Attention

- Use a separate folder for each problem.
- Create a project (.mzp) for each problem, if there is any data file involved.
  - Add the model files (*.mzn) and the data files (*.dzn)
- Configure the solver to obtain the solution statistics and to set a time limit (300 seconds).
- Use commas when reporting big numbers. E.g.,
  - 976474 instead of 976,474
- Submit one single zip file.
- Indicate the group partner in the submission comments.

N-Queens

- Using the alldifferent model (without symmetry breaking), search for **a solution** for N = 30, 35, 45, 50, using the following 6 variable - value ordering heuristics of Gecode:
  - input order – min value, input order – random value
  - min domain size – min value, min domain size – random value
  - domWdeg – min value, domWdeg – random value
- Record the number of failures (‘-’ for timed out instances).
- For each instance, indicate the best results in bold.

Poster Placement

- Proceed **similarly to n-Queens**, by using the global model and the instances provided in the data files.
- Given the two sequences of variables X and Y, let the solver search on X_i and Y_i for all i in [1..n].
- In addition, re-order the rectangles in the data file in decreasing order by their perimeter and try input order – min value and input order – random value.
- Record the number failures and time (‘-’ for timed out instances).
- For each instance, indicate the best results in bold.

Quasigroup Completion Problem

- Implement a model using alldifferent constraints.
- Search for **a solution** to the instances given in the data files using Gecode and experiment with:
  - default search
  - domWdeg – random value
  - domWdeg – random value + restarting (employing the Luby strategy with L = 250)
- Record the number failures and time (‘-’ for timed out instances).
- For each instance, indicate the best results in bold.

Comments

- For each problem, make observations on your results and write them down (avoid repeating the numerical results in text).
- While doing so, pay attention to the following points:
  - When are random decisions (not) useful? Why?
  - Are dynamic heuristics always better than static heuristics? Why?
  - Is programming search and/or restarting always a good idea? Why?