An Interactive Constraint-Based Approach to Minesweeper

Ken Bayer, Josh Snyder, and Berthe Y. Choueiry
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Email: `kbayer|jsnyde|choueiry@cse.unl.edu`

Introduction

We present a Java applet that uses Constraint Processing (CP) to assist a human in playing the popular game Minesweeper. Our goal is to illustrate the power of CP techniques to model and solve combinatorial problems in a context accessible to the general public.

Minesweeper is a video game that has been included with Microsoft Windows since 1989. In this game, the player is presented with a grid of squares. Each of these squares may conceal a mine. When the player clicks on a square, it is revealed. If the square is a mine, the game is over. If the square is not a mine, it is replaced by a number indicating how many of the adjacent squares are mines. The goal of the game is to reveal all squares that are not mines. These simple rules yield a complex problem: Kaye proved that the minesweeper-consistency problem is **NP**-complete (2000). The minesweeper-consistency problem is to determine if, given a board with some known squares, there exists a layout of mines in the unknown squares that is consistent with the numbers displayed.

A few programs have been written to solve Minesweeper. One notable such program was developed by Collet (2004), and uses the Oz language to model and solve Minesweeper as a set of Boolean linear constraints. In his implementation, Collet defines constraints intensionally and generates dual variables to achieve higher-level consistency[1]. This model artificially and unnecessarily increases the number of variables (thus increasing the cost in terms of time and space), and obscures the concept of 'consistency level,' which is central to Constraint Processing.

Like (Collet 2004), our program assists the player in solving Minesweeper puzzles by modeling them as Constraint Satisfaction Problems (CSPs). Although our model is similar to that of Collet, it was developed independently. Also, our constraints are defined in extension, and, importantly, our implementation uses textbook propagation-algorithms to determine the locations of mines, thus better serving our pedagogical goals.

We briefly discuss our motivations, then describe our model, interface and implementation.

[1]Collet also generates all solutions (while exploiting symmetries) and infers the probability that a square is mined. We have not yet considered these extensions.

Motivation: an educational tool

The primary motivation for developing this application is to use it as an educational tool. At the University of Nebraska-Lincoln, Constraint Processing is taught in two introductory courses offered every year (Introduction to Artificial Intelligence and Foundations of Constraint Processing), and one advanced course offered every 2 years (Recent Advances in Constraint Processing). This application is being used to help students visualize and understand modeling with constraints and the mechanisms of constraint propagation.

Many students are familiar with Minesweeper and likely already have an intuition about how to solve it. With this application, we can help them demystify their intuitions about puzzle solving and relate these intuitions to the more formal concept of consistency level.

CSP model

In our CSP model for Minesweeper, we generate a Boolean variable for every square on the board with the two possible values *mined* and *safe*. We generate a constraint for every revealed square on the board, whose scope is the set of squares adjacent to the revealed square, see Figure 1. This is a "sum" constraint

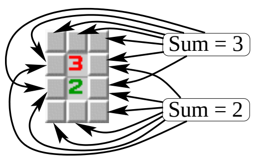

Figure 1: CSP model.

that specifies the total number of mines in the adjacent squares. For example, a square labeled 3 yields a constraint stating that the square be surrounded by 3 mines. All constraints are defined in extension, that is by enumerating the set of allowed assignments. While this representation requires storing constraints of up to 70 tuples, it allows us to perform higher of levels consistency by simple constraint composition (Dechter 2003).

Interface

Our interface (see Figure 2) implements Minesweeper using the same rules as the version that comes with Microsoft Windows. The player can click on a square to

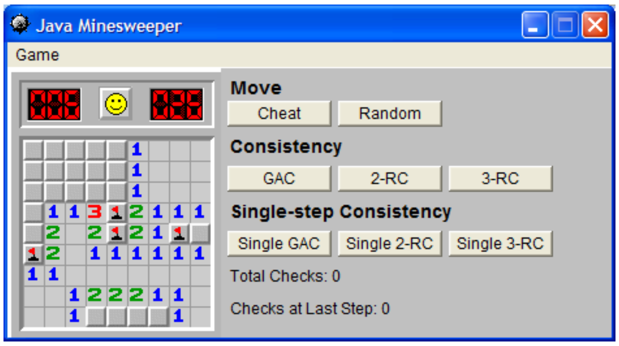

Figure 2: Interface.

reveal it, or right-click on a square to mark it as a mine. The player can choose one of three difficulty levels corresponding to different board sizes and mine densities. The player can also create a custom-size board. In addition to the basic features, we can also load a predefined board configuration from an XML file, which allows us to test specific configurations of interest.

Our interface has buttons to activate the various constraint-based 'assistants.' These buttons are **GAC**, **2-RC**, and **3-RC**. Pressing the **GAC**-button launches the Generalized Arc Consistency (GAC) algorithm (Mohr & Masini 1988) on the current state of the game. This algorithm considers every constraint independently, and attempts to infer the values of the unexplored squares in the scope of the constraint from the values of the explored ones. If an explored square is deemed to have exactly one possible value, it is immediately flagged accordingly. In particular, if it is safe, a new constraint is dynamically generated using the number revealed by the square. This process is similar to a human player flagging and expanding squares that are obviously mined or safe.

GAC is also called 1-Relational Consistency (RC), because it looks at every constraint independently. Higher levels of consistency, such as 2-RC and 3-RC, look at combinations of constraints (Dechter 2003). 2-RC (respectively, 3-RC) looks at every combination of 2 (respectively, 3) constraints with overlapping scopes. It computes the join of these constraints to create the list of consistent assignments, then filters accordingly the domains of the corresponding variables. We choose not to store the constraints generated by the consistency algorithms for space consideration. The **2-RC** and **3-RC** buttons enforce these levels of consistency in cascaded manner, that is until no propagation can be done.

The display of the game board always reflects exactly the state of the CP model. Thus, as constraints are propagated, the player actually sees what is happening. Because propagation can be too quick for a student to observe, our application offers a single-step version for each of the propagation algorithms. These buttons perform constraint propagation, but they stop as soon as the solver either expands a safe square or flags a mine. Using these buttons, the player can step through the algorithms observing one change at a time. Because the interface is not performing search by doing constraint propagation, there is no concept of backtracking and no need to undo propagations. Further, we are implementing a preview mode that highlights the squares a given algorithm will expand or flag before the propagation is actually executed in an effort to provide a visual support to the actual operation of the algorithms.

Implementation

We implemented this application as a Java applet. This allows us to embed the program in a web page, and make it publicly accessible at `consystlab.unl.edu/our_work/minesweeper.html`.

The architecture of the application is as follows. There is a `MineSweeperWindow` object that handles all of the display. This stores a `MineField` object that manages the current game board. The last major object is the `RC` object, which implements the constraint propagation techniques described above. Note that our solver does not store a separate copy of the variables. Rather, it reads and writes directly to the `MineField` object. This decision is intentional and allows our display to be continually updated as the solver proceeds. Likewise, our solver is able to immediately take into account any flags or expansions performed by the player.

Conclusion

Our application implements the game of Minesweeper and three different levels of consistency-enforcement techniques. Our effort allows us to illustrate the concept of constraint satisfaction and its related mechanisms of constraint propagation in the context of an application understandable to most people. As such, our program is useful for explaining the significance and power of Constraint Processing to students and also to the public. In the future, we plan to expand the current implementation to illustrate more advanced concepts such as tree-width and backbone variables.

References

Collet, R. 2004. Playing the Minesweeper with Constraints. In *Multiparadigm Programming in Mozart/OZ: Second International Conference, MOZ 2004*, 251–262. Springer, LNCS 3389.

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

Kaye, R. 2000. Minesweeper is np-complete. *Mathematical Intelligencer* 22(2):9–15.

Mohr, R., and Masini, G. 1988. Good Old Discrete Relaxation. In *European Conference on Artificial Intelligence (ECAI-88)*, 651–656.