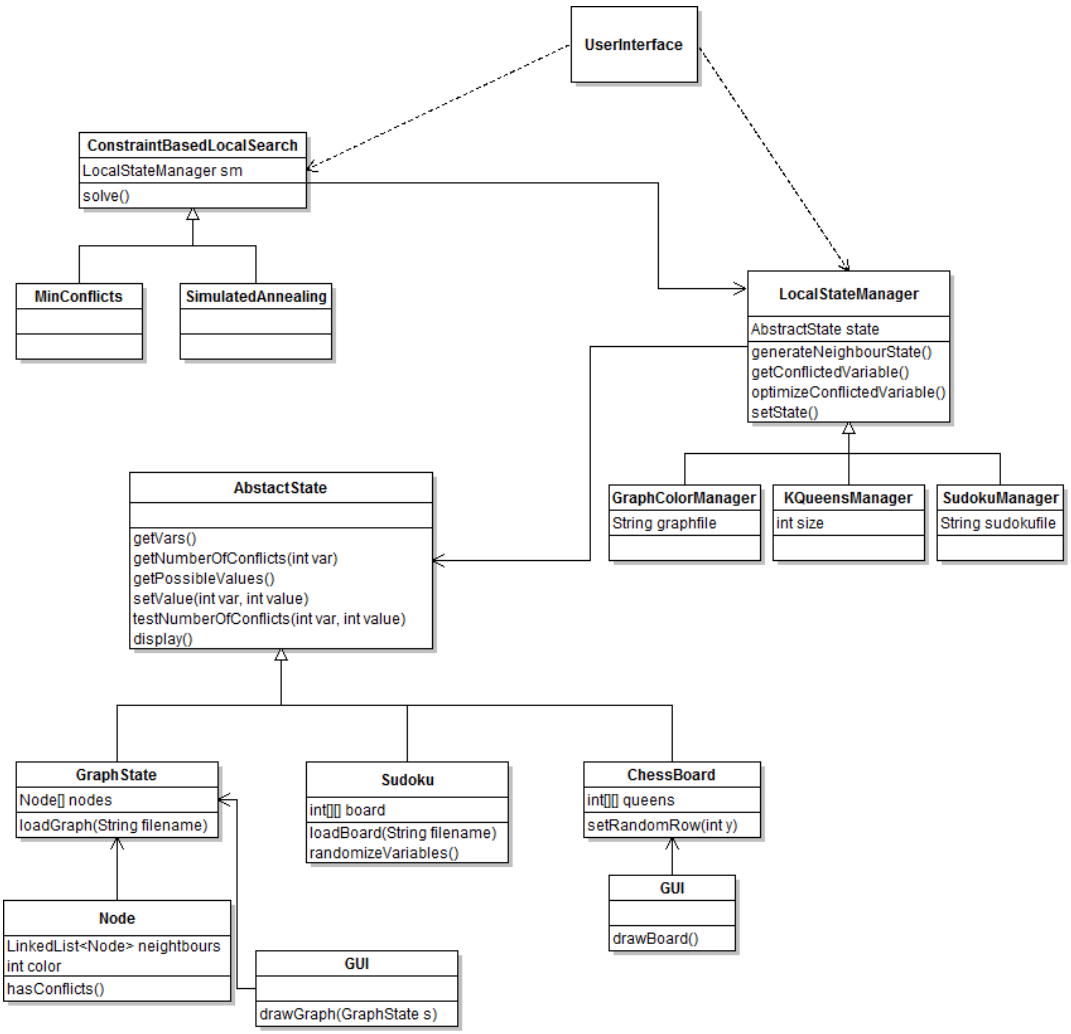
Building a General Constraint-Based Puzzle Solver

An IT3105 Project

Sondre Engebråten and Kristoffer Hagen

Table of contents:

1. Structure of code



When implementing our general puzzle solver focus was to make it as modular as possible. This means that an object oriented language (such as Java) is a prime choice due to easy modularity by subclassing / extensions of classes. Our implementation borrows much from the diagram as presented by this exercise (ref. Puzzle-solver Figure 1). However in order to archive a better distinction and breakdown of the code we opted to use an additional state class hierarchy. This allows us to have much of the logic required to implement the different problems hidden away and greatly simplifies the state managers. The specialized state managers are simplified to the point where they only have to create the initial state.

LocalStateManager and AbstractState defines a set of functions and attributes that are required by the problem solver algorithms. Quite a few functions are implemented fully in the abstract base classes (LocalStateManager and AbstractState). This was made possible by declaring all the required functions abstract in the base classes. There were cases where we saw fit to reimplement some of the functions, this was done in cases where a much faster implementation was possible given knowledge specific to the problem. For some functions this meant that they went from being O(n^2) to O(n). Clearly this is a worthwhile improvement.

The split between which parts of the code should be a part of the state manager and the constraint-based local search was a challenge. Some of the code in its present state could have been moved from LocalStateManager to MinConflict or SimulatedAnnealing. Before hand this was hard to determine. Since reuse of functionality was a criteria we chose to have more of the functionality in the LocalStateManager.

2. The 3rd puzzle

2.1 Equation problem

This problem solves a linear system of equations using constraint-based search. In short we are looking at mathematical problems such as:

$$a + b = 5$$
$$a - b = 3$$

The solution to this problem is trivial, $a = 4$ and $b = 1$. However these problems can be made significantly harder by introducing N equations instead of just two. The equation problem will attempt to solve such a problem.

In order to guarantee that at least one solution is possible the problem will be generated by randomizing a matrix A and a vector x. Another vector b is then computed from the two by the expression:

$$Ax = b$$

Vector x is then deleted and the problem solver handed the newly generated problem to solve (find x). For each variable in the solution vector x a constraint is considered broken; if an equation that it is a part of (where the coefficient of the variable is not equal to zero) contributes to an incorrect value in vector b.

When randomizing vector x a constraint is set on the possible values, a value can only be in the range of -50 and 50. This does not affect the difficulty of the problem since N (the number of equations) can easily be increased to more than make up for this slight restriction.

2.2 Sudoku

As we had varied results with our first attempt for the custom puzzle we decided to implement a second puzzle. Sudoku also proved by be a challenge for the solver algorithms, but is a well-known and easy-to-understand puzzle.

The initial problem is loaded from a file, very similar to that of the graph coloring problem. This file contain the size of the puzzle and the initial state of the problem, for example:

```
4
2
0 3 4 1
4 1 2 0
3 2 1 0
0 4 3 2
```

Our sudoku1.txt as an example of input

The numbers are all stored in a 2d integer array, and all the slots that initially contained “0” are saved in a list so that we can get access to just the variables that are supposed to be modified later.

In the example above (sudoku1.txt). The variable list will be: 0,7,11,12, and refer to the locations: (0,0), (1,3), (2,3), (3,0) (0-indexed, (y,x)). The method getPossibleValues would return a list with 1,2,3,4 which are the possible values for every variable.

In order to check a variable (a number-slot) for conflicts we have three methods. These methods return a list containing all the other numbers that could be in conflict with the selected variable: the horizontal line, the vertical line and the square. Note that the same slot in the sudoku table can be in more than one of these (both box and vertical for example). If the value of the current variable is found to be in either of these three lists, this is counted as a violation of the constraint. Special care had to be taken to not count conflicts with oneself, because any given variable will clearly be a part of its own row, column or square.

A query for the total number of constraint violations will simply run the “check for conflicts method” on every variable and sum the result. Evaluation of the board (as used in simulated annealing) is the number of conflicts for the entire board negated. A perfect evaluation is therefore 0 in order to accommodate simulated annealing that maximizes the value of a state.

3. Results

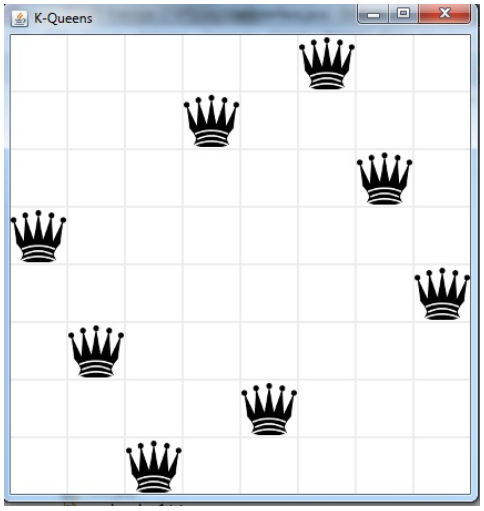
3.1 Simulated annealing parameters

We experimented with different parameters for simulated annealing, this ended up being a very time consuming part of the assignment. In the end we settled on one set of parameters that are used for all problems and appear to give reasonable performance in most cases. The parameters used for simulated annealing are:

Neighbours: 50
Max temperature: 20
Delta temperature = 0.02

3.2 K-Queens problem

The easy K-Queens problem was with K=8:



This figure shows the solved state of a K-Queens problem with K = 8. As can be seen in the image no two queens are threatening each other, in other words no constraints are broken.

Solver	Problem	Avg. steps	Std. dev steps	Least steps	Avg. conf. sol.	Std. dev conf. sol.	Min conf. solution
SA	**K=8**	748	298	35	0	0	0
MC	**K=8**	556	2167	5	0	0	0
SA	**K=25**	1082	58	1008	0	0	0
MC	**K=25**	101	56	26	0	0	0
SA	**K=1000**	10000	0	10000	36	6	26
MC	**K=1000**	699	34	659	0	0	0

3.2.1 Simulated annealing

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.KQueensManager K = 8
Number of runs: 20

AvgStepCount 748.0
StddevStepCount 298.0
MinStepCount 35.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.KQueensManager K = 25
Number of runs: 20

AvgStepCount 1082.0
StddevStepCount 58.0
MinStepCount 1008.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.KQueensManager K = 1000
Number of runs: 20

AvgStepCount 10000.0
StddevStepCount 0.0
MinStepCount 10000.0

AvgConflictsInSolution 36.0
StddevConflictsInSolution 6.0
MinConflictsInSolution 26.0

3.2.2 Min-Conflicts

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.KQueensManager K = 8
Number of runs: 20

AvgStepCount 556.0
StddevStepCount 2167.0
MinStepCount 5.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.KQueensManager K = 25
Number of runs: 20

AvgStepCount 101.0
StddevStepCount 56.0
MinStepCount 26.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

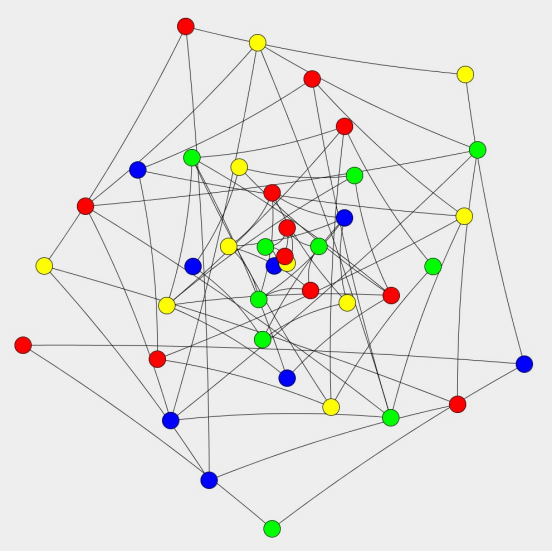
Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.KQueensManager K = 1000
Number of runs: 20

AvgStepCount 699.0
StddevStepCount 34.0
MinStepCount 659.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

3.3 Graph-Coloring problem

The easiest of the graph coloring problems is “graph-color-2.txt”, this problem is to color the nodes of this graph:



This figure illustrates the graph when solves, this is just one of many possible solutions to this problem. Important to note that no two connected nodes have the same color, our representation would outline constraint violations as red lines (of which there are none in the given picture).

Solver	Problem	Avg. steps	Std. dev steps	Least steps	Avg. conf. sol.	Std. dev conf. sol.	Min conf. solution
SA	**GC1**	751	228	317	0	0	0
MC	**GC1**	104	82	29	0	0	0
SA	**GC2**	74	23	40	0	0	0
MC	**GC2**	33	22	15	0	0	0
SA	**GC3**	1381	108	1216	0	0	0
MC	**GC3**	804	133	485	0	0	0

3.3.1 Simulated annealing

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.GraphColorManager graph-color-1.txt
Number of runs: 20

AvgStepCount 751.0
StddevStepCount 228.0
MinStepCount 317.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.GraphColorManager graph-color-2.txt
Number of runs: 20

AvgStepCount 74.0
StddevStepCount 23.0
MinStepCount 40.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.GraphColorManager graph-color-3.txt
Number of runs: 20

AvgStepCount 1381.0
StddevStepCount 108.0
MinStepCount 1216.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

3.3.2 Min-Conflicts

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.GraphColorManager graph-color-1.txt
Number of runs: 20

AvgStepCount 104.0
StddevStepCount 82.0
MinStepCount 29.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.GraphColorManager graph-color-2.txt
Number of runs: 20

AvgStepCount 33.0
StddevStepCount 22.0
MinStepCount 15.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.GraphColorManager graph-color-3.txt
Number of runs: 20

AvgStepCount 804.0
StddevStepCount 133.0
MinStepCount 485.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

3.4 Equation problem

The simplest of the equation problems is a set of equations with 10 equations and 10 unknowns. This is equivalent to solving a system of equations such as:

$$Ax = b$$

Where A is a 10x10 matrix, x is a vector of length 10 and b is a vector of length 10.

Solver	Problem	Avg. steps	Std. dev steps	Least steps	Avg. conf. sol.	Std. dev conf. sol.	Min conf. solution
SA	**K=10**	8304	2749	1420	4	3	0
MC	**K=10**	27	10	12	0	0	0
SA	**K=25**	10000	0	10000	14	2	12
MC	**K=25**	88	22	58	0	0	0
SA	**K=50**	10000	0	10000	30	4	22
MC	**K=50**	226	67	118	0	0	0

3.4.1 Simulated annealing

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.EquationManager K=10
Number of runs: 20

AvgStepCount 8304.0
StddevStepCount 2749.0
MinStepCount 1420.0

AvgConflictsInSolution 4.0
StddevConflictsInSolution 3.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.EquationManager K=25
Number of runs: 20

AvgStepCount 10000.0
StddevStepCount 0.0
MinStepCount 10000.0

AvgConflictsInSolution 14.0
StddevConflictsInSolution 2.0
MinConflictsInSolution 12.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.EquationManager K=50
Number of runs: 20

AvgStepCount 10000.0
StddevStepCount 0.0
MinStepCount 10000.0

AvgConflictsInSolution 30.0
StddevConflictsInSolution 4.0
MinConflictsInSolution 22.0

3.4.2 Min-Conflicts

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.EquationManager K=10
Number of runs: 20

AvgStepCount 27.0
StddevStepCount 10.0
MinStepCount 12.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.EquationManager K=25
Number of runs: 20

AvgStepCount 88.0
StddevStepCount 22.0
MinStepCount 58.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

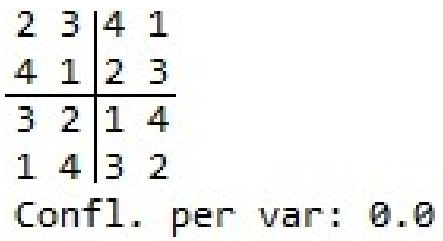
Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.EquationManager K=50
Number of runs: 20

AvgStepCount 226.0
StddevStepCount 67.0
MinStepCount 118.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

3.5 Sudoku problem

The easiest sudoku we could think of was a 4x4 sudoku using the numbers from 1 to 4. As shown below:



This is an example of how a solved board could look like, in our case it's unlikely that there are more than one solution (due to few values to be placed) however in general a board can be solved by a number of valid states. A sudoku board is constraint free if the same number (1-4) is only present once in the square, row and column of that number. A square is a division of the board into a number of smaller segments, in this case each square is 2x2 and there are four of them as shown by the figure.

Solver	Problem	Avg. steps	Std. dev steps	Least steps	Avg. conf. sol.	Std. dev conf. sol.	Min conf. solution
SA	**SU1**	15	9	1	0	0	0
MC	**SU1**	4	2	0	0	0	0
SA	**SU2**	412	173	55	0	0	0
MC	**SU2**	4510	4966	12	1	1	0
SA	**SU3**	1428	199	1184	0	0	0
MC	**SU3**	10000	0	10000	17	5	12

3.5.1 Simulated annealing

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.SudokuManager sudoku1.txt
Number of runs: 20

AvgStepCount 15.0
StddevStepCount 9.0
MinStepCount 1.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.SudokuManager sudoku2.txt
Number of runs: 20

AvgStepCount 412.0
StddevStepCount 173.0
MinStepCount 55.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.SimulatedAnnealing
Problem: StateManagers.SudokuManager sudoku3.txt
Number of runs: 20

AvgStepCount 1428.0
StddevStepCount 199.0
MinStepCount 1184.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

3.5.2 Min-Conflicts

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.SudokuManager sudoku1.txt
Number of runs: 20

AvgStepCount 4.0
StddevStepCount 2.0
MinStepCount 0.0

AvgConflictsInSolution 0.0
StddevConflictsInSolution 0.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.SudokuManager sudoku2.txt
Number of runs: 20

AvgStepCount 4510.0
StddevStepCount 4966.0
MinStepCount 12.0

AvgConflictsInSolution 1.0
StddevConflictsInSolution 1.0
MinConflictsInSolution 0.0

Algorithm: LocalSearch.MinConflicts
Problem: StateManagers.SudokuManager sudoku3.txt
Number of runs: 20

AvgStepCount 10000.0
StddevStepCount 0.0
MinStepCount 10000.0

AvgConflictsInSolution 17.0
StddevConflictsInSolution 5.0
MinConflictsInSolution 12.0