University of
St Andrews

CS4402 Constraint Programming

The Bombastic Modelling Problem

February 28, 2018

Lecturer:
Ian Miguel

Submitted By:
140011146

1 Introduction

2 Design and Implementation

2.1 Initial model

2.1.1 Initial and goal states

There are three sets of state variables that need to be set up as the initial states: the avatar's position, the locations of the blocks and the cells of the grid.

```
$ Avatar's initial position
avatarCurrentRow[0] = avatarInitRow,
avatarCurrentCol[0] = avatarInitCol,

$ Initial locations for blocks
forall block : int(1..numBlocks) .
    blocksCurrentRow[0,block] = blocksInitRow[block] /\
    blocksCurrentCol[0,block] = blocksInitCol[block],

$ Initial cells of grid
forall row : int(1..r) .
    forall col : int(1..c) .
        gridCurrent[0,row,col] = gridInit[row,col],
```

This sets all "current" decision variables for step 0 based on the given "init" parameter variables. All further constraints will be based on these "current" matrices and their values. Next is the constraint for the goal state.

```
forall block : int(1..numBlocks) .
    exists goal : int(1..numBlocks) .
        blocksCurrentRow[steps,block] = blocksGoalRow[goal] /\
        blocksCurrentCol[steps,block] = blocksGoalCol[goal],
```

Because it does not matter which blocks is pushed into which goal, we can say that for every block there must exist a goal it is in. This combined with the constraint that blocks cannot be in the same position means each block must be in a different goal.

2.1.2 Invalid states

Next are the constraints for invalid states of the game. This restricts the model to not have states such as having the avatar and a block be in the same position.

```
$ Avatar current row/col cannot be on dead cells
forall step : int(0..steps) .
    forall row : int(1..r) .
        forall col : int(1..c) .
            gridCurrent[step,row,col] = 0
```

```
                -> avatarCurrentRow[step] != row \/
                   avatarCurrentCol[step] != col,


$ Blocks and avatar cannot share same cell
forall step : int(0..steps) .
    forall block : int(1..numBlocks) .
        avatarCurrentRow[step] != blocksCurrentRow[step,block] \/
        avatarCurrentCol[step] != blocksCurrentCol[step,block],

$ Block cannot be on dead cells
forall step : int(0..steps) .
    forall block : int(1..numBlocks) .
        forall row : int(1..r) .
        forall col : int(1..c) .
            gridCurrent[step,row,col] = 0
                -> blocksCurrentRow[step,block] != row \/
                   blocksCurrentCol[step,block] != col,

$ Blocks cannot share same cell
forall step : int(0..steps) .
    forall checkBlock : int(1..numBlocks) .
        forall otherBlock : int(1..numBlocks) .
            checkBlock != otherBlock ->
                blocksCurrentRow[step, checkBlock] !=
                    blocksCurrentRow[step, otherBlock] \/
                blocksCurrentCol[step, checkBlock] !=
                    blocksCurrentCol[step, otherBlock],
```

2.1.3 Movement

Now for movement. We need to ensure that if the `avatarCurrentRow` and `avatarCurrentCol` have different positions in different steps (i.e the avatar has moved its position), then `moveRol` and `moveCol` must be updated. Furthermore, the movement cannot be more than one step vertically or horizontally and not diagonally.

```
$ Update moveRow/moveCol for avatar movement
forall step : int(1..steps) .
       moveRow[step] = avatarCurrentRow[step] - avatarCurrentRow[step-1]
           /\
       moveCol[step] = avatarCurrentCol[step] - avatarCurrentCol[step-1],

$ Diagonal movement not allowed
forall step : int(1..steps) .
    | moveRow[step] | + | moveCol[step] | = 1,
```

2.1.4 Grid and ice

Finally, we have to make sure than none of the grid changes unless it is ice and it was stepped on.

2.2 Model improvements

2.2.1 Direct row/col

3 Results

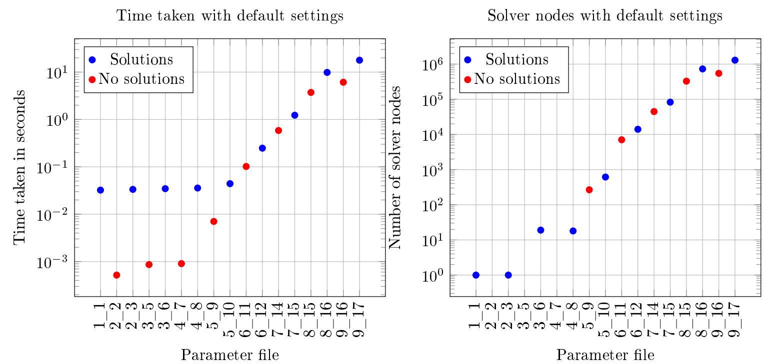

Figure 1: Time taken and number of solver nodes for all given parameters

3.1 Optimisations

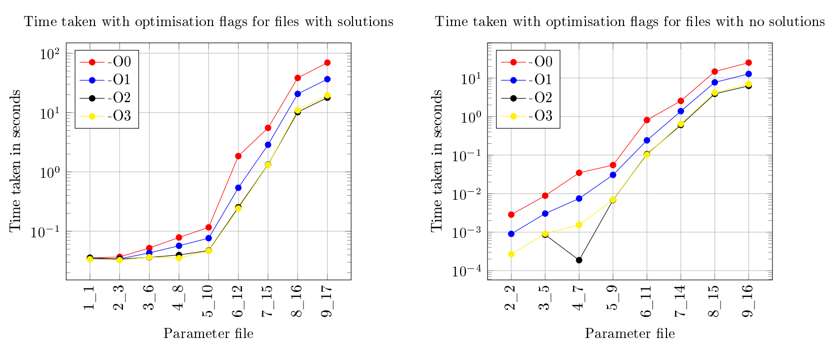

Figure 2: Results with different optimisation flags

3.2 Heuristics

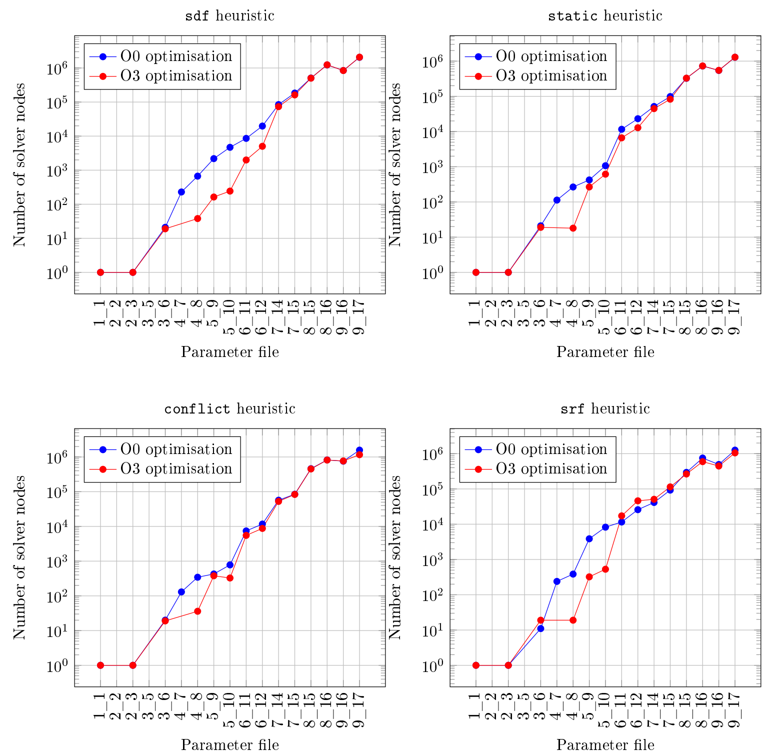

Figure 3: Number of solver nodes for different heuristics

3.3 Custom instances

4 Conclusion and evaluation