Automatically Generating Puzzles of Different Complexity

September 22, 2014

1 General Approach

Definition 1. *We define a puzzle board $\mathcal{P}$ to be a two dimensional $n \times m$ board. Let V denote a function that assigns values to the squares on the puzzle board such that the value of the square (i, j) is defined by $V(i, j)$, where $1 \leq i \leq n$, $1 \leq j \leq m$, $V : \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{D}$, and $\mathcal{D}$ denotes the set of all possible values the puzzle squares can take.*

The main components of our technique are:

- A declarative definition of puzzle D
- A Complexity Function : C
- A set of Transformation Functions : $\tilde{T}$

Definition 2. *A declarative definition of puzzle D defines constraints over the set of valid values $V(i, j)$ that puzzle squares can take.*

Definition 3. *The complexity function $C : \mathcal{P} \rightarrow H$ takes a puzzle board puzzleboard as input and maps it to a finite class of hardness levels denoted by H.*

Definition 4. *A transformation function $T : \mathcal{P} \rightarrow \mathcal{P}$ takes a puzzle board as input and transforms it another puzzle board such that the new puzzle board also satisfies the puzzle constraints. The set of all transformation functions are denoted by $\tilde{T}$.*

```
The General Puzzle Creation Algorithm
P_I = GetRandomPuzzle(P_D)

While isRemoveValid(P_c,i,j):
  P_c = P_c - {i,j}
  For T \in T_P:
    R = R U T(P_c)

For P \in R:
  D[F_c(P)] = P
```

The algorithm first uses an off-the-shelf constraint solver to solve the puzzle constraints P_D to get an initial random board configuration. It then starts removing the value at a square (i,j) until there are no more square values remaining to be removed. The algorithm uses the isValidSquare function to check if certain puzzle constraints hold after removing the square value (i,j). An example isRemoveOK function is that the Number of solutions to the puzzle after removing the square value is only 1. Let P_c denote the puzzle obtained after removing a valid square. We then apply a set of transformation functions T_P to get a set of new puzzles.