Modelling for Combinatorial Optimisation (1DL451) Uppsala University – Autumn 2025 Assignment 2: Spacecraft Assembly Problem (SAP)

— Deadline: 13:00 on Friday 28 November 2025 —

The scope of this assignment is Topics 1 to 4: you need ***not*** show any knowledge of subsequent topics. Read the **Report Instructions** and **Grading Rules** at the end of this document ***before*** tackling the following problem(s) upon reading them a first time. It is strongly recommended to prepare and attend the help sessions, as huge time savings may ensue.

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 spacecraft types they should be. Your job is to decide which week to assemble each ordered spacecraft (or, equivalently, which spacecraft type 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 spacecraft types. An instance of the SAP 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..types` and each `w` in `1..weeks`, the number `Order[t,w]` of spacecrafts of type `t` to assemble by the end of week `w`; note that each `Order[t,w]` can be any non-negative integer;
- the cost `storageCost` of storing one spacecraft during one week;
- for each `t1` and `t2` in `1..types`, the cost `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..types`, we have `SetupCost[i,k]` $+$ `SetupCost[k,j]` $\geq$ `SetupCost[i,j]`), but might be asymmetrical, and there is no setup cost when not changing the spacecraft type (for all `i` in `1..types`, we have `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 available. 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 `t1` and directly before the assembly of a spacecraft of type `t2`, then one must still pay the cost `SetupCost[t1,t2]`.

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

A. In order to reason with individual spacecrafts, pre-compute at least the derived parameter `DueWeek`, specified in the skeleton model.

B. Write and evaluate a MiniZinc model called `SAP1.mzn` using viewpoint (1) above.
Write and evaluate a MiniZinc model called `SAP2.mzn` using viewpoint (2) above.

Hints. Try both viewpoints in parallel for a while and then finish the model that comes easiest for you. *Finish the other model only afterwards: some insights should carry over, which can save you a lot of time.* Model incrementally, based on the specification: first model the assembly schedule, then extend the model with the storage costs, and finally extend it with the setup costs. Make sure each model is correct before and after each extension. Make sure your two models yield, for each instance, the same objective value upon proven optimality. The predicates of the following examples may be useful:

- `inverse([3,1,4,2],[2,4,1,3])` (seen in Topic 6, *before* the deadline);
- `inverse_in_range([3,1,5,2],[2,4,1,0]);`
- `arg_sort([7,5,8,6],[2,4,1,3]).`

These predicates are specified in the MiniZinc Handbook.

For each evaluation, use all the provided instances and report the results for the chosen backends for all the considered technologies. ***Use a time-out of 5 CPU minutes per instance in order to avoid too long solving times.*** Note that the numbers of weeks and spacecraft types are not necessarily indicative of the difficulty of an instance. For your convenience, here are the minimal objective values for some instances (in real life, you do not know any of the optima when you start modelling a problem):

instance	minimum	minimum without `storageCost`
sap_005_02	171	131
sap_006_02	280	130
sap_008_04	431	366
sap_010_05	675	505

C. Which model was easier to write? Why? Which model is easier to understand? Why?

D. 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 C, as one may also want to consider the maintainability of the chosen model.

E. Briefly describe a real-world situation where (a variation) of the SAP can occur.

[1]Solo teams, except PhD students, may skip using `storageCost` within their models, but are highly encouraged to try nevertheless.

Report Instructions

1. You ***must*** fill in the provided skeleton report and skeleton model(s). You ***must*** inspect your model(s) using our checklist. See the demo report (LaTeX source) for generic instructions and for the expected quality of a report. The specific writing instructions are in comments that start with '`%%`' in the LaTeX source of the provided skeleton report and should be followed even when not using LaTeX. All running text should be black: before submitting, comment away line 15 (which typesets the placeholders in blue) and uncomment line 16 (which typesets them in black). In the provided MiniZinc skeleton model(s), the placeholders are '`...`' and the main model items are delimited by comments that start with '`%%`'.

2. Spellcheck the report and the comments in ***all*** models, in order to show both self-respect and respect for your readers. Thoroughly proofread the report, at least once per teammate, and ideally grammar-check it. In case you are curious about technical writing: see the English Style Guide of UU, the technical-writing Style Manual of the Optimisation research group, a list of common errors in English usage, and a list of common errors in English usage by native Swedish speakers.

3. Remember that when submitting you implicitly certify (a) that your report and all its uploaded attachments 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 report, and (c) that your report and attachments are not freely accessible on a public repository.

4. Submit (by only ***one*** of the teammates) by the given ***hard*** deadline two files via *Studium*: your report as a ***PDF*** file (all other formats will be rejected) and a compressed folder in ***ZIP*** format with all source-code files (including all MiniZinc models, plus possibly a software pipeline for the project) and datafiles (only for the project).

Grading Rules (stricter than the ones for Assignment 1)

If *all* tasks have been tackled, ***and*** *all* requested models are uploaded in files with the imposed names, structure, comments (in the model), and explanations (in the report) exemplified in the demo report, ***and*** *all* models ultimately produce *correct* solutions to *all* instances under backends of *all* considered solving technologies, ***and*** *all* models produce (*near-*)*optimal* solutions in *reasonable* time to *most* instances under backends of *at least two* technologies (under MiniZinc version 2.9.4 on a Linux computer of the IT department), ***then*** you get a score of at least 1 point (read on), ***else*** your *final* score is 0 points. Furthermore:

- ***If*** *all* models have *good enough* comments and explanations ***and*** *all* task answers are *correct enough*, ***then*** you get a *final* score of 3 or 4 or 5 points and are *not* invited to the grading session.

- ***If*** *some* models have *poor* comments or explanations ***or*** *some* task answers have *severe errors*, ***then*** you get an *initial* score of 1 or 2 points and *might* be invited to the grading session, at the end of which you are informed whether your initial score is increased or not by 1 point into your *final* score. A non-invitation leads to your final score being the initial one, and the same holds for each invited student who makes a no-show.

Also, ***if*** an assistant figures out a minor fix that is needed to make some model run as per our instructions above, ***then***, instead of giving 0 points, the assistant may at their discretion deduct 1 point from the score earned upon the fix.