**CS 171 Sudoku Project Shell: Cpp Edition**

TL;DR:

* Main File to modify is cspSolver/BTSolver.cpp

1. OVERVIEW OF THE CODE

Files for the main program (located in the cspSolver folder):

|  |  |
| --- | --- |
| Variable.cpp/hpp | represents a variable in the CSP |
| Domain.cpp/hpp | represents the domain of a variable |
| Constraint.cpp/hpp | represents a constraint in the CSP |
| ConstraintNetwork.cpp/hpp | represents the CSP model |
| BTSolver.cpp/hpp | a backtracking solver |
| Trail.cpp/hpp | a trail of assignments that have occurred |
| LocalSolver.cpp/hpp | a local search solver |
| Assignment.cpp/hpp | an assignment of a value to a variable in local search |

Modify **BTSolver.cpp** and then run Makefile. **BTSolver.cpp** implements that constraint satisfaction problem solvers you have studied in class and is an implementation of the pseudocode for Recursive Bactracking on **page 142** in **chapter 5** of the Russell-Norvig book.

1. RUNNING THE EXAMPLE CODE

To Run the Example code:

* Run the Makefile and it will automatically compile and link all the needed files.
  + Note when running the Makefile, you will generate numerous .o files. Ignore those files.
* After running the Makefile, it will create three executable files in the parent dirctory:
  + **SudokuFileExample**: This is a demo on how the program randomly generates a new sudoku problem. Modify examples/**SudokuFileExample.cpp** to change the behavior of this demo.
  + **ConverterExample**: This is a demo on how the program convert a sudoku file into a constraint network. It prints out the variable numbers in the sudoku file and then prints out each group of varibles that are in the same constraint. Modify examples/**ConverterExample.cpp** to change the behavior of this demo.
  + **BTSolverExample**: This is a demo on how the program use a "dummy" backtracking solver to solver a problem. After each round, the program will print out its current process. It will print out a final solution if it succeeded. Modify examples/**BTSolverExample.cpp** to change the behavior of this demo(e.g. the size of the sudoku problem).

Current state of the example program:

* Currently, the shell is really "dumb", so the size of the sudoku problem in the sample is a "4\*4".
* The probability of using this shell to solve a "9\*9" problem is very low, so don’t even think about it.

1. TO DO For BTSolver

In order to improve upon the backtracking search, we can introduce heuristics and consistency checks. Having heuristics allows us to speed up our search by only checking promising assignments first. Meanwhile, consistency checks significantly prune the search space, reducing the number of potential assignments to check.

Variable Selection Heuristics

* Minimum Remaining Value: Select variable with fewest remaining values
* Degree: Select the variable involved with the largest number of constraints on other unassigned variables

Value Selection Heuristics

* Least Constraining Value: Select value which is involved with least conflicts with neighboring domains

Consistency Checks:

* Forward Checking: If you assign a variable X, if another variable Y is involved in a constraint with X, you prune the domain of Y with any inconsistent values Arc Consistency: For every pair of variables participating in constraints with each other, every value in one variable's domain must be satisfiable by at least one value in the other variable's domain.

1. Some Issues with the CPP code

* If there is an issue running Local Solver, let us know, we are still trying to find that bug.
* If the sudoku board turns by 90 degrees, let us know.
* The Makefile may sometimes fail to update the executable files after updating the source code. If you encounter any weird problems after editing the source code, please do "make clean" and then re-"make" it.