Fourth ASP Competition
Detailed scoring regulations

The Competition Organizing Committee

Università della Calabria, Technical University of Vienna

Change Log

- **V.2.2, Feb 24th 2013**
  - Added details to the instance selection procedure
- **V.2.1, Feb 14th 2013**
  - Added instance selection procedure
  - Added policy concerning multiple runs
- **V.2.00, Nov 16th 2012**
  - First public version (this document)

1 Scoring System

The final score obtained by a system $\mathcal{S}$ in a track $\mathcal{T}$ consists of the sum over the scores obtained by $\mathcal{S}$ in all benchmarks selected for $\mathcal{T}$. In particular, a system could get a maximum of 100 points for each given benchmark problem P considered for $\mathcal{T}$. The overall score of a system on a problem P counting N instances, hereafter denoted by $S(P)$, is computed according to the following formulas that depend on whether P is a search, query or optimization problem. For each problem domain we set the maximum execution time $t_{out} = 600$ seconds and the value $\alpha = 50$;

Wrong Answers. Should a system $\mathcal{S}$ produce an output detected as incorrect[1] for at least one instance of P, then $\mathcal{S}$ is disqualified from P and $S(P)$ is accordingly set to zero (i.e., $S(P) = 0$ in case of incorrect output); otherwise, the following formulas are applied for computing $S(P)$.

Search and Query Problems. In case of both search and query problems the score $S(P)$ is computed by the sum

$$S(P) = S_{solve}(P) + S_{time}(P)$$

where $S_{solve}(P)$ and $S_{time}(P)$ take into account the number of instances solved by $\mathcal{S}$ in P and the corresponding running times, respectively; in particular

$$S_{solve}(P) = \alpha \frac{N_S}{N} \tag{1}$$

$$S_{time}(P) = \frac{100 - \alpha}{N\gamma} \sum_{i=1}^{N} \left(1 - \left(\frac{\log(\max(1, t_i) + s)}{\log(t_{out} + s)}\right)\right) \tag{2}$$

where: N_S is the number of instances solved by P within the time limit; t_{out} is the maximum allowed time; t_i the time spent by S while solving instance i (t_i is assumed to be lesser or equal to t_out); s is a shifting factor which controls how much the logarithmic effect of S_{time} is mildened; γ is a normalization factor which is set in order to have $S_{time} = 0$ for $t_i = t_{out}$, and $S_{time} = (100 - \alpha)\ N$ when $0 \leq t_i \leq 1$; and, α is a percentage factor balancing the impact of $S_{solve}(P)$ and $S_{time}(P)$ on the overall score. Both $S_{solve}(P)$ and $S_{time}(P)$ are rounded to the nearest integer.

As in the 3d ASP Competition, $S_{time}(P)$ is specified in order to take into account the "perceived" performance of a system according to a logarithmic scoring. We set for this edition of the competition $s = 10$ and accordingly,

$$\gamma = 1 - \frac{\log(1 + s)}{\log(t_{out} + s)}$$

The term $\max(t_i, 1)$ has been introduced in order to avoid significant changes in scores when $t_i \leq 1$. This choice prevents that measurement errors can have

[1] Incorrect answers are determined as specified in 1.1

impact on the value of S_{time}, especially when t_i is within the order of magnitude of measurement errors themselves.

The 2011's definition for S_{time} can be substantially seen as similar to the above, where 2011's S_{time} can be seen as having s and γ both set to 1, and without correction for $t_i \leq 1$. The new value for s avoids the distribution of too much points for small differences in t_i in the lower range of S_{time}, while the correction $\max(1, t_i)$ prevents any difference at all when t_i is below 1 second.

Optimization Problems. As in the previous edition, the score of a system $\mathcal{S}$ in the case of optimization problems depends on whether $\mathcal{S}$ was able to find a solution or not, and in the former case, the score depends on the quality of the produced solution. In addition, as in the case of decision problems, time performance is taken into account. We assume the cost function associated with optimization problems must be minimized (the lower, the better). Optimum values for objective functions are normalized to 100.

The overall score of a system for an optimization problem P is given by the sum

$$S(P) = S_{opt}(P) + S_{time}(P)$$

where $S_{time}(P)$ is defined as for search problems, and $S_{opt}(P)$ takes into account the quality of the solution found. In particular, for each problem P, system $\mathcal{S}$ is rewarded of a number of points defined as

$$S_{opt}(P) = \alpha \cdot \sum_{i=1}^{N} S_{opt}^{i}$$

where, as before, α is a percentage factor balancing the impact of $S_{opt}(P)$ and $S_{time}(P)$ on the overall score, and S_{opt}^{i} is computed by properly summing, for each instance i of P, one or more of these rewards:

1. $\frac{1}{N}$ points, if the system correctly recognizes an unsatisfiable instance;
2. $\frac{1}{4N}$ points, if the system produces a correct witness;
3. $\frac{1}{4N}$ points, if the system correctly recognizes an optimum solution and outputs it (can be awarded together with point 2 above);
4. $\frac{1}{2N} \cdot e^{M-Q}$ points, where Q denotes the quality of the solution produced by the system and M denotes the quality of the best answer produced by any system for the current instance, for M conventionally set to 100, and Q normalized accordingly.

Taking into account that an incorrect answer causes the whole benchmark to pay no points, three scenarios may come out: timeout, unsatisfiable instance, or solution produced. Note thus that the score of point 1 cannot be rewarded for the same instance together with the other quotas.

Note that a system producing a solution with a quality gap of 1% with respect to the best solution gets only 35 points (over a range of 100) and the quality score quota rapidly decreases (it is basically 0 for quality gap $> 4\%$), so that small incremental gains in the quality of a solution determine a strong difference in scoring. Recall that S_{time} is awarded only if the optimal solution is found.

1.1 Detection of Incorrect Answers.

Each benchmark domain P is equipped with a checker program C_P taking as input values a witness A and an instance I, and such that $C_P(A, I) =$ “true” in case A is a valid witness for I w.r.t problem P. The collection of checkers underwent a proper review process and is pragmatically assumed to be correct.

Suppose that a system $\mathcal{S}$ is faulty for instance I of problem P; then, there are two possible scenarios in which incorrect answers need detection and subsequent disqualification for a given system:

- $\mathcal{S}$ produces an answer A, and A is not a correct solution (either because I is actually unsatisfiable or A is wrong at all). This scenario is detected by checking the output of $C_P(A, I)$;
- $\mathcal{S}$ answers that the instance is not satisfiable, but actually I has some witness. In this case, we check whether a second system $\mathcal{S}'$ produced a solution A' for which $C_P(A', I)$ is true.

Concerning optimization problems, checkers produce also the cost C of the given witness. This latter value is considered when computing scores and for assessing answers of systems. Note that cases of general failure (e.g. out of memory, other abrupt system failures) are not subject of disqualification on a given benchmark.

If a solver marks an answer as optimal for the instance of an optimization problem, and no solver finds a better witness, this is pragmatically assumed to be the optimal solution. In general, we take the best witness found as the “imperfect optimal solution”.

As a last remark, note that in the setting of the System Track, where problem encodings are fixed, a single stability checker for answer sets could replace our collection of checkers. We prefer to exploit already available checker modules, which will be also used for assessing the correctness of fixed official encodings set for the System Track.

2 Score averaging policy in case of multiple runs

The Competition is run multiple times across all domains and selected instances. The following averaging policies apply depending on whether we consider S_{solve}, S_{opt} or S_{time}. Let n be the chosen number of runs:

- S_{solve} and S_{opt}. As for these score quotas we aim at measuring which is the *minimum performance guaranteed* by a solver over a given instance. For instance, if over the n runs a given participant times out at least once for a given instance I, this is indicative of the fact that the solver does not guarantee termination within time-out for I, thus we should attribute $S_{solve} = 0$.
  To this end, for each instance, the minimum value measured for S_{solve} and S_{opt} over the n runs is awarded.

- S_{time}. This quota, which is the most affected by computation glitches, is averaged over all runs. Note that we average S_{time} values, instead of computing S_{time} over the average of time values.

Other settings. The organizing committee keeps a neutral position and do not discloses any material submitted by participants until the end of the competition: however, participants are allowed to share their own work willingly at any moment. All participants agree implicitly that any kind of submitted material (system binaries, scripts, problems encodings, etc.) will be made public after the competition, so to guarantee transparency and reproducibility.

3 Instance Selection

3.1 Selection Requirements

- The selection system depends on a unique, not controllable by the organizer, random seed value;
- Instances are roughly ordered by some difficulty criterion provided by domain maintainers;
- Hash values of instance files, and the fixed ordering of instances is known before the Competition run;
- The selection criterion must be unique and applied rigidly to each benchmark domain. I.e. it must be impossible in practice, for organizers, to possibly forge the selection of instances in one domain without altering, out of control, the selection of instances in the other domains.

3.2 Instance Selection procedure

In the following, let S be the Competition seed, R be the number of instances per benchmark to be selected. Let $norm = 255$. Let D a benchmark domain, L_D its ordered set of available instances with $|L_D| = N_D$. We denote as $L_D[i]$ the i-th instance. We adopt a variant of systematic sampling in order to roughly ensure a fair selection over the whole family L_D, as follows: Let $Start, Perturb_1, \ldots, Perturb_R$ be values systematically generated from S where $Start$ ranges from 0 to $Norm$ and each $Perturb_i$ ranges from -1.5 to 1.5. Then, we have:

$Step = \frac{N_D}{R}$, $Start_D = Step\frac{Start}{norm}$. Then we select, for all $i\,(1 \leq i \leq R)$, all the instances

$$L_D\left[round(\max(0, \min(N_D, Start_D + i * Step + Perturb)))\right]$$

Here round(n) is n rounded to the nearest integer. In the cases in which $Step < Perturb_{i+1} - Perturb_i$ for some i (a case possible when $Step \leq 3$), we set $Perturbi + 1 = Perturb_i + 1$.

Technical notes. Given the random seed S (a positive integer value), $Start$ and $Perturb$ values are given by generating a RC4 byte stream with key S, and taking values from its 1001-th byte. Let $R[i]$ be the i-th byte of the obtained stream. We set $Start = R[1001]$ and for any i, $Perturb_i = R[1000 + i]/Norm * 3 - 1.5$.