Toulbar2
Testing and benchmarking doc

toulbar2 is able to perform benchmarking and tests using cmake and perl scripts (cf. `cmake_script` directory and `misc/script` directory). The current document describes the testing procedure in the toulbar2 project.

i) Validation tests

The validation tests perform "unit" tests (testing one mode of toulbar2). If you modified toulbar2 code, you may quickly want to check if the new code produces the correct result for a given set of wcsp problem. This is easily achieved by executing the following commands :

```
mkdir build
cd build
cmake ..
make
make test
```

The cmake script is going to scan the `./validation` directory located in the root of the project. For each file found with a "`.wcsp`" extension, a unit test will be generated with an associated error detection. The error detection happens during test execution, if a string defined in the test is not found in the test output. The "end." string (the final word in a toulbar2 execution) is used by default for each test. However, if a `.ub` file containing the right optimal cost is located in the same directory of a given instance, this value will be extracted from the files and used to detect inconsistent result for the related test.

Overall, for each `foo.wcsp` file if a `foo.ub` file is found in the same directory, a test is produced and an error raised if the optimum computed by the toulbar2 execution is different from the value in the `.ub` file.

Exemple: `validation/default/example.wcsp` and `validation/default/example.ub` (containing just "`27`") exist and toulbar2 will be tested on the wcsp file with a detected error if the optimal cost differs from 27. Actually, ctest launches `toulbar2 example.wcsp` and extracts the output. By default "`end.`" is sought or otherwise an optimal cost identified through a defined regular expression is checked (cf. toulbar2 output).

```
"...
Optimum: 27 in 106 backtracks and 215 nodes ( 278 removals by DEE) and
0.02 seconds.
```

```
end.
..”
```

NB: a timeout is used for each test. The default timeout value is 20 seconds and it can be modified in two ways:

1. by editing the `test-opt.cmake` file located in each test directory. Here is an example of a `test-opt.cmake` file:

```
set (command_line_option -B=0 -v -e: )
# test timeout ( used for all wcsp founded in the directory
# in this case test timeout is set to 100 second

set (test_timeout 100)

#regex to define successful end.
# cmake var $UB will be grab into the test output using the
# following regular expression
set (test_regexp  "Optimum: ${UB}")
```

NB: when a `test-option.cmake` file cannot be found in a given directory, test_timeout, option, location and test activation are globally set by the `cmake` or `ccmake` command s(i..,e “`ccmake ..`” from build directory). One can customise the default values by modifying the following variables:

```
Default_bench_timeout       30        # default timeout (seconds)
Default_regexp             end.       # default regexp
Default_test_option        -v         # default toulbar2 option
Default_validation_dir     validation # default root directory
#----------------
COVER_TEST                    OFF  # flag for cover test activation
Default_cover_dir             cover
#----------------
BENCH                         OFF
Default_BenchDir              benchmarks
Default_BenchFormat           wcsp
Default_bench_option          TOULBAR2_OPTION
Default_bench_regexp          “test ok”
```

`Command_line_option` and `test_regexp` can be modified for specific test purposes: you can customize your own test specifying your own option, duration and location (relative path from the root directory of the toulbar2 project).

ii) Cover test

Cover tests have been developed to check non-regression of new toulbar2 binary releases. We want to test a given benchmark set with different toulbar2 options in order to check if new codes modifications haven't impacted the global behaviour of Toulbar2.

In the `cover` directory directory, in addition to the `test_option.cmake`, a `cover_test.cmake` file contains a double entry list allowing to perform several tests with different options on the same instance. A given "`.wcsp`" file declared in the `$instances` array will generate N tests corresponding to each element of the command lines stored into the cmake array named as the benchmark file.

The following example describes a minimal `test_option.cmake`:

```
# The file must define a "instances" variable as a set of benchmark
# files. For each file, a specific set of options can be used.
# Example: if "instances" contains "foo.wcsp", the "foo.wcsp"
# variable defines the list of options to test on this file: each
# element of the foo.wcsp variable will create a unit test with the
# foo.wcsp instance and the toulbar2 options given in the "foo.wcsp"
# variable.

SET(instances CELAR6-SUB0.wcsp CELAR6-SUB1.wcsp)

# please beware to use one space between each file

SET (CELAR6-SUB0.wcsp
     "-A"
     "-A -S -ub=160"
     "-A -V"
     "-A -V -ub=160"
     "-A=10 -V"
     "-A=10"
     "-A=10 -V"
     "-A=10 -v"
     "-B=0"
     "-B=1"
     "-B=1 -Z"
     "-B=1 -Z -R=1"
     "-B=2"
     "-B=3"
     )
```

```
SET (CELAR6-SUB1.wcsp
      "-A"
      "-A -V"
      "-B=2"
      "-B=3"
      )
```

This example will generate 14 and 4 different tests on `CELAR6-SUB0.wcsp` and `CELAR6-SUB1.wcsp`. The whole set can be tested using the `make test` command.

3) Benchmarking

Discrete graphical model optimisation is an NP Hard problem and experimentation on various benchmark provides essential metrics. In this section, we want to apply toulbar2 onto a given set of instances and extract metrics from toulbar2 output and generate a report containing all metrics on all instances and global information such as the overall total number of solved instances. For this, perl has to be installed on your workstation. On debian derived machines this can be achieved by executing "`sudo apt-get install perl perl-modules` ".

The perl scripts developed for benchmarking are:

```
misc/script/run_test.pl* /* simple perl launcher for toulbar2 bench test
misc/script/exp_opt.pl*   /
misc/script/make_report.pl*
misc/script/MatchRegexp.txt
```

They use the following perl module (available in debian based distributions in the `perl-modules` debian package):

```
use File::Basename;
use List::Util ;
use Getopt::Long;
use Pod::Usage;
use File::Copy;
```

Benchmarking step by step

The following instructions assume that a dedicated directory has been created:

```
mkdir  Build_bench, cd Build_bench; ccmake ..
```

The benchmarking mode can be activated using the `ccmake` interface and the following options (by default benchmark are OFF, validation test and cover test are ON):

```
BENCH                       *OFF
COVER_TEST                  *ON
Default_BenchDir            *benchmarks     /* benchmark location
Default_BenchFormat         *wcsp           /* extension definition
Default_bench_timeout       *30             /* default timeout
Default_cover_dir           *cover          /* cover test location
Default_regexp              *end.           /* default regexp
Default_test_option         *              /* default option
```

If you want activate the toulbar2 benchmark mode you must first set the first cmake variable to ON and COVER_test to OFF. In this case, wcsp benchmark have to be located in the benchmark directory and will be executed with a 30 seconds timeout .

In the current example, the `toulbar2/benchmark` directory includes the following files:

```
./benchmarks/zebra.wcsp
./benchmarks/CELAR6-SUB1.wcsp
```

The following commands:

```
cd build_bench ; cmake ..
```

will start the benchmarking and cmake will output:

```
####################################
-- Bench found: /home/dallouche/bug/toulbar2/benchmarks/zebra.wcsp
command line : TOULBAR2_OPTION timeout=30;regexp=test ok

-- Bench found: /home/dallouche/bug/toulbar2/benchmarks/CELAR6-SUB1.wcsp
command line : TOULBAR2_OPTION timeout=30;regexp=test ok
-- UB found ==> -ub=160
-- ################################
```

NB: like in the previous test modes, if a `.ub` file containing the optimal cost is located in the `foo.wcsp` directory then ???

All the benchmark tests need to be defined in the `build_bench/CTestTestfile.cmake` file:

```
---
ADD_TEST(Phase1_Toulbar_/home/dallouche/bug/toulbar2/benchmarks/zebra
"bin/Linux/run_test.pl" "-wcsp"
```

```
"/home/dallouche/bug/toulbar2/benchmarks/zebra.wcsp" "-rank" "1"
"-regexp" "test ok" "-option" "TOULBAR2_OPTION" "-timeout" "30" "-path"
"/home/dallouche/bug/toulbar2/build4")

SET_TESTS_PROPERTIES(Phase1_Toulbar_/home/dallouche/toulbar2/benchmarks/z
ebra PROPERTIES  PASS_REGULAR_EXPRESSION "test ok" TIMEOUT "30")

ADD_TEST(Phase1_Toulbar_/home/dallouche/toulbar2/benchmarks/CELAR6-SUB0
"bin/Linux/run_test.pl" "-wcsp"
"/home/dallouche/toulbar2/benchmarks/CELAR6-SUB1.wcsp" "-rank" "2"
"-ub=160" "-regexp" "test ok" "-option" "TOULBAR2_OPTION" "-timeout" "30"
"-path" "/home/dallouche/bug/toulbar2/build4")

SET_TESTS_PROPERTIES(Phase1_Toulbar_/home/dallouche/toulbar2/benchmarks/C
ELAR6-SUB0 PROPERTIES  PASS_REGULAR_EXPRESSION "test ok" TIMEOUT "30")
```

Before benchmarking, you need to generate the toulbar2 binary using "`make`". You can now define the benchmark campaign. To compare different toulbar2 options, you need to create in the working directory (i.e : `build_bench`) a text file including the various options that you will benchmark. For example the following text file include :

Data extraction: for each test, ctest is not going to directly launch toulbar2 but the perl wrapper `run_test.pl`. This script is located in the `misc/script/` directory. it will be copied in the same location as toulbar2 binary during the compilation process (`build/bin/Linux`).

Match

```
#file structure :
#label;value_postion;regular expression;mandatory (1 = yes,0=no)
#label = name used for data storage and colonne name in the report file
#value position = rank of wanted value in the regular expression specified in
the next field
#regular expression ==> reg exp in perl syntax fence define extract value
# mandatory ==> boolean flag 1=> current element is mandatory .i.e test will be
successful or failed is this field is not found in the jobs output

Optimum;2;(Optimum:)\s(\d*);1
backtracks;1;[in]\s(\d+)\s+(backtracks);0
nodes;1;[and]\s(\d+)\s+(nodes);0
seconds;1;[and]\s(\d+.\d*|\d+)\s+(seconds.);0
Pretime;1;[Preprocessing Time]\s+[:]\s*(\d+.\d*|\d+)\s+(seconds);0
Solution;1;New solution:\s(\d+);0
logLike;1;log10like:\s([+-]\d*.\d*)\s;0
```

Match allows to define regular expressions in order to extract metrics from each output.