ROADEF challenge revival : the checker

The *Java* checker[1] is supposedly a strict copy of the original one (written in *C#* for the 2005 Renault challenge). This document serves two purposes :

- it lists the potential commands you can use to run the checker.
- it helps you understand the results returned by the checker.

Were you in seek of information on the challenge subject, you should take a look at the ROADEF challenge website[2].

How is a solution checked ?

Consistency

The first thing checked when you run the checker on a solution file is the consistency of the latter. The test is simple and passes if and only if each car appears once and only once in the solution. Note that cars from the previous day must not be included in the solution file (as they are not to be rescheduled).

Constraints and violations

The original problem was a multi-criteria extension of the well-known car sequencing as two types of constraints were involved : paint batches and classical ratio constraints. The candidate has now a choice since he can choose to solve the original problem, but he also can ignore paint batches and compete on *academic* versions of the instances.

Paint batches

Each instance of the problem provides the candidate with a paint limit, K. The checker computes the number of paint batches from a solution file as follows :

- The first car to be considered is the last one from the previous day.
- It then compares - for each car in the solution - the color of the current car and the previous one. If they are different and if there were no batch after the previous car, then the checker counts a batch.
- If at some point, there are K consecutive vehicles with the same color, then the checker counts a batch.

[1] *carSeqCheck.jar*
[2] http://challenge.roadef.org/2005/en/

Ratio constraints

To remain consistent with the historical challenge, ratio constraints violation computation is subtle. Being given a solution file and a N/P ratio constraint the checker proceeds as follows :

- Extend the solution with the $P-1$ last cars from the previous day.
- For each length P interval in the extended solution, if there are more than N constrained vehicles then the checker counts $nbVehicles - N$ violations.
- For each length $p < P$ interval such that the last vehicle in the interval is the last vehicle in the solution, if there are k constrained vehicles then the checker counts $max(0, k - N)$ violations.

Solution score

The score, s, depends on the choice the candidate made. In the case the candidate provides a solution for a *historical* instance, then the procedure is the same and depends on the number of objectives :

Case 1 : two objectives then
$s = 1000 \times \#violation\_obj_1 + \#violation\_obj_2$

Case 2 : three objectives then
$s = 1000000 \times \#violation\_obj_1 + 1000 \times \#violation\_obj_2 + \#violation\_obj_3$

In the case of a solution for an *academic* instance - without color batches - then the procedure is the same as above removing objectives linked to color batches. For example, if obj_2 concerns color batches, it simply is removed from the objectives : an instance with 3 objectives becomes an instance with 2 objectives.

How to use *carSeqCheck.jar*

You need to format your workspace prior to checking anything. Make sure you own a directory *dir* (name it) in which there are two sub-directories : *Solutions* and *Instances*. There are several ways of launching the checker :

- java -jar carSeqCheck.jar **dir name**
  seeks the instance at **dir/Instances/name** and the solution at **dir/Solutions/name**.
- java -jar carSeqCheck.jar **dir instance solution**
  seeks the instance at **dir/Instances/instance** and the solution at **dir/Solutions/solution**.
- java -jar carSeqCheck.jar *-r* **dir**
  for each i in **dir/Instances/***, seeks the solution at **dir/Solutions/i**.

You can add the option *-ac* to a launch line to trigger the *academic* mode on. For instance : java -jar carSeqCheck.jar *-r -ac* **dir**, will process all instances in the given directory with the academic versions.