Github [24].

Acknowledgments. We thank the many contributors of SMT-LIB benchmarks, and the organizers of the SMT competition. Geoff Sutcliffe provided valuable insights into benchmark difficulty ratings. We also thank the anonymous reviewers for their valuable feedback on the paper and the design of the database.

References

1. 2.2 manifold learning – scikit-learn 1.7.2 documentation. https://scikit-learn.org/stable/modules/manifold.html#isomap, accessed: 2025-10-16
2. nan_euclidean_distances – scikit-learn 1.7.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.nan_euclidean_distances.html, accessed: 2025-10-16
3. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24
4. Barrett, C., Deters, M., de Moura, L., Oliveras, A., Stump, A.: 6 years of SMT-COMP. vol. 50, pp. 243–277 (Mar 2013). https://doi.org/10.1007/s10817-012-9246-5
5. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org (2016)
6. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.7. Tech. rep., Department of Computer Science, The University of Iowa (2025), available at www.SMT-LIB.org
7. Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_14, https://doi.org/10.1007/978-3-642-22110-1_14
8. Barrett, C.W., Deters, M., de Moura, L.M., Oliveras, A., Stump, A.: 6 years of SMT-COMP. J. Autom. Reason. **50**(3), 243–277 (2013). https://doi.org/10.1007/S10817-012-9246-5, https://doi.org/10.1007/s10817-012-9246-5
9. Biere, A., Fleury, M., Froleyks, N., Heule, J.M.: The SAT museum. In: Järvisalo, M., Le Berre, D. (eds.) Proceedings of the 14th International Workshop on Pragmatics of SAT Co-located with the 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023), Alghero, Italy, July, 4, 2023. CEUR Workshop Proceedings, vol. 3545, pp. 72–87. CEUR-WS.org (2023), http://ceur-ws.org/Vol-3545/paper6.pdf
10. Bobot, F., Hara, H.R.A.E., Correnson, A., Junke, C.: colibri2 (Jun 2025). https://doi.org/10.5281/zenodo.15769941
11. Bury, G.: Dolmen: A validator for SMT-LIB and much more. In: Nadel, A., Niemetz, A. (eds.) Proceedings of the 19th International Workshop on Satisfiability Modulo Theories. CEUR Workshop Proceedings, vol. 2908, pp. 32–39. CEUR-WS.org (2021)
12. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT solver. In: Piterman, N., Smolka, S.A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7795, pp. 93–107. Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_7

13. Cok, D.R., Stump, A., Weber, T.: The 2013 evaluation of SMT-COMP and SMT-LIB. J. Autom. Reason. **55**(1), 61–90 (2015). https://doi.org/10.1007/S10817-015-9328-2
14. Fu, Z., Su, Z.: Xsat: A fast floating-point satisfiability solver. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9780, pp. 187–209. Springer (2016). https://doi.org/10.1007/978-3-319-41540-6_11, https://doi.org/10.1007/978-3-319-41540-6_11
15. Iser, M., Jabs, C.: Global Benchmark Database. In: Chakraborty, S., Jiang, J.H.R. (eds.) 27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024). Leibniz International Proceedings in Informatics (LIPIcs), vol. 305, pp. 18:1–18:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2024). https://doi.org/10.4230/LIPIcs.SAT.2024.18
16. Kulczynski, M., Lotz, K., Manea, F., Poulsen, D.B., Sarnighausen-Cahn, P.: SMT-Query: Analysing SMT-LIB string benchmarks. In: C. Nogueira, S., Teodorov, C. (eds.) Formal Methods: Foundations and Applications. pp. 22–34. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-78116-2_2
17. Marre, B., Blanc, B., Mouy, P., Chihani, Z., Vedrine, F., Bobot, F.: Colibri (2019), https://smt-comp.github.io/2019/system-descriptions/colibri.pdf
18. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
19. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13965, pp. 3–17. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_1
20. Niemetz, A., Preiner, M., Zohar, Y.: Scalable bit-blasting with abstractions. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14681, pp. 178–200. Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_9
21. Organizers, S.C.: The SMT competition. https://smt-comp.github.io (2025)
22. Preiner, M., Schurr, H.J., Barrett, C., Fontaine, P., Niemetz, A., Tinelli, C.: SMT-LIB release 2025 (incremental benchmarks) (May 2025). https://doi.org/10.5281/zenodo.15493096
23. Preiner, M., Schurr, H.J., Barrett, C., Fontaine, P., Niemetz, A., Tinelli, C.: SMT-LIB release 2025 (non-incremental benchmarks) (Aug 2025). https://doi.org/10.5281/zenodo.15493089
24. Schurr, H.J., Bobot, F.: Github reporsitory: SMT-LIB / SMT-LIB-db. https://github.com/SMT-LIB/SMT-LIB-db/releases/tag/tacas26 (2026)
25. Schurr, H.J., Preiner, M., Niemetz, A., Barrett, C., Fontaine, P., Tinelli, C.: SMT-LIB catalog 2025 (Jul 2025). https://doi.org/10.5281/zenodo.16290040
26. Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: Algorithm selection for SMT. Int. J. Softw. Tools Technol. Transf. **25**(2), 219–239 (2023). https://doi.org/10.1007/S10009-023-00696-0

27. Stump, A., Sutcliffe, G., Tinelli, C.: StarExec: A cross-community infrastructure for logic solving. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8562, pp. 367–373. Springer (2014). https://doi.org/10.1007/978-3-319-08587-6_28
28. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017). https://doi.org/10.1007/s10817-017-9407-7
29. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science **290**(5500), 2319–2323 (2000). https://doi.org/10.1126/science.290.5500.2319
30. Weber, T.: Par4 system description (2019), https://smt-comp.github.io/2019/system-descriptions/Par4.pdf
31. Weber, T., Conchon, S., Déharbe, D., Heizmann, M., Niemetz, A., Reger, G.: The SMT competition 2015–2018. J. Satisf. Boolean Model. Comput. **11**(1), 221–259 (2019). https://doi.org/10.3233/SAT190123