FIT5220 Solving Discrete Optimization Assignment 1: Implementing propagators

Semester 2, 2020

July 7, 2020

Overview

For this assignment, your task is to extend the `sprue` CP solver with a propagator for a specified constraint.

- Submit your work to the MiniZinc auto grading system (using the submit button in the MiniZinc IDE).
- Submit your propagator (copy and paste the contents of the .py file) using the Moodle assignment. You have to submit by the due date (20th July 2020, 11:59pm), using **both** MiniZinc and using the Moodle assignment, to receive full marks. You can submit as often as you want before the due date. Late submissions without special consideration receive a penalty of 10% per day. Submissions are not accepted more than 3 days after the original deadline. This is an individual assignment. Your submission has to be entirely your own work. We will use similarity detection software to detect any attempt at collusion, and the penalties are quite harsh. If in doubt, contact your teaching team with any questions!

For local testing, you should have `sprue-cp` installed, and MiniZinc configured to use it. See the *Week 1 workshop* for details on how to do this.

Introduction

One of the most frequently used constraints is the `alldifferent` constraint:

$$\texttt{alldifferent}([x_1, \ldots, x_n]) \equiv \bigwedge_{i,j \in 1..N, i<j} x_i \neq x_j$$

that is, no two (distinct) x_i variables can take the same value. There are several propagation algorithms for `alldifferent` – you will see a domain consistent propagation algorithm later in the unit, and there is a bounds consistent algorithm which runs in linear time.

In this assignment, you will build a propagator for the `alldifferent` constraint.

Tasks

Part A - Value Propagation [2 marks]

For this task, you will implement the `value` propagator for `alldifferent`. The `vaue` propagator is woken up whenever a variable becomes fixed (e.g. x_i is set to k). When the propagator runs, it then removes k from the domain of all *other* variables.

Hint: Some of the arguments to `alldifferent` will be fixed values (type `int`), so will not support `Var` operations. Remember to handle this!

Part B - Variable Elimination [1 mark]

Once a variable has become fixed, we don't need to consider it again: its value has been removed from all other domains, so cannot be fixed to it.

For this task, modify your propagator to take advantage of this: *persistently* track the fixed and non-fixed variables, so each value is removed at most once, and already fixed variables are ignored.
Hint: This should not change the number of nodes/failures for your model.

Part C - Hall Violators [2 marks]

The bounds- and domain-consistent algorithms for `alldifferent` are based on Hall's *Marriage theorem*. In terms of `alldifferent`, the marriage theorem states that `alldifferent`(X) is satisfiable over domain $\mathcal{D}$ if and only if for every subset X' of X, we have:

$$|X'| \leq |\bigcup_{x \in X'} \mathcal{D}(x)|$$

Several propagation algorithms for `alldifferent` rely on finding variables X' where the domains of X' cover exactly $|X'|$ values – so those values can be removed from the domains of every other variable.

For this task you will use a restricted form of this reasoning. If the domains of *all* X variables cover fewer than $|X|$ values (that is, $|\bigcup_{x \in X} \mathcal{D}(x)| < |X|$), the constraint is unsatisfiable.

Modify your propagator to detect when this occurs, and indicate failure.

Instructions

Edit the provided Python source files to implement the propagators described above. You are provided with some sample model and data files to try your propagators on. Your implementations can be tested locally by using the `Run` icon in the MiniZincIDE or by using:
`minizinc --solver org.fit5220.sprue-cp ./modelname.mzn ./datafile.dzn`
at the command line.

Marking

The marks for Tasks 1 and 3 are automatically calculated. We will be checking both the correctness of solutions (incorrect propagators receive 0 marks), and the expected number of failures (to see that propagation has not been missed). Task 2 will receive 1 mark for a correct implementation.