

A Tool for Benchmarking Command-line Systems¹

Wolfgang Faber

Dipartimento di Matematica, Università della Calabria, 87030 Rende (CS), Italy

Email: faber@mat.unical.it

SOMMARIO/ABSTRACT

In this paper we describe a system for representing and running benchmarks. The tool was started for benchmarking AI systems, but is actually usable for arbitrary command-line systems. It is fully implemented and available on the web.

Keywords: Benchmarking Tool, Command-line Systems

1 Introduction

Whoever has implemented tools for Artificial Intelligence, has faced the problem of benchmarking them, be it to demonstrate the feasibility of the approach, or to compare it to existing systems. Several considerations have to be done, for example suitable and representative instances have to be identified, resource restrictions have to be imposed, and statistical methods for evaluating the data have to be fixed.

While all of these issues are very important, you also have to actually *run* the benchmarks. We are aware of people, who actually do this manually. Most others hack together or re-use some ad-hoc scripts. Both approaches are not very satisfactory, the former because it is clearly time-consuming and error-prone, the second because new benchmarks often require changing some code in the script. It is also easy to introduce hard-to-debug errors in this way. Moreover, issues like evaluation of obtained data are usually not covered by such script-suites.

This paper describes BMTOOL, a tool for aiding the researcher to accomplish this task. In particular, this tool differentiates two phases of benchmarking: 1. Actually running benchmarks, and 2. evaluating obtained results. The tool allows for a declarative representation of benchmark instances, matching patterns and obtained results. It is

*This work was supported by an APART grant of the Austrian Academy of Sciences and the European Commission under projects IST-2002-33570 INFOMIX, IST-2001-37004 WASP.

designed to be generic, and should work for almost any benchmarking task for command-line systems. While the system had been developed for benchmarking an AI system, it can be used (and actually is used) for benchmarking arbitrary command-line systems.

2 System Data

We will start by describing the input and output data dealt with by BMTOOL.

2.1 Problem Descriptions

The descriptions of the problems or testcases to be solved, reside in a `problem` file, the syntax of which is as follows:

```
problem(2QBF1)
description(2QBF, 1000 univ, 20 exist vars,)
description(10000 clauses, 5CNF)
files(2qbf_1000x_20y_10000rules_5cnf.dl)
options(-FB)
```

Here, `problem` is a label describing the testcase, in the example `2QBF1`. The next two lines give a natural language description of the testcase. Then, the input files are specified, the filenames should be separated by spaces and relative to the location of the `problem` file itself. The last line fixes the options to be passed to the executable. Multiple tests must be separated by an empty line.

2.2 Extraction Rules

Another file specifies the values to match in the output (on the standard error stream). As an example, a part of the output of a GNU time command is as follows:

```
3.61user 2.62system 2:10.02elapsed 4%CPU
```

The first two numbers (user and system time) are output in seconds, while the real or elapsed time is output in integer minutes and seconds, separated by a colon. Matching these values can be accomplished by the following extraction rule file:

```

USER      "@NUMBER@user"
SYS       "@NUMBER@system"
REAL     "@TIME@elapsed"

```

Each line contains a label for the measure, followed by a TAB, and a regular expression, which must contain one of the special symbols @NUMBER@ and @TIME@.

2.3 Runs, Intermediate Results, Errors

In order to avoid distortions of the data by singularities, BMT00L can be instructed to run each combination of executable and test case n times.

Often it is useful or even necessary to separate the benchmarking proper from processing the obtained data. For instance, one could decide to collect data of as many parameters as possible, while for subsequent analysis they will be considered separately. To this end, BMT00L can be instructed to produce an *intermediate* file, which it can later re-use. If a testcase exits with an error code (non-zero), this is recorded in an error file.

2.4 Output Schemata

For producing output based on the gathered data, one has to specify an output schema. This is done by specifying a label, which is to be associated with this output, a TAB, a string in *printf* format, and specification(s) of how the value is to be computed, for each numerical value in the *printf* string. The latter should be written in Tcl format, where several predefined functions (like *arithmetic_mean*, *standard_deviation*, *plus*, *min*, *max*) exist. The measured values themselves should be “wrapped” using the function *measure*. All of these functions are defined over vectors of measures (the n runs of a particular executable and test). For example, the following will produce the average runtime of the sums of USER and SYS timings for each executable and test.

```

AVGTIME    %.2f    [arithmetic_mean \
                [plus [measure USER] [measure SYS]]]

```

2.5 Output Format

Four output formats are currently available: ASCII table, L^AT_EXtable, list and tabbed table. An ASCII table will produce the following output, which is for example useful for sending by email:

```

[0]: /usr/bin/time executable1
[1]: /usr/bin/time executable2

```

```

-----+-----+-----+
| [0] | [1] |
-----+-----+-----+
Problem 1 | 102.56 | 99.12 |
Problem 2 | 4.43 | 0.01 |
-----+-----+-----+

```

```

Problem 1:
Description 1

```

```

Problem 2:
Description 2

```

A L^AT_EXtable is suitable for importing into a paper:

	Executable 1	Executable 2
Problem 1	102.56	99.12
Problem 2	4.43	0.01

If more than one output schema is defined, one table is created for each. Lists and tabbed tables are useful for further processing by scripts or statistics software.

3 Installation and Invocation

The tool is available via the website <http://www.wfaber.com/software/bmtool/>. It is written in the language Tcl, which needs to be installed. It is tested with Tcl versions 8.4, but may run also with earlier 8.x versions. Tcl can be obtained via <http://www.tcl.tk/>. Note that you do not need Tk for running BMT00L. You should make sure that *tclsh* is in your path.

3.1 Running Benchmarks

The tool can be invoked as follows:

```

bmtool [--prefix <prefix>] [--postfix <postfix>]
        [--errorfile <filename>] [--intermediatefile <filename>]
        [--stopafterfirstfailure] --extractionschemafile <filename>
        --problemfile <filename> --outputschemafile <filename>
        [--backend=(table|list|latex|tabtable)]
        -n=<N> [--] <executable> ...

```

--prefix (resp. --postfix) will cause the specified string to be appended before (resp. after) each executable. We use --prefix to specify /usr/bin/time and --postfix for options common to all executables. By --stopafterfirstfailure one will cause the testing for each executable to be terminated as soon as one test failed. This is useful for running tests until the first time-out occurs.

3.2 Processing Results

If one just want to postprocess already obtained data, the switch --postprocess must be specified and the options and argument regarding execution are omitted.

```

bmtool --postprocess (-i|--intermediatefile) <filename>
        (-o|--outputschemafile) <filename>
        [--backend=(table|list|latex|tabtable)]

```

4 Conclusions and Future Work

The tool has been used frequently since 1999 for various benchmarks (mainly DLV and gcc), and should run very stable on arbitrary UNIX-like platforms (including Mac OS X). However, we have never used it under Windows, and it is not clear whether starting subprocesses in the way we used them is supported by Tcl under Windows.

The data format has been an ad hoc decision. A future goal is to use XML as in- and output. With large datasets, performance of the postprocessing can degrade considerably. This is probably due to a suboptimal use of Tcl. Work should be done on optimizing the code.