Solving Sudoku Puzzles with Wisdom of Artificial Crowds

Ryan Hughes
Speed School of Engineering
University of Louisville
Louisville, USA
rmhugh03@louisville.edu

Roman V. Yampolskiy
Speed School of Engineering
University of Louisville
Louisville, USA
roman.yampolskiy@louisville.edu

***Abstract*— A genetic algorithm for solving Sudoku using a wisdom of crowds heuristic is presented. This paper compares the performance of a genetic algorithm for solving Sudoku puzzles against a genetic algorithm which makes use of the wisdom of crowds heuristic. The idea behind the wisdom of crowds heuristic is that multiple people working together can produce a better solution than they would working separately. In this implementation that means that with each generation, we choose a certain percentage of the fittest individuals from the population and deem them experts (they're the best of what population has to offer). The solutions from these experts are then combined in a way that offers up one solution that is an aggregation of their solutions (one which often has a better fitness than any of the individual expert solutions that went into making it).**

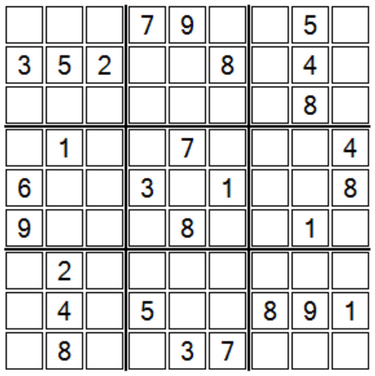
***Keywords*- Wisdom of Artificial Crowds; Sudoku**

I. Introduction

Genetic algorithms are algorithms that attempt to mimic the concept of natural selection found in nature [1]. The algorithm treats possible solutions to a problem as chromosomes in a genetic pool. Chromosome pairs will be mixed and matched (and mutated), and of those, the most fit will see their offspring make it to the next generation; while the most unfit will not (on average). The benefits of genetic algorithms are that they provide close estimations, or exact solutions, to intractable problems. They are topic agnostic, in the sense that the underlying architecture can easily be altered for various problem types, and the algorithm requires very little in the way of specifics about a problem space [2-3].

Sudoku is a puzzle game that has become incredibly popular within the last decade. There are entire books of Sudoku puzzles, and many newspapers feature a Sudoku puzzle in their puzzles section. Like many of the best games and puzzles, Sudoku is easy to learn, but hard to master. It has been calculated that there are 6,670,903,752,021,072,936,960 valid potential starting boards (for a 9x9 board) [4], so we are in no danger of running out anytime soon.

The starting state is an $n^2 x n^2$ (most commonly 9x9 – $3^2 x 3^2$) board where some values have been placed (we say they are "given"), and the rest of the values are blank.



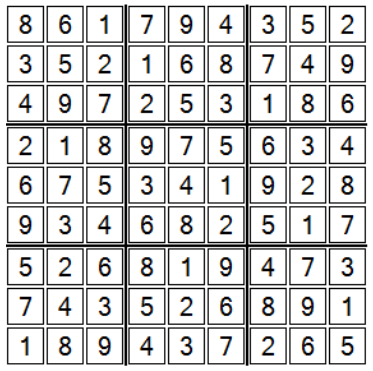
			7	9			5	
3	5	2			8		4	
							8	
	1			7				4
6			3		1			8
9				8			1	
	2							
	4		5			8	9	1
	8			3	7			

Figure 1: Sample starting state of a Sudoku board [5].

The object of Sudoku is to fill out each puzzle in such a way that the following three criteria are met:

1. Each row contains each of the integers 1 through 9 exactly once.
2. Each column contains each of the integers 1 through 9 exactly once.
3. Each 3x3 sub-grid (outlined by the double lines above) contains each of the integers 1 through 9 exactly once [6].

Typically, Sudoku starting states are designed so that they produce only one solution [7]. Figure 2 shows the solution for the initial board example shown in Figure 1.



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

Figure 2: Solution of the sample Sudoku board shown in Figure 1 [5].

For easier boards, logic and a process of elimination are typically enough to solve the puzzle. For some of the harder boards, an element of guess and check often comes into play.

Though Sudoku is most commonly played on a 9x9 board, it can actually be played on any n^2xn^2 board [8]. Solving an n^2xn^2 Sudoku puzzle is an NP-Complete problem [9]. An NP-Complete problem is a problem whose time complexity increases very quickly (exponentially) in relation to the problem size, but if given a solution, it can be checked for correctness efficiently [10]. Since solving arbitrarily large boards can quickly become intractable, this paper proposes that a genetic algorithm approach be taken to solving Sudoku. The belief is that the use of a genetic algorithm in combination with the wisdom of crowds heuristic can stand a good chance of solving many Sudoku puzzles.

The idea behind the wisdom of crowds is that multiple people (chromosomes) working together can produce a better solution than they would working separately. In this implementation that means that with each generation, we choose a certain percentage of the fittest individuals from the population and deem them experts (they're the best of what our population has to offer). We then take this group of experts and combine their solutions in a way that offers up one solution that is an aggregation of their solutions (and one which often has a better fitness than any of the individual expert solutions that went into making it).

II. Overview of Proposed Algorithm

A. *Initialization*

The algorithm creates an initial population by reading the input file containing the given positions and placing those given positions into every board in the population. It then randomly generates values, on a row by row basis for each board, such that each row is a valid (non-repeating) permutation.

B. *Fitness*

A fitness function is a crucial aspect of a genetic algorithm because it is what defines the worth of each chromosome [11]. For a Sudoku board, the fitness is a reflection of the number of conflicts in the board. A conflict is defined as a value missing from a row, column, or sub-grid (can also be defined as anytime there is a duplicate value in a row, column, or sub-grid). The more conflicts a board has, the worse its fitness.

The various chromosomes (boards) in a generation were put into ascending order, and the fitness score was given as (total population of the generation - i), where i started at 0, and went up to the total population of the generation – 1.

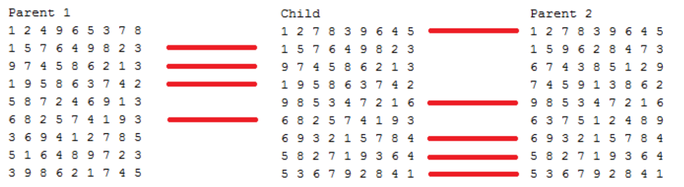
C. *Selection*

For this algorithm, a roulette wheel selection process was chosen. A roulette wheel allows us to select parents (chromosomes) for mating based on a probability, where the better scoring chromosomes have a higher chance of being selected [12]. As an example, let's say we have 5 chromosomes with fitness scores of 1, 2, 3, 4, and 5. The total fitness of the generation is 15 (1+2+3+4+5). The chromosome with a score of 5 is given a 5/15 chance of being selected, the chromosome with a score of 4 is given a 4/15 chance of being selected, etc. We select two parents for the creation of each child and use the roulette wheel to weight the randomization in favor of the more fit chromosomes.

D. *Crossover*

A crossover function takes aspects from each parent and uses them to create a child [13]. In this implementation, the algorithm chooses between two crossover functions (with a 50% chance of choosing each). Both of the functions use rows as the crossover information, which allows for the permutations in each row to hold (even when merging two parents to make a child).

The first of these crossover functions randomly chooses a parent (from the two parents chosen in the selection process) and takes the first row from the chosen parent and places it in the child. It then randomly chooses one of the parents and takes the second row from the chosen parent, etc. Figure 3 shows an example crossover of this type, where the lines indicate which parent provided the row to the child.

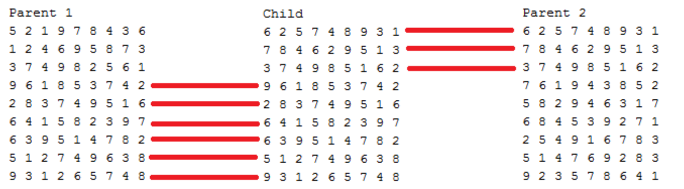


```
Parent 1                    Child                       Parent 2
1 2 4 9 6 5 3 7 8           1 2 7 8 3 9 6 4 5           1 2 7 8 3 9 6 4 5
1 5 7 6 4 9 8 2 3           1 5 7 6 4 9 8 2 3           1 5 9 6 2 8 4 7 3
9 7 4 5 8 6 2 1 3           9 7 4 5 8 6 2 1 3           6 7 4 3 8 5 1 2 9
1 9 5 8 6 3 7 4 2           1 9 5 8 6 3 7 4 2           7 4 5 9 1 3 8 6 2
5 8 7 2 4 6 9 1 3           9 8 5 3 4 7 2 1 6           9 8 5 3 4 7 2 1 6
6 8 2 5 7 4 1 9 3           6 8 2 5 7 4 1 9 3           6 3 7 5 1 2 4 8 9
3 6 9 4 1 2 7 8 5           6 9 3 2 1 5 7 8 4           6 9 3 2 1 5 7 8 4
5 1 6 4 8 9 7 2 3           5 8 2 7 1 9 3 6 4           5 8 2 7 1 9 3 6 4
3 9 8 6 2 1 7 4 5           5 3 6 7 9 2 8 4 1           5 3 6 7 9 2 8 4 1
```

Figure 3: One-row crossover.

This crossover function works well at preserving the row information while allowing for a large variety in the possible children two parents could make. However, we stand a high chance of losing any valued information we may have gathered in the sub-grids.

The second crossover function acts similarly to the first, except this function views three row segments as the smallest value that can be passed on from parent to child. This allows sub-grid retention from boards that provided valuable (low-conflict) sub-grids.



```
Parent 1                    Child                       Parent 2
5 2 1 9 7 8 4 3 6           6 2 5 7 4 8 9 3 1           6 2 5 7 4 8 9 3 1
1 2 4 6 9 5 8 7 3           7 8 4 6 2 9 5 1 3           7 8 4 6 2 9 5 1 3
3 7 4 9 8 2 5 6 1           3 7 4 9 8 5 1 6 2           3 7 4 9 8 5 1 6 2
9 6 1 8 5 3 7 4 2           9 6 1 8 5 3 7 4 2           7 6 1 9 4 3 8 5 2
2 8 3 7 4 9 5 1 6           2 8 3 7 4 9 5 1 6           5 8 2 9 4 6 3 1 7
6 4 1 5 8 2 3 9 7           6 4 1 5 8 2 3 9 7           6 8 4 5 3 9 2 7 1
6 3 9 5 1 4 7 8 2           6 3 9 5 1 4 7 8 2           2 5 4 9 1 6 7 8 3
5 1 2 7 4 9 6 3 8           5 1 2 7 4 9 6 3 8           5 1 4 7 6 9 2 8 3
9 3 1 2 6 5 7 4 8           9 3 1 2 6 5 7 4 8           9 2 3 5 7 8 6 4 1
```

Figure 4: Three-row crossover.

This test data shown in Figure 4 was taken from an early generation, but as more generations are run and the boards have a better average fitness, the ability to preserve sub-grid information becomes more valuable. The goal of having these two crossover functions is that the one-row crossover will provide variation, but as a run goes on, the three-row crossover will aid in preserving low conflict sub-grids.

E. *Mutation*

A mutation randomly alters part of a child's chromosome. The chance of mutation is based on the mutation rate; so if the mutation rate is 5%, then there is a 5 in 100 chance that the child in question will be chosen to be mutated (on average). Mutation is an incredibly valuable part of any genetic algorithm, but it seems to be of even greater importance when attempting to solve Sudoku puzzles using genetic algorithms. The reason for the heightened importance is that the crossover function allows for variations of rows (which allows for varied columns and sub-grids), but it does not provide variation within the rows themselves. Mutations are critical, as they are what continually provide variety in the rows [14].

The mutation method that seemed to work best was one in which a randomly selected row was randomly re-sorted (with the exception of the given values, those are never changed). This seemed to provide a large amount of variety, by both randomizing the entire row and performing the mutation on a random row each time.

F. *Wisdom of Crowds*

The purpose of the wisdom of crowds heuristic is to aggregate the best solutions from each generation to try and make an even better solution [15]. Through trial and error the value of 5% was decided on as the percentage of each population, with the best fitness, to be considered experts. There is a balance that needs to be fine-tuned for each problem type attempted when implementing the wisdom of crowds heuristic. If the percentage of experts is too low then the aggregation function will not get enough input to be valuable, but make the percentage too high and the results will be watered down, and possibly compromised by poor solutions.

The algorithm to combine the expert solutions into one solution consists of three steps. For the sake of explanation, row 1 is used as an example (but this algorithm is run for all rows in the board, and the process is repeated for each generation):

Step 1: To create an aggregated solution, the first step is to determine what the various expert solutions have in common. To do this, we take the pool of experts and create a list that contains the row permutations the experts used for row 1. From that, the count for how many times each row permutation was used is determined. A high count for a row permutation means that many experts all have the same row permutation for row 1.

Step 2: Determine which row-permutation was used the most frequently for row 1 and place that row in the aggregated solution as row 1.

Step 3: Sometimes the experts won't agree on a row permutation. When this happens, we generate a random row and insert that into the aggregated solution.

This approach was chosen over breaking down the board into individual cells and counting the frequency of the values in each cell because there is a high chance that with that approach we would end up with heavily duplicated rows, which we want to avoid.

G. *Stopping Criteria*

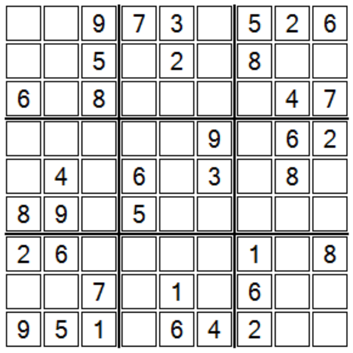
The stopping criteria for this algorithm are when a solution has been found, or when the current generation's conflict count is equal to, or worse than, the conflict count from 249 generations ago. On average, the algorithm takes only 87.81 generations to either solve the Sudoku puzzle, or to reach a low conflict count from which it is unlikely to improve (local minima). The check for generations without progress is set to 249 because that provided plenty of clearance over the average number of generations required, while also encompassing many of the statistical outliers.

Often times when performing test runs of the algorithm the process would get stuck at some low conflict count but be unable to get itself out of the groove and continue improving. To help alleviate this issue a shake-up function was implemented. The goal of the shake-up function is to try and get the algorithm out of local minima. If the algorithm hasn't made progress for 124 generations we "shake-up" the population by completely replacing the poorest performing half of the population with randomized boards.

III. RESULTS

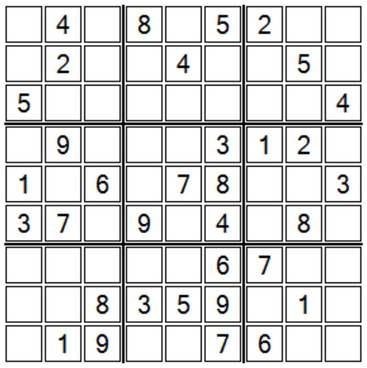
A. *Test Data*

Sudoku puzzles are often divided into the difficulty categories of very easy, easy, medium, hard, and very hard (which roughly correlate to how many values are "given"). Five boards were used in the testing and analysis of the algorithm, one from each difficulty level.



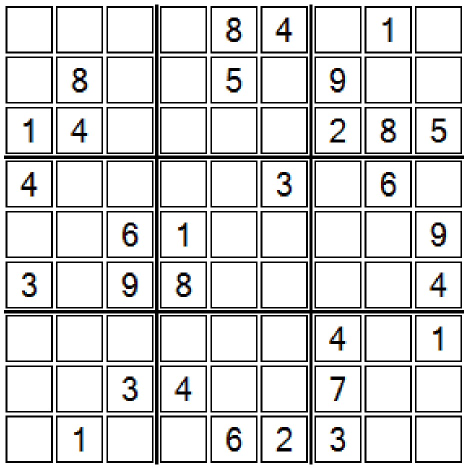
		9	7	3		5	2	6
		5		2		8		
6		8					4	7
					9		6	2
	4		6		3		8	
8	9		5					
2	6					1		8
		7		1		6		
9	5	1		6	4	2		

Figure 5: Very easy difficulty Sudoku board used for testing (36 givens).



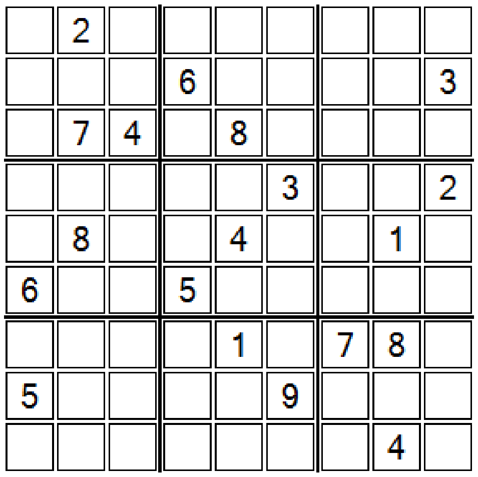
	4		8		5	2		
	2			4			5	
5								4
	9				3	1	2	
1		6		7	8			3
3	7		9		4		8	
					6	7		
		8	3	5	9		1	
	1	9			7	6		

Figure 6: Easy difficulty Sudoku board used for testing (34 givens) [16].



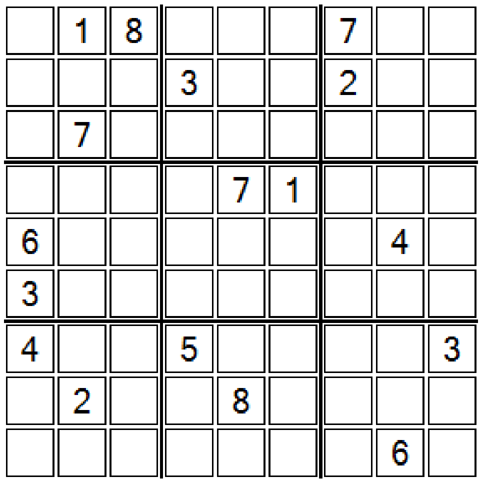
				8	4		1	
	8			5		9		
1	4					2	8	5
4					3		6	
		6	1					9
3		9	8					4
						4		1
		3	4			7		
	1			6	2	3		

Figure 7: Medium difficulty Sudoku board used for testing (30 givens).



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

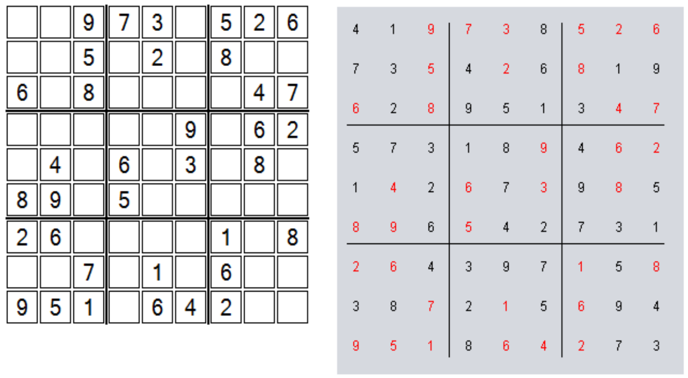
Figure 8: Hard difficulty Sudoku board used for testing (19 givens) [17].



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

Figure 9: Very hard difficulty Sudoku board used for testing (17 givens) [18].

Figure 10 shows the input board for the very easy solution, along with the solution provided by the algorithm.



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

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

Figure 10: Very easy difficulty Sudoku board and the solution provided by the algorithm ("given" values are shown in red in the solution).

B. *Genetic Algorithm vs. Genetic Algorithm using the Wisdom of Crowds Heuristic*

All runs performed used the following parameters:

Population: 5,000

Mutation rate: 40%

Expert percentage: 5%

Values averaged over 10 runs

Genetic Algorithm will be referred from this point on as GA and Genetic Algorithm using the Wisdom of Crowds heuristic may be referred to as GA+WoC.

As illustrated in Figure 11, the GA and GA+WoC were fairly even when it came to the average number of conflicts left on the board when a run would end. The GA+WoC performed slightly better, with an overall average of 3.78, compared to the 3.82 for the GA.

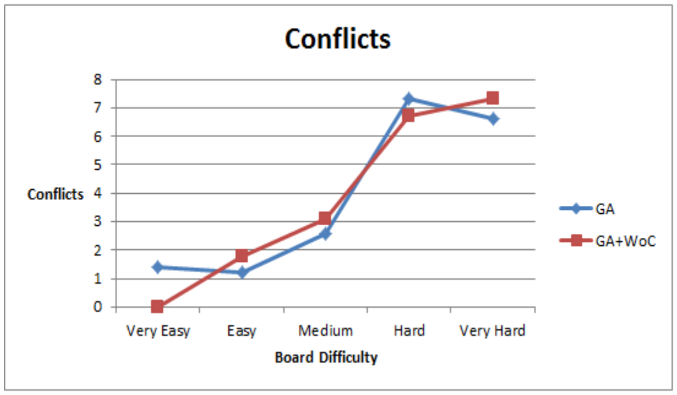


Figure 11: Graph comparing number of conflicts by board difficulty for GA and GA+WoC.

The GA was marginally faster than the GA+WoC, with an overall average of 131.34 seconds, compared to 139.34 seconds for the GA+WoC.

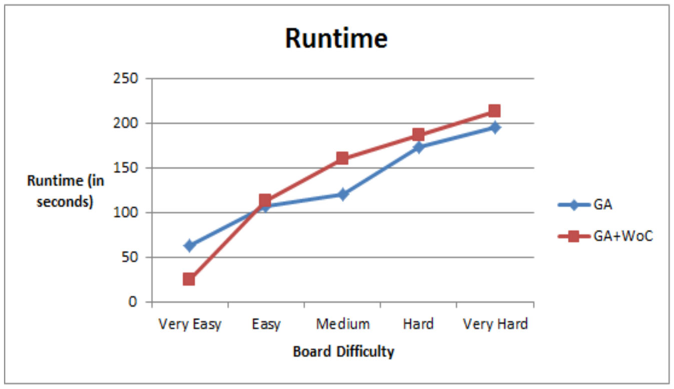


Figure 12: Graph comparing runtime by board difficulty for GA and GA+WoC.

GA and GA+WoC were fairly even when it came to the average number of generations needed on the easy, hard, and very hard boards. However, GA+WoC was significantly more efficient when attempting to solve the very easy board, and GA was moderately more efficient when attempting to solve the medium board. Overall, GA+WoC was slightly more efficient, averaging 252.04 generations compared to GA's 262.26.

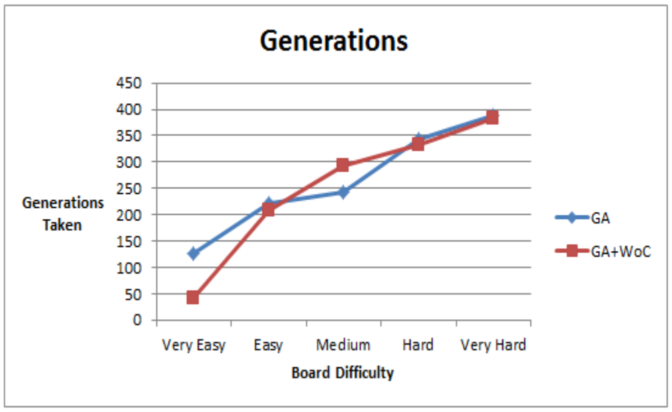


Figure 13: Graph comparing number of generations taken by board difficulty for GA and GA+WoC.

Figure 14 depicts the performance of the wisdom of crowds heuristic while solving the very easy Sudoku puzzle (averaged over 10 runs as well). As can be seen, the wisdom of crowd's aggregate solution always performed better than the overall expert average. The aggregate provided a better solution than the best expert for roughly half of the generations. The best expert solution performs better toward the end of the run, presumably because the aggregate has a smaller window for improvement. It should be stated that the aggregate solution appears to always improve; it is simply that the best expert improves more quickly toward the end of the run.

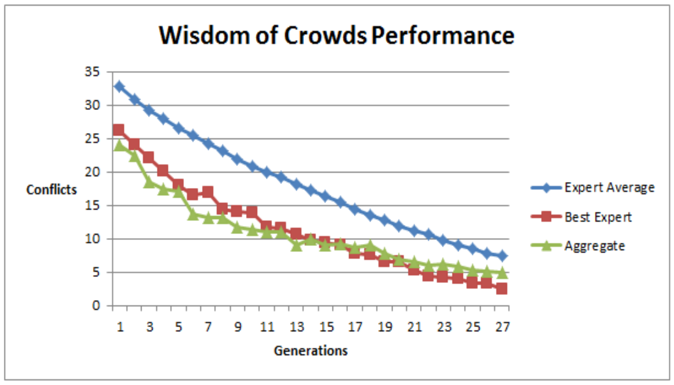


Figure 14: Graph showing the performance of the wisdom of crowds heuristic

Overall, the GA+WoC averaged .348 conflicts less per generation than the best expert solution. The GA+WoC provided a solution that was, on average, 4.2% better than the best expert solution.

IV. CONCLUSIONS

An algorithm was presented that can reliably solve very easy and easy Sudoku boards, can occasionally solve medium difficulty boards, and comes very close to solving hard and very hard boards. It has been shown that making use of the wisdom of crowds heuristic when using genetic algorithms will typically result in a similar quality solution to a regular genetic algorithm, but will do so in fewer generations (at the cost of a slightly higher runtime).

Table 1: Overall performance of GA and GA+WoC in three areas

	Average conflict count	Average runtime	Average generations needed
GA	3.82	131.34	264.26
GA+WoC	3.78	139.34	252.04

Future work will consist of improving the algorithm so that it does a better job in overcoming local maxima points and of increasing number of puzzles in each testing level to increase statistical significance of the achieved results. Even on the very hard problems the algorithm managed to average less than 8 conflicts, which would seem to suggest that the core logic in the algorithm is sound, it just needs more fine-tuning. The 'Shake-Up' functionality is a good start and can be improved upon. Future work should also focus on improving the speed of the wisdom of crowds heuristic so that the GA+WoC implementation averages a better runtime to go along with the lower average number of generations needed.

The core issue of this algorithm might lead one to question the usefulness of genetic algorithms for solving Sudoku. Such questioning definitely has merit. Genetic algorithms seem best suited for situations where finding a near-optimal solution is sufficient. In Sudoku, however, there is typically only one solution per board, and a near-optimal final board state is not worth anything; it is still an invalid solution. Sudoku solving algorithms seem to fair better when they view Sudoku as a constraint satisfaction problem [19]. Sudoku can easily be viewed as a constraint satisfaction problem with four constraints:

1. Given values cannot be altered
2. Each row has to be a permutation of the values 1 through 9
3. Each column has to be a permutation of the values 1 through 9
4. Each sub-grid has to be a permutation of the values 1 through 9 [20]

Despite these difficulties, the algorithm presented in this paper provides a solid framework for improvement and is well within reach of being able to reliably solve very hard Sudoku puzzles. The data presented here clearly shows the benefit of using the wisdom of crowds heuristic when solving Sudoku puzzles with a genetic algorithm. As stated in the introduction, genetic algorithms are topic agnostic, in the sense that the underlying architecture can be easily altered for various problem types, and the algorithm requires very little in the way of specifics about a problem space. This suggests that the performance boost obtained from the Wisdom of Crowds heuristic can be expected in implementations of this algorithm on other problem types [21]. Some promising results have already been demonstrated with respect to other puzzles such as Light Up [22] and Mastermind [23].

REFERENCES

[1] C. Darwin, “The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life”, Ed. William F. Bynum. AL Burt, 2009.

[2] H. Sengoku and I. Yoshihara, “A Fast TSP Solver using GA on Java”, 3rd International Symposium on Artificial Life and Robotics (AROB-III), pp. 283–288, 1997.

[3] K. De Jong, “Learning with genetic algorithms: An overview”, Machine learning 3.2, p. 121-138, 1988.

[4] K. Das, S. Bhatia, S. Puri, and K. Deep, “A Retrievable GA for Solving Sudoku Puzzles”, Technical Report, Department of Mathematics, IIT Roorkee, May 2008. http://www.cse.psu.edu/~sub194/papers/sudokuTechReport.pdf.

[5] C. Chang, Z. Fan, and Y. Sun, “A Difficulty Metric and Puzzle Generator for Sudoku”, The UMAP Journal, Vol. 29, no. 3, p. 305-326, 2008.

[6] M. Perez and T. Marwala, “Stochastic Optimization Approaches for Solving Sudoku”, arxiv.org/pdf/0805.0697, May 6, 2008.

[7] L. Taalman, “Taking Sudoku Seriously”, Math Horizons 15.1, p. 5-9, 2007.

[8] G. Kendall, A. Parkes, and K. Spoerer, “A survey of NP-complete puzzles”, International Computer Games Association Journal 31, no. 1, p. 13-34, 2008.

[9] T. Yato and T. Seta, “Complexity and Completeness of Finding Another Solution and Its Application to Puzzles”, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 86, No. 5, p. 1052–1060, 2003.

[10] L. Fortnow, “The Status of the P versus NP Problem”, Communications of the ACM, Vol. 52, p. 78-86, September 2009.

[11] T. Mantere and J. Koljonen. “Sudoku solving with cultural swarms”, AI and Machine Consciousness, 2008. www.stes.fi/step2008/proceedings/step2008proceedings.pdf#page=60

[12] C. Reeves, “Handbook of Metaheuristics”, International Series in Operations Research & Management Science, Vol. 57, p. 55-82, 2003.

[13] T. Mantere and J. Koljonen, “Solving and Rating Sudoku Puzzles with Genetic Algorithms”, Publications of the Finnish Artificial Intelligence Society, Vol. 12, no. 23, p. 86-92, October 26-27, 2006.

[14] D. Waters, “SudoKube: Using Genetic Algorithms to Simultaneously Solve Multiple Combinatorial Problems”, MS Thesis, Oklahoma State University, 2007. http://dc.library.okstate.edu/utils/getfile/collection/theses/id/3203/filename/3204.pdf

[15] R. Yampolskiy, L. Ashby and L. Hassan, “Wisdom of Artificial Crowds—A Metaheuristic Algorithm for Optimization”, Journal of Intelligent Learning Systems and Applications, Vol. 4 No. 2, 2012, p. 98-107. doi: 10.4236/jilsa.2012.42009.

[16] M. Ercsey-Ravasz and Z. Toroczkai, “The Chaos Within Sudoku”, Scientific Reports, Vol. 2, no. 725, October 11, 2012.

[17] T. Weber, “A SAT-based sudoku solver”, In Geoff Sutcliffe and Andrei Voronkov, editors, LPAR-12, p. 11–15, December 2005.

[18] I. Lynce and J. Ouaknine, “Sudoku as a SAT problem”, In proceedings of the 9th Symposium on Artificial Intelligence and Mathematics, January 2006.

[19] P. Norvig, “Solving Every Sudoku Puzzle”, http://norvig.com/sudoku.html

[20] A. Moraglio, J. Togelius, and S. M. Lucas, “Product Geometric Crossover for the Sudoku Puzzle”, 2006 IEEE Congress on Evolutionary Computation (CEC2006), 470-476, Vancouver, BC, Canada, July 16-21, 2006.

[21] R. V. Yampolskiy and A. El-Barkouky, Wisdom of Artificial Crowds Algorithm for Solving NP-Hard Problems, International Journal of Bio-Inspired Computation (IJBIC), 3(6) (2011), pp. 358-369.

[22] L. H. Ashby and R. V. Yampolskiy, Genetic Algorithm and Wisdom of Artificial Crowds Algorithm Applied to Light Up, 16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games, Louisville, KY, USA July 27 - 30, 2011, pp. 27-32.

[23] A. B. Khalifa and R. V. Yampolskiy, GA with Wisdom of Artificial Crowds for Solving Mastermind Satisfiability Problem, International Journal of Intelligent Games & Simulation (IJIGS), 6(2), December 2011.