Lab 3: Modelling

Algorithms for NP-Hard Problems (CSE2310) 2023

Maarten Flippo, Emir Demirović

March 1, 2023

The goal of this assignment is to gain hands-on experience with modelling and solving combinatorial optimisation problems.

MiniZinc. We will be using the MiniZinc[1] modelling language. Once you devise a model in MiniZinc, you may use many of the available solvers to find solutions to your problem. Please visit the webpage, download MiniZinc, and familiarise yourself with the software. You may find a basic MiniZinc guide as part of this assignment folder, but you may also have a look at the MiniZinc tutorial[2]. We are also providing you with two example models for the graph colouring problem[3]. All of this should allow you to explore the MiniZinc IDE and get to understand how to use it.

Assignment. The assignment consists of two problems, see below. For each problem, you are given a skeleton model (`.mzn` file) that contains an incomplete model of the problem. Your task is to fill in the missing pieces to complete the model and find solutions to ten instances of the problem (`.dzn` files).

To help you test if you model is correct, we are providing you with the optimal objective value for each instance. Compare your result with these values!

Note that you are free to add, remove, and change existing variables in the models should you chose to do so. However please leave the parameters untouched, as they have to match with the `.dzn` files.

Tip #1. By default, solvers in MiniZincare configured to run without a timeout. However, for this assignment you should set a timeout of 1 minute. You can do so by clicking "Show configuration editor", and under "Options" (make sure "Maintain these options across solver configurations" is checked) check the "Time limit" checkbox and set the time limit to 60 seconds.

Tip #2. Recall that different solvers may exhibit different performance depending on the problem being solver. We invite you to experiment with *Chuffed*, *COIN-BC* and *Gecode* solvers. Note that you do not necessarily need to understand how these solvers work, it is enough to know that they implement different algorithms and they can be used with MiniZinc!

[1] `https://www.minizinc.org/`

[2] `https://www.minizinc.org/doc-2.5.3/en/modelling.html`

[3] `https://en.wikipedia.org/wiki/Graph_coloring`

1 Weighted Set Cover

The set cover problem is a classical combinatorial optimisation problem, where the goal is cover a number of elements by selecting the least amount of sets. See the Wikipedia page for the definition. [4]

A concrete example would to form a team of people, such that the team has a certain set of skills. Let us say a team needs to have the following skills: Planning, Written communication, Programming, System Design, Presenting. We can choose from the following pool of people:

Person	**Skills**
Anika	Planning, Written communication, Programming
Bao	Written communication, System Design
Cees	Programming, System Design
Dave	System Design, Presenting

In this situation, the smallest team to cover all skills would consist of Anika and Dave. This example is encoded in the `small-1.dzn` file.

The problem can be expanded to the weighted set cover problem, by attaching a cost to each of the people (their salary for example). The goal is to minimise the combined cost of the team.

For this problem, you should model the weighted set cover problem.

[4] `https://en.wikipedia.org/wiki/Set_cover_problem`

2 Scheduling

For this problem you will be scheduling tasks on a single resource. This resource has a capacity $C \in \mathbb{N}$. Each task i has three properties associated with it:

- A duration $d[i] \in \mathbb{N}$ which specifies the number of time steps the tasks needs to complete.
- A demand $rr[i] \in \mathbb{N}$ with $rr[i] \leq C$ which specifies the resource requirements of a task. All the tasks running at a single point in time can never exceed the capacity C of the resource. That is, if the resource capacity is 5, and we have $rr = [3, 4]$, then task 1 cannot run when task 2 is running or vice versa.
- A set of tasks $succ[i]$ that depend on i to be completed. For example, if we have $succ[1] = \{3, 5\}$, then tasks 3 and 5 cannot start before task 1 has completed.

The goal is to schedule all the tasks such that the dependencies are always satisfied, whilst minimising the time it takes to complete all tasks (i.e. the makespan). This problem is a simple case of the resource-constrained project scheduling problem (RCPSP). [5]

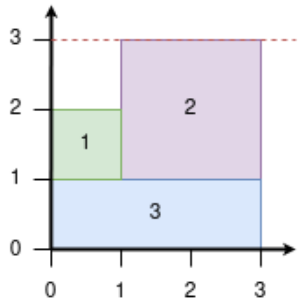

Figure 1: An example of a schedule for this assignment. The boxes are tasks, with the horizontal axis denoting the duration of each task and the vertical axis denoting the demand. The resource capacity is indicated with the dashed line.

An example in the problem is illustrated in Figure 1. We have 3 tasks with $d = [1, 2, 3]$, $rr = [1, 2, 1]$ and $suc = [\{2\}, \emptyset, \emptyset]$. The resource capacity $C = 3$.

[5] `https://www.csplib.org/Problems/prob061/`