Generating and Solving Logic Puzzles through Constraint Satisfaction

Barry O'Sullivan and **John Horan**
Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
b.osullivan@4c.ucc.ie and jjh2@student.cs.ucc.ie

Abstract

Solving logic puzzles has become a very popular past-time, particularly since the Sudoku puzzle started appearing in newspapers all over the world. We have developed a puzzle generator for a modification of Sudoku, called Jidoku, in which clues are binary disequalities between cells on a 9×9 grid. Our generator guarantees that puzzles have unique solutions, have graded difficulty, and can be solved using inference alone. This demonstration provides a fun application of many standard constraint satisfaction techniques, such as problem formulation, global constraints, search and inference. It is ideal as both an education and outreach tool. Our demonstration will allow people to generate and interactively solve puzzles of user-selected difficulty, with the aid of hints if required, through a specifically built Java applet.

Introduction

The primary objective of this work is to use the current popularity of logic puzzles, such as Sudoku (Hayes 2006; Jussien 2007), amongst the general public as an opportunity to explain the power of artificial intelligence techniques such as constraint satisfaction. We present a demonstration of a puzzle generator and interactive solver for a well-known variant of Sudoku called Jidoku, or Less-Than-Sudoku.

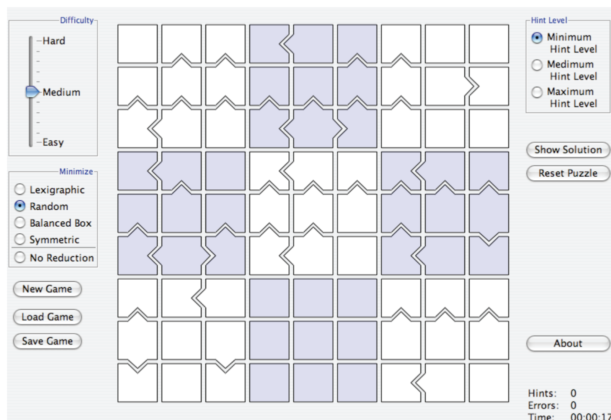


Figure 1: An example Jidoku puzzle.

Copyright © 2007, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

An example of a Jidoku puzzle is presented in Figure 1. The rules of the puzzle are simple. Given a 9×9 grid, one must place the integers $1 \ldots 9$ on each row, column and 3×3 box such that the constraints between the cells, displayed by modifying the edges of the cells to appear as $<$ or $>$, are satisfied. The solution to the puzzle in Figure 1 is presented in Figure 2.

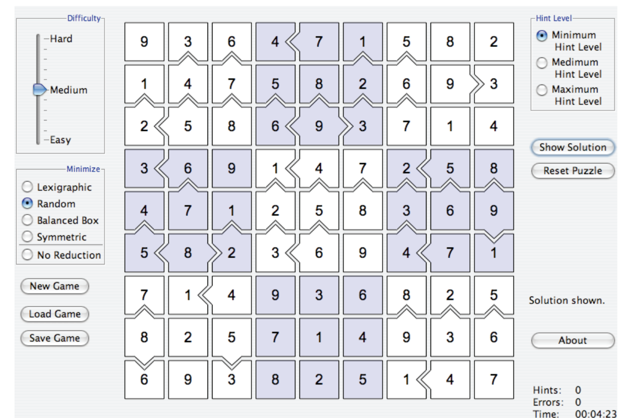
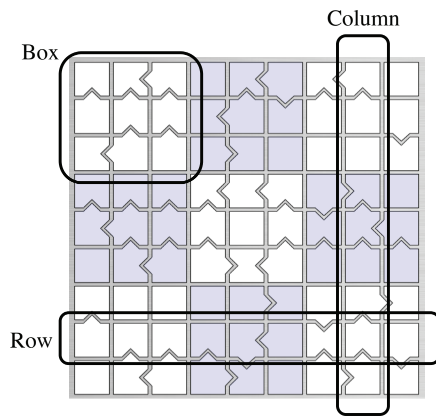


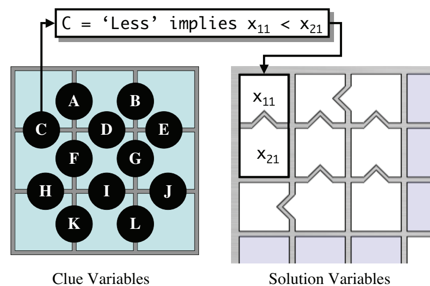
Figure 2: The solution to the puzzle in Figure 1.

There are a number of important properties that a puzzle should have. Firstly, a puzzle must have a unique solution so that it can be printed in a newspaper, book or presented on-screen for the player. Secondly, a puzzle should be soluble by using inference and logic alone in order to make the puzzle interesting and engaging from a human player's point of view. Thirdly, the puzzle must have a level of difficulty associated with it so that a player has some expectation of the complexity of the inference that is required to solve it. Finally, there may be some aesthetic visual property that we may prefer, e.g. that the clues are positioned in a rotationally symmetric manner (this is a cardinal rule of style for crossword puzzles published in The New York Times). The puzzle generator we present generates puzzles that have all these properties. This is an interesting application for constraint satisfaction, in particular problem formulation, global constraints, search and inference (Dechter 2003).

The Sudoku puzzle has been formulated as a constraint satisfaction problem (CSP) in the past (Simonis 2005), but



(a) The constraints of the basic puzzle.



(b) The constraints between clue and solution variables.

Figure 3: Constraint model for generating Jidoku puzzles.

there the focus was on solving rather than generating the puzzles. Our primary concern here is the generation of interesting human soluble puzzles. The generator is presented in the form of a Java applet that has been implemented using Choco, an open-source constraint programming system, which is available from `http://choco-solver.net`.

Approach

Basic Puzzle CSP. The basic underlying model of a Jidoku puzzle is the same as that of a Sudoku: in a 9×9 puzzle each integer $1 \dots 9$ must appear in each row, column and box. The CSP formulation is simple: we have 81 variables, each representing a cell in the puzzle; the domain of each variable is the set of values $1 \dots 9$; we post an `alldifferent` constraint (Régin 1994) on each column, row and box (see Figure 3(a)), giving 27 global constraints.

Formulating the Puzzle Generation Problem. To generate the puzzle we add additional *clue variables* to the basic model of the puzzle to represent the relations between adjacent cells in the puzzle, represented by *solution variables*, as shown in Figure 3(b): in this example we show the variables $A, \dots, L$ showing the cells whose relation is determined by them. The domain of each of these variables is $\{Less, Greater\}$. Channelling constraints between the clue variables and the solution variables define the relationship between them for every possible assignment to the clue variables. For example, in Figure 3(b) we show the constraint that states than if the clue variable C is set to be $Less$ then the value taken by solution variable x_{11} must be less than the value of solution variable x_{21}, indicated by the shape of the border between these solution variables.

The Search Procedure. The search procedure is a depth-first search algorithm that finds an assignment to the clue variables than is sufficient to reduce the domains of the solution variables to singletons by propagating the constraints of the puzzle generation problem. Relying on propagation here is important since it can allow us to control the difficulty of the puzzles, as we will see later.

Minimising the Puzzle. Based on the reduced domains of the solution variables, a complete assignment to the clue variables is immediately obtained by constraint propagation. We are certain that this set of clue assignments gives rise to a puzzle with a unique solution. It now remains to obtain a puzzle of the desired level of difficulty, and satisfying any aesthetic preferences we may have, by removing redundant clues from the final puzzle.

We revisit each clue variable and test if unassigning it, thus removing a constraint between a pair of solution variables, would still ensure that the domains of the solution variables remain singletons, and if so we unassign the variable, otherwise we mark it as required. We iterate over all clue variables until we cannot unassign any more of them. We can obtain clue-symmetric puzzles by visiting the clues variables in an order that is consistent with a rotational symmetry of the grid, for example.

The difficulty of the puzzle is controlled by replacing some of the `alldifferent` constraints with cliques of binary inequality constraints. The effect of this is that a weaker level of consistency is maintained on these constraints. We have a maximum of 27 `alldifferent` constraints (one for each row, column and box). Therefore, we can define 27 levels of "difficulty" for our puzzles by varying the formulation of the puzzle we use in the puzzle minimisation process.

Acknowledgements. This work was supported by Science Foundation Ireland (Grant 05/IN/I886). We thank Hadrien Cambazard for his help with the Choco constraint programming system.

References

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Hayes, B. 2006. Unwed numbers. *American Scientist* 94:12–15.

Jussien, N. 2007. *A to Z of Sudoku*. ISTE/Hermes.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *AAAI*, 362–367.

Simonis, H. 2005. Sudoku as a constraint problem. In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 13–27.