

# Dominoes as a Constraint Problem

Helmut Simonis

No Institute Given

**Abstract.** In this note we describe a model for the Dominoes puzzle using finite domain constraint programming.

## 1 Introduction

Dominoes is a small logic puzzle from [5, 6]. The rules are as follows:

0	5	2	2	5	4	6	5
3	6	2	2	4	4	4	1
3	6	1	2	3	4	6	1
0	1	4	3	0	2	2	1
3	5	3	0	3	1	5	6
6	4	0	3	6	0	4	1
1	6	0	0	2	5	5	5

**Fig. 1.** Example Problem

1. You are given a board consisting of 7 rows and 8 columns. Each cell contains a number from 0 to 6.
2. The board is made up from using all dominoes starting from 0-0 to 6-6 exactly once.
3. The dominoes cover the board completely, but they don't overlap.
4. The dominoes may be rotated in any direction, the orientation of the numbers is not significant.

An example problem from [5] is shown in figure 1. The solution to this puzzle is shown in figure 2.

## 2 Related Work

The puzzle is related to *Shikaku*, whose model is described in [4]. We use the same decomposition method to solve the puzzle. We first identify the places

0	5	2	2	5	4	6	5
3	6	2	2	4	4	4	1
3	6	1	2	3	4	6	1
0	1	4	3	0	2	2	1
3	5	3	0	3	1	5	6
6	4	0	3	6	0	4	1
1	6	0	0	2	5	5	5

**Fig. 2.** Example Solution

where the different dominoes can be placed, and then select a subset of these patterns which uses every domino once and covers the board.

A very different problem, but also using domino pieces, are *Domino Portraits* [1], where grayscale images are approximated by using multiple sets of domino pieces.

### 3 Model

One way of modelling this puzzle would use a 2-D *diffn* constraint, where the set of dominoes must be placed inside a 7x8 rectangle without overlap. Expressing the restrictions on domino placement by the given hints would require extra constraints, and possibly a switch to a Cartesian product domain. As ECLiPSe does not provide the *diffn* constraint, we choose an alternative model.

We introduce 0/1 integer variables for every 2x1 pattern that can be placed on the board. The pattern can be placed either horizontally or vertically. Looking at the hints inside the pattern we know which domino could be placed at this pattern.

We now have two sets of constraints, setting up a set partitioning problem.

- The sum of all variables in pattern for the same domino must be equal to one, as each domino must be used once and once only.
- The sum of all pattern covering a cell in the grid must be equal to one as well, as every cell must be covered, and no dominos may overlap.

Finding an assignment for the variables defines which pattern are used, this tiles the board completely.

The model can be easily expressed as a finite domain program, or as a MIP model, or a Pseudo-Boolean model. As described in [4], we can also derive a SAT model directly from the non-overlap condition. As the puzzle seems quite simple, we did not explore these alternatives, and only implemented the finite domain model.

## 4 Results

In table 1 we show the results of testing the program on a few dominoes problems from [5, 6, 3, 2]. Each line shows the data for one puzzle, the columns have the following meaning:

**Set** the name of the puzzle collection

**Nr** the number of the puzzle in the set

**Grade** the grade assigned to the puzzle

**X,Y** the board size

**Single** number of dominoes occurring only once on the board, this pattern will be fixed

**Cover** number of cells covered by a single pattern, this is zero as every cell is at least covered by two dominoes

**Setup** number of variables after initial constraint setup, before shaving

**Shave** number of variables left after shaving

**Back** number of backtracking steps in the search routine

**Time** total time to solve the puzzle

Set	Nr	Grade	X	Y	Single	Cover	Setup	Shave	Back	Time
wh	0	easy	7	8	3	0	73	0	0	0.02
wh	9071	easy	7	8	4	0	0	0	0	0.01
wh	9072	easy	7	8	3	0	0	0	0	0.00
wh	9073	easy	7	8	3	0	0	0	0	0.00
wh	9074	easy	7	8	3	0	64	0	0	0.01
wh	10071	easy	7	8	5	0	0	0	0	0.01
wh	10072	easy	7	8	4	0	0	0	0	0.00
wh	10073	easy	7	8	3	0	66	0	0	0.01
wh	10074	easy	7	8	0	0	84	0	0	0.00
handley	1	easy	7	8	3	0	0	0	0	0.01
keller	1	easy	7	8	0	0	97	0	0	0.00
keller	2	easy	7	8	0	0	97	0	0	0.02
keller	3	easy	7	8	0	0	97	0	0	0.01
keller	4	easy	7	8	0	0	97	0	0	0.01

Table 1: Summary

We can see from the results that some puzzles are just solved by constraint propagation, while others require shaving to avoid search. It is possible to derive redundant constraints which avoid the shaving, but this was not attempted here.

## 5 Summary

In this note we have described a simple constraint program to solve a small logic puzzle to cover a given board with domino pieces. Instead of treating the

placement problem directly, we decompose the problem into a pattern generating step and a set partitioning problem. With this approach the given problems are quite simple.

## References

1. Robert Bosch. Opt art. In J. Christopher Beck and Barbara M. Smith, editors, *CPAIOR*, volume 3990 of *Lecture Notes in Computer Science*, page 1. Springer, 2006.
2. John Handley Highschool Mathematics Department. Domino puzzle, 2007. <http://www.pen.k12.va.us/Div/Winchester/jhhs/math/puzzles/domino.html>.
3. Michael Keller. Dissected dominoes, 2007. <http://www.gamepuzzles.com/tlog/tlog33.htm>.
4. H. Simonis. Shikaku as a constraint problem, 2007.
5. various. Beyond soduko issue 9, 2007.
6. various. Beyond soduko issue 9, 2007.