Dot Dot Line Sudoku

Written report

Cheng Wenhao 3i1 (03) Leader

Wang Kwang Ji 3i1 (26)

Ye Rongyang 3i2 (29)

Hwa Chong Institution

(High School)

1. Introduction and Rationale

Sudoku is a logic-based, combinatorial number-placement puzzle. It is a mathematical challenge in which the objective is to fill a 9x9 grid with digits one to nine so that every row, column, and each of the nine 3x3 regions does not have the same number occurring twice. It dates back to the 18th century AD, where it was modelled after a Swiss mathematician's game called “Latin Squares”. To solve these Sudoku puzzles, researchers have used different types of algorithms such as pencil and paper algorithm, flower pollination algorithm, harmonising search algorithm, bee colony optimization algorithm and integrating two different algorithms together.

This project aims to extend on the methods used to solve Sudoku and also look at the difficulties of puzzles from the graph theory point of view. We will be focusing on the integration of the Flower Pollination Algorithm and the Message Passing Algorithm for solving Sudoku puzzles, as well as the Sudoku Generation Algorithm.

1.1. Objectives

- To find out how Flower Pollination and the Message Passing algorithms can be applied when it comes to sudoku
- To find out the different methods to evaluate easy, intermediate and challenging difficulties of Sudoku puzzles
- To find a general way to solve Sudoku boards with size $n \times n$

1.2. Research Problems

- How can hybridisation of Flower Pollination and Message Passing algorithms be improved in terms of speed and success rate for Sudoku games of different difficulty levels?
- How to create a Sudoku game with increased difficulty?
- How can the algorithms be modified to solve Sudoku games for all sizes?

1.3. Field of Math

- Probability theory
- Graph theory

1.4. Definition

Term	Definition
Region	A part of the sudoku board, a 3×3 in a usual sudoku board, such that every number in this region is not repeated
Speed	The amount of time taken for the algorithm to complete the code
Success Rate	How likely the algorithm is able to solve the Sudoku puzzle
Flower pollination algorithm	The flower pollination algorithm is a nature-inspired algorithm that imitates the pollination behaviour of flowering plants.
Harmony search algorithm	Harmony Search (HS) is an evolutionary algorithm, which mimics a musicians' actions, such as random play, memory-based play, and pitch-adjusted play when they perform. A musician randomly playing can be associated with choosing any value from harmony

	memory, which is also known as memory consideration. A pitch adjusted play can be associated with choosing an adjacent value of one value from harmony memory. A musician playing a totally random sound from memory can be associated with choosing a random value from the possible numbers.
Message passing algorithm	An iterative decoding algorithm factorizes the global function of many variables into a product of simpler local functions, whose arguments are the subset of variables. It includes algorithms like the max-product algorithm, sum product algorithm and belief propagation algorithm.

2. Literature Review

2.1. Flower Pollination Algorithm and Chaotic Harmony Search Algorithm

Osama, Ibrahim and Mohamed (2014) observed that the success rate for bee colony optimisation over fifty runs was not as high as other methods applied to the same problems. Osama, Ibrahim and Mohamed (2014) observed that the chaotic harmony search algorithm has been a highly efficient algorithm that can be used to solve Sudoku puzzles. As a result, numerous projects have been done to attempt to increase the efficiency of this algorithm. However, even by integrating chaotic harmony search and flower pollination algorithms together, the run time still took 9 seconds.

2.2. Generation of puzzles

Boothby, Svec, and Zhang, (2008) observed that there exists a Sudoku Generation algorithm which provides great control over the resulting puzzle and it is possible to generate a puzzle at the desired difficulty. Although each puzzle is generated and analyzed within a few seconds, thousands have to be generated to increase the likelihood of creating a difficult puzzle. Although the run time of this search engine was constant, the time required to successfully run this engine was incredibly high. By finding out how a given Sudoku puzzle can be made harder, thousands of puzzles do not need to be generated.

2.3. Belief propagation / Message passing algorithm

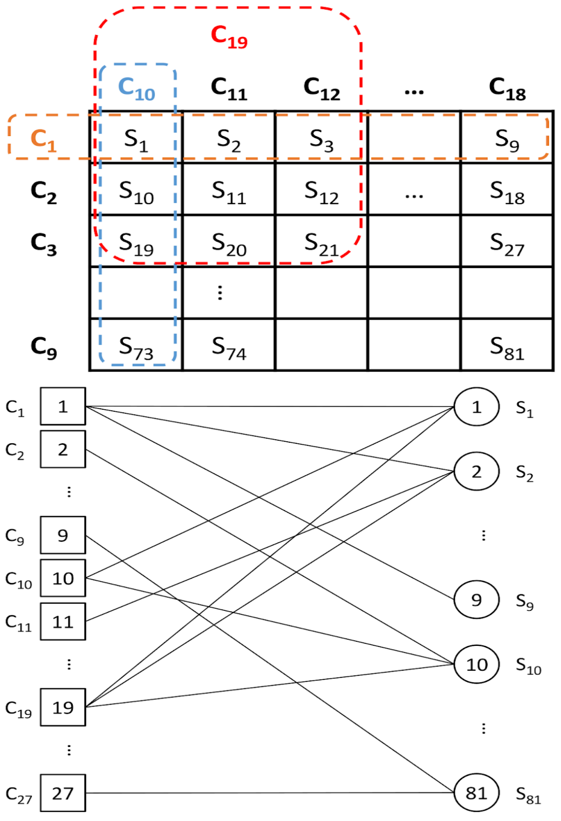
Goldberger (2007) found out that the method which would have the highest success rate on sudoku puzzles with 17 given numbers was by using the sum-product algorithm on the stopping-set obtained by the extended version of max-product. This method also took a total of 0.223s. Porcu (2015) made use of the belief propagation algorithm and when the sudoku provided within 17 to 20 numbers, the percentage success rate of solving the sudokus was really low at 6.66%. Message passing algorithms with and without schedules had average times from 52s to 125s (Porcu, 2015). Message passing algorithms are very powerful for their simplicity of implementation, but at the same time, they have limits due to the presence of cycles, which exist in the sudoku graph, in the Factor Graph and Stopping-Sets during the solution. If Stopping-Sets are able to be minimised, through the alteration of the algorithm or other algorithms, the success rate and speed should be maximised.

Algorithm	Average Speed	Success Rate
Integrating harmony search and flower pollination algorithms	9s	100%
Belief propagation algorithm	52s to 125s	6.66%
Sum-product algorithm	0.223s to 2.81s	71.3% to 89.5%

3. Study and Methodology

3.1. Research Question 1

For the 1st research question, the Belief Propagation Algorithm will attempt to solve the puzzle. Flower Pollination Algorithm can be implemented after the Belief Propagation Algorithm to solve the puzzle.



Each constraint, C_m, is associated with a row, column, or grid. Each constraint is responsible for ensuring that every number in said row, column or grid is unique and that there are no repeats. S_n represents the value of cell n. There are a total of 27 constraints for a 9 * 9 puzzle, with C_1 to C_9 representing the row constraints, C_{10} to C_{18} representing the column constraints, and C_{19} to C_{27} representing the column constraint. x_n refers to a value between 1 to 9 for a 9 by 9 puzzle. N_{mn} refers to all cells associated with constraint m except for cell n. M_{nm} refers to all constraints associated with cell n except for cell m.

Each n cell has a probability vector associated with it called p_n with a size of 9 for a 9 by 9 puzzle. Each index of the probability vector represents the probability of the value of the cell being x_{index}. All probabilities are uniformly distributed initially and values are then eliminated via simple elimination by the program. The probabilities will then be readjusted. For example, if the first cell is empty and the values that cannot be used by the first cell are 1, 3, 5, 7 and 9, the probability vector associated with the cell p_1 , will be (0,0.25, 0, 0.25, 0 , 0.25, 0 , 0.25, 0). Similarly, if the initial value of a cell is 9, the corresponding probability vector will be (0, 0, 0, 0, 0, 0, 0, 0, 1).

Messages are sent between constraints and n cells. The message which constraint C_m sends to cell n is a probability vector, which can be represented as R_{mn}. R_{mn} for a 9 by 9 puzzle can be represented as R_{mnx} = (R_{mn}(1), R_{mn}(2), R_{mn}(3), R_{mn}(4), R_{mn}(5), R_{mn}(6), R_{mn}(7), R_{mn}(8), R_{mn}(9))

R_{mn}(x) can be represented by the equation: $P(C_m|S_n = x_n)$, which can be rewritten as:

$$\sum_{(x_{n'},\, n' \in N_{mn})} P(C_m|\, S_n = x_n\,,\ \{S_{n'} = x_n\ \ n' \in N_{mn}\}) \times P(\{S_{n'} = x_{n'}\,, n' \in N_{mn}\}|S_n = x)$$ (K. Moon, H. Gunther , 2011)

$P(C_m|\, S_n = x_n\,,\ \{S_{n'} = x_n\ \ n' \in N_{mn}\})$ refers to the probability of C_m being satisfied when S_n = x and all the other cells associated with C_m have a certain value. Hence, there are only two different return values for this equation, that being 0 or 1. $P(C_m|\, S_n = x_n\,,\ \{S_{n'} = x_n\ \ n' \in N_{mn}\}) = 1$ if all numbers are unique and 0 if otherwise. $P(S_{n'} = x_{n'}\,, n' \in N_{mn}|S_n = x)$ simply refers to: for n' $\in N_{mn}$, the probability of n' being $x_{n'}$ will be calculated and multiplied together with the other probabilities. Therefore, the equation can be simplified further to:

$$\sum_{(x_{n'},\, n' \in N_{mn})} \prod_{(l \in N_{mn})} q_{ml}(x_l)\text{ , where } q_{ml} = P(S_l = x_l|\, S_n = x_n) \text{ (equation 2)}$$

Here is an example of the implementation of the equation for a 4 by 4 sudoku puzzle to a row constraint(C_1):

$R_{11}(1)$ = $q_{12}(2) \times q_{13}(3) \times q_{14}(4) + q_{12}(2) \times q_{13}(4) \times q_{14}(3) + q_{12}(3) \times q_{13}(2) \times q_{14}(4) + q_{12}(3) \times q_{13}(4) \times q_{14}(2) + q_{12}(4) \times q_{m3}(2) \times q_{14}(3) + q_{12}(4) \times q_{13}(3) \times q_{14}(2)$

After the message R_{mn} is sent from the constraint to n, a message will be sent back to the constraint from n, also known as Q_{mnx}, which can be represented by a probability vector ($Q_{mn}(1)$, $Q_{mn}(2)$, $Q_{mn}(3)$, $Q_{mn}(4)$, $Q_{mn}(5)$, $Q_{mn}(6)$, $Q_{mn}(7)$, $Q_{mn}(8)$, $Q_{mn}(9)$), $P(S_n = x|\{C_{m'}, m' \in M_{mn}\})$. The equation for $Q_{mn}(x)$ is $P(S_n = x|\{C_{m'}, m' \in M_{mn}\})$. Using the conditional probability formula and simple modifications, we can simplify this to $P(S_n = x) * \prod_{(m' \in M_{mn})} P(C_{m'}|S_n = x)$

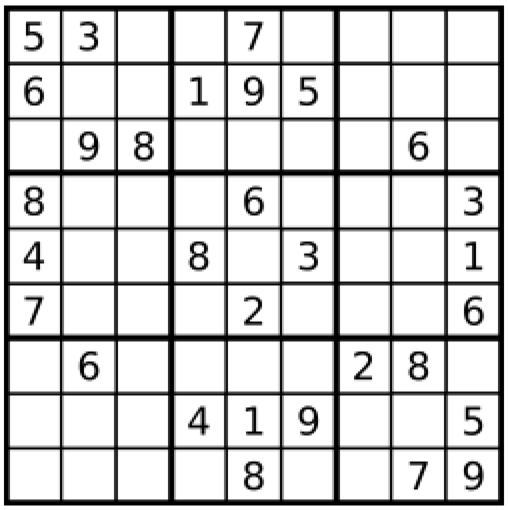
From observation, it can be seen that $P(C_{m'}|S_n = x) = R_{m'n}(x)$. Therefore, $Q_{mn}(x) = \alpha P(S_n = x) * \prod_{(m' \in M_{mn})} R_{m'n}(x)$, where α is a normalising constant to ensure that the area under the graph of this function is 1. The equation $Q_{mn}(x)$ is used to provide more information to the constraint the cell is currently sending to about what values are not allowed and the probabilities of the values that are allowed. The program iterates between the two messages $R_{mn}(x)$ and $Q_{mn}(x)$ and when the probability vectors or 'beliefs' converge and all of a probability vector's mass is placed onto one *x* value, the program will make a hard decision and assign the value to that node.

After the puzzle is partially solved by the Belief Propagation Algorithm, the solution is set as the initial puzzle for the Flower Pollination Algorithm.

	Col 0	Col 1	...	Col 8
Row 0	0	1	...	8
Row 1	9	10	...	17
..	...	...	...	...
Row 8	72	73	...	80

(This is the index of a Sudoku board, row number and column number for Flower Pollination)

An optimisation equation is needed for the Flower Pollination Algorithm.



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

We can observe that the sum of each row, column and region is 45, so a penalty function as an optimisation equation can be used.

This is the equation:

$$f(\{x_i\}) = \sum_{i=0}^{8}\left(\left|\sum_{j=0}^{8} x_{i+9j} - 45\right|\right) + \sum_{i=0}^{8}\left(\left|\sum_{j=0}^{8} x_{9i+j} - 45\right|\right) + \sum_{i=0}^{8}\left(\left|\sum_{j \in B_i} x_j - 45\right|\right) \text{ (Geem, 2007)}$$

where x_i is the value of cell with index *i*, and B_i is the set of indexes of each 3 by 3 region. Because of the absolute values, the minimum value of this function is 0 when the sum of each row, column and region is 45.

However, when the sum of each row, column and region is 45, it does not mean that the board is correctly solved. A simple example is when all numbers are 5. To prevent this error, we have to determine the possible solutions of each cell by adding a new function, known as pencil marks in Sudoku solving. From an index of a cell, we can find the row number, column number, and the region. Then we can eliminate all numbers that already exist in that row, column or region.

$c \equiv i \ (mod\ 9)$, $r = 9\lfloor \frac{i}{9} \rfloor$, $b = 3\lfloor \frac{c}{3} \rfloor + 27\lfloor \frac{r}{27} \rfloor$, where *i* is the index of a given cell, *r* is the leftmost index of that row, *c* is the topmost index of that column, *b* is the top left index of that region. The indexes of cells in the same row, column and region of cell *i* are $\{x_{c+9m}\}, \{x_{r+n}\}, \{x_{b+9p+q}\}$, where $m, n \in \{0, 1, ..., 8\}$ and $p, q \in \{0, 1, 2\}$.

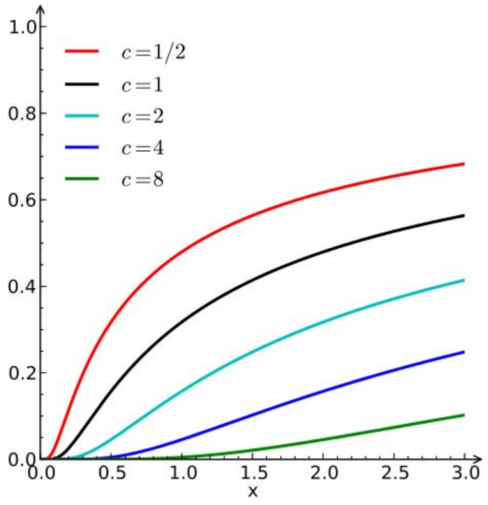
This is the pseudocode.

```
● Define optimisation equation, pencil marks function
● Input partially solved puzzle from Belief Propagation Algorithm
● Initialise $n$ sudoku boards with random solutions based on pencil marks
● Find best solution out of the sudoku boards, based on the optimisation equation
● Define switch probability p=0.8 (Osama Abdel-Raouf, Ibrahim El-henawy, &
  Mohamed Abdel-Baset, 2014)
● For each sudoku board,
    ○ Carry out Global or Local Pollination for all cells in the sudoku board
```

- Round off updated values to closest possible solution based on pencil marks
- If new board returns a lower value in the optimisation equation, replace the old sudoku board with the new sudoku board
- Find best solution

Global Pollination

The equation is $x_i^{t+1} = x_i^t + L(x_i^t - B)$, where x_i^t is the value of cell with index i at iteration t; B is the value of cell with the same index from the sudoku board that has the lowest value in the optimisation equation, and L is a step size drawn from a Levy's distribution. To draw from a distribution, we use the method of inverse transform sampling where a random y is taken and finding the value of x in a cumulative distribution function. Each cell in a sudoku board is updated using this equation.



Levy's distribution. Equation: $y = \text{erfc}\left(\sqrt{\frac{c}{2x}}\right)$

Local Pollination

The equation is $x_i^{t+1} = x_i^t + U(x_j^t - x_k^t)$, where $U \in [0,1]$ is drawn from a uniform distribution. x_j^t and x_k^t are the values of cell with the same index but from 2 different and random sudoku boards. Each cell in a sudoku board is updated using this equation.

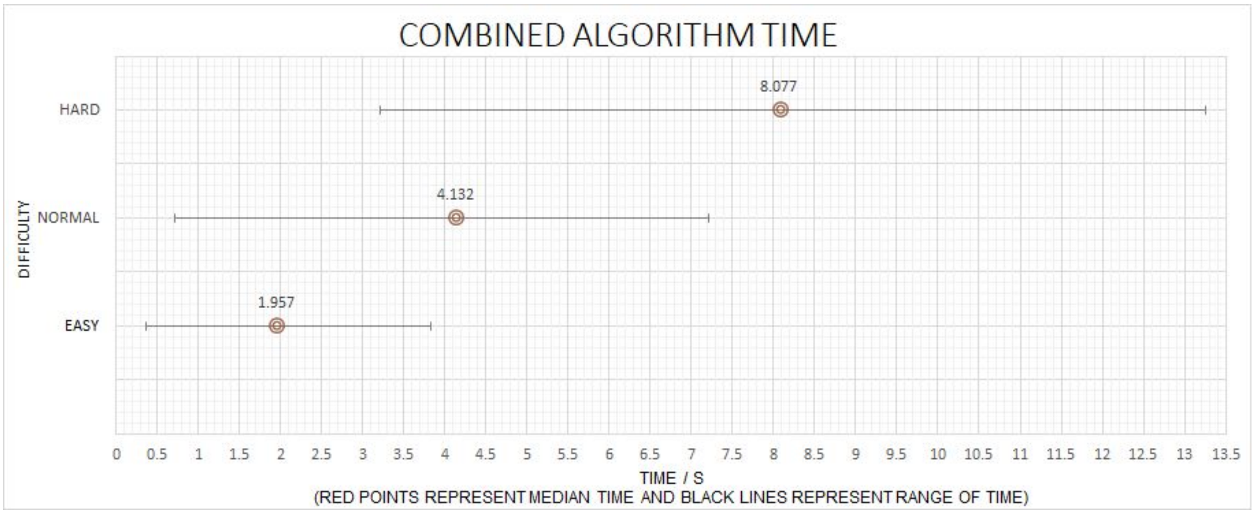
Results and Conclusion for Research Question 1

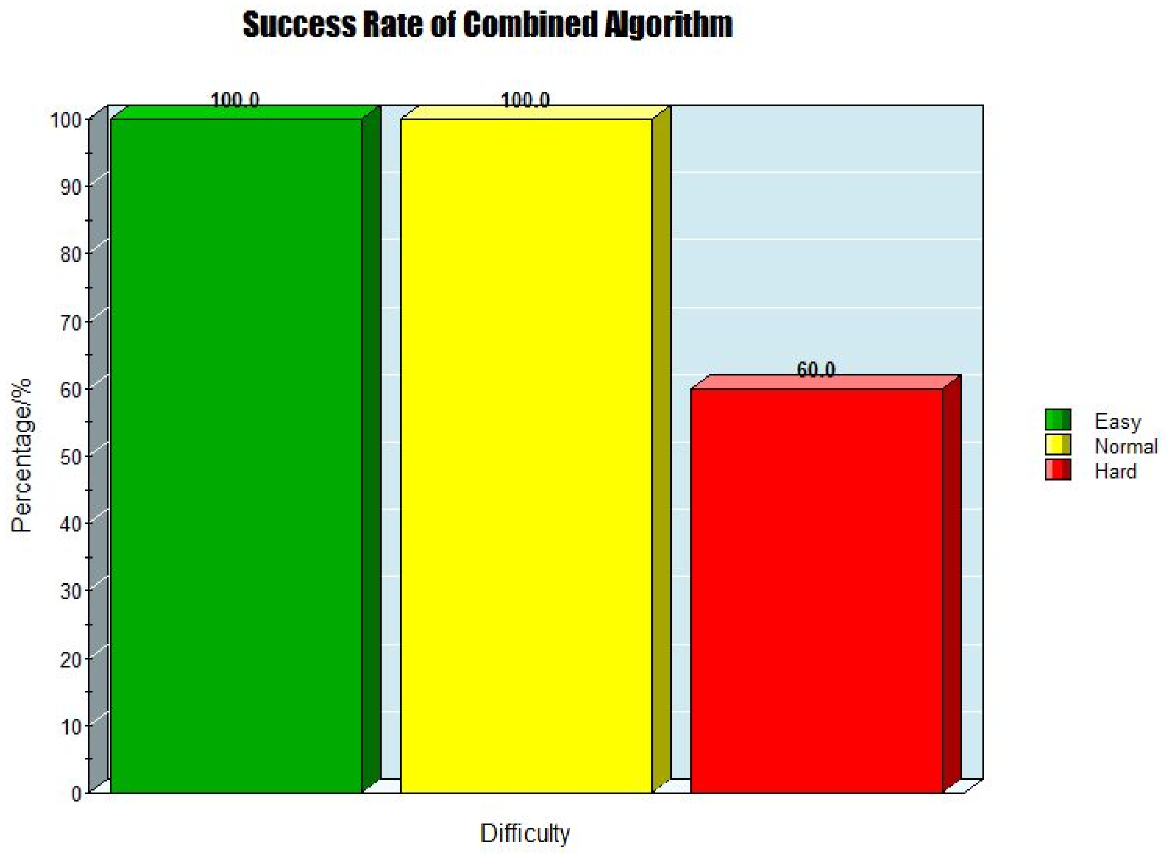
The combined algorithm of belief propagation and flower pollination has achieved a relatively high success rate in solving sudoku puzzles for all difficulties of the puzzle. This is because running the belief propagation algorithm first lowers the probability of stopping sets. Belief propagation is also very fast and can quickly simplify the original board so that the initial puzzle for flower pollination has more numbers, reducing the number of evaluations needed from the flower pollination algorithm. The flower pollination algorithm does not have stopping sets and helps obtain a more accurate solution.

The average run time of the algorithm is considered short. When compared to Osama, Ibrahim and Mohamed (2014) integrated chaotic harmony search and flower pollination algorithms together, with a run time of 9 seconds and message passing algorithms with and without schedules with average times from 52s to 125s (Porcu, 2015). Goldberger (2007) using the sum-product algorithm took a total of 0.223s showing that the algorithm still has room for improvement in terms of time.

However, the range of run time varies with the number of iterations by belief propagation algorithm and evaluations of flower pollination algorithm. This causes a wide range of time required to solve the sudoku puzzles, varying greatly as the difficulty increases. The difficulty increases as the number of cells decreases as a lower number of cells with result in a higher time complexity of the algorithm.

Combined Algorithm Results (Belief Propagation and Flower Pollination)				
Difficulties	Tested Puzzles	Range of Time/s	Average Time/s	Success Rate/%
Easy puzzles (29-32 numbers)	10	0.3691 - 3.836	1.957	100.0
Normal puzzles (25-28 numbers)	12	0.7149 to 7.212	4.132	100.0
Hard puzzles (21-24 numbers)	10	3.205 - 13.252	8.077	60.0





Further extensions to our combined algorithm can include an improved BP algorithm without statistical biases due to loopy graphs, researching the optimal number of iterations belief propagation algorithm should perform to get the shortest run time and highest success rate.

Our combined algorithm could be altered to solve other constraint satisfaction problems in graph theory.

3.2. Research Question 2

For the 2nd research question, our goal is to increase the difficulty of a puzzle by removing more numbers, while ensuring that the solution remains unique. A graph theory approach can be used to replace the backtracking methods used in many other Sudoku Generation Algorithms, when it comes to finding how many solutions there are. This will greatly reduce the memory used, and the complexity of the code.

After random numbers are removed, we look at individual rows and generate all possible solutions for that row only. To reduce the number of solutions generated, other given numbers are considered so all solutions will not have any conflicts with given numbers. For each row, there is a set of solutions. The largest number of solutions for a row with n given numbers is $(9-n)!$. Taking other given numbers into consideration decreases the number of solutions significantly, as 1 possible number as a solution eliminated from a cell in the row will reduce the number of solutions by half. For any 2 rows, a solution from each row is selected and compared. If they do not have any conflicts, an edge can be drawn between the 2 solutions. This process is repeated for every pair of solutions and every pair of row numbers. After that, there will be a constructed graph which has edges connecting different solutions (nodes) to show which pairs of solutions are non-conflicting. To find how many solutions the Sudoku board has, the graph is checked to find complete graphs. A complete graph is a set of nodes where edges connect every pair of nodes. A complete graph within this constructed Sudoku graphs will mean that the solutions of the 9 rows are non-conflicting, and if these solutions are put together, a valid solution of the board will be constructed. The number of complete graphs found is the number of solutions for that puzzle. The puzzle is rejected if it has more than one solution.

This is a puzzle generated by the algorithm that has a unique solution in 1.3s.

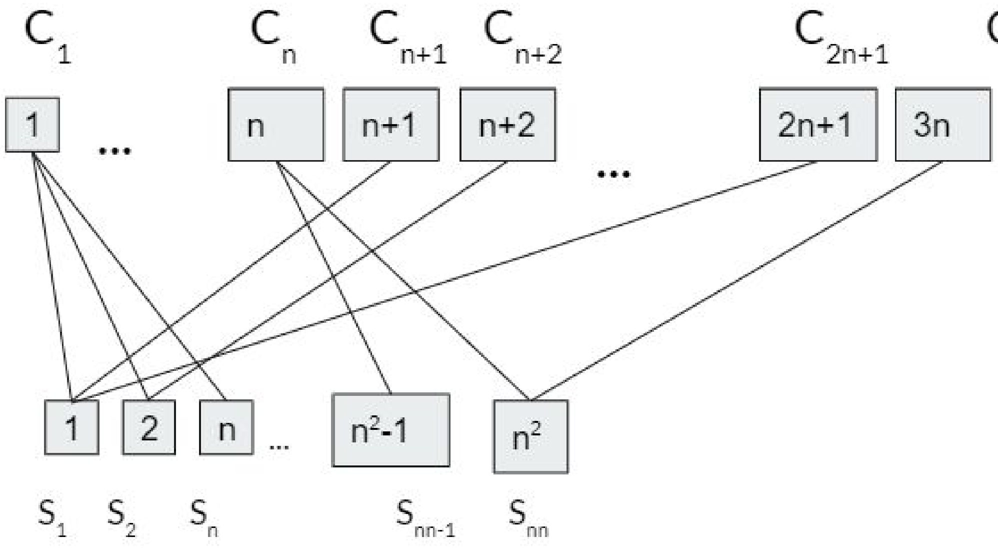
7	4		5	1		2		
	9				4		7	
2	3			7	8			
4	8		7		6	3	9	1
3			4			6	5	7
	6		9		1			4
	2	3		4		9		
6	5	8		9		7		2
					5		3	8

However, the run time of this algorithm will increase significantly when more numbers are removed, as there are more pairs of solutions to check for. Limitations include high run time complexity and memory space limit.

A possible extension is to generate Sudoku boards with a desired difficulty based on techniques needed to solve the board, rather than using the number of given cells, or to generate *n* by *n* Sudoku boards.

3.3. Research Question 3

To solve Sudoku boards for all *n* by *n*, we have to modify the tanner graph and constraints for Belief Propagation Algorithm and the optimisation equation and pencil marks function in Flower Pollination Algorithm. For *n* by *n* Sudoku board, there are $3n$ constraints and n^2 cells. This is how the tanner graph will look like.



Constraints 1 to *n* is for the *n* rows; constraints *n*+1 to 2*n* is for the *n* columns, and constraints 2*n* +1 to 3*n* is for the *n* regions.

In the Flower Pollination Algorithm, the method of iterating through each region needs a general formula. The optimisation equation needs to iterate through all of the regions, while the pencil marks function needs to iterate through the region based on the index only. These are the modified equations:

$c \equiv i \pmod{n}$, $r = n\lfloor \frac{i}{n} \rfloor$, $b = 3\lfloor \frac{c}{\sqrt{n}} \rfloor + n\sqrt{n}\lfloor \frac{r}{n\sqrt{n}} \rfloor$ where *i* is the index of a given cell, *r* is the leftmost index of that row, *c* is the topmost index of that column, *b* is the top left index of that region, and n is the size of the board. The indexes of cells in the same row, column and region of cell i are $\{x_{c+nm}\}, \{x_{r+k}\}, \{x_{b+np+q}\}$, where $m, k \in \{0, 1, ..., n-1\}$ and $p, q \in \{0, 1, ..., \sqrt{n}-1\}$. With these equations, the pencil marks function can be modified to find all indexes of cells that are in the same row, column and region as cell *i.*

The sum of numbers in each row, column and region is $\binom{n+1}{2}$, since $\binom{n+1}{2} \equiv \sum_{i=1}^{n} i$.

Hence the modified optimisation equation is

$$f(\{x_i\}) = \sum_{i=0}^{n-1}\left(\left|\sum_{j=0}^{n-1} x_{i+nj} - \binom{n+1}{2}\right|\right) + \sum_{i=0}^{n-1}\left(\left|\sum_{j=0}^{n-1} x_{ni+j} - \binom{n+1}{2}\right|\right) + \sum_{i=0}^{\sqrt{n}-1}\left(\sum_{j=0}^{\sqrt{n}-1}\left(\left|\sum_{p=0}^{\sqrt{n}-1}\left(\sum_{q=0}^{\sqrt{n}-1}\left(x_{in\sqrt{n}+j\sqrt{n}+pn+q}\right)\right) - \binom{n+1}{2}\right|\right)\right)$$

However, the modified algorithms will not be able to solve difficult Sudoku boards due to combinatorial explosion as size increases. This algorithm can only solve easier puzzles of larger sizes.

4. References

Ananthagopal, Lakshmi, "Application of Message Passing and Sinkhorn Balancing Algorithms for Probabilistic Graphical Models" (2014). Master's Projects. 365. DOI: https://doi.org/10.31979/etd.8afz-w6k8 https://scholarworks.sjsu.edu/etd_projects/365

Bartlett A., Chartier T. P., Langville A. N & Rankin T. D (2008). *An Integer Programming Model for the Sudoku Problem*, Journal of Online Mathematics and its Applications, Vol 8, No. 1

Boothby, T., Svec, L., & Zhang, T. (2008), *Generating Sudoku Puzzles as an Inverse Problem.* Retrieved on 20 Feb 2020 from https://sites.math.washington.edu/~morrow/mcm/team2306.pdf.

D.Koller (2017) , Belief Propagation Algorithm. Retrieved on 19 May 2020 from https://www.coursera.org/lecture/probabilistic-graphical-models-2-inference/belief-propagation-algorithm-1FE96

Freeman B. , Torralba A. Lecture 7: graphical models and belief propagation. Retrieved on 5 July 2020 from http://helper.ipam.ucla.edu/publications/gss2013/gss2013_11344.pdf

Goldberger, J. (2007). Solving sudoku using combined message passing algorithms. School of Engineering, Bar-Ilan University. Retrieved from http://www.gatsby.ucl.ac.uk/~turner/workshop/GBPpapers/Goldberger%202007.pdf on 20 Feb 2020.

Osama Abdel-Raouf, Ibrahim El-Henawy, & Mohamed Abdel-Baset (2014). *Novel Hybrid Flower Pollination Algorithm with Chaotic Harmony Search for Solving Sudoku Puzzles*, IJMECS, vol.6, no.3, pp.38-44, 2014.
DOI: 10.5815/ijmecs.2014.03.05

Porcu, F. (2015). Application of Belief Propagation Algorithms on Factor Graphs: from Sudoku solving to LDPC decoding. Retrieved from https://core.ac.uk/download/pdf/79619165.pdf on 20 Feb 2020.

Richard E. Turner Belief Propagation and Sudoku Worksheet. Retrieved on 13 July 2020 from http://www.gatsby.ucl.ac.uk/~turner/workshop/GBPpapers/GBPWorksheet.pdf

Wainwright, M. (2003). Graphical model, message-passing algorithms and convex optimization. Retrieved From https://people.eecs.berkeley.edu/~wainwrig/Talks/A_GraphModel_Tutorial.

Yang, X.-S.: Flower pollination algorithm for global optimization. In: International Conference on Unconventional Computing and Natural Computation, pp. 240–249. Springer (2012)

Z. W. Geem (2007) Harmony search algorithm for solving sudoku, Knowledge-Based Intelligent Information and Engineering Systems, pp. 371-378.