Model Counting Competition 2021: Call for Benchmarks/Participation

Johannes K. Fichte Markus Hecher

February 19, 2021

Abstract

Model counting is a very vibrant field that provided both recent advances in theory as well as in practical solving including various applications. Various solving techniques have been established and implemented into search engines ranging from SAT-based solving with caching, knowledge compilation, approximate solving by means of sampling using SAT solvers, or dynamic programming. There have been also successful implementations for parallel and distributed computation as well as massively parallel computation approaches. An initial competition to compare the advances and the applicability of solvers has been established in 2020, which will go into its next iteration.

In this document, we provide a call for benchmarks and participation. We briefly summarize timeline, formats, requirements, and evaluation setting used in the competition.

More information on the competition can be found at `modelcounting.org`.

Contents

1 Call for Benchmarks

`https://mccompetition.org/2021/mc_description`

Setting Model counting is very vibrant field that provided both recent advances in theory as well as in practical solving including various applications. State-of-the-art search engines rely on techniques from SAT-based solving, knowledge compilation, dynamic programming, or approximate solving by means of sampling using SAT solvers. The success of solving various problems, in the area of satisfiability and declarative languages in the last two decades, can be seen in both the availability of numerous efficient solver implementations and the growing number of applications. Designing efficient solvers requires both understanding of the fundamental algorithms underlying the solvers, as well as in-depth insights into how to implement the algorithms for obtaining efficient and robust solvers. Several competitive events are regularly organized for different declarative solving paradigms to evaluate available solvers on a wide range of problems. Winners of such events often set new standards in the area. The Model Counting (MC) Competition [FHH20] aims to identify new challenging benchmarks and to promote new solvers for the problem as well as to compare them with state-of-the-art solvers.

Benchmarks Challenging and representative benchmarks are essential to perform significant comparisons of solvers. We invite submissions of both real world benchmarks and benchmark generators to ensure a diverse benchmark set for

the competition. In the case of (randomly) generated benchmarks we would be happy if authors also publicly provide the generator. Submissions of real world benchmarks are most welcome no matter if they come directly from an application or if they have been obtained via a translation from another formalism. Note that last year's instances are publicly available. Please do not send us those instances or permuted versions again. We encourage contributors to provide the instances as a dataset on the public data repository Zenodo (`https://zenodo.org/`) for submission. Since all instances will be made available to the community after the event, we expect that the copyright of the dataset allows for publication under a CC-BY license. If instances are sensitive (e.g., infrastructure/health-care), sufficient anonymization has to be applied prior to submission.

Selection From existing and submitted benchmarks, we will select instances on which we evaluate submissions in MC2021. Instances and mappings will be publicly released on Zenodo.

Submission Before March 5, 2021, contributors of benchmarks are expected to

- Register for benchmark submission at `tinyurl.com/fm7ucg3z`, and decide between (A) providing us with a link to download the instances or (B) asking us for an upload space to place the instances.
- Submissions are expected to be as one tar-archive of bz2 compressed files in the format described below in Section 2.
- The submission has to contain a short abstract (at most 2 pages single column pdf) describing the dataset (and generator if applicable).
- We appreciate if the submissions contain in addition the following information:
  - a file (`checksums.txt`) listing the sha256 checkum for each uncompressed, submitted instance and
  - a file (`counts.txt`) listing the known (weighted/projected) model counts, the expected hardness, and if applicable in addition the runtime and used tool to obtain the result.

2 Data Format (DIMACS-like)

The input format is similar to the formats used for last year's iteration. But in order to provide a more uniform input without re-encoding headers between tracks (and SAT solvers), we moved additional descriptions lines that would be interpreted as comments in SAT solvers.

2.1 Input Format

In the Model Counting Competition 2021, we use a DIMACS CNF-like input format [TCC+93]. The format extends the format used in SAT competitions [JBRS12, BFH+20][1] by introducing statements for weights and projections similar as in Cachet [KS05] or Ganak [SRSM19].

The following description gives an idea on the expected input format:

```
c c this is a comment and will be ignored
c c REPRODUCIBILITY LINE MANDATORY FOR SUBMITTED BENCHMARKS
c r originUrl/doi descUrl/doi [generatorUrl/doi]
c c HEADER AS IN DIMACS CNF
p cnf n m
c c OPTIONAL MODEL COUNTING HEADER
c t mc|wmc|pmc
c c PROBLEM SPECIFIC LINES for wmc|pmc
c p ...
c p ...
c c CLAUSES AS IN DIMACS CNF
-1 -2 0
2 3 -4 0
c c this is a comment and will be ignored
4 5 0
4 6 0
```

In more details (note that we print symbols in typewriter font):

- Line separator is the symbol \n. Expressions are separated by space.
- A line starting with character `c r` is a comment. The solver may ignore it. The line aims for open data and reproducibility of the submitted instances expressing the origin of the benchmarks. The line will be added after competition when publishing the instances. `originUrl/doi` states where the instance can be downloaded (either as URL or DOI). `descUrl/doi` links to a description of the benchmarks. `generatorUrl/doi` provides an optional URL/DOI to where a problem generator can be found.
- A line starting with character `c t` is a comment. The starting character will be followed by `mc`, `wmc`, `pmc`, or `pwmc` indicating possible problem specific lines starting with `cs` in the file. If present, the line will occur prior to problem specific line. The solver may ignore it.
- A line starting with character `c p` provides problem specific details for weighted or projected model counting. Lines may occur anywhere in the file. We assume that lines are consistent and provide no contradicting information. We provide more details on its meaning below.

[1] See, for example, `http://www.satcompetition.org/2009/format-benchmarks2009.html`

- Lines starting with character `c` *followed by* a second *character differing* from `p` and `s` are comments and can occur anywhere in the file. For convenience, we provide comments by lines starting with `c c`.
- Variables are consecutively numbered from `1` to `n`.
- The problem description is given by a unique line of the form `p cnf NumVariables NumClauses` that we expect to be the first line (except comments). More precisely, the line starts with character p (no other line may start with p), followed by the problem descriptor `cnf`, followed by number `n` of variables followed by number `m` of clauses each symbol is separated by space each time.
- The remaining lines indicate clauses consisting of decimal integers separated by space. Lines are terminated by character `0`. The Line `2 -1 3 0\n` indicates the clause "2 or not 1 or 3" ($v_2 \vee \neg v_1 \vee v_3$). If more lines than announced are present, the solver shall return a parser error and terminate without solving the instance.
- Empty lines or lines consisting of spaces/tabs only may occur and can be ignored.

Weighted Model Counting For weighted model counting, we introduce optional problem specific lines:

```
c p weight  1 0.4 0
c p weight -1 0.6 0
c p ...
```

In more details (note that we print symbols in typewriter font):

- The weight function is given by lines of the form `c p weight` ℓ w_ℓ `0` defining the weight w_ℓ for literal ℓ, where $0 \leq w_\ell$. The weight will be given as floating point (e.g., `0.0003`) with at most 9 significant digits after the decimal point, or in 32-bit scientific floating point notation (e.g., `1.23e+4`), or as fraction (e.g., `3/10`) consisting of two integers separated by the symbol `/`.
- We expect that the submission tests instances and output if it cannot be handled correctly. For example, if an instance specifies a function that has more than 9 significant digits (e.g., `0.00000000009`), larger floating point values, or large fractional values. In such a case, the solver should output a parsing error as specified in the return code section.
- We will provide only instances where $0 \leq w_\ell \leq 1$ and $w_{\neg\ell} + w_\ell = 1$.
- If a weight w_ℓ is defined for a literal ℓ but $\neg\ell$ is not given or $\neg x$, we assume $w_{\neg\ell} = 1 - w_\ell$ (assume that $\neg\neg a = a$). In other words:

  ```
  c p weight  1 0.4 0
  ```

or

```
c p weight -1 0.6 0
```

are the same as

```
c p weight  1 0.4 0
c p weight -1 0.6 0
```

- If the solver can handle weights $w_\ell > 1$ or $w_\ell + w_{\neg\ell} > 1$ unless $w_\ell = w_{\neg\ell} = 1$, the solver must output a warning. Otherwise, the solver must output an error on those instances.
- For compatibility, with #SAT we say that if for a variable x, there is neither a weight for x nor $\neg x$ given, it is considered 1.
  Note: this differs from the format used in Cachet.

Projected Model Counting For projected model counting, we introduce optional problem specific lines:

```
c c INDICATES VARIABLES THAT SHOULD BE USED
c p show varid1 varid2 ... 0
....
c c MORE MIGHT BE GIVEN LATER
c p show varid7 varid23 ... 0
```

In more details (note that we print symbols in typewriter font):

- Projection variables will be given by lines starting with `c p show` followed by the identifier of the variables. Lines describing to add variables to a projection set may occur anywhere in the files and will be terminated by symbol `0`.

Note: if all variables are stated using `show`, we consider model counting. if no variables are stated the problem is simply to decide satisfiability.

2.2 Output Format

We expect that the solver outputs result to `stdout` in the following format.

```
c o This output describes a result of a run from
c o a [weighted|projected] model counter.
c o
c o The following line keeps backwards compatibility
c o with SAT solvers and avoids underflows if result is 0.
c o MANDATORY
s SATISFIABLE|UNSATISFIABLE|UNKNOWN
c o The following solution line is optional.
```

```
c o It allows a user to double check whether a solver
c o that provides multiple options was called correct.
c o
c o MANDATORY
c s type [mc|wmc|pmc]
c o The solver has to output an estimate in scientific notation
c o on the solution size, even if it was an exact solver.
c o MANDATORY
c s log10-estimate VALUE
c o The solver outputs the solution in its highest precision.
c o MANDATORY
c s SOLVERTYPE PRECISION NOTATION VALUE
c o OPTIONAL
c o InternalValueForOpt=X
c o Internal=X
```

In more details:

- While the solver may use a line staring with `c` for comments in the output, we suggest to use lines starting with `c o` to indicate a comment.
- The solver has to announce whether the instance is satisfiable or unsatisfiable by a line starting with `s` followed by `SATISFIABLE` or `UNSATISFIABLE`. The solver may output `UNKNOWN`, but is not allowed to output any other value than these three if a line starting with `s` is present.
- The solver has to announce an estimate on the solution by a line starting with `c s log10-estimate VALUE` where `VALUE` is a string representing the result (model count/weighted model count/projected model count) in $\log_{10}$-Notation (see Section 3.3) of double precision, i.e., 15 significant digits.
- The output is has to be announced by a line starting with `c s` followed by strings indicating the solver type, the precision, the notation, and the result. The solver developer may use for `SOLVERTYPE` the following strings: `approx` and `exact`. In place of `PRECISION`, the solver developer has to specify which the internal precision the solver used, allowed values are `arb`, `single`, `double`, `quadruple`, or other values according to IEEE754 Standard. For `NOTATION`, the developer may chose `log10`, `float`, `prec-sci`, or `int`. Then, in place of `VALUE`, the solver has to output its computed result.

  Note: If the solver developer announces as output a higher precision than the actual solver theoretically allows, we reserve the right to disqualify all submissions by the team.
- The solver may output a result in $\log_{10}$ *notation* which is similar to the format used in the probabilistic inference competitions UAI [GRS+16].

For computing the relative accuracy of the solution of a solver, we refer to Section 7.5.

- If the solver consists of a run script, which calls a pre-processor, consists of multiple phases, or consists of a solving portfolio, we expect the developer to output by a comment line which tool was started, what parameters it used, and when the tool ended. Preferably as follows:
  ```
  c o CALLS(1) ./preproc -parameters
  c o stat CALL1 STARTED RFC3339-TIMESTAMP
  ...
  c o stat CALL1 FINISHED RFC3339-TIMESTAMP
  c o CALLS(2) ./postproc -parameters
  ...
  ```
- We suggest that the solver outputs internal statistics by lines starting with `c o DESCRIPTION=VALUE` or `c DESCRIPTION : VALUE`.

2.3 Examples

The following sections provide a few brief examples for each track with expected input and output.

Model Counting

Example 1. *The following text describes the CNF formula (set of clauses)*

$$\{\{\neg x_1, \neg x_2\}, \{x_2, x_3, \neg x_4\}, \{x_4, x_5\}, \{x_4, x_6\}\}.$$

```
c c This file describes a DIMACS-line CNF in MC 2021 format
c c The instance has 6 variables and 4 clauses.
p cnf 6 4
c t mc
-1 -2    0
 2  3 -4 0
c c This line is a comment and can be ignored.
4 5 0
c The line contains a comment and can be ignored as well.
4 6 0
```

A solution is given as follows, but can also be modified according to the technique of the solver:

```
c o This file describes a solution to a model counting instance.
s SATISFIABLE
c s type mc
c o The solver log10-estimates a solution of 22.
c s log10-estimate 1.342422680822206
c o Arbitrary precision result is 22.
c s exact arb int 22
```

Weighted Model Counting

Example 2. *The following text describes the CNF formula (set of clauses)*

$$\{\neg x_1, \neg x_2\}, \{x_2, x_3, \neg x_4\}, \{x_4, x_5\}, \{x_4, x_6\}\}$$

with weight function $\{x_1 \mapsto 0.4, \neg x_1 \mapsto 0.6, x_2 \mapsto 0.5, \neg x_2 \mapsto 0.5, x_3 \mapsto 0.4, \neg x_3 \mapsto 0.6, x_4 \mapsto 0.3, \neg x_4 \mapsto 0.7, x_5 \mapsto 0.5, \neg x_5 \mapsto 0.5, x_6 \mapsto 0.7, \neg x_6 \mapsto 0.3\}$.

```
c c This file describes a weighted CNF in MC 2021 format
c c with 6 variables and 4 clauses
p cnf 6 4
c t wmc
c c Weights are given as follows, spaces may be added
c c to improve readability.
c p weight  1 0.4 0
c p weight  2 0.5 0
c p weight  3 0.4 0
c p weight  4 0.3 0
c p weight  5 0.5 0
c p weight  6 0.7 0
-1 -2 0
 2  3 -4 0
c this is a comment and will be ignored
 4  5 0
 4  6 0
c same
```

The solution should be given in the following format (modified according to the technique of the solver):

```
c o This file describes a solution to a weighted
c o model counting instance.
s SATISFIABILE
c s type wmc
c o This file describes that the weighted model count is 0.345
c o
c s log10-estimate -0.460924
c s exact double float 0.346
```

Example 3 (Optional)**.** *The following text describes the CNF formula (set of clauses)*

$$\{\neg x_1, x_2\}, \{x_3, \neg x_2\}, \{x_2, x_1\}, \{x_3, x_2\}\}$$

with weight function $\{x_1 \mapsto 0.1, \neg x_1 \mapsto 0.1, x_2 \mapsto 0.1, \neg x_2 \mapsto 0.9, x_3 \mapsto 0.0235, \neg x_3 \mapsto 0.0125\}$ *including a problem description line and two comments.*

```
c c This file describes a weighted CNF in MC 2021 format
c c with 3 variables and 4 clauses
p cnf 3 4
c t wmc
-1  2 0
 3 -2 0
 2  1 0
 3  2 0
c p weight  1 0.1
c p weight -1 0.1
c p weight  2 0.1
c p weight  3 0.0235
c p weight -3 0.0125
```

The solution should be given in the following format:

```
c o WARNING
c o  L9:Sum of positive and negative literal is not equal to 1.
c o WARNING
c o  L12:Sum of positive and negative literal is not equal 1.
c o This file describes a solution to a weighted
c o model counting instance.
s SATISFIABILE
c s type wmc
c o This file describes that the weighted model count is
c o 0.000470000000000000532907051822
c s type wmc
c s log10-estimate -3.327902142064282
c s exact arb log10 -3.32790214206428248634352698 91
```

Projected Model Counting

Example 4. *The following text describes the CNF formula (set of clauses)*

$$\{\neg x_1, \neg x_2\}, \{x_2, x_3, \neg x_4\}, \{x_4, x_5\}, \{x_4, x_6\}\}$$

with projection set $\{x_1, x_2\}$ *including a problem description line and two comments.*

```
c c This file describes a projected CNF in MC 2021 format
c c with 6 variables and 4 clauses and 2 projected variables
p cnf 6 4 2
c t pmc
c p show 1 2
-1 -2 0
 2  3 -4 0
```

```
cc this is a comment and will be ignored
 4  5 0
 4  6 0
```

A solution can be given in the following format:

```
c o This file describes that the projected model count is 3
s SATISFIABILE
c s log10-estimate 0.47712125471966
c s type pmc
c s exact arb int 3
```

3 Problems

In the following section, we briefly describe the considered problems.

3.1 Preliminaries

Let U be a universe of propositional variables. A *literal* is a variable x or its negation $\neg x$. We call x *positive* literal and $\neg x$ *negative* literal. A *clause* is a finite set of literals, interpreted as the disjunction of these literals. A (Boolean) *formula* (in conjunctive normal form) is a finite set of clauses, interpreted as the conjunction of its clauses. An *assignment* is a mapping $\tau : X \to \{0, 1\}$ defined for a set $X \subseteq U$ of variables. For $x \in X$, we define $\tau(\neg x) = 1 - \tau(x)$. By $2^{|X|}$ we denote the set of all assignments $\tau : X \to \{0, 1\}$. By $\tau^{-1}(b)$ we denote the preimage $\tau^{-1}(b) := \{a \mid a \in X, \tau(a) = b\}$ of the truth assignment τ for some truth value $b \in \{0, 1\}$. The formula F *under assignment* τ is the formula F_τ obtained from F by (i) removing all clauses c that contain a literal set to 1 by τ and then (ii) removing from the remaining clauses all literals set to 0 by τ. An assignment τ *satisfies* a given formula F if $F_\tau = \emptyset$. For a satisfying assignment τ, we call the set M of variables that are assigned to true by τ a *model* of F, i.e., $M(\tau) = \{x \mid x \in \tau^{-1}(1)\}$.

3.2 Competition Problems

Let $\mathrm{Mod}(F)$ be the set of all models of F, i.e.,

$$\mathrm{Mod}(F) = \{\tau^{-1}(1) \mid F_\tau = \emptyset, \tau \in 2^{|F|}\}.$$

Problem:	Model Counting (`MC`)
Input:	A Boolean formula F in conjunctive normal form.
Task:	Output the number of models of the formula F, $$\mathrm{mc}(F, P) := |\mathrm{Mod}(F)|\,,$$ which we call *model count* of F.

Problem:	Weighted Model Counting (`WMC`)
Input:	A Boolean formula F in conjunctive normal form and a function $w : \text{lits}(F) \to [0, 1]$, which we call *weight function*.
Task:	Output $\text{wmc}(F, w) :=$ $$\sum_{M \in \text{Mod}(F)} \Big(\prod_{v \in \text{var}(F) \cap M} w(v) \cdot \prod_{v \in \text{var}(F) \setminus M} w(\neg v) \Big),$$ which we call *weighted model count* of F wrt w.

Problem:	Projected Model Counting Problem (`PMC`)
Input:	A Boolean formula F in conjunctive normal form and a set $P \subseteq \text{var}(F)$ of *projection variables*.
Task:	Output $\text{pmc}(F, P) :=$ $$\left| \{ M \cap P \mid M \in \text{Mod}(F) \} \right|,$$ which we call *projected model count* of F wrt P.

3.3 Output $\log_{10}$-Notation

We say that an output is in $\log_{10}$-Notation if the result of the problem `MC`, `WMC`, or `PMC`, respectively, is the value v, but the solver outputs $\log_{10} v$.

4 House Keeping

By March 2020, we hope to provide runtime scripts and a few test cases that allows you to check compatibility with the input format `github.com/daajoe/mc2021`. While the requirements below might seem tedious, it really is necessary to allow for stable runs with cluster systems were specialized benchmarking frameworks cannot be installed due to restrictions or strict policies.

Last year, some runtime scripts were writing large files onto disks (which resulted in problems on one cluster; as we ran over-quota) or would not terminate sub-solvers, if we had to terminate a program, or would hard-code timeouts.

For simplicity, we suggest that adapt our provided wrapper scripts (see `https://github.com/daajoe/mc2021`). We will also provide scripts to allow for quick transformation of the old format. To avoid errors in the tooling, we suggest that teams consider the use of a recent fuzzing tool for testing the solver, e.g., [UWK20].

Note that we will test whether the submission (both solver and starting script) can handle the requirements correctly. If the submission fails a basic test, we reserve the right to disqualify the solver.

4.1 Output: Return Codes

We suggest that the submission passes return codes from the pre-processors/ solvers and that the submission gives a *return code zero only if the instance was solved*.

4.2 Handling of Temporary Files

We will specify a temporary path for storing files, we will provide the location in both the environment variable `$TMPDIR` and parameter `--tmpdir=path` to allow easy handling in compiled submissions or submissions that use wrapper scripts. The path will preferably be space on the shared memory (`/run/shm`). You are *not* allowed to try to store files anywhere else other than in the given location for temporary files. If you create a temporary file consider to use `TMPDIR=$(dirname $(mktemp -u -t tmp.XXXXXXXXXX))` in bash to create a temporary file or the according function in your script language. *If we find `rm filename`, `rm -r projection`, or `rm *` or something similar in your script, your submission will be disqualified.* We do not want anyone to use such submission scripts.

4.3 Signal Handling

If the solver receives the unix signal 15/SIGTERM, the solver has to terminate itself and any child process within 2 seconds. The submission may specify, which sub-solver was terminated. You do not required to cleanup temporary files. If the solver receives the unix signal 2/SIGINT, the solver has to terminate within 10 seconds, but shall cleanup any temporary files.
Since various scripts last year produced a variety of errors, we *highly encourage* the developers to create submission scripts, which provide at least basic fail safe behavior. In order to save valuable time when preparing submissions, we provide you with templates. The files `mysubmissionX.sh` and `mysubmissionX.py` in the github repository (`https://github.com/daajoe/mc2021`) should provide a good idea on to run the instances. Feel free to adapt the scripts or copy from them.

4.4 Tooling/Solver Calls

Please, prepare your submission with StarExec virtual machine. For details, see: `https://www.starexec.org/starexec/public/about.jsp`.

In addition, place a start script named as your submission-name ending with a file extension in the usual way. The wrapper/ELF should be capable of reading the instance both from `stdin` and a textfile, which is given as first (unnamed) parameter. We will provide the following parameters:

Parameter/Env	Description
`$TMPDIR`	Path to the temporary directory.
`--tmpdir=/givenpath`	Path to the temporary directory.
`--maxrss=X`	Size (integer) of the max available RAM in GB.
`--maxtmp=X`	Size (integer) of the max available TMP in GB.
`--timeout=X`	Runtime limit in seconds.
`--task=X`	Solver Task $X \in \{\text{mc}, \text{wmc}, \text{pmc}\}$.

5 Submission Requirements

The number of submissions is limited to two per team. An author may be part of only one exactly one team. This requirement is valid over all tracks. Note that is is a quite weak limitation as we announce the problem to the script/solver and a team can prepare a submission for multiple tracks.

5.1 Type of Solver

Portfolio solvers or pre-processors may be used, but the submission shall announce which tool is running in each step (see above).

Track 1: We only allow exact solvers. If your solver does not allow for arbitrary precision, you need to specify it in the solution accordingly.

Track 2: We allow for exact as well as approximate solvers.

Track 3: We allow for exact as well as approximate solvers. Exact solvers will receive a preference bias in the measurement.

5.2 Publication of the Submission

Each team has to publish all final submissions on the public data repository Zenodo two weeks after the submission deadline. The Zenodo repository has to fulfill the following requirements:

1. Provide a recognizable name.
   We suggest "MC2021 (Submission): Team/SubmissionName").
2. Provide the own solver as binary (ELF 64bit). The binaries need to be *statically linked* (no external dependencies to libc etc.). All necessary submission scripts have to be included. If the submission uses an external third party library, which is not under an open source license, we expect that a script for downloading the library is included. Libraries, which require a registration or are not free for academic use, are not allowed.
3. We highly encourage solver developers to publish the source code in the data repository and give a standard open source license.

4. Place a PDF that briefly describes your submission. Use the article style, single column, and at maximum 2 pages.

6 Expected Timeline

Date	
January 10, 2021	Announcement of the challenge (Tracks)
February 19, 2021	Format Description Online
February 21, 2021	Call for Benchmarks
March 15, 2021	Test Instances are available
March 19, 2021	Benchmarks (submit ASAP)
	Please use the Form `tinyurl.com/fm7ucg3z`
March 25, 2021	Intent to Participate
	Please use the Form `tinyurl.com/2ys3tz4v`
April 10, 2021	Public Instances are available
April 15, 2021	Submission of the Solvers (Feedback Phase starts)
April 30, 2021	Last Update to submissions (End of Feedback Phase)
May 10, 2021	Evaluation Phase starts
May 30, 2021	Submissions + Descriptions (Zenodo)
	Please use the Form `tinyurl.com/10v5vw38`
July 8, 2021	Presentation of the Results

7 Arena

The following briefly describes the evaluation setting of the competition.

7.1 Instances

The instance set will contain 200 instances of which we will make the even numbered instances (`trackX_000.ecnf`, `trackX_002.ecnf`, ..., `trackX_198.ecnf`) public for all participants during the testing phase of the solvers. The private instances (`trackX_001.ecnf`, `trackX_003.ecnf`, ..., `trackX_199.ecnf`) will be used for the final evaluation and disclosed after the submission deadline. We will run two exact solvers on the competition instances with a timeout of at most 3 days to pre-determine the solution. We will only use instances on which the solvers agreed if the instance was solved within the timeout. However, we may use instances, which could not be solved. Note that there will be instances of which we do not know the solution. *We will not apply pre-processing to the instances. It might make sense to spend some time on choosing the right pre-processor and decide on the runtime you want to spend for pre-processing.*

7.2 Cluster

As we have seen last year, we might not be able to fix the cluster environment in advance due to unexpected external circumstances. However, we will use the clusters in the following order and only move to the next, in case of backup.

1. StarExec: `starexec.org/starexec/public/machine-specs.txt`
2. Taurus (ISLAND 4–6 Haswell):
   `doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium node_topology`; `Description`
3. Cobra:
   ```
   model name :  Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
   cache size :  30720 KB
   Linux 4.4.0-184-generic
   ldd (Ubuntu GLIBC 2.23-0ubuntu11.2) 2.23
   gcc (Ubuntu 5.4.0-6ubuntu1 16.04.12) 5.4.0 20160609
   ```

We run instances at base-frequency, at most one process per memory channel, and transparent huge pages activated.

7.3 Timeouts

Track 1: 3600 seconds (total 2 days?)

Track 2: 3600 seconds (total 2 days?)

Track 3: 3600 seconds (total 2 days?)

While you may use the linux program `timeout` *in your script. Please, do not hardcode timeouts. Make sure that your submission script can handle the given timeout in seconds (see above).*

7.4 Judges

tba

7.5 Evaluation Measure

Disqualification The solver has to satisfy all requirements specified above and to pass few simple test cases. Among those test cases will be additionally 10 instances (5 public, 5 private) of which the solution is known. If a solver fails to produce a correct solution (within the expected margin), it will be disqualified.

Accepted Solutions A submission has to mark an instance as solved (return code 0).

$$\text{solved}(I, S) := \begin{cases} 1, & \text{if solver } S \text{ outputs return code 0 on instance } I \\ 0, & \text{otherwise} \end{cases}$$

An instance that remains unsolved will be ignored. An instance will be marked as accepted if the outputted solution is within a relative margin of error depending on the track as given in Table 1.

Track	t
Model Counting Track	0.1%
Weighted Model Counting Track	1.0%
Projected Model Counting Track	1.0%

Table 1: Expected accuracy of participating solvers.

A submission may output incorrect or not accepted solutions to solved instances. However, all instances that were not accepted solutions will be listed in the report. Furthermore, a submission might be disqualified, if it outputs a not accepted solution to more than 20 solved private instances and more than 5 solved private where the solution was unknown.

Let v_o refer to the outputted result by the submission ("observed value") and v_e to the pre-computed result ("expected value") in $\log_{10}$ notation. We measure the margin of error by the relative percentage difference:

$$\text{RLPD}(v_e, v_o) := \frac{100}{\log_{10}(e)} \cdot \begin{cases} |(v_o - v_e)|, & \text{if } v_o \text{ in } \log_{10} \text{ notation} \\ |\log_{10}(v_o) - v_e|, & \text{otherwise.} \end{cases}$$

We consider an instance as accepted

$$\text{accept}(v_o, v_e) := \begin{cases} 1, & v_e = \text{unknown and } v_o \neq \text{unknown;} \\ 1, & \text{RLPD}(v_e, v_o) \leq t; \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

where $S(I)$ refers to the output of S on the instance I.

Scoring Function The task for a submission s is to optimize the number of acceptably solved instances. In other words, for the given set I of private instances, to maximize the following function:

$$\text{score}(S) := \sum_{i \in I} \Big(\text{accept}(v_e^I, v_o^{I,S}) \cdot \text{solved}(I, S) \Big)$$

where v_e^I refers to the pre-computed solution for instance I and $v_o^{I,S}$ the solution given by submission S on instance I.

Tie Breaking We will not break ties. If two submissions receive a tie, we will assign the same places and the next place will be vacant.

8 Contact

- Main contact: `benchmarks@mccompetition.org`
- Competition website:
  `https://mccompetition.org/2021/mc_description`
- For up-to-date information consult:
  the Webpage at `https://mccompetition.org/news` or
  our Slack Channel at `https://mccompetition.slack.com/`

Organizers

- Markus Hecher (TU Wien & Uni Potsdam)
- Johannes K. Fichte (UC Berkeley)

Scientific Partners

- Adnan Darwiche (University of California at Los Angeles)
- Arthur Choi (University of California at Los Angeles)
- Armin Biere (Johannes Kepler Universität Linz)
- Fahim Bacchus (University of Toronto)
- Jean-Marie Lagniez (CNRS at Centre de Recherche en Informatique de Lens and Université d'Artois)
- Kenji Hashimoto (Nagoya University)
- Kuldeep S. Meel (National University of Singapore)
- Markus Hecher (TU Wien)
- Johannes K. Fichte (TU Dresden)
- Mate Soos
- Norbert Manthey
- Pierre Marquis (CNRS at Centre de Recherche en Informatique de Lens and Université d'Artois)

References

[BFH+20] Tomáš Balyo, Nils Froleyks, Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. University of Helsinki, Department of Computer Science, 2020.

[FHH20] Johannes K. Fichte, Markus Hecher, and Florim Hamiti. Model counting competition 2020: Competition instances. Zenodo, November 2020.

[GRS+16] Vibhav Gogate, Tahrima Rahman, Somdeb Sarkhel, David Smith, and Deepak Venugopal. Uai 2016 inference evaluation. `http://www.hlt.utdallas.edu/~vgogate/uai16-evaluation/tuning.html`, 2016.

[JBRS12] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. In *AI Magazin*. The AAAI Press, 2012.

[KS05] Henry Kautz and Tian Sang. Model counting using component caching and clause learning. `https://www.cs.rochester.edu/u/kautz/Cachet/cachet-wmc-1-21.zip`., 2005.

[SRSM19] Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. GANAK: A scalable probabilistic exact model counter. In Sarit Kraus, editor, *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*, pages 1169–1176, Macao, China, 2019. IJCAI.

[TCC+93] Michael Trick, Vavsek Chvatal, Bill Cook, David Johnson, Cathy McGeoch, and Bob Tarjan. The 2nd DIMACS implementation challenge: 1992–1993 on NP hard problems: Maximum clique, graph coloring, and satisfiability. `http://archive.dimacs.rutgers.edu/pub/challenge/sat/benchmarks/`, 1993.

[UWK20] Muhammad Usman, Wenxi Wang, and Sarfraz Khurshid. Testmc: Testing model counters using differential and metamorphic testing. In Claire Le Goues and David Lo, editors, *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE'20)*, pages 709–721, 2020.