Modelling for Combinatorial Optimisation (1DL451) and Constraint Programming (1DL442)
Uppsala University – Autumn 2023
Assignment 2: Spacecraft Assembly Problem (SAP)

Prepared by Pierre Flener, Gustav Björdal, and Jean-Noël Monette

— Deadline: 13:00 on Friday 22 September 2023 —

The scope of this assignment is Topics 1 to 4: you need ***not*** show any knowledge of subsequent topics. The source code of the demo report (at `https://user.it.uu.se/~pierref/courses/COCP/demoReport`) has problem-independent indications on how to proceed. Read the *Submission Instructions* and *Grading Rules* at the end of this document. 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 at `https://user.it.uu.se/~pierref/courses/COCP/assignments/assignment2/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 `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 the first viewpoint above. Write and evaluate a MiniZinc model called `SAP2.mzn` using the second viewpoint 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.

   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. See Table 1 at the end of this document for the minimum requirements for each possible score on this assignment. 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
sap_005_02	171
sap_006_02	280
sap_008_04	431
sap_010_05	675

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

D. Which improvements of your models have you performed to render them more efficient? A purely textual reply is expected: 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.

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

E. 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.

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

Submission Instructions

In order to protect yourself against an unnecessary loss of points, use the following to-do list before submitting:

- Tackle ***each*** task of ***each*** problem, using (in order to accelerate the grading) the numbering and the ordering in which they appear in this assignment statement.
- Take the instructions of the demo report at `https://user.it.uu.se/~pierref/courses/COCP/demoReport` as a ***strict*** guideline for the structure and content of a model description, model evaluation, and task answer, and as an indication of the expected quality of content: write with the precision that you would expect from a textbook.
- You ***must*** use the MiniZinc experiment script explained in `https://user.it.uu.se/~pierref/courses/COCP/assignments/cheatsheet.pdf`: it conducts the experiments and generates a result table that can be automatically imported (rather than manually copied) into a LaTeX report, so that each time you change a model, it suffices to re-run the script and re-compile your report, without any tedious number copying!
- If a MiniZinc model does not compile and run error-free under backends of ***all*** the considered solving technologies, then obtain a teacher's approval in due time ***before*** submitting your 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: the *English Style Guide* of UU at `https://mp.uu.se/en/web/info/stod/kommunikation-riktlinjer/sprak/eng-skrivregler` and the technical-writing *Checklist & Style Manual* of the Optimisation group at `https://optimisation.research.it.uu.se/checkList.pdf` offer many pieces of advice; common errors in English usage are discussed at `https://brians.wsu.edu/common-errors`; in particular, common errors in English usage by native Swedish speakers are listed at `https://www.crisluengo.net/english-language`.
- Match ***exactly*** the uppercase, lowercase, and layout conventions of any filenames and I/O texts imposed by the tasks, as we will process submitted source code automatically. However, do not worry when *Studium* appends a version number to the filenames when you make multiple submission attempts until the deadline.
- Do ***not*** rename any provided problem-specific skeleton model, for the same reason.
- Import all the MiniZinc models ***also*** into the report: for brevity, it is allowed to import only the lines after the copyright notice.
- 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 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.
- Submit (by only ***one*** of the teammates) the solution files (one report and all MiniZinc source code) ***without*** folder structure and ***without*** compression via *Studium*, whose clock may differ from yours, by the given ***hard*** deadline.

Grading Rules

If *all* the tasks have been tackled, ***and*** *all* the requested models are in files with the imposed names, comments, and explanations exemplified in the demo report, ***and***, for *all* our grading instances, *all* models produce *correct* outputs under backends of *all* the considered solving technologies and (*near-*)*optimal* outputs under backends of *at least two* technologies in *reasonable* time under MiniZinc version 2.7.6 on a Linux computer of the IT department, ***then*** you get at least 1 point (read on), ***else*** your *final* score is 0 points. Furthermore:

- ***If*** *all* models meet the *minimum* requirements in the first four or five rows in Table 1 below, ***and*** *all* models are good (in terms of comments and against the checklists of Topics 2 and 3 at `https://user.it.uu.se/~pierref/courses/COCP/demoReport/checklist.pdf`), ***and*** *all* task answers are *mostly correct*, ***then*** you get a *final* score of 3 or 4 or 5 points and are *not* invited to the grading session.
- ***If*** *some* models meet the *minimum* requirements in at most the first three rows in Table 1 below, ***or*** *some* models are flawed (in terms of comments and against the checklists of Topics 2 and 3), ***or*** the task answers have *many errors*, ***then*** you get an *initial* score of 1 or 2 or 3 points and *might* be invited to the grading session, where you can try and increase your initial score by 1 point into your *final* score, a no-show or non-invitation leading to your final score being the initial one.

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

	minimum requirement	
score	optimality	feasibility
1..5	sap_005_02	sap_008_04
2..5	sap_008_04	sap_015_10
3..5	sap_010_05	sap_030_05
4..5	sap_010_06	sap_100_15
5	sap_030_05	

Table 1: In each row, a solution to the instance in the 'optimality' column must be found and proven optimal by at least one backend under at least one viewpoint, and a solution to the instance in the 'feasibility' column must be found by at least one backend under at least one viewpoint, each time within 5 CPU minutes and no matter how slow the other backends and viewpoints are, under MiniZinc version 2.7.6 on any Linux computer of the IT department.