AI PRINCIPLES & TECHNIQUES

Assignment 2: Sudoku

Jort Post *s1114712*
Victor Janssen *s1121859*
Radboud University *22-11-2024*

Contents

1 Introduction

Constraint satisfactions problems (CPSs) are mathematical problems commonly used in artificial intelligence. These problems are useful for making multiple decisions at the same time, making them ideal for a Sudoku game. Sudoku is a logic-based game in which you have to place numbers into a 9x9 grid, which is divided into nine smaller 3x3 grids. The goal of the game is to fill the entire grid with numbers 1-9, while following the rules that each number can only appear once in every row, column, and 3x3 grid. A partially finished grid, which offers a single solution for a well-posed puzzle, is provided by the creator of a Sudoku puzzle.

This project addresses our implementation of the AC-3 algorithm applied in Sudoku puzzles. Additionally, several heuristics were applied to the algorithm, and the corresponding complexity has been measured.

2 Constraint Satisfaction Problems

A CSP is defined as a tuple which includes: a set of variables, each with a domain which represents a set of choices that can be made for the specific value of the variable and

a set of constraints, which make it impossible to assign certain values to the variables. For example, a general constraint could be that one variable has to have a higher value than the other, or that two variables cannot have the same value. In Sudoku, one of the constraints is that a number can only appear once in every row, column and smaller 3x3 block. The solution to a CSP can be found by assigning values to variables in such a way that all the constraints are satisfied, and that no constraints are violated. One of the most important challenges about Sudoku is limiting the number of assignments that can be made to each cell without breaking any constraints. There are several methods and algorithms to solve these problems. In section the following section, the focus lies on the implementation of the **AC-3 algorithm** which enforces arc consistency.

2.1 AC-3 Algorithm

The Arc Consistency 3 algorithm is one of the algorithms which can be used to solve CSPs. The algorithm turns every binary constraint of the CSP into two different arcs, ensuring that each pair of related variables has at least one valid assignment in their domains that satisfies the requirement. For example, the constraint $A \neq B$ becomes $A \neq B$ and $B \neq A$. The algorithm adds all arcs to a queue, which is a list of arcs to be processed. Any arcs impacted by the algorithm's revisions to the domain of variables are returned to this queue for additional processing. The algorithm specifically takes the following actions:

1. Place all arcs (variable pairs with constraints) in a queue.
2. For each arc (X, Y), check if each value in X's domain has a compatible value in Y's domain. Time complexity: $O(d^2)$.
3. Remove values from X's domain that do not satisfy the constraint with any value in Y's domain (Revise).
4. If X's domain was modified, add arcs (Z, X) for each neighbor Z of X back to the queue.
5. Continue processing arcs until the queue is empty.
6. If any domain becomes empty, the problem is unsolvable for the current implementation, otherwise the problem is arc-consistent.

The time complexity of the AC-3 algorithm depends on the number of arcs in the queue and the number of revisions made to the domains of the variables. The worst-case time complexity depends on e, the number of constraints in the problem and d the maximum domain size. Each binary constraint in a CSP creates two arcs, one for each direction, so the number of arcs is related to the number of constraints. Therefore, the worst-case time complexity of the AC-3 algorithm is $O(ed^2)$: the algorithm will check every arc and perform $O(d^2)$ operations for each one.

2.2 Heuristics

Heuristics can be used in CSPs to help selecting the most promising variable or value for each step, this will ensure a lower search space and an improved efficiency. These include the Minimum Remaining Values (MRV) heuristic, the combination of MRV with the Degree heuristic, and the Least Constraining Values (LCV) heuristic.

The Minimum Remaining Values (MRV) heuristic focuses on choosing the variable with the fewest legal values remaining in its domain. The method can more effectively reduce the search space and possibly find the best answer more quickly by choosing the variable with the fewest remaining values.

In some cases, two or more variables may have the same number of remaining values in their domains. Just MRV on its own will not be able to give the most optimal variable choice. If this is the case, the **degree heuristic** is used as a tiebreaker. The variable that is most constrained by other unassigned variables is chosen when using the degree heuristic.

The Least Constraining Values (LCV) heuristic is another commonly used heuristic that is applied when selecting values for a chosen variable. Given a variable to assign a value to, the LCV heuristic suggests choosing the value that rules out the fewest values for the remaining variables. By maintaining flexibility for the remaining variables, this method helps in determining the most promising areas of the search space.

3 Methodology

3.1 Implementation

The AC-3 algorithm is implemented in Python as solve() with two helper functions revise() and fill_queue(). The only packages used are the heapq and the os module.

Variables

The most important variables used in the implementation of the AC-3 algorithm are: Field(), grid, queue and arc.

The Field object has three properties: a value, a list of neighbours and a domain. The value of a field is determined by a template Sudoku file, but it is most likely equal to zero at the start. The list of neighbours is filled with the neighbouring fields in the grid that constrain the current field. In other words, the fields in the same 3x3 block, in the same 1x9 row and in the same 9x1 column. The domain is represented as a list of integers ranging from 0 to 9 by default or none if the field already has a non-zero starting value. While iterating through the AC-3 algorithm, the domain of a field shrinks until it might have only one value left: this becomes the value of this Field() instance.

The grid is a list which consists of 9 lists made up of 9 Field() instances. It is used primarily in the add_neighbours() function to add all the constraining neighbours of a specific field in the grid to the neighbour attribute of that field.

The queue is a heapq data-structure consisting of a priority value and an arc in this implementation. The arc is represented as a tuple containing a field instance with a constraining neighbour which is also a field instance. The arcs together form the constraints.

Heuristics

The heuristics are a part of the AC-3 algorithm and the blocks of code below are mere snippets of said algorithm.

Default: (not really a heuristic)

Listing 1: Default (No heuristic)

```
heapq.heappush(queue, (id(neighbour), id(field1), (neighbour, field1)))
unresolved_arcs.add((neighbour, field1))
```

Minimum Remaining Value: Looks at the field with the smallest domain size. The degree heuristic is used as a backup which looks at the field with the most amount of neighbours without a finalized value.

Listing 2: Minimum Remaining Value Heuristic

```
heuristic = neighbour.get_domain_size()
try:
    heapq.heappush(queue, (heuristic, id(neighbour), id(field1), (
        neighbour, field1)))
    unresolved_arcs.add((neighbour, field1))
except:
    #Degree Heuristic:
    heuristic = 0
    for n in neighbour.get_neighbours():
        if not n.is_finalized():
            heuristic += 1
    heapq.heappush(queue, (heuristic, id(neighbour), id(field1), (
        neighbour, field1)))
    unresolved_arcs.add((neighbour, field1))
```

Most constraining neighbour: Looks at the field with the most amount of neighbours with a finalized field.

Listing 3: Most Constraining Neighbour Heuristic

```
heuristic = 0
for n in neighbour.get_neighbours():
    if n.is_finalized():
        heuristic += 1
heapq.heappush(queue, (heuristic, id(neighbour), id(field1), (neighbour,
    field1)))
unresolved_arcs.add((neighbour, field1))
```

Output verification

Below is the code for the valid_solution() function which checks the validity of the output of the AC-3 algorithm by looping over all the fields in the grid and check if they dissatisfy either of two conditions:

1. The field has a non-zero value and is thus finalized
2. The field has a value which is not equal to any other value of it's constraining neighbours.

As soon as a field does not satisfy one of these conditions, the value of that field is not valid and therefore the solution is invalid and the function breaks the loop by returning False.

Listing 4: Output Verification Function

```
    def valid_solution(self) -> bool:
        self.show_sudoku()
        grid = self.sudoku.get_board()
        print(f'Number of arcs revised: {self.constraints_revised}')

        for row in range(9):
            for col in range(9):
```

```
                if not grid[row][col].is_finalized():
                    return False
                elif any(grid[row][col] == neighbour.get_value() for
                    neighbour in grid[row][col].get_neighbours()):
                    return False
        return True
```

3.2 Experiment

To analyze the complexity of the AC-3 algorithm based on different heuristics, a runtime measure is used. The runtime measure is a variable that keeps track of the number of revisions the algorithm makes. Every time the domain of a field gets modified inside the revise() helper function, this variable gets incremented by one. After the AC-3 algorithm terminates, this measure is printed and stored in a dataset (Appendix A). To gather enough data for the comparison of this measure, the experiment is conducted in 10 trials for each heuristic, for each Sudoku. Resulting in 200 data-points. After the data has been gathered, the runtime measure of each heuristic will be compared across different Sudokus, which are used to answer the question: "How do different heuristics impact the complexity of the AC-3 algorithm?"

Hypothesis
All heuristics used in AC-3 will significantly reduce the number of revisions of arcs compared to not using a heuristic.

4 Results

The results of the experiment can be found in the table in appendix A. The figures mentioned below are a visual representation of said data:

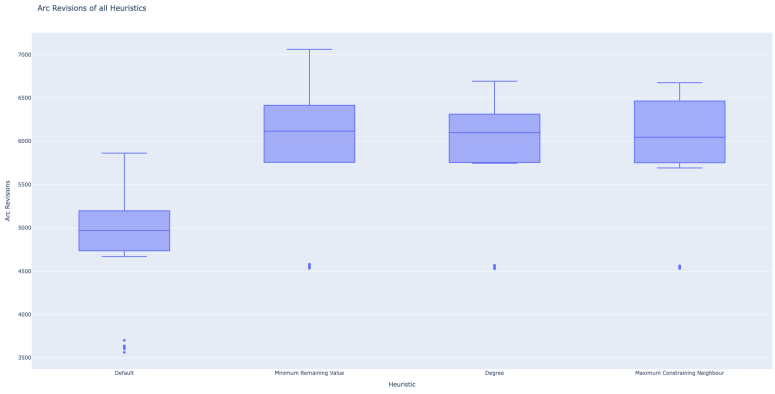

Figure 1: Arc revisions of each heuristic across all Sudokus

The results for the default heuristic, the minimum remaining value heuristic, the degree heuristic and the maximum constraining neighbour heuristic, each across all Sudokus is plotted in Figure 2, Figure 3, Figure 4 and Figure 5 respectively, which can be found in Appendix B.

5 Discussion

When comparing the runtime measures of all heuristics to the runtime measures of the default 'heuristic', it appears the heuristics are in fact slowing the AC-3 algorithm down. In Figure 1, the median runtime measure of all heuristics lies above 6000 arc revisions, whereas the median runtime measure of the default lies just under 5000 arc revisions.

The problem lies in the initial queue: here every arc is added based on the id of a Field() instance or the id of it's constraining neighbour. The first value for every item in the queue is therefore an id. But inside the AC-3 algorithm the heuristic is now the first value or priority of every item added to the queue. The heapq now compares the heuristic to the first value of all other items which in most cases is the id of an field object. This results in a queue where all items with an heuristic have an unfair priority over the fields with an id as first value since heuristic $<$ id. Therefore, the algorithm must iterate over more arcs in order to get to a result. A solution might be to initialize the queue with heuristic values as priority already. But due to a constraint in time, it is not feasible to implement this and rerun the experiments.

When comparing the runtime measure across the five different Sudokus for every heuristic, a clear pattern arises: if the AC-3 algorithm cannot solve the Sudoku, it takes less revisions to come to that conclusion than it takes to come to the conclusion that the constraints can be satisfied.

Lastly, in the experiments all data-points fall in the 3000 to 7000 range of arc revisions. The theoretical . Therefore, the number of revised arcs seems to fall into the range of the theoretical complexity mentioned in Section 2.

5.1 Conclusion

In conclusion, since the data was gathered while using an incorrect implementation of the queue, the hypothesis cannot be rejected or accepted because the data is likely unreliable. Although the data contradicted the hypothesis, the complexity of the implemented algorithm remains within the theoretical complexity range of the AC-3 algorithm which means that this implementation is consistent with the theory. Lastly, the data does suggest that there is a slight decrease in the number of arc revisions if the AC-3 algorithm is unable to find a valid solution. (Figure 2-5)

A Data

Sudoku ID	Trial	Heuristic	Arc Revisions	Solved
1	1	Default	5055	Yes
	2	Default	5062	Yes
	3	Default	4932	Yes
	4	Default	5062	Yes
	5	Default	5123	Yes
	6	Default	5540	Yes
	7	Default	5062	Yes
	8	Default	5197	Yes
	9	Default	5129	Yes
	10	Default	5326	Yes
	1	Minimum Remaining Value	6415	Yes
	2	Minimum Remaining Value	6451	Yes
	3	Minimum Remaining Value	6423	Yes
	4	Minimum Remaining Value	6423	Yes
	5	Minimum Remaining Value	6342	Yes
	6	Minimum Remaining Value	6319	Yes
	7	Minimum Remaining Value	6157	Yes
	8	Minimum Remaining Value	6995	Yes
	9	Minimum Remaining Value	6668	Yes
	10	Minimum Remaining Value	5901	Yes
	1	Degree	6242	Yes
	2	Degree	6245	Yes
	3	Degree	6245	Yes
	4	Degree	6313	Yes
	5	Degree	6522	Yes
	6	Degree	6470	Yes
	7	Degree	6293	Yes
	8	Degree	6279	Yes
	9	Degree	6291	Yes
	10	Degree	6309	Yes
	1	Minimal Constraining Neighbour	6259	Yes
	2	Minimal Constraining Neighbour	6262	Yes
	3	Minimal Constraining Neighbour	6593	Yes
	4	Minimal Constraining Neighbour	6586	Yes
	5	Minimal Constraining Neighbour	6282	Yes
	6	Minimal Constraining Neighbour	6451	Yes
	7	Minimal Constraining Neighbour	6262	Yes
	8	Minimal Constraining Neighbour	6262	Yes
	9	Minimal Constraining Neighbour	6464	Yes
	10	Minimal Constraining Neighbour	6259	Yes
	1	Default	5287	Yes
	2	Default	5413	Yes
	3	Default	5513	Yes
	4	Default	5862	Yes
	5	Default	5413	Yes
	6	Default	5608	Yes

Sudoku ID	Trial	Heuristic	Arc Revisions	Solved
	7	Default	5115	Yes
	8	Default	5413	Yes
	9	Default	5452	Yes
	10	Default	5721	Yes
	1	Minimum Remaining Value	6343	Yes
	2	Minimum Remaining Value	6621	Yes
	3	Minimum Remaining Value	7059	Yes
	4	Minimum Remaining Value	6343	Yes
	5	Minimum Remaining Value	6672	Yes
	6	Minimum Remaining Value	6339	Yes
	7	Minimum Remaining Value	6610	Yes
	8	Minimum Remaining Value	6718	Yes
	9	Minimum Remaining Value	6552	Yes
	10	Minimum Remaining Value	6659	Yes
	1	Degree	6444	Yes
	2	Degree	6571	Yes
	3	Degree	6569	Yes
	4	Degree	6569	Yes
	5	Degree	6692	Yes
	6	Degree	6586	Yes
	7	Degree	6553	Yes
	8	Degree	6662	Yes
	9	Degree	6518	Yes
	10	Degree	6624	Yes
	1	Minimal Constraining Neighbour	6558	Yes
	2	Minimal Constraining Neighbour	6558	Yes
	3	Minimal Constraining Neighbour	6534	Yes
	4	Minimal Constraining Neighbour	6558	Yes
	5	Minimal Constraining Neighbour	6558	Yes
	6	Minimal Constraining Neighbour	6470	Yes
	7	Minimal Constraining Neighbour	6552	Yes
	8	Minimal Constraining Neighbour	6558	Yes
	9	Minimal Constraining Neighbour	6595	Yes
	10	Minimal Constraining Neighbour	6675	Yes
	1	Default	4736	No
	2	Default	4669	No
	3	Default	4808	No
	4	Default	4736	No
	5	Default	4760	No
	6	Default	4832	No
	7	Default	4736	No
	8	Default	4844	No
	9	Default	4792	No
	10	Default	4736	No
	1	Minimum Remaining Value	5755	No
	2	Minimum Remaining Value	5869	No
	3	Minimum Remaining Value	5805	No
	4	Minimum Remaining Value	5755	No

3

Sudoku ID	Trial	Heuristic	Arc Revisions	Solved
	5	Minimum Remaining Value	5867	No
	6	Minimum Remaining Value	5852	No
	7	Minimum Remaining Value	5755	No
	8	Minimum Remaining Value	5787	No
	9	Minimum Remaining Value	5825	No
	10	Minimum Remaining Value	5755	No
	1	Degree	5745	No
	2	Degree	5799	No
	3	Degree	5903	No
	4	Degree	5754	No
	5	Degree	5815	No
	6	Degree	5827	No
	7	Degree	5754	No
	8	Degree	5899	No
	9	Degree	5754	No
	10	Degree	5806	No
	1	Minimal Constraining Neighbour	5691	No
	2	Minimal Constraining Neighbour	5850	No
	3	Minimal Constraining Neighbour	5886	No
	4	Minimal Constraining Neighbour	5752	No
	5	Minimal Constraining Neighbour	5959	No
	6	Minimal Constraining Neighbour	5739	No
	7	Minimal Constraining Neighbour	5752	No
	8	Minimal Constraining Neighbour	5793	No
	9	Minimal Constraining Neighbour	5783	No
	10	Minimal Constraining Neighbour	5752	No
	1	Default	3564	No
	2	Default	3702	No
	3	Default	3633	No
	4	Default	3637	No
	5	Default	3605	No
	6	Default	3624	No
	7	Default	3633	No
	8	Default	3633	No
	9	Default	3633	No
	10	Default	3639	No
	1	Minimum Remaining Value	4540	No
	2	Minimum Remaining Value	4575	No
	3	Minimum Remaining Value	4536	No
	4	Minimum Remaining Value	4577	No
	5	Minimum Remaining Value	4571	No
	6	Minimum Remaining Value	4581	No
	7	Minimum Remaining Value	4545	No
	8	Minimum Remaining Value	4534	No
	9	Minimum Remaining Value	4559	No
4	10	Minimum Remaining Value	4542	No
	1	Degree	4557	No
	2	Degree	4536	No

Sudoku ID	Trial	Heuristic	Arc Revisions	Solved
	3	Degree	4570	No
	4	Degree	4536	No
	5	Degree	4545	No
	6	Degree	4536	No
	7	Degree	4536	No
	8	Degree	4550	No
	9	Degree	4529	No
	10	Degree	4529	No
	1	Minimal Constraining Neighbour	4558	No
	2	Minimal Constraining Neighbour	4537	No
	3	Minimal Constraining Neighbour	4548	No
	4	Minimal Constraining Neighbour	4562	No
	5	Minimal Constraining Neighbour	4536	No
	6	Minimal Constraining Neighbour	4544	No
	7	Minimal Constraining Neighbour	4530	No
	8	Minimal Constraining Neighbour	4536	No
	9	Minimal Constraining Neighbour	4549	No
	10	Minimal Constraining Neighbour	4550	No
	1	Default	5163	Yes
	2	Default	4930	Yes
	3	Default	5008	Yes
	4	Default	4884	Yes
	5	Default	5186	Yes
	6	Default	5007	Yes
	7	Default	5237	Yes
	8	Default	4750	Yes
	9	Default	5008	Yes
	10	Default	4912	Yes
	1	Minimum Remaining Value	6115	Yes
	2	Minimum Remaining Value	6115	Yes
	3	Minimum Remaining Value	6115	Yes
	4	Minimum Remaining Value	6115	Yes
	5	Minimum Remaining Value	6153	Yes
	6	Minimum Remaining Value	6119	Yes
	7	Minimum Remaining Value	6128	Yes
	8	Minimum Remaining Value	6057	Yes
	9	Minimum Remaining Value	6082	Yes
5	10	Minimum Remaining Value	6183	Yes
	1	Degree	6056	Yes
	2	Degree	6057	Yes
	3	Degree	6123	Yes
	4	Degree	6066	Yes
	5	Degree	5996	Yes
	6	Degree	6123	Yes
	7	Degree	6103	Yes
	8	Degree	6095	Yes
	9	Degree	6123	Yes
	10	Degree	6182	Yes

Sudoku ID	Trial	Heuristic	Arc Revisions	Solved
	1	Minimal Constraining Neighbour	6050	Yes
	2	Minimal Constraining Neighbour	6012	Yes
	3	Minimal Constraining Neighbour	6043	Yes
	4	Minimal Constraining Neighbour	6050	Yes
	5	Minimal Constraining Neighbour	5897	Yes
	6	Minimal Constraining Neighbour	6025	Yes
	7	Minimal Constraining Neighbour	6050	Yes
	8	Minimal Constraining Neighbour	6033	Yes
	9	Minimal Constraining Neighbour	6243	Yes
	10	Minimal Constraining Neighbour	6050	Yes

B Plots

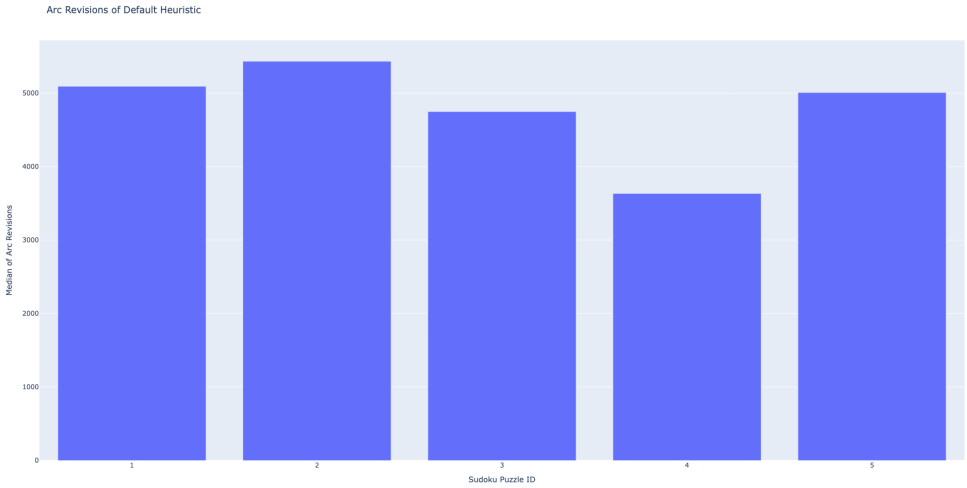

Figure 2

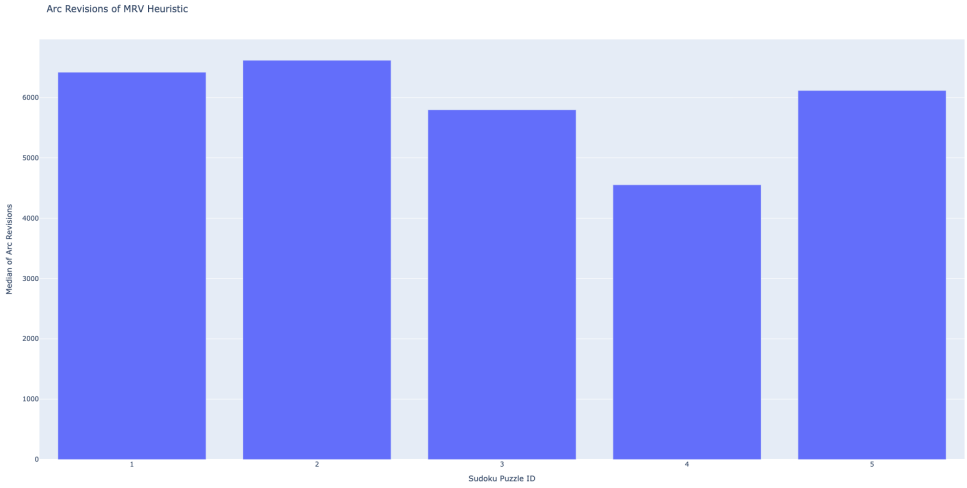

Figure 3

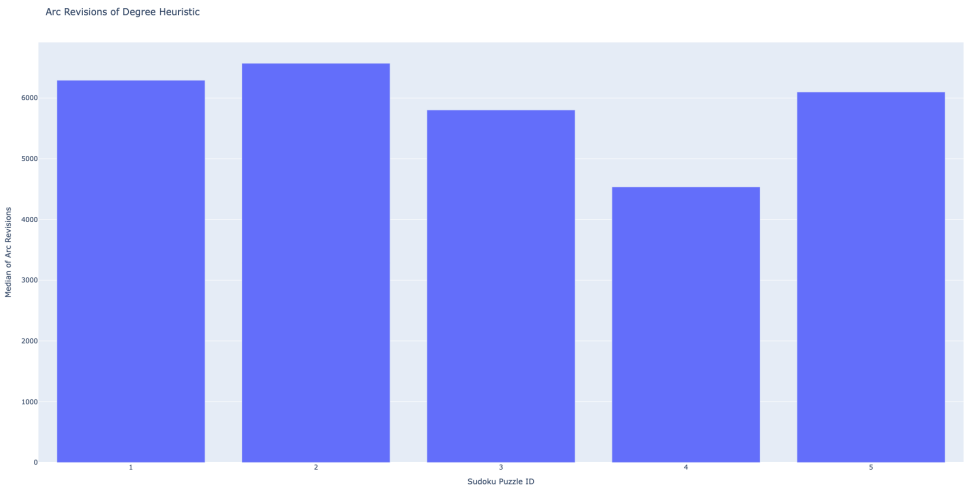

Figure 4

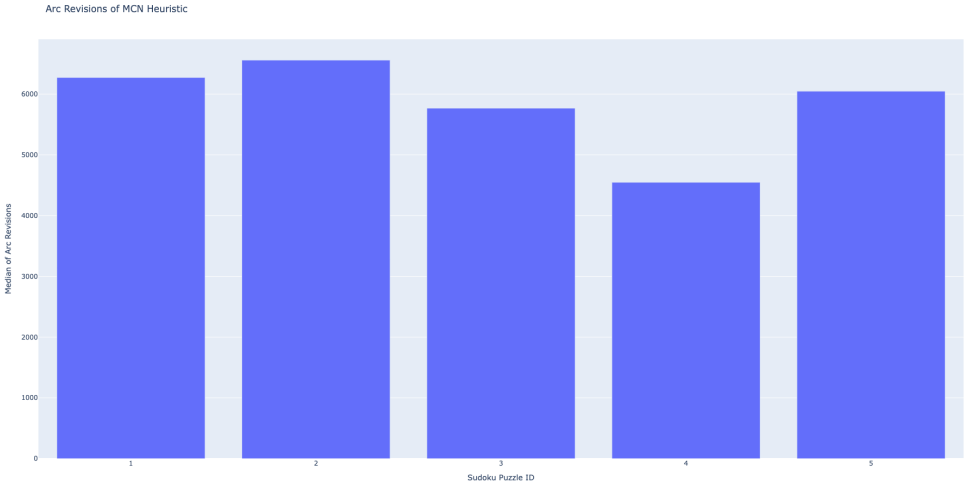

Figure 5