Automated Benchmarking of KR-Systems

Christoph Redl

redl@kr.tuwien.ac.at

November 28, 2016

Motivation

Benchmarking is a time-consuming task

- Benchmarking is an important part of scientific work on solving techniques for KR systems.
- The implementation of hand-crafted scripts for each benchmark problem is cumbersome.
- Most benchmarks are similar such that the process appears to be largely automatable.

Motivation

Benchmarking is a time-consuming task

- Benchmarking is an important part of scientific work on solving techniques for KR systems.
- The implementation of hand-crafted scripts for each benchmark problem is cumbersome.
- Most benchmarks are similar such that the process appears to be largely automatable.

Issues

- However, automating the process is not straightforward.
- While there are similarities between benchmarks, details may differ:
  - Systems/configurations to compare.
  - Input/Output of such systems.
  - Values to measure.
  - ...

Motivation

Goal

- Identify similarities between benchmarks.
- Create a benchmarking system with a default behavior which is good for many benchmarks . . .
- . . . but also flexible to be adaptable to a large variety of benchmarks.

Motivation

Goal

- Identify similarities between benchmarks.
- Create a benchmarking system with a default behavior which is good for many benchmarks ...
- ...but also flexible to be adaptable to a large variety of benchmarks.

Contributions

- Formalization of benchmarks in a customizable fashion.
- Design of a benchmark system.
- Implementation in the ABC-system.

Formalization of Benchmarks

Definition

A benchmark problem is a tuple

$$B = \langle (I_1, \ldots, I_\ell), C, o, a \rangle$$

where

- $I_1, \ldots, I_\ell \subseteq \mathcal{I}$ is a list of sets of instances,
- $C \subseteq \mathcal{C}$ is a list of configurations,
- o is an output builder function, and
- a is an aggregation function.

Formalization of Benchmarks

Example

Suppose we want to compare the runtime of multiple SAT-solvers.

Then:

- $\mathcal{I}$ is the set of all syntactically wellformed DIMACS files
- $\mathcal{C}$ is a set of SAT solver calls
- $\mathcal{D}$ is the set of all floating point values

Suppose we have two different instance sizes 1 and 2 (wrt. the number of variables) containing $|I_1| = |I_2| = 2$ instances each.

Then:

- I_1, I_2 are sets of SAT-instances to be run
- The configurations are $C = (minisat, clasp, manysat)$

Evaluating Instances

The (benchmark-independent) evaluation function ϵ maps an instance and a configuration to the output from an abstract output domain $\mathcal{O}$ (e.g. the set of all strings).

Evaluating Instances

The (benchmark-independent) evaluation function ϵ maps an instance and a configuration to the output from an abstract output domain $\mathcal{O}$ (e.g. the set of all strings).

Definition

The evaluation function $\epsilon\colon \mathcal{I} \times \mathcal{C} \to \mathcal{O}$ associates each instance $i \in \mathcal{I}$ and configuration $c \in \mathcal{C}$ with an output from $\mathcal{O}$.

Evaluating Instances

The (benchmark-independent) evaluation function ϵ maps an instance and a configuration to the output from an abstract output domain $\mathcal{O}$ (e.g. the set of all strings).

Definition

The evaluation function $\epsilon\colon \mathcal{I} \times \mathcal{C} \to \mathcal{O}$ associates each instance $i \in \mathcal{I}$ and configuration $c \in \mathcal{C}$ with an output from $\mathcal{O}$.

However, normally not the full output is relevant for the benchmark results.

Evaluating Instances

The (benchmark-independent) evaluation function ϵ maps an instance and a configuration to the output from an abstract output domain $\mathcal{O}$ (e.g. the set of all strings).

Definition

The evaluation function $\epsilon\colon \mathcal{I} \times \mathcal{C} \to \mathcal{O}$ associates each instance $i \in \mathcal{I}$ and configuration $c \in \mathcal{C}$ with an output from $\mathcal{O}$.

However, normally not the full output is relevant for the benchmark results.

Definition

For a benchmark with domain $\mathcal{D}$, an output builder o is a function $o\colon \mathcal{O} \to \mathcal{D}^n$, where n is the number of values per instance and configuration measured by o.

Evaluating Instances

Example (cont'd)

Continuing the previous example (SAT-solvers), the output domain $\mathcal{O}$ contains all possible outputs consisting of:

- the standard output (e.g. a satisfiability flag, possibly models),
- the standard error output (e.g. log information),
- the return value of the call (e.g. indicating satisfiability), and
- meta-information (e.g. observed runtime and memory consumption).

Evaluating Instances

Example (cont'd)

Continuing the previous example (SAT-solvers), the output domain $\mathcal{O}$ contains all possible outputs consisting of:

- the standard output (e.g. a satisfiability flag, possibly models),
- the standard error output (e.g. log information),
- the return value of the call (e.g. indicating satisfiability), and
- meta-information (e.g. observed runtime and memory consumption).

The output builder o extracts from this information the observed runtime and maximum memory usage and returns it as two floating point values, hence $n = 2$.

Evaluating Instances

Example (cont'd)

Continuing the previous example (SAT-solvers), the output domain $\mathcal{O}$ contains all possible outputs consisting of:

- the standard output (e.g. a satisfiability flag, possibly models),
- the standard error output (e.g. log information),
- the return value of the call (e.g. indicating satisfiability), and
- meta-information (e.g. observed runtime and memory consumption).

The output builder o extracts from this information the observed runtime and maximum memory usage and returns it as two floating point values, hence $n = 2$.

For $C = (minisat, clasp, manysat)$ and instance $i_{2,1} \in I_2$,
we have that $o(\epsilon(i_{2,1}, c))$ evaluates to a vector of floating point values of length 2 for each $c \in C$.

Representing the Output as Table

The results of individual instances can then be arranged in a table:

Definition (Instance Results Table)

The instance results table $T_I(B)$ associated with a benchmark B is the unique table of size $|I| \times |C| \cdot n$ such that $(ti_{u,v \cdot n+1}, \ldots, ti_{u,v \cdot n+n}) = o(\epsilon(I_u, C_{v+1}))$ for all $1 \leq u \leq |I|, 0 \leq v < |C|$.

Representing the Output as Table

The results of individual instances can then be arranged in a table:

Definition (Instance Results Table)

The instance results table $T_I(B)$ associated with a benchmark B is the unique table of size $|I| \times |C| \cdot n$ such that $(ti_{u,v \cdot n+1}, \ldots, ti_{u,v \cdot n+n}) = o(\epsilon(I_u, C_{v+1}))$ for all $1 \leq u \leq |I|, 0 \leq v < |C|$.

However, normally the final results should not show individual instances, but aggregated results, where the aggregation might be benchmark-dependent.

Definition

An aggregation function for a benchmark B as by Definition 1 is a function $a\colon 2^{\mathcal{D}^{|C| \cdot n}} \to \mathcal{D}^{|C| \cdot n}$.

Representing the Output as Table

The results of individual instances can then be arranged in a table:

Definition (Instance Results Table)

The instance results table $T_I(B)$ associated with a benchmark B is the unique table of size $|I| \times |C| \cdot n$ such that $(ti_{u,v \cdot n+1}, \ldots, ti_{u,v \cdot n+n}) = o(\epsilon(I_u, C_{v+1}))$ for all $1 \leq u \leq |I|$, $0 \leq v < |C|$.

However, normally the final results should not show individual instances, but aggregated results, where the aggregation might be benchmark-dependent.

Definition

An aggregation function for a benchmark B as by Definition 1 is a function $a\colon 2^{\mathcal{D}^{|C| \cdot n}} \to \mathcal{D}^{|C| \cdot n}$.

Definition (Aggregated Results Table)

The aggregated results table $T_A(B)$ associated with a benchmark B has rows

$$r_i = a(\{T_I(B)_{s+1}, \ldots, T_I(B)_{s+|I_i|}\}) \text{ for all } 1 \leq i \leq \ell,$$

where $s = \Sigma_{1 \leq j < i} |I_j|$ is the number of instances preceding instance group i.

Representing the Output as Table

Example (cont'd)

Continuing the previous example (SAT-solvers), each row of $T_I(B)$ consists of $|C| \cdot 2$ columns because the output builder returns two values (runtime and memory consumption) for each instance and configuration.

Suppose the instance results table looks as follows:

$T_I(B)$	minisat		clasp		manysat		
	runtime	memory	runtime	memory	runtime	memory	
$T_I(B)_1$	0.04	0.10	1.21	1.00	0.51	0.40	I_1
$T_I(B)_2$	1.64	0.90	5.23	2.20	0.20	0.20	I_1
$T_I(B)_3$	6.44	2.40	3.53	1.30	1.12	5.00	I_2
$T_I(B)_4$	7.70	2.80	6.11	3.30	8.32	7.20	I_2

Representing the Output as Table

Example (cont'd)

Continuing the previous example (SAT-solvers), each row of $T_I(B)$ consists of $|C| \cdot 2$ columns because the output builder returns two values (runtime and memory consumption) for each instance and configuration.

Suppose the instance results table looks as follows:

$T_I(B)$	minisat		clasp		manysat		
	runtime	memory	runtime	memory	runtime	memory	
$T_I(B)_1$	0.04	0.10	1.21	1.00	0.51	0.40	I_1
$T_I(B)_2$	1.64	0.90	5.23	2.20	0.20	0.20	
$T_I(B)_3$	6.44	2.40	3.53	1.30	1.12	5.00	I_2
$T_I(B)_4$	7.70	2.80	6.11	3.30	8.32	7.20	

The aggregation function a is separately applied to $\{T_I(B)_1, T_I(B)_2\}$ and $\{T_I(B)_3, T_I(B)_4\}$ and computes the columnwise average values. As above, for a table $T_A(B)$ let $T_A(B)_k$ be its k-th row.

Representing the Output as Table

Example (cont'd)

This yields table $T_A(B)$ with two rows:

$T_A(B)$	minisat		clasp		manysat	
	runtime	memory	runtime	memory	runtime	memory
$T_A(B)_1$	0.84	0.50	3.22	1.60	0.36	0.30
$T_A(B)_2$	7.07	2.60	4.82	2.30	4.72	6.10

Implementation

The ABC-System

- Automated benchmarking based on HTCondor: `https://github.com/credl/abcbenchmarking`. A detailed system documentation is included in the repository.
- Implemented as a set of shell scripts.
- Based on HTCondor (`https://research.cs.wisc.edu/htcondor`) and the R-system (`https://www.r-project.org`).

Implementation

The ABC-System

- Automated benchmarking based on HTCondor: `https://github.com/credl/abcbenchmarking`. A detailed system documentation is included in the repository.
- Implemented as a set of shell scripts.
- Based on HTCondor (`https://research.cs.wisc.edu/htcondor`) and the R-system (`https://www.r-project.org`).

Basic usage

In order the user the system, add its patch to the `$PATH` variable, and for each benchmark create a file `run.sh` which:

1. Include the ABC header file:

   ```
   source run_header.sh
   ```

2. Call the `run` method with appropriate parameters (see system documentation and the following example).

Implementation

Example

Consider the following scenario:

- Our instances are given by all files of type `*.dlv` (DLV programs) in the directory `instances`.
- We compare the configurations `dlv` and `dlv -n=1`.

Implementation

Example

Consider the following scenario:

- Our instances are given by all files of type `*.dlv` (DLV programs) in the directory `instances`.
- We compare the configurations `dlv` and `dlv -n=1`.

This is implemented in the following file `run.sh`:

```
source run_header.sh

instances="instances/*.dlv"
configurations="dlv;dlv -n=1"
combine="CONF INST"
benchmarkname="dlv"
aggregationfunc=""
outputbuilder=""

run "$instances" "$configurations" "$combine" \
    "$benchmarkname" "$aggregationfunc" "$outputbuilder"
```

Implementation

Example (cont'd)

Assuming that there are three groups of 10 instances of sizes 1, 2 and 3, the output of the call `./run.sh` is a table of the following form:

```
1 10 0.12 0 0.07 0
2 10 1.08 0 43.15 1
3 10 22.81 0 270.01 9
```

Implementation

Example (cont'd)

Assuming that there are three groups of 10 instances of sizes 1, 2 and 3, the output of the call `./run.sh` is a table of the following form:

```
1 10 0.12 0 0.07 0
2 10 1.08 0 43.15 1
3 10 22.81 0 270.01 9
```

This ABC system allows for an automatic translation of this table to LaTeX code:

```
\begin{table}[t]
\scriptsize
\centering
\begin{tabular}[t]{r|rrr}
\hline
instance & \verb+dlv+ & \verb+dlv -n=1+ \\
\hline
1 (10) & 0.12 (0) & 0.07 (0) \\
2 (10) & 1.08 (0) & 43.15 (1) \\
3 (10) & 22.81 (0) & 270.01 (9) \\
\hline
\end{tabular}
\caption{Benchmark Results}
\label{tab:results}
\end{table}
```

Figure: Benchmark Results: LaTeX Code

System Architecture

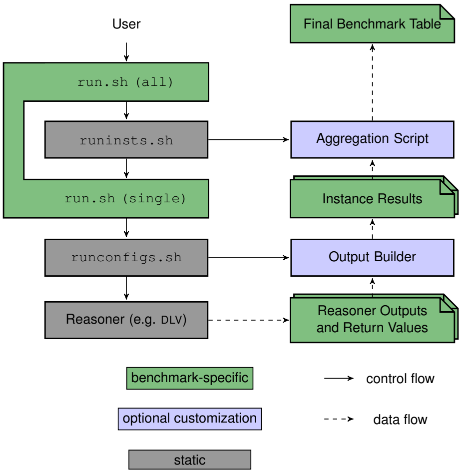

Figure: ABC System Architecture

Further Features of the System

Customization

- Custom output builders implemented as shell script.
- Custom aggregation functions implemented either by
  - specifying the function for each column, or by
  - providing a completely customized R script.

Further Features of the System

Customization

- Custom output builders implemented as shell script.
- Custom aggregation functions implemented either by
  - specifying the function for each column, or by
  - providing a completely customized R script.

Output processing

- Scripts for processing final benchmark tables,
  e.g. projection, joining, etc., and
- E-mail notifications upon finishing benchmarks.

Further Features of the System

Customization

- Custom output builders implemented as shell script.
- Custom aggregation functions implemented either by
  - specifying the function for each column, or by
  - providing a completely customized R script.

Output processing

- Scripts for processing final benchmark tables,
  e.g. projection, joining, etc., and
- E-mail notifications upon finishing benchmarks.

Comparisons

- Results may be (statistically) compared to previous results.

Conclusion

Benefits of our system

- Largely automates benchmarking from the evaluation of individual instances up to generating the final LaTeX table.
- Focused on command-line tools including many KR-tools.
- Default settings are good for many benchmarks.
- But customizable to allow for adaption to less standardized benchmarks.

Future Work

- The benchmark specification is declarative, thus a declarative language might be supported as frontend.
- Additional backends (as an alternative to HTCondor) might be supported.