A Sudoku Solver

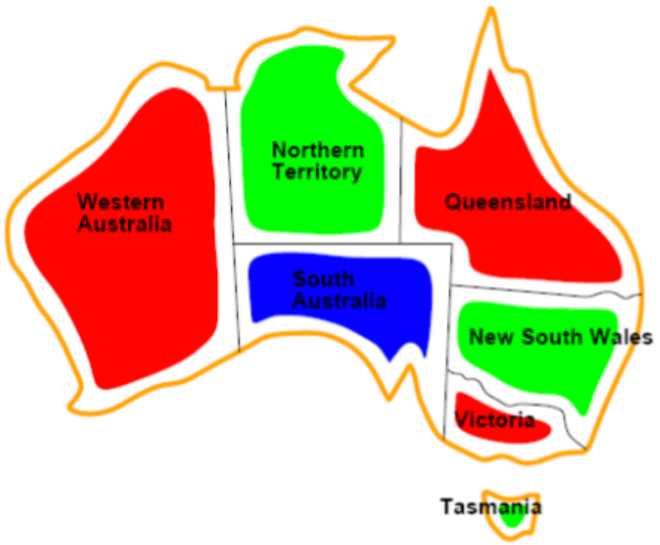
Hao Ni, Rui Liu, Sijie Yu, Yue Ding

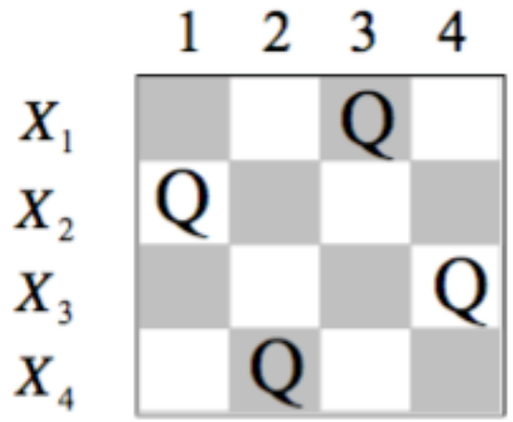
Outline

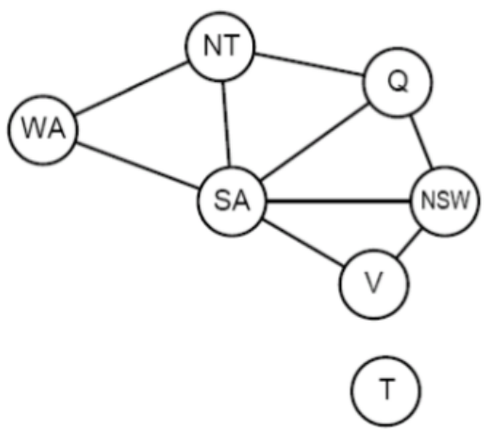
- CSP(Constrained Satisfaction Problem)
- Sudoku problem
- Existing solutions and adopted method
- Performance and future improvement

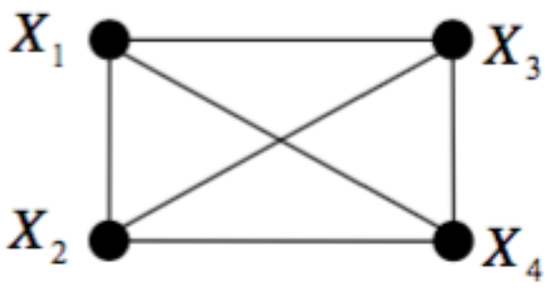
Constrained Satisfaction Problem

- **State**: defined by variable, with value from domain
- **Goal test**: a set of constraints









source: lecture note 05 constraint satisfaction

Sudoku

- Grid: 9 × 9 cells, 3 × 3 sub-grids

  Variables: 81 cells Domain:{1,2,3,4,5,6,7,8,9}
- Each row, column, sub-grid contains a permutation of 1-9

  Constraints: 27 non-equals

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

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

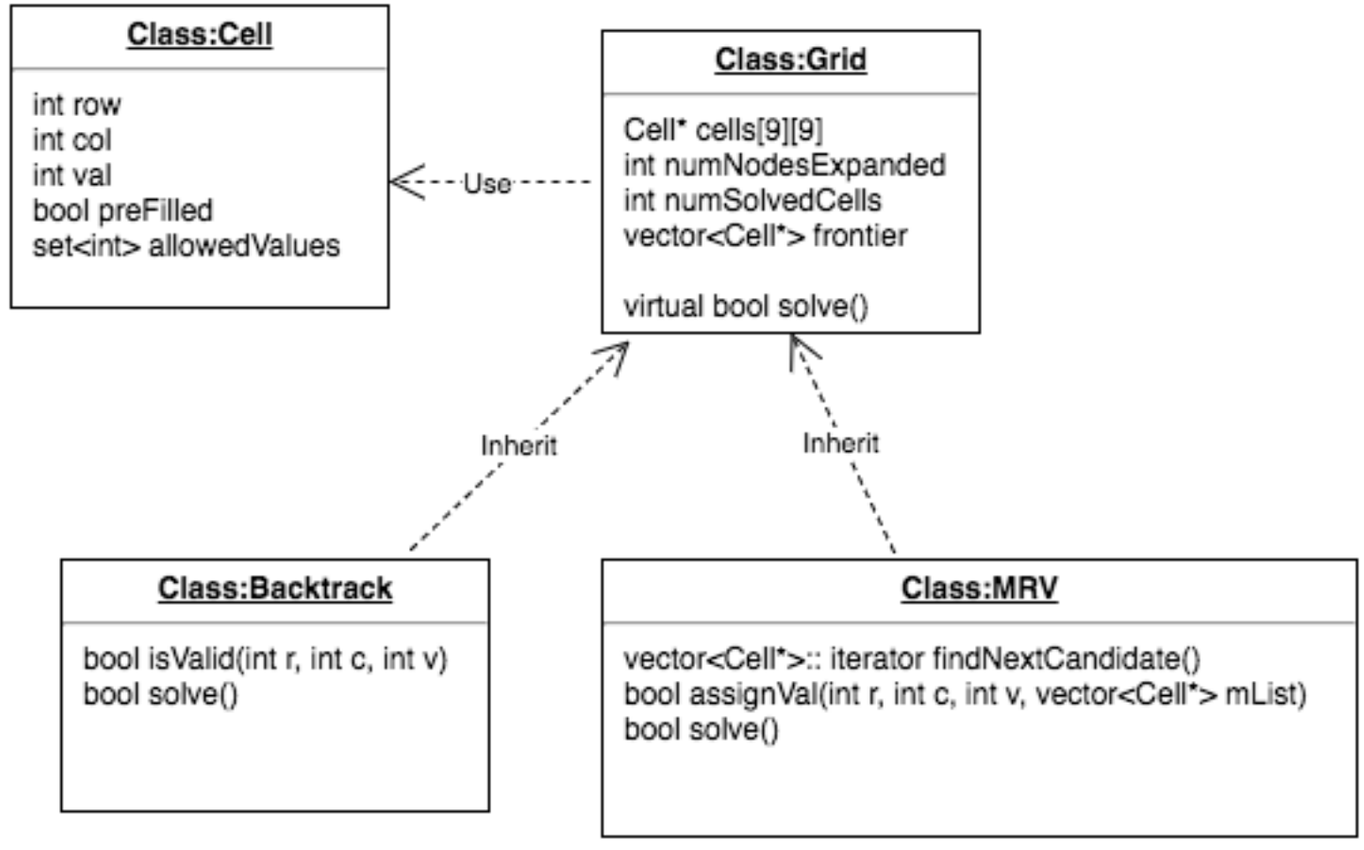
source: https://en.wikipedia.org/wiki/Sudoku

Existing solutions

- Base: backtrack
- Reducing search space:
  - arc/path/k consistency
- During search:
  - look ahead: variable/ value ordering
  - look-back: buckjump, constraint recording, etc

Our approach (C++)

- Naive backtrack and Forward Checking with MRV
- Classes: Cell, Grid, Backtrack, MRV
- Grid inherited by Backtrack and MRV



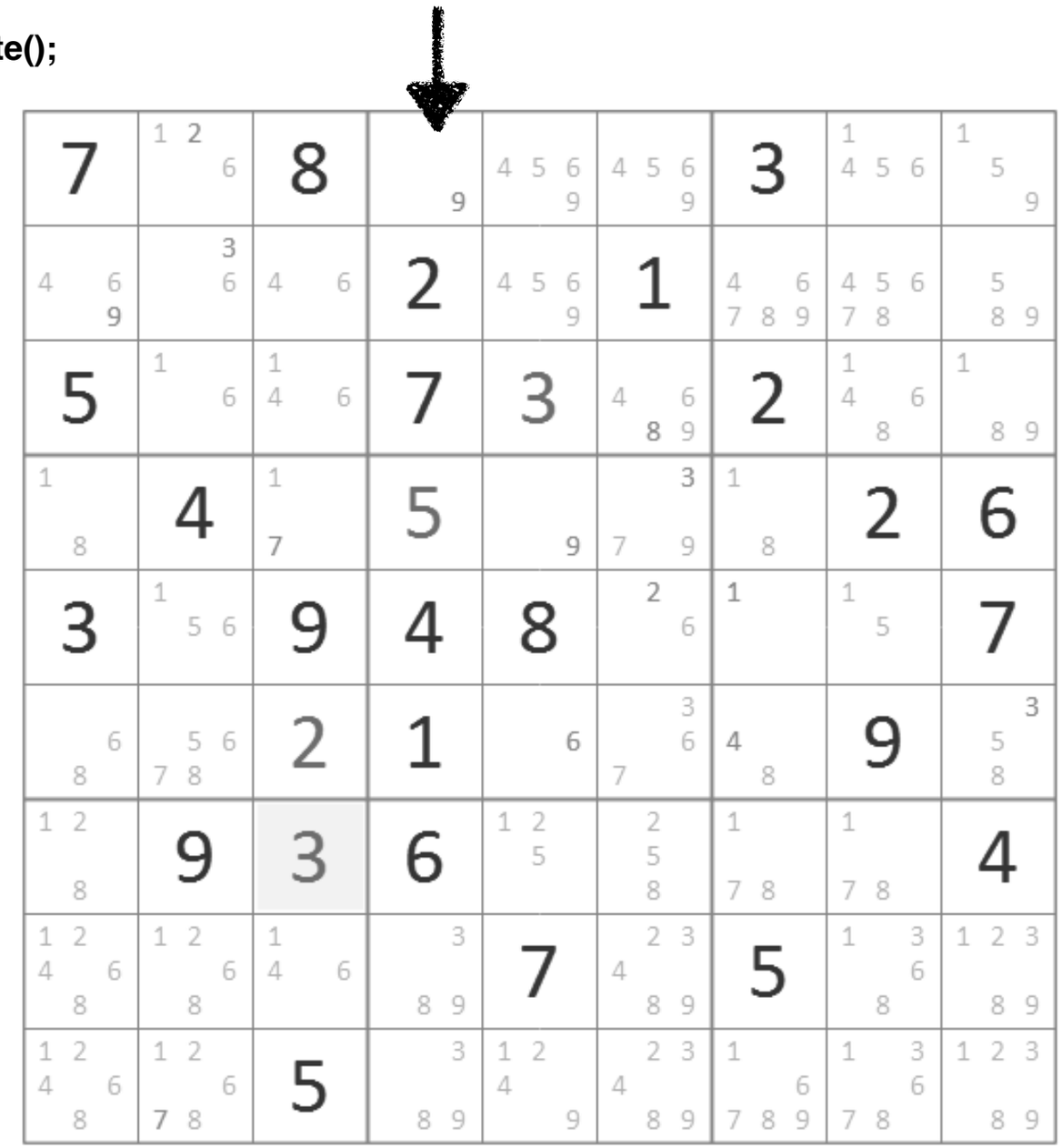
Naive backtrack

```
bool solveFrom(int row, int col) {
  if (numSolvedCell == TOTAL_CELL) return true;

   if (cells[row][col]->preFilled)
      return solveFrom(col == GRID_DIM - 1 ? row + 1 : row, col == GRID_DIM - 1 ? 0 : col + 1);

   for (int val = 1; val <= 9; val++)
      if (isValid(row, col, val)) {

         cells[row][col]->value = val;

         if(solveFrom(col == GRID_DIM - 1 ? row + 1 : row, col == GRID_DIM - 1 ? 0 : col + 1))
            return true;

         cells[row][col]->value = Cell::EMPTY_VALUE;
         numSolvedCell--;
      }
   return false;
}
```

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

FCMRV

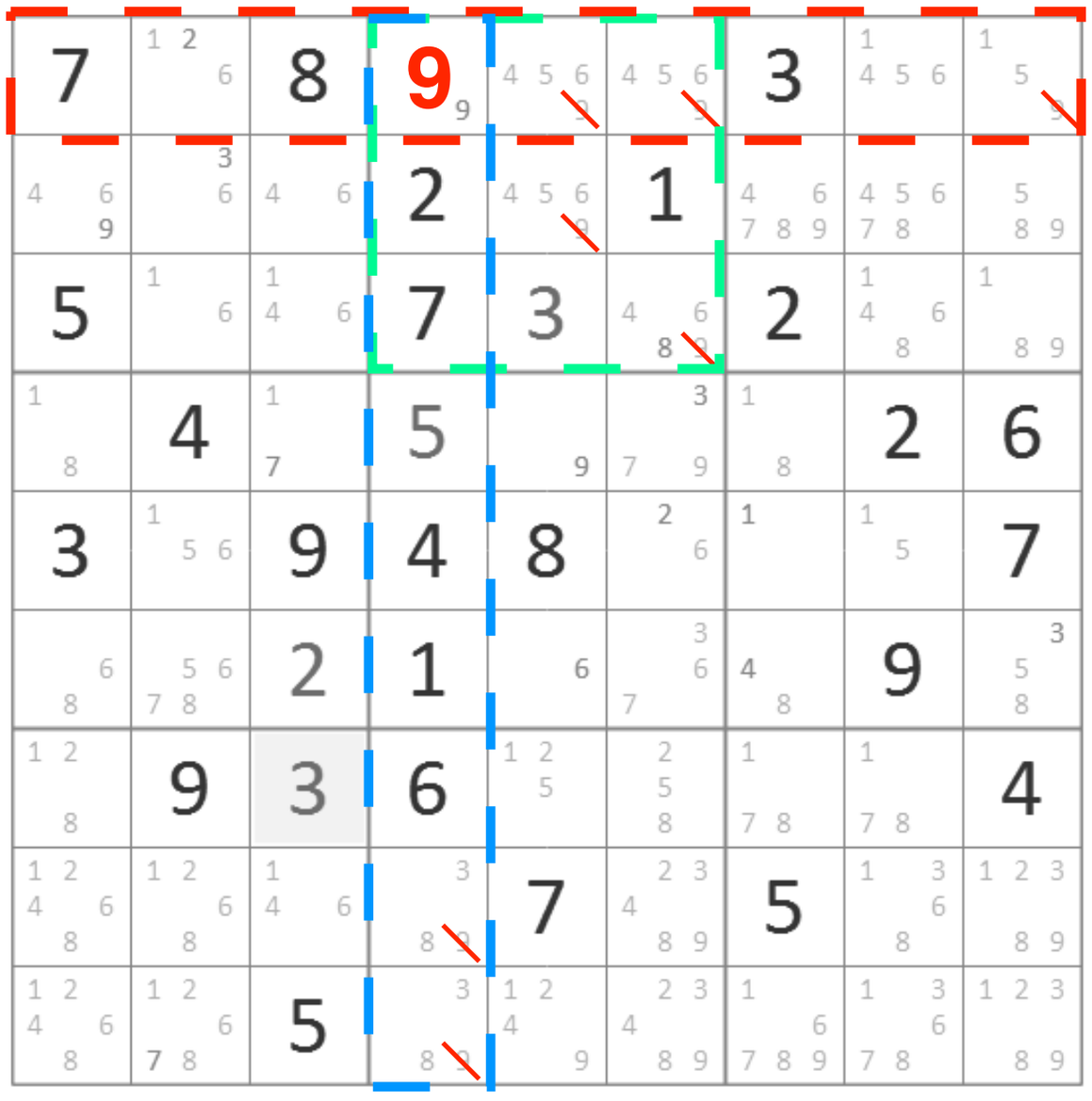
```
bool solve() {
    if (frontier.size() == 0)
        return true;

    std::vector<Cell*>::iterator curCell_itr = findNextCandidate();
    Cell* curCell = * curCell_itr;
    frontier.erase(curCell_itr);

    std::set<int> possibleValues = curCell->allowedValues;

    for (std::set<int>::iterator it = possibleValues.begin();
          it != possibleValues.end(); it++) {

        int curVal = *it;
        std::vector<Cell*> modifiedCells;

        if(!assignCell(curCell, curVal, modifiedCells))
            continue;

        if (solve())
            return true;

        for (int i = 0; i < modifiedCells.size(); i++)
            modifiedCells[i]->allowedValues.insert(curVal);
    }
    curCell->value = 0;
    frontier.push_back(curCell);
    return false;
}
```



source: https://www.codeproject.com/articles/292692/html-sudoku-solver

FCMRV

```
bool solve() {
    if (frontier.size() == 0)
        return true;

    std::vector<Cell*>::iterator curCell_itr = findNextCandidate();
    Cell* curCell = * curCell_itr;
    frontier.erase(curCell_itr);

    std::set<int> possibleValues = curCell->allowedValues;

    for (std::set<int>::iterator it = possibleValues.begin();
            it != possibleValues.end(); it++) {

        int curVal = *it;
        std::vector<Cell*> modifiedCells;

        if(!assignCell(curCell, curVal, modifiedCells))
            continue;

        if (solve())
            return true;

        for (int i = 0; i < modifiedCells.size(); i++)
            modifiedCells[i]->allowedValues.insert(curVal);
    }
    curCell->value = 0;
    frontier.push_back(curCell);
    return false;
}
```



source: https://www.codeproject.com/articles/292692/html-sudoku-solver

Performance

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

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

method	#nodes expanded	time
Naive backtrack	1150	500 us
FCMRV	63	250 us

Future improvement

- Use min-heap as frontier for MRV to find the next candidate
- Apply value ordering: e.g. least constraining value
- Introduce other heuristic