Constraint Technology for Solving Combinatorial Problems – Project Assignment: PictureLogic Puzzles*

Magnus Ågren

Computing Science, Information Technology, Uppsala University

agren@csd.uu.se

— To be handed in before midnight July 9.[1] —

1 General Instructions

1. This assignment should be handed in individually.[2]
2. Submit a listing of your program.
3. There should be accompanying documentation describing how your program works. This includes describing the model and any particular relevant features, e.g., heuristics used, custom propagators, search technique(s), special data structures (if any), etc. Don't forget to explain how to compile and run your program.
4. Include sample test runs with input and output. Please make sure that the test runs are reproducible by the programs you hand in.
5. Discuss your results where this is relevant.
6. A *hard-copy* of 2–5 above should be handed in to Magnus *before* the deadline. Please put it in his mailbox (number 55) on the fourth floor.
7. A *soft-copy* should also be sent by email to agren@csd.uu.se. Add your name and information about the email's content to the subject line. If your submission consists of several files, please tar or zip them together before submission.
8. If you need a deadline extension then contact Magnus *before* the deadline.

2 Picture Logic Puzzles

In this assignment, we want you to write a program that solves PictureLogic puzzles. These are very popular number grid puzzles available on Palm PDAs. If you have a Palm PDA, the PictureLogic program and various puzzles can be found on http://www.palmgear.com.

PictureLogic puzzles are played on a rectangular board. The idea is that the cells of the board are used to represent a picture painted with black and white pixels. A cell which is black is a black

*This is a modified version of an assignment in the course Constraint and Logic Programming at National University of Singapore in the spring of 2001. It was originally written by Roland Yap.

[1] This means that I will empty my mailbox on the morning of July 12.

[2] Please do not work in pairs and hand in nearly identical solutions. It is allowed to discuss solution alternatives but not to copy each others programs.

pixel, and a white pixel is simply a blank cell. Clues (or constraints) are provided for rows and columns – usually for all of them. A clue is a list of numbers representing a consecutive sequence of pixels of exactly that length separated by blanks (except at the two ends of the sequence).

In Figure 1, you see a board with 10 rows and 7 columns depicting a candle. The @-sign is used to represent the black pixels and the blanks are the white ones.

```
    3 3
   32223
  3224223
1    @
3   @@@
11  @ @
22 @@ @@
11 @   @
22 @@ @@
3   @@@
111 @  @  @
7 @@@@@@@
7 @@@@@@@
```

Figure 1: Candle (10×7)

In the last row, there is only one 7 clue, which means that that whole row is black since the number of columns is 7. The first row only has a 1 clue, which means that exactly one pixel in that row is black. The third column has the clue $\langle 3, 2, 2 \rangle$, which means that in this column there should first be 3 black pixels, then at least 1 blank pixel, then 2 black pixels, then again at least 1 blank pixel, and finally 2 more black pixels. While in general, the clue constraints can be quite loose, it is the combination of row- and column clues which produces the picture. Most of the puzzles are constructed so that there is only 1 unique solution (however, you will see one example of a puzzle with more than 1 solution). Another example is given in Figure 2 which shows the picture of a bird.

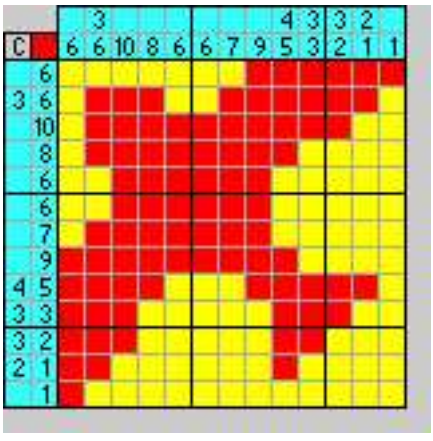

Figure 2: Bird (13×13)

3 The Task

Write a program using your favourite constraint solver that solves such PictureLogic puzzles given the board size, row, and column clue constraints. You can use any convenient input data format that you like. (A fixed file format is probably a good idea. Make it as simple as possible!)

You will probably need more than just arithmetic constraints to model the problem. Browse the solver documentation for appropriate constraints to use. You **may** find out that writing your own custom constraint for propagating information between the row and column constraints is a good idea. You will also need to devise some consistency checking to determine when some cells should be black or white. In our example, you can reason that the last row should be black, the 4 in the middle column should be the last four cells since the last row is black, which means that there should be some space above, and so on...

Many example can be solved by purely applying consistency techniques. However, in some other examples search is necessary, in particular when the solution is not unique.

– Hints –

- On July 21 or 23, Magnus will go through an example of how to create your own (global) constraints in Koalog.
- You can deduce not only when a cell should be black but of course also when it should be white. Use this information!
- You **must** perform some kind of consistency. Naïve search will be too inefficient!

4 Simple Puzzles

Heart (9×9)

```
        2 2 2 2 2 2 2
      3 3 2 2 2 2 2 3 3
  2 2
  4 4
1 3 1
2 1 2
  1 1
  2 2
  2 2
    3
    1
```

Bear (8×13)

```
      2 3   1             1
    2 1 2 6 4 3 4 4 4 5 4 3 2
  1
  2
4 4
 12
  8
  9
3 4
2 2
```

Crocodile (9×15)

```
            2 3 2 3 2 4 3   1 1 1
          3 4 2 1 3 2 3 2 2 6 3 3 4 5 5
        3
    2 3 2
     10 3
       15
1 1 1 1 6
      1 7
      1 4
      1 4
        4
```

5 Harder Puzzles

Unknown (10×10)

```
              1   1 1
          2 2 2 2 1 4 1 3
        3 1 2 1 1 1 1 2 1 4
      3
    2 1
    1 1
    1 4
1 1 1 1
2 1 1 1
  2 1 1
    1 2
    2 3
      3
```

Pinwheel (6×6)

```
    1         2
    2 1 2 2 1 1
2 1
  1
  2
  2
  1
1 2
```

Difficult (15×15)

```
                        2 1 1 1
    3 2 2 1 2 3 2 4 3 4 1 1 1 1 3
  3
1 1
1 1
1 1
1 2
  5
  1
  2
  1
  1
1 2
1 2
2 1
2 2
  3
```

Non-unique (15×11)

```
                    2 2
              2 1   1 1
          1 2 2 1   1 1 1
          2 1 1 1 1 3 1 4 1
        5 4 3 1 1 5 2 1 1 1 1
    2 2
    2 2
      4
    1 1
    1 1
1 1 1 1
    1 1
    1 4
  1 1 1
  1 1 4
    1 3
    1 2
      5
    2 2
    3 3
```