Combinatorial Optimisation and Constraint Programming (1DL442) Uppsala University – Autumn 2024 Assignment 5: The Sum, Element, Table, and AllDifferent Constraints

Prepared by Pierre Flener and Frej Knutar Lewander

— Deadline: 13:00 on Friday 6 December 2024 —

It is strongly recommended to read the *Grading Rules* below and the *Submission Instructions* at the end of this document even ***before*** attempting to tackle its tasks. It is also strongly recommended to prepare and attend the help sessions, as huge time savings may ensue.

Questions and Grading Rules

Assignment 5 is graded 0..5 and covers **Modules 3 to 5** of the MiniCP teaching materials [1]. The tasks are as follows (solo teams or designated sub-teams may choose in Tasks B and C one of the two problems starred there, but *Eternity* only if their MiniZinc project was not *Carcassonne*), a report being only needed if you take on Task C and possibly also Task D (those tasks are mandatory for PhD students):

A. Individually: Pass *all* the **Theoretical Questions** at INGInious of *all* those modules.

B. As a team:

- Pass *all* the unit tests at INGInious for `Element1D` (Module 3), `Element1DDomainConsistent` (Module 3), `Element1DVar` (Module 3, the hybrid domain-bound consistent propagator suffices, but see Task D), `StableMatching` (*, Module 3), `StateSparseBitSet` (Module 4), `TableCT` (Module 4), `Eternity` (*, Module 4), `AllDifferentFWC` (Module 5), and `AllDifferentDC` (Module 5).
- Upload *also* at Studium *all* `*.java` (except the `*Test.java`) mentioned in the questions, for a local archive at UU.

C. As a team or designated sub-team: Write a report on the problems *Stable Matching* (*, Module 3) and *Eternity* (*, Module 4):

- Evaluate (in the style of Section C of the MiniZinc demo report) each of the models under a suitable time-out (of at least 10 minutes per instance) in terms of:
  - the number of solutions for *Stable Matching* and satisfiability for *Eternity*;
  - the runtime in seconds, with 1 decimal place, to *all* the solutions for *Stable Matching* and to the *first* solution for *Eternity*;
  - the number of failures;

for *all* the instances in `data/stable_matching`, and for the instances `data/eternity/brendan/pieces_NxN.txt` with `N` in 5..10 for *Eternity*. Note that we are not grading for speed, but for correctness.

D. As a team or designated sub-team, if you think you pass Task C, then do the following:

- Implement (here, or within Task B, or both) a fully domain-consistent propagator for `Element1DVar` (Module 3) and add unit tests. Give a high-level description and correctness argument for your propagator and tests in the report.
- Mark in the report section for Task C that it is this propagator that underlies your evaluation of *Stable Matching*. Note that, for this problem and model, domain consistency for `Element1DVar` implies that search must be failure-free: indeed, a polynomial-time algorithm exists for this problem.
- Upload *also* at Studium your code, to be called `Element1DVarDC.java` and `Element1DVarDCTest.java` there, for a local archive at UU.

If you pass only Tasks A and B, then your score is 3 points. If you pass only Tasks A to C, then your score is 4 points. If you pass Tasks A to D, then your score is 5 points. In all other cases, your score is 0 points and there is no grading session. The solution session will be for questions and answers on the **Theoretical Questions** of Modules 3 to 5.

References

[1] Laurent Michel, Pierre Schaus, and Pascal Van Hentenryck. MiniCP: A lightweight solver for constraint programming. *Mathematical Programming Computation*, 13(1):133–184, 2021. The source code is available at http://minicp.org and the teaching materials are available at https://www.edx.org/course/constraint-programming.

Submission Instructions

Use the following to-do list before submitting:

- If you write a report (Tasks C & D): there is no demo report, but remember best practice on comments for code and on experimental evaluation from Sections B and C of the MiniZinc demo report and use your best judgement; do ***not*** import code into the report.
- ***Thoroughly*** proofread, spellcheck, and grammar-check the report, at least once per teammate, including the comments in ***all*** code. In case you are curious about technical writing: see the English Style Guide of UU, the technical-writing Check List & Style Manual of the Optimisation group, common errors in English usage, and common errors in English usage by native Swedish speakers.
- Produce the report as a ***single*** file in ***PDF*** format; all other formats will be rejected.
- Remember that when submitting you implicitly certify (a) that your files were produced solely by your team, except where explicitly stated otherwise and clearly referenced, (b) that each teammate can individually explain any part starting from the moment of submitting your files, and (c) that your files are not freely accessible on a public repository.
- Submit (by only ***one*** of the teammates) the files (all `*.java` mentioned in the questions, except the `*Test.java`, and possibly a report) ***without*** folder structure and ***without*** compression via *Studium*, whose clock may differ from yours, by the given ***hard*** deadline.