Read the source code at http://user.it.uu.se/~pierref/courses/M4CO/demoReport before tackling any assignment: it contains problem-independent instructions on what to hand in, as well as useful problem-independent indications on how to proceed.

It is also strongly recommended to read the Submission Instructions and Grading Rules at the end of this document before attempting to address its tasks.

Finally, students who have taken the Combinatorial Optimisation with Constraint Programming course (code 1DL441, formerly 1DL023, 1DL118, and 1DL440), or an equivalent course elsewhere, must show experiments with annotations for CP inference and CP search, due to the slight overlap with such a course: they must self-study and apply the slides of Topic 8: Inference and Search for CP and LCG (also see Chapter 6 in the MiniZinc Tutorial).

It is the year 2137. You are consultants and the factory of a client can assemble one spacecraft per week. The factory manager is given a list of orders, each order specifying at the end of which week some ordered spacecrafts should be ready and which types of spacecraft they should be. Your job is to decide which week to assemble each ordered spacecraft (or, equivalently, which type of spacecraft to assemble each week) in order to minimise the total cost incurred by the storage of the spacecrafts that are completed before their due date and by the adaptation of the factory when switching between types of spacecraft. An instance of the SPP is defined by:

- the number weeks of weeks for the planning;
- the number types of spacecraft types the factory can assemble;
- for each \( t \) in \( 1 .. \text{types} \) and each \( w \) in \( 1 .. \text{weeks} \), the number \( \text{Order}[t,w] \) of spacecrafts of type \( t \) to assemble by the end of week \( w \); you can assume that each \( \text{Order}[t,w] \) is in the integer range \( 0 .. 1 \);
- the cost \( \text{storageCost} \) of storing one spacecraft during one week;
- for each \( t1 \) and \( t2 \) in \( 1 .. \text{types} \), the cost \( \text{SetupCost}[t1,t2] \) of adapting the factory from assembling spacecrafts of type \( t1 \) to assembling spacecrafts of type \( t2 \); this cost matrix respects the triangular inequality (for all \( i, j, k \) in \( 1 .. \text{types} \), we have \( \text{SetupCost}[i,k] + \text{SetupCost}[k,j] \geq \text{SetupCost}[i,j] \)), but might be asymmetrical, and there is no setup cost when not changing the spacecraft type (for all \( i \) in \( 1 .. \text{types} \), we have \( \text{SetupCost}[i,i] = 0 \)).

A skeleton MiniZinc model and instances of varying sizes and difficulty, in the form of datafiles using the parameter names above, are at http://user.it.uu.se/~pierref/courses/M4CO/assignments/assignment2.zip. Here are some clarifications by the factory manager:
A spacecraft assembled during the week it is due incurs no storage cost.

There is no limit on storage space: one can always store as many spacecrafts as needed.

One cannot assemble an ordered spacecraft after its due date.

There is no setup cost before the first spacecraft is assembled and there is no setup cost after the last spacecraft is assembled.

If there is a stretch of one or more weeks with zero assembly directly after the assembly of a spacecraft of type $t_1$ and directly before the assembly of a spacecraft of type $t_2$, then one must still pay the cost $\text{SetupCost}[t_1,t_2]$.

This problem can be modelled using at least two viewpoints: either (1) decide, for each week, which, if any, (type of) spacecraft to assemble; or (2) decide, for each (type of) spacecraft, during which week(s) to assemble it. Perform the following sequence of tasks:

A. Design and evaluate a model using the first viewpoint above.

B. Design and evaluate a model using the second viewpoint above.

C. Design and evaluate a model using the two viewpoints and channelling constraints.

Note the parenthesised use of “(type of)” in the previous paragraph: it is up to you to decide, for each viewpoint, whether to reason with individual spacecrafts or with types of spacecraft.

Hint: We suggest you start with the first viewpoint and implement it incrementally, based on the specification: first model the assembly schedule, then extend the model with the storage costs, and finally extend the model with the setup costs. Make sure the model is correct before and after each extension. Start working on the second viewpoint only after finishing the model for the first viewpoint: some insights should carry over, which can save you a lot of time. After implementing the first two viewpoints, the third model should essentially be a concatenation of the first two models, augmented with channelling constraints. Some constraints might become redundant when channelling two models: removing some of them might improve or worsen the solving time.

For the evaluations, pick at least six (of the ten) provided instances and report the results for the chosen backends for all the considered technologies. Use any time-out of a few CPU minutes per instance to avoid too long solving times. Note that the number of weeks and types is not necessarily indicative of the difficulty of an instance.

After performing all the evaluations, elaborate on the following sequence of questions:

D. Which model was easier to write and which one is easier to understand?

E. Which improvements of your models have you performed to render them more efficient? Quantify the effect of each improvement, without necessarily running all versions of a model on all the instances under all the chosen backends: use your best judgement.

F. For each of the chosen technologies and backends, which model is more efficient? Why?

G. Which combination of model, technology, and backend would you recommend to the factory manager for solving future instances of the problem? Why? Factor in your answer to Question D, as one may also want to consider the maintainability of the chosen model.

Solo teams may skip using $\text{storageCost}$ within their models, but are highly encouraged to try nevertheless.
Submission Instructions

All task answers, other than source code, must be in a single report in PDF format; all other formats are rejected. Furthermore:

- Identify the team members and state the team number inside the report.
- Address each task of each problem, using the numbering and the ordering in which they appear in the assignment statement.
- Take the instructions of the demo report at [http://user.it.uu.se/~pierref/courses/M4C0/demoreport](http://user.it.uu.se/~pierref/courses/M4C0/demoreport) as a strict guideline for the structure and content of a model description and evaluation in the report, as well as an indication of its expected quality.
- If at least one teammate has taken a course on CP, then for Assignments 2 and 3 show experiments with annotations for CP inference and CP search: self-study and apply the slides of Topic 8: Inference and Search for CP and LCG (also see Chapter 6 in the MiniZinc Tutorial).
- Write clear task answers, source code, and comments.
- Justify all task answers, except where explicitly not required.
- State any assumptions you make that are not in this document.
- Thoroughly proofread, spellcheck, and grammar-check your report.
- Upload all models.
- Match exactly the uppercase, lowercase, and layout conventions of any filenames and I/O texts imposed by the tasks, as we will process your source code automatically.
- Write a paragraph, which will not be graded, describing your experience with this assignment: which aspects were too difficult or too easy, which aspects were interesting or boring? This will help us improve the course in the coming years.
- Remember that when submitting you implicitly certify that your report and all its uploaded attachments were produced solely by your team, except where explicitly stated otherwise and clearly referenced, that each teammate can individually explain any part starting from the moment of submitting your report, and that your report and attachments are not freely accessible on a public repository.

Only one of the teammates submits the solution files (one PDF report with answers to all the tasks, and all model files), without folder structure and without compression, via the Student Portal, whose clock may differ from yours, by the given hard deadline.

Grading Rules

If all tasks have been seriously attempted and all requested models exist, have the comments exemplified in the demo report, and produce correct outputs for some of our grading instance data in reasonable time on the Linux computers of the IT department, within the given time bounds (if any), then you get at least 1 point (of 5), otherwise you get 0 points and fail the Assignments part of the course. Furthermore:
If your models pass most of our grading tests and your report is complete, then you get 4 or 5 points, depending also on the quality of the model comments and the report; you are not invited to the grading session.

If your models fail many of our grading tests or your report is incomplete, then you get an initial mark of 1 or 2 points, depending also on the quality of the model comments and the report; you are invited to the grading session, where you can try and increase your initial mark by 1 point into your final mark.

However, if the assistant figures out a minor fix that is needed to make your model run as per our instructions above, then, instead of giving 0 points up front, the assistant may deduct 1 point at his discretion.

Considering that there are two or three help sessions for each assignment, you must get minimum 1 point (of 5) on this assignment until the end of its grading session, and minimum 8 points (of 15) over all three assignments in order to pass the Assignments part (3 credits) of the course.