CMPSCI-683 Homework Assignment #2: Constraint Satisfaction and Adversarial Search

Patrick Pegus II

February 19, 2016

Problem 1. Show that any CSP that has a finite domain for each variable can be transformed into a CSP with only binary constraints. Follow these steps:

1. Show how a single ternary constraint such as $A + B = C$ can be turned into three binary constraints by using an auxiliary variable. (Hint: consider a new variable AB whose domain is pairs of numbers.)
2. Next, show how constraints with more than three variables can be treated similarly.
3. Finally, show how unary constraints can be eliminated by altering the domains of variables.

1. Let $D_{AB} = \{a+b | a \in D_a, b \in D_b\}$ be the domain of the variable AB. Then $A+B = C$ can be replaced by $AB = C$, $A = a$, $B = b$.
2. Iteratively remove variables from non-binary constraints, such as

   $$V_1 * \cdots * V_i * V_{i+1} * \cdots * V_m = V_n$$

   by replacing it with

   $$V_1 * \cdots * V_i V_{i+1} * \cdots * V_m = V_n,\ V_i = v_i,\ V_{i+1} = v_{i+1}$$

   until only binary constraints exist. Where $D_{V_i V_{i+1}} = \{v_i * v_{i+1} | v_i \in D_{v_i}, v_{i+1} \in D_{v_{i+1}}\}$.
3. Any unary constraint U that imposes a condition on variable V, such as $U(V) = True$, can be eliminated after reducing the domain of V to only values v where $U(v) = True$.

Problem 2. Sudoku is a fairly old puzzle that is now a worldwide phenomenon. You can type "sudoku" into Google, or read the Wikipedia article to get more information than you could possibly imagine. Key facts about standard Sudoku puzzles:

- Every puzzle has a unique solution.
- Every puzzle is solvable without trial-and-error, given suitable inference methods.

- The designated difficulty of a puzzle is usually determined by the difficulty of the inference methods required to solve it.

In addition to the rules, many web sites offer extensive discussion of methods that humans can use to solve Sudoku without using trial-and-error search. You need to write a program that can solve Sudoku problem instances. To test your program, you should apply it to the following 16 puzzles: See `/data/sudoku_puzzles/`.
These puzzles are from "Sudoku: Easy to Hard", by Will Shortz. The original collection includes 100 puzzles (from 1-25 are 'light and easy', 26-50 are 'moderate', 51-75 are 'demanding', and 76-100 are 'beware! very challenging'). Each puzzle is stored in a file that looks as follows:

```
- 1 9 - - - - - -
- - 8 - - 3 - 5 -
- 7 - 6 - - - 8 -
- - 1 - - 6 8 - 9
8 - - - 4 - - - 7
9 4 - - - - - 1 -
- - - - - 2 - - -
- - - - 8 - 5 6 1
- - 3 7 - - - 9 -
```

1. Explain how Sudoku can be represented as a CSP (how many variables are needed? what are their domains? what are the constraints?).

2. Write a program that can read a Sudoku puzzle from a file and solve it using the backtracking-search algorithm that was discussed in class (slide 16). Keep in mind that instead of representing the constraints explicitly in your program, it might be easier to write a function that checks if a given variable assignment results in a conflict with the values already placed in its row/column/square.

   Try to solve a few of the Sudoku instances using this algorithm and make sure that it works. Explain why can plain backtracking search solve Sudoku puzzles quickly despite the size of the problem?

1. Sudoku can be represented as a CSP with a matrix of variables $M = V_{ij}$ for the entries in the puzzle. i and j are the row and column index, respectively. Since i and j are integers in $[1, 9]$, there are 81 variables. The domains of each V_{ij} are the integers in $[1, 9]$. The constraints are:

   (a) $V_{ij} \neq V_{ik}$, if $j \neq k$

   (b) $V_{ij} \neq V_{kj}$, if $i \neq k$

   (c) When M is partitioned into 9 3x3 matrices $m = V_{ab}$, $V_{ab} \neq V_{cd}$, if $a \neq c$ or $b \neq d$.

2. Please see the `simpleBacktrackSearch` function in csp.py for the implementation of the backtracking algorithm. **TODO: Write explanation.**