Randomised Analysis of Backtracking based Search Algorithms in Elucidating Sudoku Puzzles Using A Dual Serial/Parallel Approach

Pramika Garg[1][0000−0002−2867−3421], Avish Jha[2][0000−0003−1479−0456], and Amogh Shukla[3][0000−0003−0357−6649]

[1] SCOPE, Vellore Institute of Technology, Vellore TN 632014, India
micepram@gmail.com
[2] SCOPE, Vellore Institute of Technology, Vellore TN 632014, India
avish.j@protonmail.com
[3] SCOPE, Vellore Institute of Technology, Vellore TN 632014, India
amoghsh369@gmail.com

Abstract. Sudoku is a 9x9 grid-based puzzle. It is a game where each row, column, and 3x3 box must have one instance of a number from 1-9. In the present paper, we shall evaluate 3 different algorithmic approaches both in serial and parallel configurations that can be utilised to solve a puzzle of Sudoku to assess their comparative performance metrics for differential randomly generated Sudoku data sets. We shall utilise Breadth First Search, Depth First Search, Depth First Search with Breadth First Search parallelisation for sub tress, for evaluating a large number of randomly generated Sudoku puzzles with a varying number of clues to find the best algorithm based on time and space complexity as well as clue complexity. With this, we shall analyse and develop a best practice algorithm that can be ideally used to solve a large number of puzzles in any given situation in the most time-efficient manner. Our analysis has found that there was a significant improvement in utilising the parallel algorithm over both the Breadth First and Depth First search approaches from 28% to over 56%. Even moving from Breadth First to Depth First search we have gauged quite a moderate improvement in performance from 15% to 21%.

Keywords: sudoku · algorithms · breadth first search · bfs · backtracking · depth first search · dfs · efficient sudoku solving · parallel sudoku · randomly generated dataset · randomised analysis.

1 Introduction

Sudoku puzzle is a logic-based game that was first mentioned in the Number Place (1979) [1]. It was adopted by the Japanese as "Suuji Wa Dokushin Ni Kagiru", and later on, the name was shortened down to Sudoku in 2005, leading to widespread popularity. It consists of a grid that is divided into squares. It consists of varying sizes ranging from 4x4 grids to 16x16 grids.

1			2
		1	3
		2	

(a) A 4x4 Sudoku.

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

(b) A 9x9 Sudoku.

Fig. 1: Unsolved Sudoku's of Varying Sizes.

It can be generalised onto an NxN grid where the square root of N is a positive integer. However, the Sudoku containing 9x9 grids is considered to be conventional due to the large prevalence in its usage among people around the world. Unsystematic numbers are assigned to the board in prior, where the ultimate goal is to fill off the remaining empty spaces. There must be a one-of-a-kind solution, with no repetitions permitted. Certain sets of rules and regulations are required to be followed by every player which includes the following (for a 9x9 grid):

- Only numbers containing integer values from 1-9 can be used.
- All the nine squares must contain different numbers each within a larger 3x3 box.
- Each column or row should also contain all the numbers exactly once.
- All the squares must be filled.
- All the numbers should only be from 1-9.

The boxes are a set of nine 3x3 smaller grids that form the bigger grid. The main principle of solving the game involves solving it by unraveling the squares with values without breaking any of the rules of the game. The initial clues given in the puzzle allow the game to have a start point. A Sudoku puzzle with 17 or more clues (clues imply filled in squares) will have a unique solution. [2]

2 Related Works

The Sudoku is an NP complete problem which allows for quite a diverse variety of solutions. We have analysed multiple approaches used previously in the upcoming text.

In [3], the authors use a stochastic search based algorithm that works with underlying meta-heuristic techniques. They also utilise an simulated annealing

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

(a) Unsolved Sudoku

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

(b) Solved Sudoku

Fig. 2: 9x9 Sudoku - Problem & Solution

based algorithm which uses a set of Markov chains. They have also proved the existence of easy-hard-easy phase transitions in Sudoku puzzles of larger grid sizes such as 16x16 & 25x25.

The authors in [4] propose a solution based on Constraint Programming, treating the Sudoku as a Constraint Satisfaction Problem. They focus on both the Enumeration Strategies of Variable Selection Heuristics as well as Value Selection Heuristics, recognising their impact on performance. In Variable Selection Heuristics, they cover both the Static & Dynamic Selection Heuristics Parameters, and in Value Selection Heuristics, they cover Smaller, Greater, Average and just Greater than Average Value of Domain to gauge the best performing parameters. Even in [5], the author Simonis, H. uses a Constraint Programming based approach implements versions of "Swordfish" and "X-Wing" which use complex propagation schemes in a more redundant nature with well defined constraints such as colored matrix and defined cardinality. This leads to a stronger model which they improve on further by using flow algorithms with bipartite matching.

Using State Space Search is another method of implementing a Sudoku solver as [6] explores using both Breadth First Search and Depth First Search. It finds that Depth First Search is more optimal for blind searching and it can be implemented using a backtracking based approach. They compared a brute force approach using both Breadth First Search and Depth First Search. They also implemented both algorithms using a tree pruning technique which essentially is backtracking.

Backtracking is an algorithmic approach to solve a Constraint Satisfaction Problem which was approached as a Depth First Traversal of a Search Tree using varied Branching Techniques as described in detail in [7]. In [8], both the authors have explored on the different forms of backtracking based search algorithm and compared it against other approaches such as rule-set or Boltzmann Machine. Hence, there has been research in the utilisation of these search algorithms but all of these have been serial approaches, there has been none in a

parallel methodology. This is what our research aims to fill the gap in since parallel processing is a major optimization as thread counts on Central Processing Units keep on increasing.

3 Methodology

3.1 Challenges Involving Bias

The challenges will involve minimising the time as well as space complexity, due to the nature of the Sudoku as well as the algorithms in question [9]. Sudoku is well known to be an NP-Complete problem and backtracking-based heuristics algorithms such as Backtracking based Depth First Search (DFS) & Breadth First Search (BFS) provide an excellent methodology to prepare an efficient solution [10].

In such a problem where we have a given Sudoku problem, and we need to assess and determine a unique solution (we will analyse Sudoku puzzles with 17 or more given clues in the present paper), we have to use search algorithms to perform a potential solution on the puzzle set. A random set of 1000 Sudoku puzzles shall be generated via a process that first involves generating a full Sudoku solution and then removing numbers as necessary to create a problem with a prior specified number of clues. This process will be repeated 1000 times to generate a random data-set of Sudoku puzzles with X number of clues, where X may be an integer from 25 to 30.

3.2 Solution to the Challenges

This will ensure that we have a large set ensuring any bias if at all can be eliminated via the large size of the trial and will allow the proper comparative analysis of the four algorithms in both their serial and parallel implementations. For each X from 25 to 30 the data-set shall be tested to all the algorithms and the analysis will be done in such a way as to see if any algorithm outpaces another based on the number of clues given to the algorithm at the start. This will allow in the future an algorithm to be developed that utilises the best algorithm amongst these for each specific situation based on the input size, ensuring an efficient solution can be employed. This may lead one to believe that the more the number of clues, the easier it is to solve, but it is far from the truth, as it is highly situational. A puzzle with 25 clues may indeed be harder to solve than one with 17. [11]

3.3 Implementation

In the proposed algorithm we shall employ the Backtracking based Breadth First Search Algorithm in a serial configuration and we will analyse the performance of the algorithm over a random data-set of size 1000 with varying X (25-30). In a similar manner, the same process will be repeated for the Backtracking based

Depth First Search to determine the best possible configuration that works for each test case (where a test case is defined is when X has a specific value from 25 to 30). Then the Depth First Search with a parallel sub-tree Breadth First Search algorithm shall be used to determine and analyse the improvement that is brought by utilising a parallel core architecture compared to a serialised solution. We have utilised a graph data structure based on Constraint Graph structure for the final code.

3.4 Specification of Test Platform

For conducting a test with no other factors playing a role in the results, we decided to approach this with a brand new fresh copy of Windows 10 LTSC, Python 3.8. The code was entirely written in python and it was set to utilise a single virtual thread for both the Serial Breadth First Search and Depth First Search implementation, whereas the parallel algorithm utilised 6 virtual threads. Each test ran within its virtual environment, with identical system utilisations. Each test was run at an interval of 1 hour before the previous to allow for heat dispensation on the Intel 8th Generation CPU. Each test ran including the generation of 1000 different random Sudoku puzzles of the given clue number.

4 Serial Breadth First Search

4.1 What is it?

Breadth First Search is a crucial search algorithm consisting of data structures for graphical representation. It is heavily utilised for traversing graph or tree-based structures. It is an uninformed graph search strategy that moves from level to level ensuring that each level is fully searched before moving on to the next deeper level. [12] It traverses a graph structure in a breath-ward direction starting from the first vertex and follows the below-mentioned rule-set strictly.

In the current situation where we implement Breadth First Search in Sudoku, we will take the first position on the Top-Left as the initial or first vertex. Each square is represented as a vertex with 2 possible states, visited and unvisited. A queue is then used by the algorithm as it executes, as mentioned below.

4.2 Algorithm

1. From the current vertex, find all the adjacent unvisited vertexes.
2. Visit each vertex as found from Step 1, and add it to a queue.
3. If the current vertex has no adjacent vertex, then dequeue the first vertex from the queue.
4. Repeat the first three steps till all the vertices have been visited.

4.3 Results

The algorithm is implemented serially utilising the above rule-set and the following are the results after 1000 randomly generated Sudoku are processed for each of the initial numbers of clues from 25 to 30.

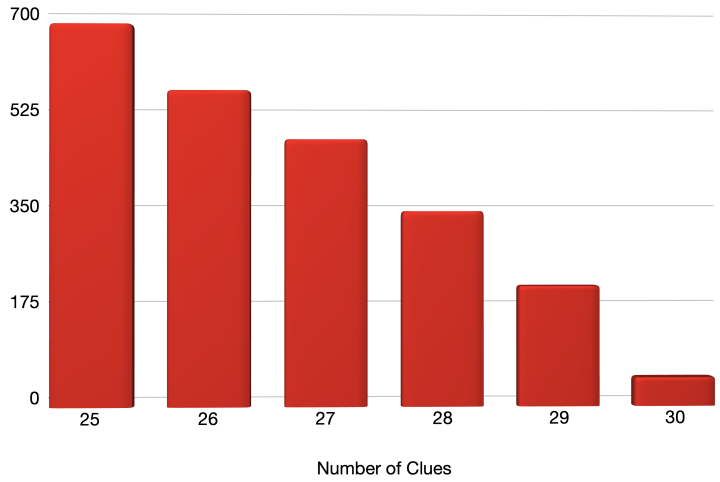

Fig. 3: BFS - Average Mean Time of 100 Random Puzzles (ms) in Y — Number of Clues in X

5 Serial Depth First Search

5.1 What is it?

Depth First Search (also known as Depth First Traversal) is a graph-cum-tree traversal and search algorithm that works on the Artificial Intelligence algorithmic approach of Backtracking [13]. The algorithm starts from a root node and moves down one of the branches fully till it reaches the end, if the answer is found along the way, then it stops, else it utilises the backtracking methodology. It continues this till the answer is found, with the eventual goal of visiting all the unvisited vertices. The backtracking methodology can be implemented via a recursive approach to allow for better time and space complexity [14].

In the current implementation of Depth First Search to port it to Sudoku, we will take the first position on the Top-Left as the initial or first vertex. Each square is represented as a vertex in the algorithm with 2 possible states, visited and unvisited. A queue is then utilised by the DFS algorithm while it executes, as mentioned below.

5.2 Algorithm

1. Take the first position in the Sudoku and put it as the starting vertex and place it onto a stack.
2. Mark the vertex on the top of the stack as visited.
3. Find all the adjacent vertices from the current vertex and store them in a list.
4. Add the unvisited ones from the list to the stack.
5. Repeat Steps 2 to 4 till all vertices are visited.

5.3 Results

The algorithm is created in a serial configuration utilising the above rule-set and below mentioned are the results after 1000 randomly generated Sudoku are processed for each of the initial numbers of clues from 25 to 30.

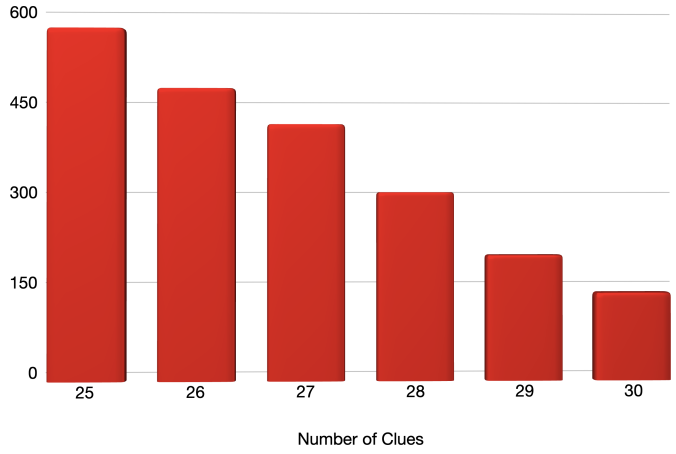

Fig. 4: DFS - Average Mean Time of 100 Random Puzzles (ms) in Y — Number of Clues in X

6 Parallel Dual DFS & BFS based Algorithm

6.1 What is it?

Parallel Depth First Search with sub-tree Breadth First Search is an approach that involves setting a search depth till which a serial Depth First Search will be run and after which a number of Breadth First Search threads shall be spawned which will each have a sub-queue as well as access to a global queue to prevent multiple copies of the Sudoku needing to be creating which would require much more memory leading to higher space complexity. The Breadth First Search threads run in parallel till the unique solution for the Sudoku is found.

6.2 Algorithm

1. A search depth is chosen.
2. Initially Depth First Search is run till the chosen depth serially with a similar rule-set as formerly given.
3. From the search depth, a set of parallel threads are spawned which run at the same time with a global queue to solve till the end of the Sudoku is reached.
4. The Breadth First Search nodes run in parallel utilising sub-queues and the global queue.

6.3 Results

The algorithm is created in a parallel configuration utilising the above rule-set so that till the pre-defined search depth, Depth First Search is run serially after which it spawns multiple Breadth First Search nodes running in parallel for the rest of the depth of the Sudoku and below mentioned are the results after 1000 randomly generated Sudoku are processed for each of the initial numbers of clues from 25 to 30.

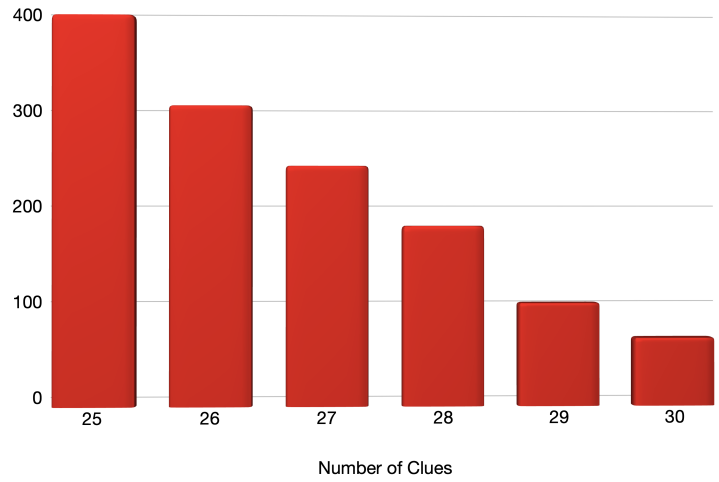

Fig. 5: Parallel Mixed DFS & BFS - Average Mean Time of 100 Random Puzzles (ms) in Y — Number of Clues in X

7 Visualisations & Analysis

Utilising the large mirage of data we collected from our testing, we can have 2 different outlooks, one based on the individual clues for each algorithm, and an overall analysis where we average out the performance across different numbers of clues.

7.1 25 Clues

When we take the initial number of clues to be 25, it can be analysed that moving from Serial Breadth First Search to Serial Depth First Search we see an improvement of over 15.86826%. Now, if we compare the Parallel Dual Depth First Search with Breadth First Search with the serial Breadth First Search algorithm we mark an improvement on 41.16766% which is quite significant. Moving from Serial Depth First Search to the Parallel algorithm we see an improvement of about 28.38926%.

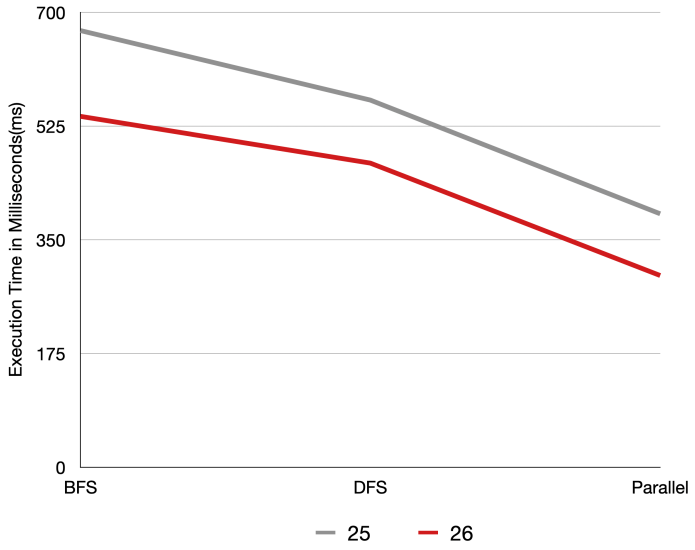

Fig. 6: Comparing the performance of 25 vs 26 Initial Clues

7.2 26 Clues

When the number of clues is taken to be 26, observations state an increase of 15.12915%, while moving from Serial Breadth First Search to Serial Depth First Search. In addition to that, the Parallel version of Depth First Search and Breadth First Search shows a notable increase of 44.83394% over Serial Breadth First Search, which concludes that the performance of the Paralleled implementation of Depth First Search and Breadth First Search is higher than the serial implementation of Breadth First Search. When comparing the Parallel Dual Depth First Search with Breadth First Search method to the Serial Depth First Search algorithm, enhancement of 35.15724% is beheld.

7.3 27 Clues

Analysing the bar plot for 27 clues, we see an improvement of about 18.73922% continuing the pattern of Serial Depth First Search being more efficient than Serial Breadth First Search. In comparison to the previous analysis, the drop from Serial Depth First Search to the Parallel algorithm is on the higher end at 51.34924%, while the gain in performance moving from the Serial Depth First Search to the Parallel algorithm stays around 38.47207%.

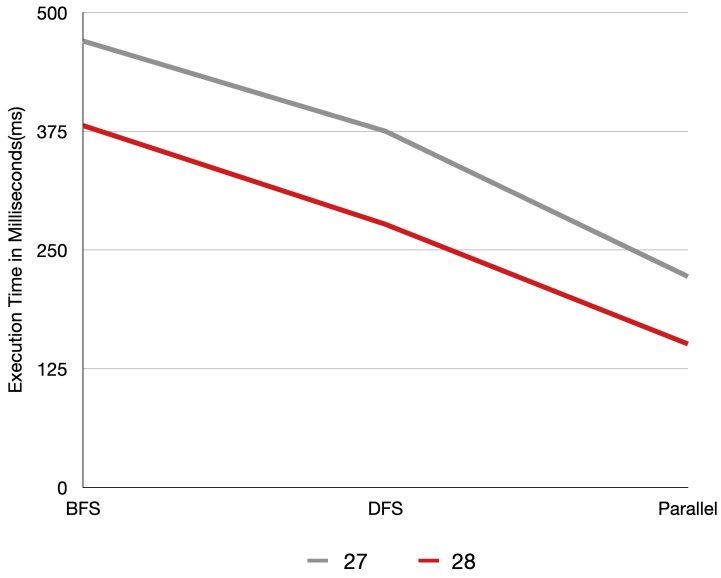

Fig. 7: Comparing the performance of 27 vs 27 Initial Clues

7.4 28 Clues

In the case of 28 clues, we see a gain in performance of approximately 21.47925% moving from Serial Breadth First Search to Serial Depth First Search. If we switch from the Serial Breadth First Search algorithm to the Dual Parallel algorithm we see a gain of over 52.64389% in performance. Whereas, moving from Serial Depth First Search to the Dual Parallel algorithm we see an improvement of 39.82891%.

7.5 29 Clues

Starting off with 29 clues, we are able to analyse that moving from Serial Breadth First Search to Serial Depth First Search we get an improvement of about 17.57393%, but there is a major improvement with over the doubling of speed

moving from Serial Breadth First Search to the Dual Parallel algorithm with over 53.73929% improvement. The movement from Serial Depth First Search to the Dual Parallel algorithm is again a pretty sharp increase in performance by about 41.32907% which is anomalously higher than the previous cases.

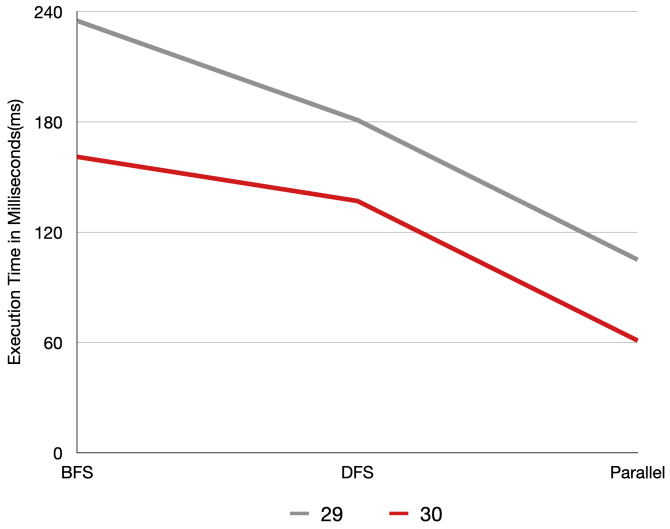

Fig. 8: Comparing the performance of 29 vs 30 Initial Clues

7.6 30 Clues

In this section, we analyse the performance where 30 clues are given initially, and we see an improvement of 19.28571% jumping from Serial Breadth First Search to Serial Depth First Search. Moving from the Serial Breadth First Search to the Parallel algorithm, we see a large improvement of over 56.38127% which confirms that this parallel algorithm continues to become more and more efficient as the number of clues is increased. The jump from Serial Depth First Search to the Parallel algorithm is quite on the higher end at 35.82937% suggesting that the Parallel algorithm starts to gain an advantage over the Serial Depth First Search algorithm with an increasing number of clues.

8 Conclusion

With the Averaging our the values as shown in Figures 3, 4 and 5, we can see that there is an improvement of approximately 20.79027% as we utilise Backtracking based Depth First Search over Breadth First Search which can be attributed to the benefits that backtracking brings to Depth First Search and hence it can be

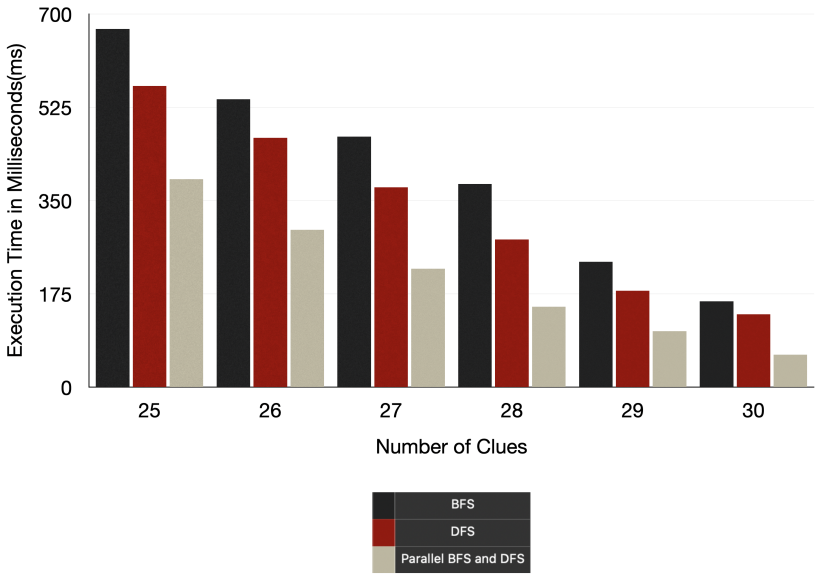

Fig. 9: Overall Summary of Execution Time

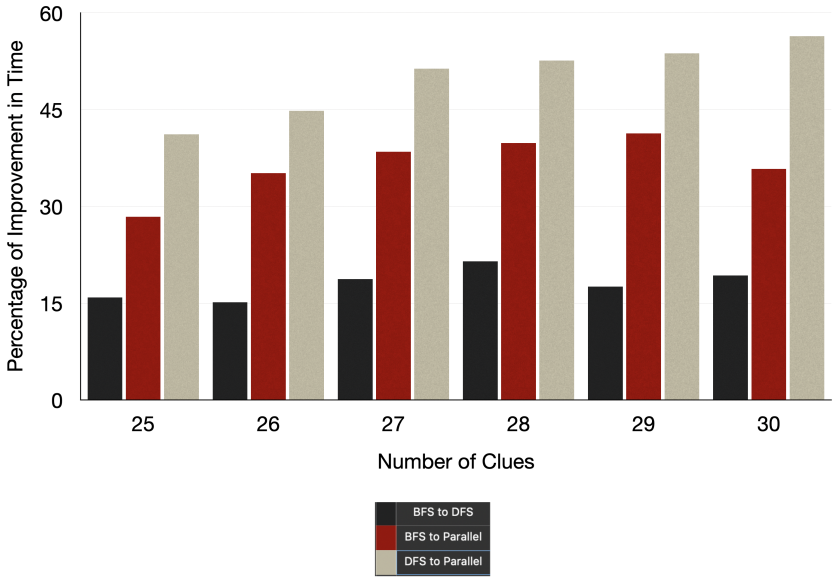

Fig. 10: Comparison of Percentage of Improvement

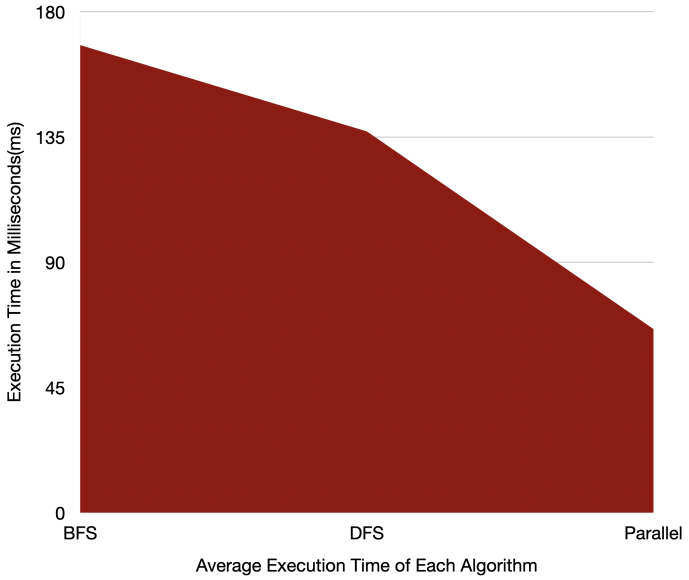

Fig. 11: Average Execution Time of the 3 Algorithms

determined that a serial implementation of Depth First Search will uniformly perform better than a serialised implementation of Breadth First Search.

Percentage Improvement			
Number of Clues	BFS to DFS	BFS to Parallel	DFS to Parallel
25	15.86826%	41.16766%	28.38926%
26	15.12915%	44.83394%	35.15724%
27	18.73922%	51.34924%	38.47207%
28	21.47925%	52.64389%	39.82891%
29	17.57393%	53.73929%	41.32907%
30	19.28571%	56.38127%	35.82937%

Table 1: This table shows the percentage gain as we move from Algorithm A to Algorithm B for all 3 possible permutations.

From the previous set of analysis at individual clue levels, we can notice that the performance gap between Serial Breadth First Search and Serial Depth First Search is approximately 20% constantly irrespective of the change in the initial number of clues. The major improvement occurs in the switch from Serial Depth First Search to the Dual Parallel Depth First Search with Breadth First Search

algorithm which ensues a constant performance gain with the increase in the number of clues, from about 28% to over 35%.

The Parallel version of Depth First Search utilising Breadth First Search for its sub-trees is a major improvement over serial Breadth First Search and even over serial Depth First Search. It manages to uphold a humongous 39.35643% over the Serial Breadth First Search Implementation. Though, it is to be noted, that this was with 6 parallel Breadth First Search threads that were launched after the pre-defined search depth has been reached. Even though utilising Breadth First Search partially in this algorithm would be expected to offer a slowdown compared to the backtracking based highly performant serial Depth First Search, the threaded Breadth First Search absorbs the complexity increase due to parallelisation and it still comes out ahead majorly of the serial Depth First Search with a 23.43783% improvement.

Overall we can see there is an immense improvement in the real world performance of our algorithm even though the time complexity is rather unchanged from that of Serial Depth First Search & Serial Breadth First Search. The novelty of our approach is adding the concept of parallelisation on not just a single algorithm rather combining 2 algorithms in usch a way that not only do we reap the benfits of parallelisation, we also get the best from both the algorithms. Hence, we can conclude that such a parallel approach is the best algorithm amongst the studied to solve a given Sudoku puzzle.

References

1. Garns, H.: Number place. Dell Pencil Puzzles & Word Games (1979)
2. Delahaye, J. P.: The science behind Sudoku. Scientific American, 294(6), 80-87, (2006).
3. Lewis, R.: Metaheuristics can solve sudoku puzzles. Journal of heuristics, 13(4), 387-401, (2007).
4. Crawford, B., Aranda, M., Castro, C., & Monfroy, E.: Using constraint programming to solve sudoku puzzles. In 2008 Third International Conference on Convergence and Hybrid Information Technology, Vol. 2, pp. 926-931. IEEE, (2008).
5. Simonis, H.: "Sudoku as a constraint problem." CP Workshop on modeling and reformulating Constraint Satisfaction Problems. Vol. 12. Citeseer, (2005).
6. Eremic, M., Adamov, R., Bilinac, N., Ognjenovic, V., Brtka, V., & Berkovic, I.: COMPARISON OF STATE SPACE SEARCH ALGORITHMS-SUDOKU PUZZLE GAME. Chief and responsible editor, 154, (2013).
7. Van Beek, P.: "Backtracking search algorithms." Foundations of artificial intelligence. Vol. 2. Elsevier, 85-134, (2006).
8. Berggren, P., & Nilsson, D.: A study of Sudoku solving algorithms. Royal Institute of Technology, Stockholm, (2012).
9. Pathak, M. J., Patel, R. L., & Rami, S. P.: Comparative Analysis of Search Algorithms. International Journal of Computer Applications, 179(50), 40-43, (2018).
10. Yato, T., & Seta, T.: Complexity and completeness of finding another solution and its application to puzzles. IEICE transactions on fundamentals of electronics, communications and computer sciences, 86(5), 1052-1060, (2003).
11. A study of Sudoku solving algorithms, https://cutt.ly/zb2GfyN. Last accessed 18 May 2021

12. Bundy, A., & Wallen, L.: Breadth-first search. In Catalogue of artificial intelligence tools (pp. 13-13). Springer, Berlin, Heidelberg, (1984)
13. Chen, L., Guo, Y., Wang, S., Xiao, S., & Kask, K.: Sudoku Solver, (2016).
14. Stone, H. S., & Sipala, P.: The average complexity of depth-first search with backtracking and cutoff. IBM Journal of Research and Development, 30(3), 242-258, (1986).
15. Multi-threaded algorithm for solving Sudoku, https://stackoverflow.com/a/850892. Last accessed 18 May 2021.