CS171: Introduction to Artificial Intelligence
Professor Richard Lathrop
Sudoku Project Report

1. A. Generate a set of 5 boards with P = 3, Q = 3, M = 15.

   B. Run your Sudoku solver on the generated boards and fill in the following table:

	Trail Pushes	Backtracks
FC	2890	85
FC MRV	1718	17
FC LCV	543293	29463
FC MRV LCV	2384	73
NOR	1730	28
NOR MRV LCV	1431	0
NOR DEG LCV	13906	788
NOR MAD LCV	1309	0

C. Which set of heuristics produced the best results? Explain.
Best: NOR MAD LCV
It has all backtracks number as 0; and the least trail pushes. NOR MRV LCV has tied amount of backtracks but because NOR MAD LCV uses DEG when MRV finds tied variables, which chooses better variables for the next assignment.

D. Describe any anomalies or patterns. Explain in detail.
1. Heuristics with FC performs worse than heuristics with NOR in backtrack numbers because to reduce the Backtracks number, we need to propagate more variables and assign more variables on checking function.
2. Usually, the more backtracks, the more trail pushes because if backtracks happen, it will do more propagates which means pushing more variables on the trail.
3. Doing without value and variable heuristics, always get the same number of backtracks on the same board. For others, the more complex heuristics apply, the variance of backtracks for running the same board each time get less. Because of changing remaining value and degrees during checking, the next variable or value choose gets varied.
4. Degree heuristics should not be used alone. The performance of using DEG depends on the board. Sometimes, it gets like 0 backtracks, the least trail pushes. But sometimes, it gets crazy number of backtracks. It's better to use with MRV

2. A. Generate a set of 5 boards with P = 4, Q = 3, M = 25.

   B. Run your Sudoku solver on the generated boards and fill in the following table:

	Trail Pushes	Backtracks
FC	484205	37165
FC MRV	4133	32
FC LCV	3669	7
FC MRV LCV	3814	5
NOR	4998	96
NOR MRV LCV	3646	0
NOR DEG LCV	14292	397
NOR MAD LCV	3583	4

C. Which set of heuristics produced the best results? Explain.
Best: NOR MRV LCV
It has all backtracks number as 0; and the least trail pushes. NOR MAD LCV and NOR MRV LCV produce very similar number of backtracks for sparse boards. Ideally, NOR MAD LCV acts better but it really depends on the board. My five boards happen to be the situation NOR MRV LCV performs better.

D. Describe any anomalies or patterns. Explain in detail.
1. Heuristics with FC performs worse than heuristics with NOR in backtrack numbers because to reduce the Backtracks number, we need to propagate more variables and assign more variables on checking function.
2. Usually, the more backtracks, the more trail pushes because if backtracks happen, it will do more propagates which means pushing more variables on the trail.
3. Doing without value and variable heuristics, always get the same number of backtracks on the same board. For others, the more complex heuristics apply, the variance of backtracks for running the same board each time get less. Because of changing remaining value and degrees during checking, the next variable or value choose gets varied.
4. Degree heuristics should not be used alone. The performance of using DEG depends on the board. Sometimes, it gets like 0 backtracks, the least trail pushes. But sometimes, it gets crazy number of backtracks. It's better to use with MRV

3. A. Generate a set of 3 boards with P = 4, Q = 4, M = 45.

B. Run your Sudoku solver on the generated boards and fill in the following table:

	Trail Pushes	Backtracks
FC	Large	Large
FC MRV	10456	299
FC LCV	27084	1065
FC MRV LCV	5920	38
NOR	Large	Large

NOR MRV LCV	5260	15
NOR DEG LCV	Large	Large
NOR MAD LCV	5026	0

C. Which set of heuristics produced the best results? Explain.

Best: NOR MAD LCV

It has all backtracks number as 0; and the least trail pushes. NOR MAD LCV and NOR MRV LCV produce very similar number of backtracks for sparse boards. Ideally, NOR MAD LCV acts better but it really depends on the board. My five boards happen to be the situation NOR MRV LCV performs better.

D. Describe any anomalies or patterns. Explain in detail.

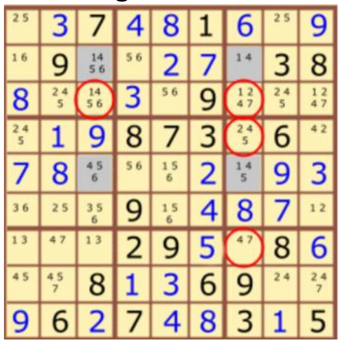
1. Heuristics with FC performs worse than heuristics with NOR in backtrack numbers because to reduce the Backtracks number, we need to propagate more variables and assign more variables on checking function.
2. Usually, the more backtracks, the more trail pushes because if backtracks happen, it will do more propagates which means pushing more variables on the trail.
3. Doing without value and variable heuristics, always get the same number of backtracks on the same board. For others, the more complex heuristics apply, the variance of backtracks for running the same board each time get less. Because of changing remaining value and degrees during checking, the next variable or value choose gets varied.
4. Degree heuristics should not be used alone. The performance of using DEG depends on the board. The occurrence of Degree heuristics gets stuck increases a lot when there are much more possibilities on larger board. It's better to use with MRV
5. As the board grows, the difference between heuristics with NOR and FC increases because second heuristic in NOR appears more often on the board.

4. Did the same set of heuristics produce the best results for (1), (2), and (3)? If not/explain why this might have happened.

   No, NOR MAD LCV and NOR MRV LCV produce very similar number of backtracks for sparse or small boards. Ideally, NOR MAD LCV acts better but it really depends on the board. DEG heuristics chose variable with the variables with the most number of unassigned neighbors on the board when MRV ties. It helps propagate more variables on the next checking. For MRV, it just randomly chooses variables when tied up. For sparse board, there are many variables with high number of unassigned neighbors. It's easy for MRV to choose one with high number of unassigned neighbors. For small board, variables usually contain small number of neighbors. MAD does a little work in these situations because there are not many unassigned neighbors to propagate. Also, usually the result depends on the board and every time it may produce different results for the same board. So, it's hard to tell which one definitely has the best performance.

5. Did you implement the Extra Credit heuristics? If so, which heuristics did you implement? Why did you choose these heuristics? How did they compare to the results in questions (1), (2), and (3) .

   For value heuristics and variable heuristics, we take LCV and MAD. We cannot think about better way to choose variable and value. So, we only make improvement on checking function. To reduce the Backtracks number, we need to propagate more variables and assign more variables on checking function. To reduce the Trail Push number, we need to detect any inconsistency on the board as soon as it appears. Two things we did to accomplish targets above: first, we implement X-Wings, an advanced Sudoku strategy and second, we add self.network.isConsistent() after there appears new assigned variable. For X-Wings, an example we found online is

   Whenever, there are two rows/columns on the board at the same column/row position contains two unassigned variables with the same value. That value exists only on these two variables on that row/column. Delete that value from other variables on the same column/row. More details see in sudokuessentials.com/ x-wing.html.
   It helps propagate more variables on the board for this situation, which decreases the number of backtracks. Also, we detect inconsistency on every constraint whenever we assign a variable. Although it costs much more time to run, it prevents from pushing more variables on the trail before finding inconsistency.
   For (1)(2)(3), it does not improve but it still can limit number of backtracks to 0-5. We think it may be because when we do extra propagation X-wings, it may appear inconsistency which should not happen in the last assigned variables. But it did much better in the case when NOR still gets large number of backtracks. We don't know how we can continue to improve that.

6. Did you encounter any problem when doing the project? How did you resolve these problems?

The shell didn't include many guidelines. The hardest thing to do is to understand the shell code at the beginning. It takes time to understand them all. Also checking function is the hardest part to code. It takes time to understand how functions, variables on the shell code provides helping for checking function. Also, Piazza helps me a lot.

7. (Optional) Do you have any suggestions to improve the shells?
   Please include more guides on the shell code, especially the part which may make confusion.

Work cited

sudokuessentials.com, “X-Wings”, *sudokuessentials.com/ x-wing.html*