School of Computer Science
University of St Andrews
2019-20
CS4402
Constraint Programming
Practical 1: Late Binding Solitaire

This Practical comprises 50% of the coursework component of CS4402. It is due 9th March at 21:00.

The deliverables consist of:

- A report, the contents of which are specified below.
- The Essence Prime constraint model for the basic specification.
- The Essence Prime constraint model(s), and possibly additional instances, for the extension part.

The practical will be marked following the standard mark descriptors as given in the Student Handbook (see link below).

Background: Patience

This practical involves modelling and solving a particular variant of Patience (aka Solitaire), a family of single-player card games. If you are not familiar with this type of game, then your first task is to read the Wikipedia description:

https://en.wikipedia.org/wiki/Patience_(game)

Late Binding Solitaire

The particular patience we are going to model is based on a traditional patience called Accordion, which you can play online here:

https://justsolitaire.com/Accordion_Solitaire/

As you can see, we begin by laying out the deck of cards in a sequence, each element of which we consider as a "pile" of one card. Each move we make consists of moving a pile on top of either the pile immediately to the left, or three to the left (i.e. with two piles between the source and destination) such that the top cards in the source and destination piles match by either rank (value of the card, e.g. both 7) or suit (clubs, hearts, diamonds, or spades). The result of each move is to reduce the number of piles by 1 and change the top card of the destination pile. The empty space left at the position of the source pile is deleted. The goal is to keep making moves until just one pile remains. There are some tips on how to play Accordion well here (note that we have been discussing the "open" version):

http://www.solitairelaboratory.com/accordion.html

Rather than playing with the full deck of 52 cards, we can play with a randomly chosen subset of cards. This version was invented and studied by Turing award winner Donald Knuth, who called this variant "late-binding solitaire". This is the version we will be modelling in this practical. See Chapter 2 of:

http://i.stanford.edu/pub/cstr/reports/cs/tr/89/1269/CS-TR-89-1269.pdf

To Play Late Binding Solitaire with `n` out of 52 cards, place the `n` cards in a sequence and make `n-1` moves as per Accordion in order to reduce the `n` initial piles to a single pile.

Examples

Consider a very simple instance of Late Binding Solitaire where `n` is 4 and we have the following four initial cards:

2♥, 3♥, 4♥, 4♦

Since each move will remove one pile, we know that we will need `n-1 = 3` moves to solve this problem.

Remembering that the source pile has to be to the right and either adjacent or 3 positions away from the destination pile, there are three possible moves from this initial state: we can move the 3♥ onto the 2♥ because they have the same suit, the 4♥ onto the 3♥, or the 4♦ onto the 4♥ because they have the same rank. Notice that if we choose the last of these three moves, resulting in the state:

2♥, 3♥, 4♦

then there is no subsequent sequence of moves that will lead to the game being solved. Can you see why?

If, on the other hand we choose the first possible move, resulting in:

3♥, 4♥, 4♦

Then a solution follows straightforwardly:

4♥, 4♦

4♦

and similarly, if we choose the second of the possible initial moves (try this for yourself).

For a slightly more complicated example, consider the initial cards:

10♣, A♥, 2♠, 2♣, A♠

This can be won, but absolutely requires a move which jumps three. We have to jump the A♠ to A♥. Following that (but not before) we can jump the 2♣ to the 10♣, then successively move the 2♠ onto the two piles now containing A♠ and 10♠ to win.

Just swapping the two twos in the above example creates an example with no solution. It is far from obvious that this is unwinnable, but we would hope your constraint model would be able to prove it.

10♣, A♥, 2♣, 2♠, A♠

Modelling Late Binding Solitaire

The goal of this practical is to formulate a model of Late Binding Solitaire in Essence Prime and test its performance on a number of supplied instances.

Parameter Format and Supplied Instances

For simplicity we will standardise the format of the parameter, for which we shall use the identifier `cards`, as follows:

```
language ESSENCE' 1.0
letting cards be [1, 2, 3, 16]
```

where each of the 52 cards in the deck is identified by an integer in the range 0, …, 51. We will assume that cards of the same suit are enumerated consecutively in ascending order. So, card 0 is the A♥, 1 is the 2♥, …, card 12 is the K♥. Similarly, cards 13, …, 25 represent the diamonds, 26, …, 38 represent the clubs, and 39, …, 51 represent the spades.

Use this identifier and format in your model as follows:

```
given cards : matrix indexed by [ int(1..n) ] of int(0..51)
where alldifferent(cards)
```

You will now be able to use the identifier `n` to refer to the size of the input card sequence in the rest of your model.

Along with this practical specification, you will find 25 .param files with which to test your model – see the section on Empirical Evaluation below.

Approaching Modelling

Late Binding Solitaire is a **planning problem**, as discussed in lectures and tutorials: the task is to find a sequence of moves to transform an initial state into a goal state. You should begin by reviewing your lecture notes on this topic and looking at the models for some other planning problems in the Savile Row examples directory, such as Sokoban or Peg Solitaire.

Planning problems are typically modelled as a sequence of states, where each element in the sequence represents the state of the world (i.e. the remaining piles in this case). In addition, there is often a representation of an action linking the previous to the current state (the source and destination piles here). Think about the decision variables you will need to represent these states and actions, and the constraints necessary to connect them.

General advice: debugging constraint models can be difficult because a conflicting set of constraints will cause the solver to return no solution without there being an obvious cause. For this reason, you are encouraged to build up the set of variables and constraints in your model **incrementally**, continually testing them on small instances.

Empirical Evaluation

Evaluate the performance of your model on the 25 instances (separated into solvable and unsolvable) supplied with this practical specification. In doing so use the default settings of Savile Row, such as:

```
./savilerow lateBindingSolitaire.eprime LBS4_13.param -run-solver
```

You should record in your report for each: the `SolverNodes`, `SolverSolveTime`, `Savile Row TotalTime`. As discussed in Tutorial 1, these are available in the `.info` file after your model has been run, irrespective of whether the instance is solvable.

You will also find supplied an instance generator. It is run as follows:

```
java LBSGen <length> <seed> > <yourFileName.param>
```

and will produce a `.param` file of the format described above. Use this generator to explore the lengths of instance to which your model will scale. Remember that not all instances of the same length are equally difficult.

There are some ideas for a more extensive empirical evaluation in the Extensions section below.

Extensions

There follow some ideas for extension activities. These are not required, but attempting at least one extension activity is required to gain a mark above 17.

You may wish to explore different models of Late Binding Solitaire, and how they perform empirically. You might also wish to explore some of the other processing Optimisation Levels Savile Row provides (SR manual Section 2.5.4) and/or the different solver backends (SR manual Section 1.2).

Report

Your report should have the following sections:

- Variables and Domains: Describe how you chose the variables and domains to represent the problem.
- Constraints: Describe each of the (sets of) constraints you have added to your model, and their purpose.
- Example Solutions: Demonstrate that your model is correct by showing the solution it produces for the three supplied instances: LBS5_0, LBS7_47, and LBS9_0. The style used in the examples given above is fine for this purpose.
- Empirical Evaluation: Describe your experimental setup, and record and analyse the output of the empirical evaluation described above.
- Extensions: Describe here the extensions you have attempted, if any. Models and empirical work should be reported as above, and compared with your original model.

Marking

The practical will be marked following the standard mark descriptors as given in the Student Handbook (see link below). There follows further guidance as to what is expected:

- To achieve a mark of 7 or higher: A rudimentary attempt at a model, incomplete or with several errors. Adequate evaluation and report.
- To achieve a mark of 11 or higher: A reasonable attempt at a model, mostly complete, or complete and with a few flaws. Reasonably well evaluated and reported.
- To achieve a mark of 14 or higher: A good attempt at a model with only minor flaws, well evaluated and reported.
- To achieve a mark of 17: A fully correct model, very well evaluated and reported.
- To achieve a mark greater than 17: In addition to the requirements for a mark of 17, an attempt at the suggested extension activities.

Pointers

Your attention is drawn to the following:

- Mark Descriptors:

  `https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html`
- Lateness:

  `https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html`
- Good Academic Practice:

  `https://info.cs.st-andrews.ac.uk/student-handbook/academic/gap.html`