Introduction to Artificial Intelligence
Prof. Richard Lathrop
Project: Sudoku Solver

Sudoku Puzzle

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

Figure 1: Sudoku puzzle

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

Figure 2: Solution

Consider the classic 9-by-9 Sudoku puzzle in Figure 1. The goal is to fill in the empty cells such that every row, every column, and every 3-by-3 box contains exactly the digits 1 through 9, each digit occurring once. The solution to the puzzle is in Figure 2, and it satisfies these constraints:

- The digits to be entered are 1, 2, 3, 4, 5, 6, 7, 8, 9.
- A row is 9 cells wide. A filled-in row must have exactly one of each digit. That means that each digit appears exactly once in the row. There are 9 rows in the grid, and the same constraint applies to each of them.
- A column is 9 cells tall. A filled-in column must have exactly one of each digit. That means that each digit appears exactly once in the column. There are 9 columns in the grid, and the same constraint applies to each of them.
- A box contains 9 cells in a 3-by-3 layout. A filled-in box must have exactly one of each digit. That means that each digit appears exactly once in the box. There are 9 boxes in the grid, and the same constraint applies to each of them.

You can find additional information on Sudoku at:

http://en.wikipedia.org/wiki/Sudoku
http://en.wikipedia.org/wiki/Sudoku_algorithms
http://www.sudokuweb.com/
http://4c.ucc.ie/~hsimonis/sudoku.pdf

You are to write a program to solve the classic 9x9 Sudoku puzzle. Note that every correct Sudoku puzzle has only **one** correct solution (see Figure 2).

Part 1

Implement a backtracking search with Forward Checking (FC) for the Sudoku puzzle. Compare time/space performance with, and without, the Minimum Remaining Values (MRV) heuristic.

In a Word document, provide a table that compares, for each of the test cases listed below, the performance (the total clock time, the search-only clock time, the number of nodes generated, and the effective branching factor [N = number of nodes generated, and d = number of initially unassigned variables; see Russell&Norvig, Section 3.6.1]) of the following two algorithms:

(**BT-FC**) naïve backtracking with FC but without MRV. Variables have a fixed ordering, left-to-right, top-to-bottom: A1, A2, …, A9, B1, …, B9, …, I9 (Russell&Norvig, Figure 6.4).

(**BT-FC-MRV**) backtracking with the MRV heuristic used to order the variables. At each step of the backtracking search, choose the variable with the Minimum Remaining Values. (For Bonus Points, you might implement the Degree heuristic [i.e., degree to unassigned variables].)

In both cases, order the variable values in fixed increasing order, i.e., 1, 2, 3, 4, 5, 6, 7, 8, 9. Of course, since you have done Forward Checking (FC), some values will be eliminated already. (For Bonus Points, you might implement the Least Constraining Value heuristic ordering.)

Discuss your results. Did you see the behavior you expected? **Why or why not?**

Hint: JAVA is slow to allocate memory and slow to do garbage collection. Can you conserve memory, i.e., can you code it to do fewer memory allocations and fewer garbage collections? E.g., it is far better to allocate a single large static block of memory at start-up (maybe an array?) than to iteratively allocate and reclaim many small blocks of memory stepwise during the search.

Part 2

Part 2 (BT-AC-MRV and BT-ACP-MRV) is similar to Part 1 (BT-FC-MRV), except that you will augment Backtracking search by using Arc Consistency (AC or ACP). Arc Consistency will be done either: (ACP) as a pre-processing step only, with (only) Forward Checking still being done after each variable assignment; or (AC) both as pre-processing and after each variable assignment, thus replacing Forward Checking entirely. (If you do AC after each assignment, there is no point to doing FC.) Both variants will use the MRV heuristic to order variables. (For Bonus Points, you might implement the Degree heuristic [i.e., degree to unassigned variables].)

Again, order the variable values in fixed increasing order, i.e., 1, 2, 3, 4, 5, 6, 7, 8, 9. Of course, since you have done AC or FC already, some values will be eliminated already. (For Bonus Points, you might implement the Least Constraining Value heuristic ordering.)

Extend the table in the Word document from Part 1 to include, for each test case, backtracking with MRV and Arc Consistency during search (BT-ACP-MRV and BT-AC-MRV). Compare the performance of the following four algorithms:

(**BT-FC**) Already done in Part 1.

(**BT-FC-MRV**) Already done in Part 1.

(**BT-ACP -MRV**) New for Part 2. Run Arc Consistency (AC, also called AC-3) as a pre-processing step on start-up, then run BT-FC-MRV as in Part 1. The point of Part 2 is to examine the performance improvement (or degradation?)

(**BT-AC -MRV**) New for Part 2. Run Arc Consistency (AC) after each variable assignment. Recall that Arc Consistency (AC) is a more general and powerful version of Forward Checking (FC), and so if you do AC there is no point in also doing FC. The point of Part 2 is to examine the performance improvement (or degradation?)

Discuss your results. Did you see the behavior you expected? **Why or why not?**

In the same document, for each test case, also provide the values remaining for each variable after running the arc consistency algorithm before search starts. This will verify that you have implemented AC correctly. (You can check this easily by hand-verifying your implementation.)

General Instructions

• You may discuss the problem with other people, but must develop your program and write your project report independently. No direct copying of code or text.
• The main class (i.e., containing the **main** function) must be named **SudokuSolver**. It is **strongly** recommended that you test your system on the command line as we will use the command line to grade each part. Projects that work "on your machine" or "only in Eclipse" will not receive full credit. Projects that do not compile will receive very little credit.
• Your program **must** take as command line arguments a file containing the Sudoku puzzle and the techniques to be used by the algorithm: MRV for minimum remaining value, AC for arc consistency to be used during search, ACP for arc consistency to be used before the search. (If you implement additional heuristics for the Bonus Points, give them similar behavior; e.g., Degree heuristic might be DH, Least Constraining Value heuristic might be LCV, etc.; define these extensions and their abbreviations in a separate section of your project report.) The file containing the puzzle is a required argument, while the techniques are optional arguments. If no techniques are specified, naïve backtracking is performed (BT-FC). The first command line argument should be the file, while the order of the other optional arguments does not matter.

Examples:

```
java SudokuSolver puzzle1.txt MRV AC //backtracking with MRV and AC
java SudokuSolver puzzle1.txt AC MRV //same as above
java SudokuSolver puzzle1.txt        //naive backtracking
```

• We **require** the following file format: the puzzle is represented by a 9x9 matrix, where a "0" means that the respective cell is unassigned (the cells are separated

by space characters). For example, the input file corresponding to the Sudoku puzzle in Figure 1 is given below.

```
0 6 0 1 0 4 0 5 0
0 0 8 3 0 5 6 0 0
2 0 0 0 0 0 0 0 1
8 0 0 4 0 7 0 0 6
0 0 6 0 0 0 3 0 0
7 0 0 9 0 1 0 0 4
5 0 0 0 0 0 0 0 2
0 0 7 2 0 6 9 0 0
0 4 0 5 0 8 0 7 0
```

• Your program **must** output to the screen the solution in a similar manner (the 0-s should be replaced by the correct digits). You **must** also output the running time in **milliseconds** (or fraction of) and the number of nodes generated. Your output, given the above file as input, should **exactly** match the following format (with variability in the time and the number of nodes generated):

```
9 6 3 1 7 4 2 5 8
1 7 8 3 2 5 6 4 9
2 5 4 6 8 9 7 3 1
8 2 1 4 3 7 5 9 6
4 9 6 8 5 2 3 1 7
7 3 5 9 6 1 8 2 4
5 8 9 7 1 3 4 6 2
3 1 7 2 4 6 9 8 5
6 4 2 5 9 8 1 7 3
Time: 1002.52
Nodes: 500
```

• You are **required** to use the Java programming language.
• Your source code **must** have comments. Java programming language.
• Please use this timing rubric:

1. PROGRAM START.
2. TOTAL_START <- current time.
3. input the problem, do all pre-processing and initialization.
4. SEARCH_START <- current time.
5. search until answer or failure.
6. END <- current time.
7. TOTAL_TIME <- (END - TOTAL_START).
8. SEARCH_TIME <- (END - SEARCH_START).
9. output the answer and record all statistics.
10. PROGRAM END.

Of course, during step 9 you can easily calculate other derivative statistics of interest, for example, INIT_TIME <- (TOTAL_TIME - SEARCH_TIME), and so on. As another example, you might be curious how much of your INIT_TIME was due to input of the problem and how much was due to pre-processing and initialization, so you might record other timing variables during step 3 and analyze them in step 9.

Test Cases

We require that you run your experiments on the following three test cases. However, make sure that your system works for any legal 9x9 Sudoku puzzle. In addition to these three test cases, we will use other test cases to grade your system.

Test Case 1

```
0 6 0 1 0 4 0 5 0
0 0 8 3 0 5 6 0 0
2 0 0 0 0 0 0 0 1
8 0 0 4 0 7 0 0 6
0 0 6 0 0 0 3 0 0
7 0 0 9 0 1 0 0 4
5 0 0 0 0 0 0 0 2
0 0 7 2 0 6 9 0 0
0 4 0 5 0 8 0 7 0
```

Test Case 2

```
0 0 0 0 0 4 9 0 0
0 0 5 3 2 0 0 0 0
2 0 0 0 0 6 0 4 0
8 0 4 0 0 0 0 6 0
0 5 0 0 6 0 0 1 0
0 1 0 0 0 0 3 0 9
0 2 0 8 0 0 0 0 6
0 0 0 0 7 9 1 0 0
0 0 9 5 0 0 0 0 0
```

Test Case 3

```
0 0 9 0 2 8 0 0 0
0 8 0 0 0 0 9 0 0
0 7 0 0 5 0 0 0 0
0 3 8 9 0 0 1 0 5
0 0 0 0 0 0 0 0 0
6 0 4 0 0 5 2 9 0
0 0 0 0 4 0 0 6 0
0 0 6 0 0 0 0 3 0
0 0 0 7 3 0 5 0 0
```

Grading

- **50%** technical correctness of your Sudoku solver.
  I.e., is every puzzle solved correctly, with substantial penalties for **any** errors in the returned solutions, and are the various speed-up algorithms implemented correctly?
- **25%** technical merit and completeness of your analysis.
  The technical soundness of your response to "Compare the performance of the algorithms," i.e., do you collect, present, and analyze data allowing the reader to judge how much each algorithmic improvement actually helped (or hurt??) performance?
- **25%** clarity, logical organization, and written communication skills
  The technical clarity of your response to "Discuss your results," i.e., if your project report were published as a Technical Manual, would it be clear, logical, and comprehensible?
- **** **15** ALGORITHM BONUS POINTS ARE AVAILABLE ****
  You are encouraged to implement algorithms that go beyond the minimum requirements stated above, and you will be rewarded if you do. You will receive up to 10% bonus for the first such extension, and up to another 5% bonus for the second such extension. For example, you could implement the Least-Constraining-Value heuristic for the first and the Degree heuristic for the second (Russell & Norvig, pages 216-217). You might find some other ideas in http://4c.ucc.ie/~hsimonis/sudoku.pdf or other sources.

  To get full bonus point credit, your project report must describe and analyze your algorithmic extensions as above, i.e., "technical merit and completeness of your analysis," and "clarity, logical organization, and written communication skills." Add performance results to the table above.

  Discuss your results. Did you see the behavior you expected? **Why or why not?**
- **** **10** SPEED BONUS POINTS ARE AVAILABLE ****
  You may designate your absolute fastest Sudoku solver, with all of your best speed hacks, and all of your best extra-credit additional heuristics, for the SPEED BONUS POINTS competition. All programs will be run in the same way on the same machine and be compared on total run time (total clock time), including input, preprocessing, and search.

  The fastest 10% of the class will get 10 bonus points, the second fastest 10% of the class will get 9 bonus points, etc. The sharp-eyed student will see that everyone gets at least one.

PLEASE NOTE:

Your project report ***MUST*** properly cite every outside source that you use (an outside source is anything that is not part of the class material). Remember that the difference between research and plagiarism is citing your sources.

You may ***NOT*** directly copy any code or any text from anywhere at any time. Both your code and your text are subject to analysis by computerized plagiarism detectors. An exception to this rule is directly quoted material set off in quotes and properly cited, e.g., "Proper citations are the foundation of rigorous scholarship" --- Jane Doe [Doe et al., page 234].

What to hand in

When submitting each part, zip all of your source code and the Word document in a file called **<last name>_<student id>.zip** (e.g. smith_12345678.zip). Do not include any *.class files. You will upload the zip file for each part into the EEE dropbox called “Project Part 1” and “Project Part 2” respectively before 4:00pm on the due dates

Also, you **must** hand in a hardcopy of your Word document in class on the due date for each part. Late projects will receive 0 credit. For a project to be on time, **both** your zip file and the hardcopy **must** be turned in on time.

Part 1 and Part 2

You must write up the full project report for both parts.

In summary: Part 1 must meet all the requirements in the Project Specification. Part 1 will be treated as a "near-final draft." You will revise part 1 to create part 2. Part 2 will be the final graded version.

(1) **Part 1 must meet all the requirements in the Project Specification.** It must be written in the format of a technical report or a technical paper. Both text and data must be complete. An ***incomplete*** part 1 will lose points.

(2) **Part 1 will be treated as a "near-final draft."** Michael Zeller, the class reader, will provide brief comments on your part 1 about both the technical content and the technical writing. You will not lose points for these feedback comments (if part 1 is complete). These comments are to help you improve for part 2. You can expect this feedback by 21 May via EEE.

(3) **You will revise part 1 to create part 2.** This means, take your part 1 document and edit it to create your part 2 document. Part of your revisions will be to include additional heuristics, as specified. Part of your revisions will be to improve your technical presentation based on the feedback from Michael Zeller.

(4) **Part 2 will be the final graded version.** You will be assigned a grade for the entire project based on the final material you submit for part 2 (and if part 1 was incomplete, you will lose points as stated above).

**** The point of this arrangement is to make the project exercise as pedagogical as possible, so that you have a chance to learn, improve, and do better as you go along. ****

Outline of your report

Write your report as if it were a technical report or a technical paper, briefly and compactly.

[My Report Title]

[My Name]
[My Institutional Affiliation]
Dept. of Computer Science [or whatever yours is]
Univ. of California, Irvine
Irvine, CA 92697, USA

Abstract: Provide a brief summary of the report in 150 words or less, including main results.

Introduction.

Very briefly, in about two or three paragraphs, orient the reader to the material that will follow. Define "The Sudoku Problem" by stating very concisely the rules of Sudoku. Briefly motivate and define Sudoku as a Constraint Satisfaction Problem (CSP). State that the goal of the paper is to test and compare the performance of various CSP heuristics on the Sudoku problem.

Write tersely and compactly, striving for a high information density. Choose your words carefully. Ensure that the sequence of thoughts is presented in a logical, easy-to-follow order.

Methods.

Define your methods.

In one subsection, briefly define the heuristics you will consider below. For heuristics that appear in your book, it is sufficient to cite [Russell & Norvig, page XXX]; it is not necessary to define them all over again. For heuristics not in your book, e.g., those you may have found on the web or in the technical literature, describe them briefly and provide a citation to the source.

In another subsection, briefly describe the run-time methodology you used to collect the data below. Low-level details are unnecessary; write so that a technically literate reader can understand what you did, providing only enough information to allow them to fill in the low-level details by inference.

Write tersely and compactly, striving for a high information density.

Results.

Present your technical results in a way that is easy for the reader to understand.

In one subsection, present the required table (discussed above) giving your main results from comparing the heuristics head-to-head on the same problems.

Optionally, in another subsection, present any other results you believe may be of technical interest to the reader. If none, omit.

Heuristic(s) Used	Total Time (msecs)	Search Time (msecs)	Nodes Generated	Effective Branching Factor
Test Case 1				
BT-FC				
BT-FC-MRV				
BT-ACP-MRV				
BT-AC-MRV				
[optionally, any others used to get Bonus Points]				
…				
Test Case 2				
BT-FC				
BT-FC-MRV				
BT-ACP-MRV				
BT-AC-MRV				
[optionally, any others used to get Bonus Points]				
…				
… and so on …				

Discussion.

Briefly discuss your results. Did you get the Results you expected? Why or why not? What follow-on studies might be done to shed additional light on the problem?

Do not be verbose in the Discussion section. The main technical content of the report is contained in the Results section. Here, you state the lessons learned, briefly and compactly.

Acknowledgments. [may be omitted if none]

Thank any individual or organization who helped you (e.g., "Thanks to Mary Smith for helpful technical discussion."). First acknowledge individuals, then organizational support, and finally funding sources (you may not need all these categories). It is unnecessary to thank the instructor.

References.

Give full citations to the textbook and any outside material. It is unnecessary to cite the class lecture notes.

Supplementary Information.

Also provide a table giving the values remaining for each variable after running the arc consistency algorithm before search starts. This will verify that you have implemented AC correctly. (You can check this easily by hand-verifying your implementation.)