An Interactive Constraint-Based Approach to Sudoku

Christopher G. Reeson
*University of Nebraska-Lincoln*, creeson@cse.unl.edu

Kai-Chen Huang
*University of Southern California*, kaichenh@usc.edu

Ken Bayer
*University of Nebraska-Lincoln*, kbayer@cse.unl.edu

Berthe Y. Choueiry
*University of Nebraska-Lincoln*, choueiry@cse.unl.edu

Follow this and additional works at: https://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

https://digitalcommons.unl.edu/cseconfwork/171

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
An Interactive Constraint-Based Approach to Sudoku

Christopher G. Reeson1 and Kai-Chen Huang2 and Kenneth M. Bayer1 and Berthe Y. Choueiry1,2
1 Constraint Systems Laboratory, Department of Computer Science & Engineering, University of Nebraska-Lincoln
2 Information Sciences Institute, University of Southern California
creeson@cse.unl.edu, kaichenh@usc.edu, kbayer@cse.unl.edu, choueiry@cse.unl.edu

Introduction and Goals
We present Solver, a Java applet that uses Constraint Processing (CP) techniques to assist human users in solving Sudoku puzzles. We also showcase Constructor, another applet that allows users to enter and store puzzles, which they can then load in Solver. These applets are available from sudoku.unl.edu/Solver and sudoku.unl.edu/Constructor. Our goals are as follows:

• To the public: Illustrate the power of CP techniques in the context of a popular easily approachable puzzle.
• For the research: Investigate how to use CP to interactively support and guide human players.
• For education: Use the Sudoku puzzle as a means to teach basic and advanced constraint propagation techniques.
• For insight in human reasoning: and understand how it differs from algorithmic approaches.

Solver is built to maximize the interactions between the human users and CP techniques. It allows users to apply various consistency algorithms to specific constraints or the entire puzzle, and interactively make assignments and domain reductions. We also designed a ‘hint’ functionality that uses increasingly complex propagation algorithms in a controlled manner to guide and train the users to play the game.

Background
The Sudoku Puzzle, originally named Number Place, was created by Howard Garns in 1979, and originally appeared in the Dell Pencil Puzzles and Word Games magazine. Nikoli began publishing Sudoku Puzzles in 1986 and introduced the Sudoku name, trademarked in Japan. More recently, newspapers across the United States have begun publishing puzzles daily. The game has become immensely popular and many web sites are devoted to this puzzle1.

Several programs have been developed and released on the Internet that allow users to solve Sudoku puzzles. These implementations vary widely. Some programs provide no support to the player2. Others offer suggestions using a set of rules loosely based on logic3. Finally, others resort to backtrack search when their rules are not able to solve the puzzle4. Those rules are often ‘tricks’ resulting from generalizing observations made while solving puzzles. They are often based on convoluted patterns and have strange names, such as naked singles, cross-hatching, locked candidates, etc. (van der Werf 2007). They are not grounded in CP techniques and, often times, a number of seemingly distinct rules are subsumed by a single constraint propagation technique.

On the theoretical side, Yato has shown that Sudoku is NP-Complete by reduction from the Latin Square (Yato 2003). Simonis (2005) studied various models of the constraints in the basic 9×9 Sudoku puzzle and their associated propagation algorithms (often based on finding matchings in bipartite graphs). He compared their ability to solve puzzles of various difficulties. Our approach differs from Simonis’s in that we restrict ourselves to the most basic model of the constraints, and use only the most common constraint propagation algorithms (a subset of those used by Simonis). Unlike Simonis, we focus on the interactive aspects of the game, and have designed a ‘hint’ functionality that uses constraint propagation mechanisms of increasing complexity to guide the human players and train them in playing the game.

Constraint Modeling and Solving
Many variations of the Sudoku puzzle exist. While we plan to extend our interfaces to handle about 10 puzzle variations, our current implementation handles only the basic puzzle. This puzzle is a 9×9 grid, where some cells in the grid are filled with numbers in [1..9]. The task is to complete the grid by placing a number from 1 to 9 in each of the empty cells, while ensuring that a number appears exactly once in every row, column, or the nine 3×3 ‘blocks’ that form the grid. A well-posed Sudoku has exactly one solution (Simonis 2005). Naturally, since the problem has a fixed size, it can be solved in fixed time (there are at most 981 combinations).

The model of the Sudoku as a Constraint Satisfaction Problem (CSP) is straightforward. We use two representations: the binary and the non-binary models. In the former, we have 810 binary ‘all-different’ constraints, one constraint between any two cells that cannot take the same value. In the

3www.sudoku.co.uk, www.sudocue.net
4www.techfinesse.com/game/sudokuSolver.php
latter, we have 27 ‘all-different’ constraints, each of arity 9, defined over the rows, columns, and blocks.

We used the following common propagation algorithms. (References not provided for lack of space.) On the binary constraints: Arc Consistency (AC) and Singleton Arc-Consistency (SAC). On the non-binary constraints: Generalized Arc-Consistency (GAC) and Singleton GAC (SGAC). We filter the domains of the CSP variables interactively as the user places values in the cells with Forward Checking (FC) and Maintaining Arc-Consistency (MAC). Finally, a depth first backtrack search displays all solutions or gives their number on demand.

**Interface**

The interface allows the user to switch between **Solver** and **Constructor**. **Solver** is shown in Figure 1. The ‘Load Instance’ button (top left) allows the user to load any puzzle stored in the library. **Constructor** (not pictured) allows users to expand the library by entering and storing new instances, and modifying existing ones. The main functional-