# IT3105: Project 2 – Building a General Constraint-Based Puzzle Solver

**Due on Wednesday, October 24, 2013**

*Lecturer: Keith Downing*

**Pablo Liste Garcia & Dominik Horb**

# Contents

# 1 Architecture Description

The general structure we chose to use for our implementation of the general puzzle solver doesn't deviate much from the proposed architecture in the task description. Figure 1 shows the two most important packages we created and their contained types. As you can see the main additions we made are *PuzzleState*, *Conflict* and *GeneralPuzzleSolver*. Our main class that creates the state manager and algorithm objects, initiates the execution and calculates the statistics is *GeneralPuzzleSolver*. It could probably be refactored to separate the different tasks it has a bit more cleanly and distribute the problem specific initialization steps, but this is trivially achievable and wasn't a priority for now. To start an algorithm it creates an object of the desired *LocalStateManager* for the puzzle that should be solved, provides it to an object of the chosen algorithm and starts the algorithm as can be seen in Listing 1.

```
this.puzzle = this.choosePuzzle();
this.searcher = this.chooseSearchAlgorithm();
this.searcher.setStateManager(this.puzzle);

this.numberOfRuns = this.chooseNumberOfRuns();

PuzzleState currentSolution;

for (int i = 0; i < this.numberOfRuns; i++) {
  currentSolution = this.searcher.run();
  currentSolution.display();
}
```

Listing 1: Combining algorithm and puzzle and starting the search.

The more important bits can be found in the *generalpuzzlesolver.puzzle* package. All three types that can be seen in the lower part of Figure 1 must be implemented by every puzzle that should be added to the application. The algorithm classes will only interact with methods that can be found within theses three types in order to generalize their execution.
*Conflict* is simply an interface without any methods, but necessary to pass around specific conflicts. For the graph colouring problem an implementation would for example store the indices of two nodes that have the same colour and an edge in the graph.
As the name suggests, an implementation of *PuzzleState* would simply store the data of the current state of the puzzle. It also needs to be able to check if it violates any constraints of the puzzle, provide a list of all the conflicts that exist and display itself.

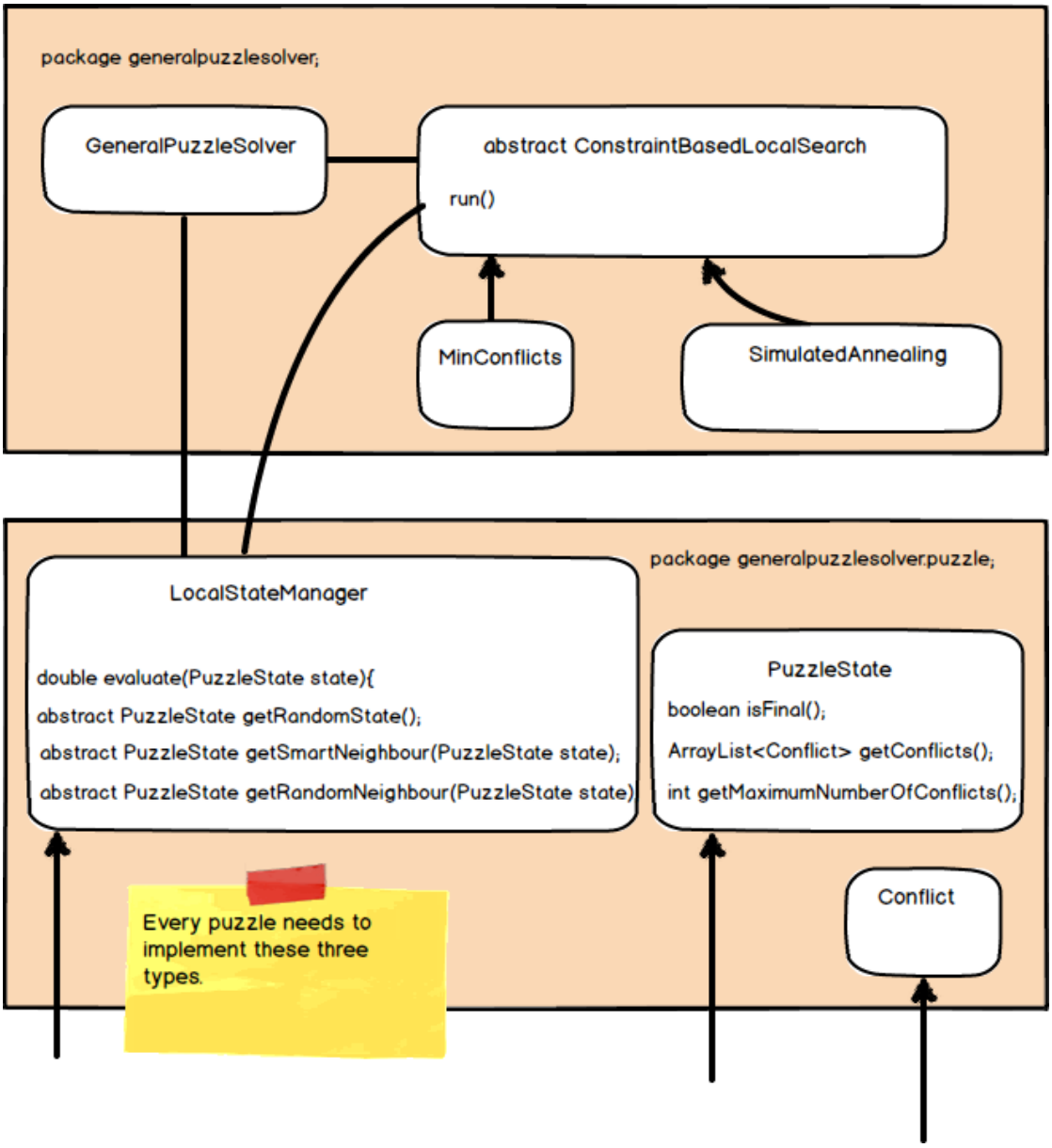


Figure 1: Architecture of our general puzzle solver.

To provide an example: In case of the graph colouring problem, it contains an adjacency matrix and a list with the data for all the vertices (colour and position).
The *PuzzleStateManager* will be a stateless object in most cases, that just takes *PuzzleState* objects into it's methods and changes them to some kind of ruleset provided inside the method.

## 2 First Puzzle: k-Queens

Table 1: Statistics for the k-Queens puzzle.

| Combination | Evaluation - Average | - SD | - Best | Steps - Average | - SD | - Lowest |
|---|---|---|---|---|---|---|
| Easy / SA | 0.0 | 0.0 | 0.0 | 76.15 | 52.12 | 8 |
| Medium / SA | 0.0 | 0.0 | 0.0 | 155.70 | 91.20 | 50 |
| Hard / SA | 0.0 | 0.0 | 0.0 | 217.05 | 69.84 | 127 |
| Easy / MC | 0.0 | 0.0 | 0.0 | 218.00 | 211.27 | 7 |
| Medium / MC | 0.0 | 0.0 | 0.0 | 392.55 | 346.81 | 22 |
| Hard / MC | 0.0 | 0.0 | 0.0 | 522.45 | 317.74 | 176 |

Both algorithms were able to solve the easy case variant of the k-queens puzzle without problems. The results as shown by our application can be seen in Figure 2 for simulated annealing and Figure 3 for min-conflicts.

```
run:
---------- Welcome to the General Puzzle Solver ----------
Which puzzle would you like to use?
1. k-Queens
2. Graph Coloring
3. Sudoku
1
How many queens should be placed?
8
Which algorithm would you like to use?
1. Simulated Annealing
2. Min-Conflicts
1
Please specify the number of runs:
1
      0  1  2  3  4  5  6  7
   0  _  _  _  _  _  _  _  Q
   1  _  Q  _  _  _  _  _  _
   2  _  _  _  _  Q  _  _  _
   3  _  _  Q  _  _  _  _  _
   4  Q  _  _  _  _  _  _  _
   5  _  _  _  _  _  _  Q  _
   6  _  _  _  Q  _  _  _  _
   7  _  _  _  _  _  Q  _  _

0.0
Used Steps: 84
Number of conflicts: 0

The average of steps is: 84.0
The lowest step count is: 84.0
The standard deviation of steps is: 0.0
The average of evaluations is: 0.0
The best evaluation is: 0.0
The standard deviation of evaluations is: 0.0
BUILD SUCCESSFUL (total time: 3 minutes 16 seconds)
```

Figure 2: Solution of 8-Queens with simulated annealing.

```
run:
---------- Welcome to the General Puzzle Solver ----------
Which puzzle would you like to use?
1. k-Queens
2. Graph Coloring
3. Sudoku
1
How many queens should be placed?
8
Which algorithm would you like to use?
1. Simulated Annealing
2. Min-Conflicts
2
Please specify the number of runs:
1
      0  1  2  3  4  5  6  7
   0  _  _  _  Q  _  _  _  _
   1  Q  _  _  _  _  _  _  _
   2  _  _  _  _  Q  _  _  _
   3  _  _  _  _  _  _  _  Q
   4  _  Q  _  _  _  _  _  _
   5  _  _  _  _  _  _  Q  _
   6  _  _  Q  _  _  _  _  _
   7  _  _  _  _  _  Q  _  _

0.0
Used Steps: 364
Number of conflicts: 0

The average of steps is: 364.0
The lowest step count is: 364.0
The standard deviation of steps is: 0.0
The average of evaluations is: 0.0
The best evaluation is: 0.0
The standard deviation of evaluations is: 0.0
BUILD SUCCESSFUL (total time: 6 seconds)
```

Figure 3: Solution of 8-Queens with min-conflicts.

# 3 Second Puzzle: Graph Colouring

Table 2: Statistics for the graph colouring puzzle.

| Combination | Evaluation - Average | - SD | - Best | Steps Average | - SD | - Lowest |
|---|---|---|---|---|---|---|
| Easy / SA | 0.0 | 0.0 | 0.0 | 68.1 | 25.10 | 40 |
| Medium / SA | 0.0 | 0.0 | 0.0 | 31.2 | 12.31 | 12 |
| Hard / SA | 0.0 | 0.0 | 0.0 | 607.94 | 48.16 | 520 |
| Easy / MC | 0.0 | 0.0 | 0.0 | 60.7 | 17.93 | 37 |
| Medium / MC | 0.0 | 0.0 | 0.0 | 33.2 | 9.12 | 12 |
| Hard / MC | 0.0 | 0.0 | 0.0 | 631.5 | 79.59 | 528 |

Both algorithms were able to solve the easy case variant of the graph colouring puzzle without problems. The results as shown by our application can be seen in Figure 4 for simulated annealing and Figure 5 for min-conflicts.

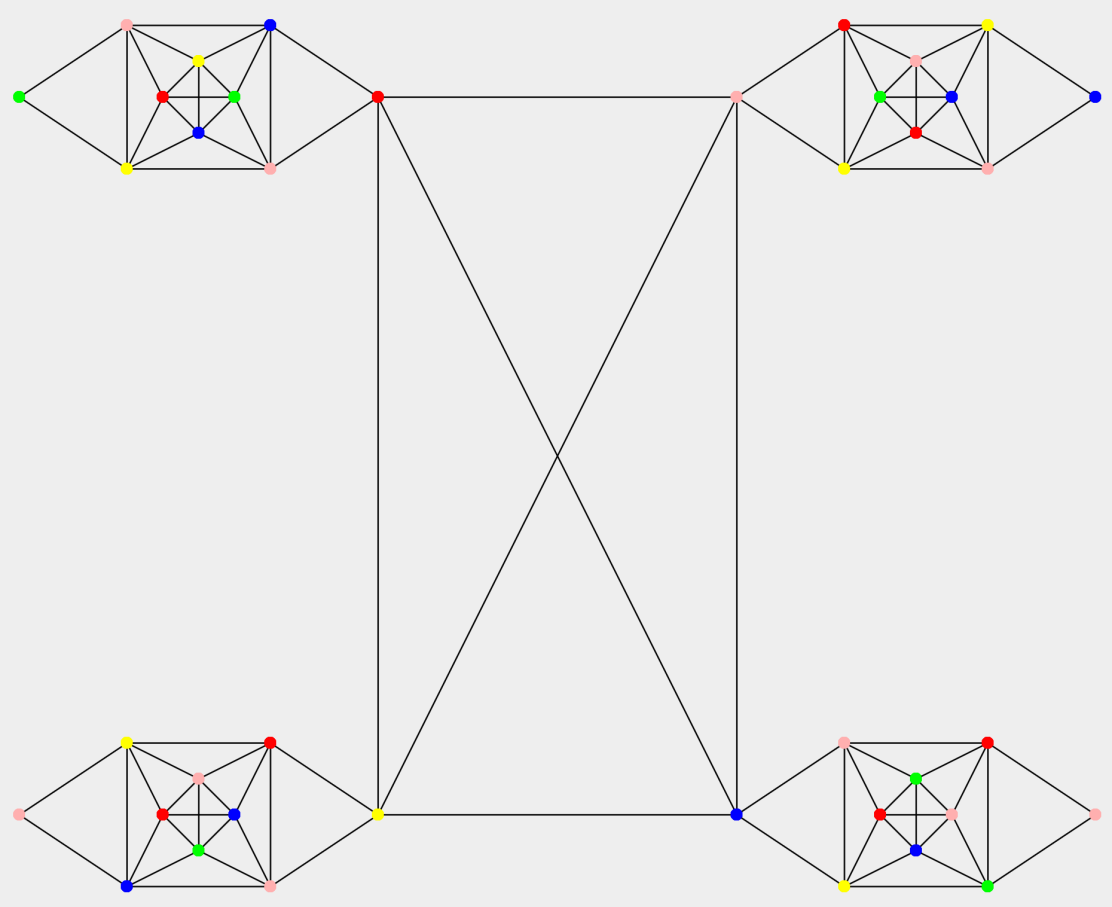

Figure 4: Solution of graph colouring with simulated annealing.

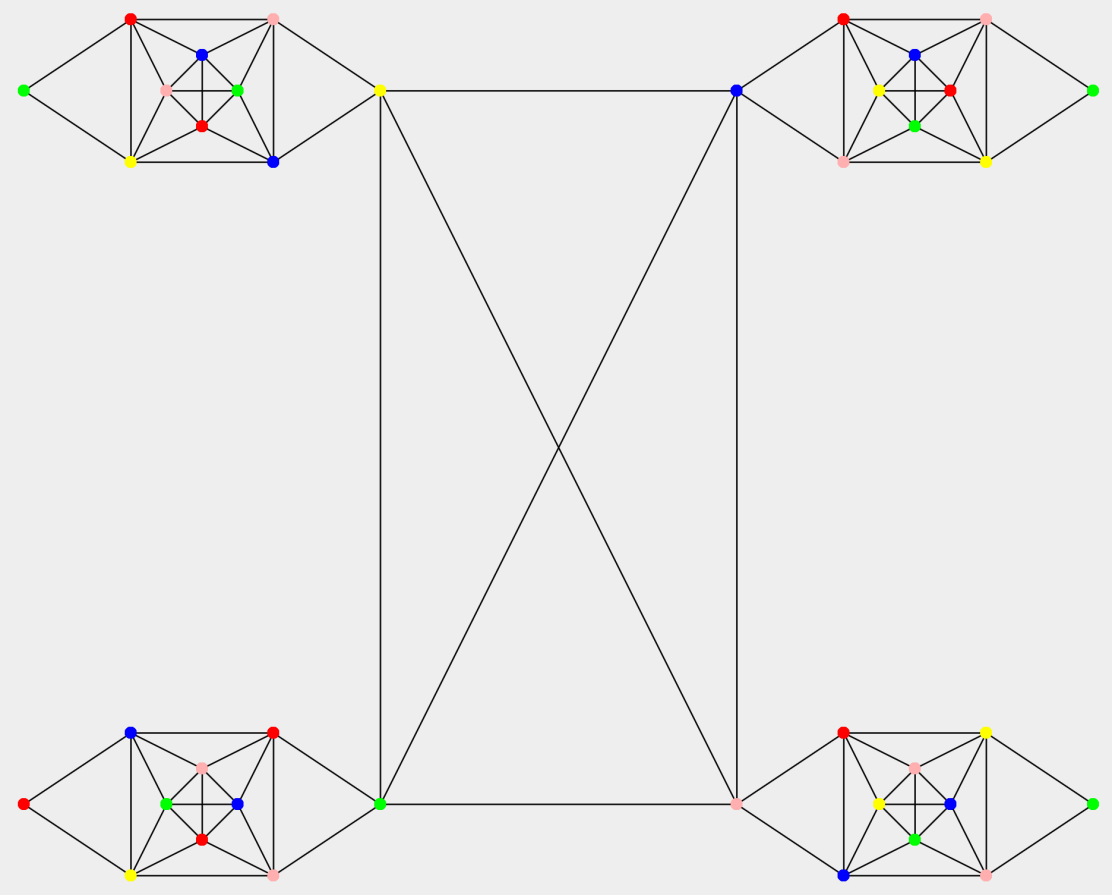

Figure 5: Solution of graph colouring with min-conflicts.

# 4 Third Puzzle: Sudoku

We chose to implement Sudoku as our third puzzle, which in hindsight wasn't a good decision because we weren't able to get the algorithms to solve it properly. With higher values for the maximum steps, it's possible to see, that the solutions seem to go in the right direction, but they somehow always get stuck at a local minimum of the count of the constraint violations.
In the *SudokuPuzzleState* class we simply represent a state as a multidimensional array of integers. To not change initial values of the sudoku grid, we store a *startingState* in the implementation of the state manager. Every position that has a value of 0 set in this starting state can be changed in other states.
Checking constraints of sudoku consists of going through all 81 numbers in the grid and comparing them to all the numbers in the same line, same column and same 3x3 block. Everytime a position with the same number is found, a new *SudokuConflict* object is created by us. These consist of the position indices of the currently checked number and the position it was equal to. This should create a maximum of $81 * 24 = 1944$ conflicts if every number in the grid would be the same. As suggested in:

- `https://amoon.netfirms.com/Portfolio/Applied%20AI%20-%20Sudoku%20Solver.pdf`,

we decided to apply higher values to conflicts with numbers from the starting state that can't be changed.
As with all the puzzles we implemented we create a double value to represent the evaluation by dividing the value of found conflicts by the maximum value for conflicts. This yields a value between 0.0 for a solution to the problem and 1.0 for the worst possible state.

Table 3: Statistics for the sudoku puzzle.

| Combination | Evaluation Average | - SD | - Best | Steps Average | - SD | - Lowest |
|---|---|---|---|---|---|---|
| Easy / SA | 0.01219 | 0.00206 | 0.00720 | 10000.0 | 10000.0 | 0.0 |
| Medium / SA | 0.01162 | 0.00272 | 0.00617 | 10000.0 | 10000.0 | 0.0 |
| Hard / SA | 0.01203 | 0.00181 | 0.00925 | 10000.0 | 10000.0 | 0.0 |
| Easy / MC | 0.06013 | 0.00682 | 0.05144 | 10000.0 | 10000.0 | 0.0 |
| Medium / MC | 0.03055 | 0.00639 | 0.01954 | 10000.0 | 10000.0 | 0.0 |
| Hard / MC | 0.04573 | 0.00691 | 0.03395 | 10000.0 | 10000.0 | 0.0 |

In Table 4 and Table 5 you can see the starting state for the easy puzzle that was used and the solution to it. Figure 6 and Figure 7 show the results that simulated annealing and min-conflicts achieved after 10.000 steps.

Table 4: Starting state of the easy Sudoku puzzle.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 6 | 2 | 4 | 0 | 0 | 0 | 0 |
| 2 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 4 | 0 | 0 | 7 | 5 | 0 | 3 | 0 |
| 0 | 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 4 | 0 | 9 | 0 | 5 | 0 |
| 0 | 0 | 0 | 3 | 0 | 0 | 0 | 1 | 0 |
| 0 | 3 | 0 | 1 | 9 | 0 | 0 | 4 | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 1 |
| 0 | 0 | 0 | 0 | 5 | 8 | 7 | 0 | 0 |

Table 5: Hand-made solution of the easy Sudoku puzzle.

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

```
run:
---------- Welcome to the General Puzzle Solver ----------
Which puzzle would you like to use?
1. k-Queens
2. Graph Coloring
3. Sudoku
3
Please select the difficulty (1-3):
1
Which algorithm would you like to use?
1. Simulated Annealing
2. Min-Conflicts
1
Please specify the number of runs:
1
5  9  6  2  4  1  8  2  7
2  7  1  6  8  3  4  9  5
8  4  2  9  7  5  1  3  6
3  8  4  5  6  1  9  7  2
7  1  3  4  2  9  6  5  8
6  5  9  3  8  7  3  1  4
4  3  5  1  9  6  2  4  8
9  2  7  7  3  4  5  6  1
1  6  8  2  5  8  7  9  3

0.012345679012345678
Used Steps: 10000
Number of conflicts: 24

The average of steps is: 10000.0
The lowest step count is: 10000.0
The standard deviation of steps is: 0.0
The average of evaluations is: 0.012345679012345678
The best evaluation is: 0.012345679012345678
The standard deviation of evaluations is: 0.0
BUILD SUCCESSFUL (total time: 7 seconds)
```

Figure 6: Best Solution of sudoku with simulated annealing.

```
run:
---------- Welcome to the General Puzzle Solver ----------
Which puzzle would you like to use?
1. k-Queens
2. Graph Coloring
3. Sudoku
3
Please select the difficulty (1-3):
1
Which algorithm would you like to use?
1. Simulated Annealing
2. Min-Conflicts
2
Please specify the number of runs:
1
3  9  6  2  4  3  5  2  7
2  7  5  8  1  6  4  8  9
8  4  1  9  7  5  6  3  2
6  8  4  3  3  1  9  7  4
7  1  9  4  4  9  3  5  6
4  5  3  3  6  9  8  1  3
5  3  6  1  9  4  7  4  8
7  8  7  4  3  2  5  6  1
4  6  2  6  5  8  7  9  3

0.03806584362139918
Used Steps: 10000
Number of conflicts: 74

The average of steps is: 10000.0
The lowest step count is: 10000.0
The standard deviation of steps is: 0.0
The average of evaluations is: 0.03806584362139918
The best evaluation is: 0.03806584362139918
The standard deviation of evaluations is: 0.0
BUILD SUCCESSFUL (total time: 5 seconds)
```

Figure 7: Best solution of sudoku with min-conflicts.