15-381 Spring 06 Assignment 2: Constraint Satisfaction Problems

Questions to Vaibhav Mehta(vaibhav@cs.cmu.edu)

Out: 2/07/06 Due: 2/21/06

Name: ______________________________ Andrew ID: ______________________________

Please turn in your answers on this assignment (extra copies can be obtained from the class web page). This written portion must be turned in at the beginning of class at 1:30pm on February 21. The code portion must be submitted electronically by 1:30pm on February 21. Please write your name and Andrew ID in the space provided on the first page, and write your Andrew ID in the space provided on each subsequent page. This is worth 5 points: if you do not write your name/Andrew ID in every space provided, you will lose 5 points.

Code submission. To submit your code, please copy all of the necessary files to the following directory:

`/afs/andrew.cmu.edu/course/15/381/hw2_submit_directory/yourandrewid`

replacing `yourandrewid` with your Andrew ID. You can use any of the following programming languages: C/C++, Java, Perl, Matlab, Lisp, ML/Ocaml, or Python. If you would like to use a language that is not on this list, please check with us first. All code will be tested on a Linux system, we will not accept Windows binaries. No matter what language you use, you must ensure that the code compiles and runs in the afs submission directory. Clearly document your program.

Late policy. Both your written work and code are due at 1:30pm on 2/21. Submitting your work late will affect its score as follows:

- If you submit it after 1:30pm on 2/21 but before 1:30pm on 2/22, it will receive 90% of its score.
- If you submit it after 1:30pm on 2/22 but before 1:30pm on 2/23, it will receive 50% of its score.
- If you submit it after 1:30pm on 2/23, it will receive no score.

Collaboration policy. You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas in the class in order to help each other answer homework questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers

- not to copy answers
- not to allow your answers to be copied

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we ask that you specifically record on the assignment the names of the people you were in discussion with (or "none" if you did not talk with anyone else). This is worth five points: for each problem, space has been provided for you to either write people's names or "none". If you leave any of these spaces blank, you will lose five points. This will help resolve the situation where a mistake in general discussion led to a replicated weird error among multiple solutions. This policy has been established in order to be fair to the rest of the students in the class. We will have a grading policy of watching for cheating and we will follow up if it is detected. For the programming part, you are supposed to write your own code for submission.

1 The N-queens Problem (10 points)

References (names of people I talked with regarding this problem or "none"):

____

Consider the N-queens problem for N=4. As stated in class, this problem can be formulated as a CSP. Assume that the rows and columns are numbered from 1 to 4.

1.1 (5 points)

Starting from a blank board, can 4-queens be solved with constraint propagation only without DFS? Why or why not.

No. because initial choice (search) needs to be made because the constraints don't lead to assignment of any variable.

1.2 (5 points)

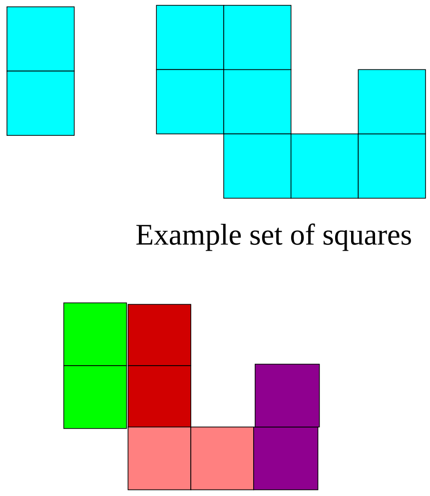
Suppose that we assign the first queen (in column 1) to row 3. Can 4-queens be solved with constraint propagation only without DFS now? Why or why not.

Yes. There is only one assignment of the remaining queens possible. The constraints lead to a fixed assignment of all the remaining variables.

2 Placing Tiles on Board(20 points)

References (names of people I talked with regarding this problem or "none"):

____



Example set of squares

Example Solution

Figure 1: Placing tiles on a board

Suppose that we have N tiles that we want to place on a board so that no two tiles overlap. Suppose that the tiles are 2 units on the side (see Figure 1). Suppose that the board is made of squares connected to each other and not overlapping. The side of the squares is 1. The squares can be in any configuration that is a subset of a regular grid (see Figure 1).

2.1 (5 points)

Formulate this problem as a CSP where the variables are the tiles (define domains and constraints). Specifically, assume that each square in the board has been assigned a co-ordinate (i, j).

Different answers are possible. Any logically correct and slightly formal notation is fine.
Variables : Let there be n tiles. Then, we form 2n variables. $T_{11}, T_{12}, T_{21}, T_{22}, ..., T_{n1}, T_{n2}$. *Each tile corresponds to two variables.*
Domain : 1 to m (m squares of the board)
Constraints : For each k, T_{k1} *and* T_{k2} *are adjacent squares on the board.*
Each variable is unique. i.e. For each k,l,i,j $T_{ki} \neq T_{lj}$.

2.2 (5 points)

Formulate this problem as a CSP where the variables are the squares (define domains and constraints).

Different answers are possible. Any logically correct and slightly formal notation is fine.
Variables : Let there be m squares on the board. Variables are $s_1, ..., s_m$
Domain : 1 to n (it can be 1 .. m, $m \leq n$*, since we do not require the index of tiles to be unique.)*
Constraints : if $s_i == a$*, exactly one of the variable corresponding to the adjacent squares is also a and no other variable is a.*

2.3 (10 points)

Construct a configuration of 8 squares for which the CSP problem is tree-structured. What is the complexity of the problem in that case.

Any linear placement of the squares would be a tree-structured CSP when using the squares as variables. Note that in this problem, we do not have the constraints that the index of one tile must be different with the indices of other tiles. Otherwise it will not be a tree-structured CSP.
Complexity : O(n)

3 The Dinner Problem

References (names of people I talked with regarding this problem or "none"):

____

Five friends (designated U,V,W,X, and Y) are planning to go to dinner but they don't all get along. Specifically, two of (V,W,and Y) will go to dinner if U goes to dinner. W never goes to dinner if either X or Y are there. V always goes to dinner with either X or Y.

Assume that U goes to dinner.

3.1 (5 points)

Formulate as a CSP problem.

Variables : U, V, W, X, Y
Domain : F,T
Constraints : $U \rightarrow (V \wedge W) \vee (V \wedge Y) \vee (W \wedge Y)$
$X \vee Y \rightarrow \neg W$
$V \rightarrow (X \vee Y)$

3.2 (5 points)

After running constraint propagation once without DFS, do any of the domains change?

Yes, V = 1, W = 0, Y = 1 are assigned. Note: in each step, constraint propagation needs to eliminate all the impossible values in the neighbor nodes until it cannot find any more. Since the definition does not state specifically what kind of methods should be used to find the impossible values, you should remove all of the impossible configurations as long as they can be found by one method. For example, in this problem, these impossible configurations can be found by 4-consistency checking rather than 2-consistency checking. Therefore the domains of the variables will be changed after the constraint propagation.

3.3 (5 points)

Assume we use CP with DFS, does the search require backtracking?

No, the search doesn't require backtracking. It is because after a CP step, only X would remain to be assigned. Both assignments of X to 0 or 1 are valid solutions. Hence, no backtracking required.

3.4 (5 points)

What solution is found?

U = 1, V = 1, W = 0, Y = 1, X = 0 or 1

4 The Soduko Puzzle

References (names of people I talked with regarding this problem or "none"):

____

Sudoku is a logic-based placement puzzle, also known as **Number Place** in the United States. The aim of the puzzle is to enter a numerical digit from 1 to N in each cell of $N \times N$ grid made up of sub-grids (called "regions"), starting with various digits in some cells (the "givens"). Each row, column, and region must contain only one instance of each numeral. Sudoku initially caught on in Japan in 1986 and gained international popularity in 2005. A number of variants of the Sudoku puzzle have been published. These variants differ in the size and configuration of the board, and the constraints enforced on the placement of numbers. To read more about the game, you can go to **http://en.wikipedia.org/wiki/Sudoku**. To try out the game, you can go to **http://www.sudoku.com**.

Solving Sudoku puzzles is a good pastime and is recommended by some teachers as an exercise in logical reasoning. However, completing a puzzle requires a lot of patience, logical ability and time!. You, as experts of finite domain constraint programming, will write a program to solve a general Sudoku puzzle. You will write a series of increasingly efficient approaches to solve a Sudoku puzzle.

Here is a formal description of the general Sudoku puzzle. You are given a square grid containing $N \times N$ cells. A region in the grid is of dimensions $A \times B$. Some cells already contain a number. You are supposed to fill the empty cells with numbers between 1 to N, such that each row, column and region contains the numbers 1 to N exactly once. The figures below show an example Sudoku puzzle alongwith its solution.

	10		2	3	5				7	11	
					2			10	3	4	5
9			5				7				
6	8	7		10						1	
	2				4	12					10
		5						3		9	
	3		8						2		
12					7	9				8	
	7						10		1	5	12
				1				5			11
10	1	11	3			4					
	5	8				6	3	12		7	

Sample Sudoku puzzle on 12×12 board with region size of 3×4

8	10	4	2	3	5	1	12	6	7	11	9
11	12	1	7	9	2	8	6	10	3	4	5
9	6	3	5	4	11	10	7	1	8	12	2
6	8	7	12	10	3	5	9	2	11	1	4
3	2	9	11	8	4	12	1	7	5	6	10
1	4	5	10	2	6	7	11	3	12	9	8
5	3	6	8	12	1	11	4	9	2	10	7
12	11	10	1	5	7	9	2	4	6	8	3
4	7	2	9	6	8	3	10	11	1	5	12
7	9	12	6	1	10	2	8	5	4	3	11
10	1	11	3	7	12	4	5	8	9	2	6
2	5	8	4	11	9	6	3	12	10	7	1

Solution to the sample puzzle given above

4.1 Input Format

Files describing a Sudoku puzzle will visually represent the board using a string of characters. The first line contains three numbers: length of the board (N), length of a region (A) and breadth of a region (B). The next N lines describe the N rows of the board. An empty cell is denoted by _. Here is a sample input board corresponding to the sample puzzle:

```
12 3 4
_ 10 _ 2 3 5 _ _ _ 7 11 9
_ _ _ _ _ 2 _ _ 10 3 4 5
9 _ _ 5 _ _ _ 7 _ _ _ _
6 8 7 _ 10 _ _ _ _ _ 1 _
_ 2 _ _ _ 4 12 _ _ _ _ 10
_ _ 5 _ _ _ _ _ 3 _ 9 _
_ 3 _ 8 _ _ _ _ _ 2 _ _
```

```
12 _ _ _ _ 7 9 _ _ _ 8 _
_ 7 _ _ _ _ _ 10 _ 1 5 12
_ _ _ _ 1 _ _ _ 5 _ _ 11
10 1 11 3 _ _ 4 _ _ _ _ _
_ 5 8 _ _ _ 6 3 12 _ 7 _
```

4.2 Output Format

The output should consist of the completed Sudoku puzzle. You should output N lines each describing a row of the completed board. The individual entries should be separated by a single space. The last line gives a number indicating the number of partial assignments explored to reach the solution. Here is a sample output of the input shown above.

```
8 10 4 2 3 5 1 12 6 7 11 9
11 12 1 7 9 2 8 6 10 3 4 5
9 6 3 5 4 11 10 7 1 8 12 2
6 8 7 12 10 3 5 9 2 11 1 4
3 2 9 11 8 4 12 1 7 5 6 10
1 4 5 10 2 6 7 11 3 12 9 8
5 3 6 8 12 1 11 4 9 2 10 7
12 11 10 1 5 7 9 2 4 6 8 3
4 7 2 9 6 8 3 10 11 1 5 12
7 9 12 6 1 10 2 8 5 4 3 11
10 1 11 3 7 12 4 5 8 9 2 6
2 5 8 4 11 9 6 3 12 11 7 1
1244312
```

4.3 (10 points)

Backtracking with consistency checking: Implement a backtracking DFS search algorithm to solve the Sudoku puzzle. Report the number of partial assignments explored and time taken before a solution is reached for the example puzzles A, B, C. Note that this naive approach might take too long for some of the puzzles.

- Puzzle A : 392888 (it is ok as long as the orders approximately match)
- Puzzle B : >40000000 (it is ok if you claim that it takes too long to run puzzle B/C)
- Puzzle C : 42368316

4.4 (10 points)

Dynamic Variable Ordering: Modify the search step so that instead of picking the first uninstantiated variable, it examines the domains of all uninstantiated variables, and picks the first one with the minimum legal domain size. Report the number of partial assignments explored and time taken before a solution is reached for the example puzzles.

- Puzzle A : 470

- Puzzle B : 891
- Puzzle C : 16236

4.5 (10 points)

Some more Constraints: At each step, apply the constraint that if a particular value can be assigned to only one variable in a row, column or a region, then the value should be assigned to that variable. Report the number of partial assignments explored and time taken before a solution is reached for the example puzzles.

- Puzzle A : 147
- Puzzle B : 360
- Puzzle C : 1448

4.6 (15 extra credit points)

Try to improve the performance using constraint propagation, value ordering or other heuristics that you think would be suitable for the problem. Report the number of partial assignments explored and time taken before a solution is reached for the example puzzles.

- Puzzle A :
- Puzzle B :
- Puzzle C :

4.7 (20 points)

You are supposed to submit code for each of the parts above. In addition to your code, please submit a one-page description of the various algorithms you have used and their implementation. Also comment on the benefits obtained by using various enhancements to the basic backtracking search. This must be attached to the rest of the part of your assignment.

5 Code Submission Details

Program Output Your code should output the solved puzzle in the output format given earlier alongwith the number of partial assignments explored. We would be testing your code on the Andrew machines, so please ensure that your code compiles and runs successfully there.
Documentation A `README` should be submitted with your code which specifies how to compile and run your code. Specifically the file must describe the command line arguments needed to run each of the algorithms implemented alongwith the example puzzles.
Grading Grading will be based on the correctness of the solution output by the program. The order of the outputs generated by your code should be correct to get credit for a solution. All

solutions will be graded using the same set of puzzles. We would be using some extra puzzles for testing other than the ones provided with the homework. You are encouraged to try your code on puzzles other than the ones provided by us. The code submitted by you should be written by you, and not copied from any other source.