# ‘Flow Free’ as a SAT problem

Michael Spivey, Michaelmas Term 2012

The puzzle game ‘flow free’ begins with a square or rectangular grid of cells. Pairs of cells are marked with coloured blobs as shown on the left in Figure 1, and the puzzle is to link the two blobs in each pair with paths that move horizontally and vertically through the grid, never crossing each other, as shown on the right. In a correct solution, all cells in the grid are filled with paths. In this note we consider only paths that do not double back on themselves, so that each point on a path other than the endpoints will have exactly two horizontal or vertical neighbours that are also part of the path.

## 1 Reduction to SAT

We can transform each instance of the puzzle as an instance of SAT. Suppose the puzzle has $N$ cells and $C$ pairs of blobs in different colours. We introduce $NC$ boolean variables $v_{kc}$ for $1 \le k \le N$ and $1 \le c \le C$, with $v_{kc}$ taking the value true if cell $k$ is part of the path for colour $c$. A number of constraints must be obeyed in any solution. First, each blob must have its specified colour; if $k$ is a blob with colour $c$, then

$$v_{kc}, \qquad (1)$$

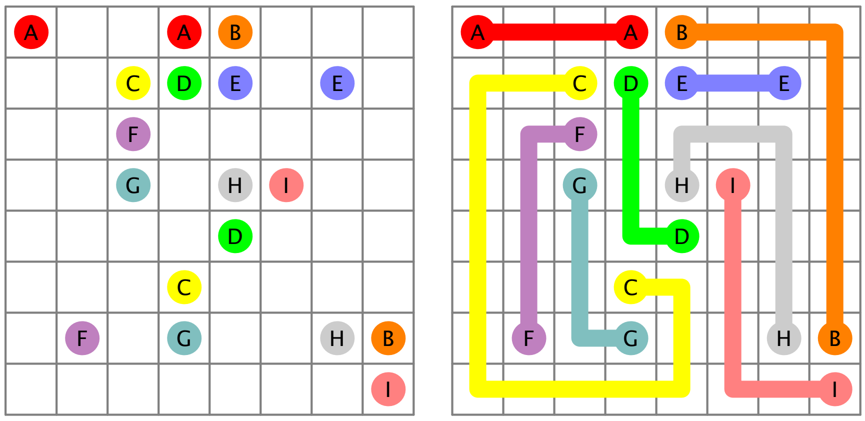

**Figure 1:** *A puzzle and its solution*

and if $c' \neq c$ then

$$\neg v_{kc'}. \tag{2}$$

Thanks to these unit clauses, we could eliminate entirely the variables $v_{kc}$ where $k$ is a blob, making the SAT instance slightly smaller. However, SAT solvers typically do such simplifications as their first step, so there is no need for us to complicate our program and duplicate the effort.

Each non-blob may have any colour we choose, but must have only one colour, so of the variables $v_{k1}, \dots, v_{kC}$, exactly one must be true:

$$\mathit{oneof}(v_{k1}, \dots, v_{kC}). \tag{3}$$

In order for the paths to join up properly, each blob must have exactly one neighbour that shares its colour. If $k$ is a blob of colour $c$ and its neighbours are $\{n, s, e, w\}$, then we must have

$$\mathit{oneof}(v_{nc}, v_{sc}, v_{ec}, v_{wc}). \tag{4}$$

If $k$ has fewer than four neighbours, the missing neighbours are to be omitted from (4). For non-blobs $k$, there must be exactly two neighbours with the same colour. So for each colour $c$, we must have

$$v_{kc} \Rightarrow \mathit{twoof}(v_{nc}, v_{sc}, v_{ec}, v_{wc}), \tag{5}$$

again with missing neighbours omitted.

These constraints make sure that all the cells join up into paths that can end only at blobs, but they don't explicitly rule out loops that don't connect to any blob. I've yet to find an example that allows a loop but also has a valid solution, or to prove that such loops are impossible.

We expressed the constraints in terms of relations $\mathit{oneof}(x_1, \dots, x_n)$ and $\mathit{twoof}(x_1, \dots, x_n)$ that repectively hold if exactly one or exactly two of the variables $x_1, \dots, x_n$ are true. The relation *oneof* may be rendered as clauses by saying that at least one of the variables is true:

$$x_1 \vee x_2 \vee \cdots x_n,$$

and for each $i$, $j$ with $1 \leq i < j \leq n$, the variables $x_i$ and $x_j$ are not true simultaneously:

$$\neg x_i \vee \neg x_j,$$

a total of $n(n+1)/2$ clauses.

We need to express the relation $y \Rightarrow \mathit{twoof}(x_1, \dots, x_n)$ only in the cases where $n$ is 2, 3, or 4, but we also need to make the relation conditional on another variable $y$. If $n = 2$, then the relation is simply $y \Rightarrow x_1 \wedge x_2$, which we can render trivially as the two clauses,

$$\neg y \vee x_1, \qquad \neg y \vee x_2.$$

If $n = 3$, then we require that in any two of the three variables, one or both are true, and in addition that not all three are true. So if $1 \leq i < j \leq 3$, we must have

$$\neg y \vee x_i \vee x_j,$$

and also

$$\neg y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3,$$

making four clauses in total.

If $n = 4$, then what we require is that in any three of the variables, one is true and one is false; for if three or more of the variables are true, we can find three that are all true, and if three or more are false, we can find three that are all false. So if $1 \leq i < j < k \leq 4$, we need the clauses

$$\neg y \vee x_i \vee x_j \vee x_k,$$

and

$$\neg y \vee \neg x_i \vee \neg x_j \vee \neg x_k,$$

making eight clauses altogether.

The pattern generalizes to $n$ variables, requiring that in any $n - 1$ of them at least one variable is true, and in any three of them at least one is false. This can be expressed in $O(n^3)$ clauses.

## Experimental results

I implemented the translation as a C program of about 250 lines that would generate a file in the input format of Minisat and other SAT solvers, and also interpret the satisfying assignment output by Minisat as a solution to the original puzzle.

For the $8 \times 8$ puzzle shown in Figure 1, the translation produces a total of 4,419 clauses in 576 variables. Minisat finds a satisfying assignment instantaneously, and this gives the solution shown in the figure. For a larger puzzle with $14 \times 24$ cells, the translation produced exactly 72,000 clauses in 5,376 variables. This problem took about 7 seconds to solve using 35MB of memory.