To Flip or Not to Flip: Pancake Sorting with Genetic Algorithms and Wisdom of Crowds

Sai CHINTHALA, Aaron FOX, Abby SPALDING
Department of Computer Science and Engineering, University of Louisville, USA
Emails: smchin01@louisville.edu, amfox005@louisville.edu, acspal03@louisville.edu

***Abstract*—A deeper investigation of the Pancake Sorting problem which involves sorting a stack of pancakes by flipping or reversing at a specific index in the pancake stack. The goal is to achieve the sorted stack in as few flips as possible. For this study, the idea of pancake flipping is applied to a string of unsorted characters in which the string can only be sorted by reversing a prefix from a certain index. A Genetic Algorithm is applied to the problem which presents greater possibility for solutions, as well as a variety of solutions. Wisdom of Crowds is also analyzed as it applies to number of sorts required for the minimum number of reversals. Python is the target language for implementing the algorithm and PyQt is used for Graphical display.**

Keywords—pancake sorting, genetics, reversal distance, Genetic Algorithm, Wisdom of Crowds, NP Complete, Single point crossover, Aggregation Method

I. Introduction

A. History

Pancake sorting finds its origins in genetics, specifically the genome (genomics) where genes and DNA evolve. Although, technically I suppose one would say it found its genetic purpose in Genome Rearrangement. In the 1920s, Alfred H. Sturtevant discovered while studying species of Drosophila (fruit flies) that some gene segments in the species had inverted or reversed in the descendent generations. These reversals can be harmful to a species mutation but can also allow for beneficial traits necessary for evolution to develop [1]. Roughly a decade after his discovery, Sturtevant partnered with Theodosius Dobzhansky from Caltech to further study Drosophila, suggesting more reversals yielded further evolutionary 'distance'. Evolutionary distance refers to the minimum number of reversals necessary to change a chromosome A to chromosome B. Measuring the minimum reversal distance, as well as determining exactly which sequence of reversals produced the mutation, was a main focus of research [2][3].

B. From Genetics to Flipping Pancakes

A little distant from the world of Genetics, flipping pancakes finds its way into the discussion of reversals and sorting algorithms. This time though, the focus steers more towards finding an upper bound where the minimum number of reversals will be no more than X.

First introduced by Jacob E. Goodman in the *American Mathematical Monthly*, the Pancake Problem involves sorting n pancakes from an unsorted stack by merely flipping a number of pancakes and varying the number each time until the stack is sorted. Since its introduction, a number of sub-genre pancake sorting algorithms have also come to fruition. Two main sub-algorithms are (1) the Burnt Pancake Problem in which each pancake has a good side and a burnt side, and the final stack must be sorted with burnt side down and (2) the Two-Spatula Problem which allows an inner set of pancakes to be flipped while holding a top stack with another spatula [4].

The original pancake sorting algorithm as well as the one studied in this paper, focuses solely on prefix reversals which can be seen in Fig. 1. In genetics, there is no limitation to where the reversal may happen in a certain set of genes.

Figure 1. Provided an unsorted list.

5	3	1	2	4

Reversing the entire list.

4	2	1	3	5

Reversing the prefix ending at 3

3	1	2	4	5

Reversing the prefix ending at 2

2	1	3	4	5

Reversing the prefix ending at 1

1	2	3	4	5

End with a sorted list.

Most studies focus on worst case sorting distances over all length-n permutations rather than the computational complexity of sorting by prefix reversals [16]. The first recognized worst case upper bound for this problem involved reversing at whichever index the highest weighted value occupied until it was in its correct place and worked backwards [2]. This naive algorithm provided an upper bound of 2n reversals (one to get it to the top position, and one to place it in its appropriate position) which was not a very tight upper bound. This upper bound was improved in the 70s when Christos H. Papadimitriou and William H. Gates devised an algorithm to sort a stack of pancakes in no more than (5n + 3)/3 moves [5]. The upper bound was improved yet again nearly 30 years later when a group of researchers at the University of Texas at Dallas set out to prove an upper bound of (18/11)n. The worst case lower bound was also improved to (15/14)n [6].

The true computational complexity for sorting by prefix reversals is still up for discussion [16]. In order to prove NP-Completeness of prefix reversal sorting (pancake flipping) for arbitrary strings, it is first easier to prove prefix reversal sorting is NP-Complete for binary strings. In other words, demonstrating that the binary subset of the problem is difficult then demonstrating any k-ary (k>2) problem is also difficult [17].

The proof of NP-Completeness for prefix reversal sorting utilizes the 3-Partition Problem, which is also a recognized NP-Complete problem [18][19].

1. Defining the problem: Related strings S and T of length n with alphabet size ≥ 2, and a bound B $\in Z^+$.
2. Is $d_r(S, T) \leq d$? Where d_r is defined as the reversal distance to transform S to T.
3. 3-Partition: A set K of 3m elements, a bound B, and a size s(k) $\in Z^+$ for each k $\in$ K such that B/4 < s(k) < B/2 and that $\Sigma_{k \in (K)} s(k) = mB$
4. Can K be partitioned into m disjoint sets, such that $1 \leq i \leq m$, $\Sigma_{k \in Ki} s(k) = B$?

Following this proof and the claims made by Hurkens et al., it can be concluded that finding the reversal distance between two strings is NP-Hard.

II. Prior Work

A. Pancake Flipping & Sorting Permutations

The *pancake flipping problem* and its algorithms are discussed and analyzed to find optimal algorithms to get the least number of flips required to sort a stack of pancakes. This problem and variations of this problem are applied in interconnection networks and computational biology. In interconnection networks, the number of flips needed to sort n pancakes serves as the diameter of the n-dimensional pancake network. The worst communication delay for broadcasting messages is equal to the diameter of a network.

A similar approach to the pancake flipping problem is the burnt pancake flipping problem. In this approach, each of the pancakes, or string in string sorting, is assigned a sign. The sign changes with each reversal (flip)[9].

B. NP-Completeness: A Retrospective

One of the most important concepts on NP complete is that "'polynomial worst-case time" is a plausible and productive mathematical surrogate of the empirical concept of the "practically solvable computational problem." For the pancake sorting problem, the question of whether it is NP complete is specified by a small detail. It is only NP complete if the number of flips to sort the stack is found. Chitturi and Sudborough and Hurkens et al. proved that pancake sorting is NP complete through the use of strings by transforming a string into another string minimum number of by prefix reversals. The upper bound solved by Bill Gates and Christos Papadimitriou was improved 30 years later by Sudborough and Hurkens et al to be 18n/11. The pancake flipping problem solves finds a least number of flips which is less than 18n/11, which makes it NP complete [8].

C. Introduction to Genetic Algorithms

Genetic algorithms are utilized due to their high capability as search and optimization tools. They differ from classical search and optimization methods because of its adaptability and applicability. Genetic algorithms were first introduced at the University of Michigan, Ann Arbor by professor John Holland. In more traditional direct search and gradient based techniques, the algorithms find suboptimal solutions and are not as efficient in handling problems with discrete variables. The simple steps of genetic algorithms include initialization population, reproduction, crossover and mutation[13][14]. The chromosomes are evaluated using a fitness function to determine the 'fitness' of the chromosomes, which are used to either eliminate or continue using the chromosomes. The crossover and mutation operators use a constant crossover and mutation rate to create 'fitter' offspring or mutate existing chromosomes. [15]

D. Wisdom of Crowds

Wisdom of crowds is the idea that aggregated results from a group have the ability to be more accurate than most individuals and even some experts in a particular field [20]. In our previous projects of Traveling Salesperson, we aggregated a percentage of experts' results in a population and compared to individuals to see if the outcome was indeed better. Most of the time, an improved result should be found, but of course if one limits the 'expert' pool to only a percentage of the population, there could be better outliers that would not be chosen for the aggregation. Typically in problems with stochastic elements, wisdom of crowds should prove better than individuals since it 'sees' the bigger picture when aggregating data to find a best result. In the case where order of aggregated opinions matters, it makes wisdom of crowds harder to implement. In the case of prefix reversals, aggregation on a particular index cannot be determined since the string could have been reversed at index 1 first in one run, but reversed at index 1 fourth in a different run. Order of reversal matters and aggregating at indices would likely have proved for the crowd to never be better.

III. Proposed Approach

A. Building the Chromosomes

Each chromosome is used to represent the indices of the string to flip, with an index of 1 representing the flip of 'ab' in 'abcd', and an index of 2 representing an index of 'abc' in 'abcd.' Each chromosome is determined to be a fixed width of the size of the ceiling of (18/11) times the length of the input string based on the .string input files. As *18n/4* evaluated to a number larger than the total number of available indices, the first *n* indices are set to be all of the possible indices in random order. The remaining *18n/4* indices of the chromosome are then set to completely random possible indices, ranging from 0 to the length of the input string minus one. After each chromosome has been filled to length of *18n/4*, each chromosome is added to the initial population until the population reaches a size of the given `population_size`.

B. Fitness and Cost Functions

To find the cost of each chromosome, the fitness of each chromosome is evaluated in the `evaluate_cost` method of the `genetic_pancake_algorithm` class. The fitness function is called on each chromosome of the population so that the roulette wheel selection for the crossover operator can take place. In the function, all of the chromosome's "flip" indices are applied to the original input string from the bespoke `.string` files that are used as input to the `read_string` method. Two weighted inputs are used to determine the cost, and, in turn, fitness of each chromosome. The first determinant regarding the cost of a chromosome concerns the number of substrings, or subarrays, found within a string and the relative length of those ordered substrings

The cost found for this determinant is found in `find_sub_arrays_length_and_occ`, and this function works by storing each subarray length and the number of occurrences inside a Python dictionary. Initially, each character found inside the string that is used in this function is assigned a relative value based on the character's order in the original string (i.e. given a string array of {'a', 'f', 'j'}, the resulting dictionary would yield {'a': 0, 'f': 1, 'j': 2}.) The length of the subarrays are then

determined by iterating through a forward-seeking while loop, which checks for subarrays like [‘a’, ‘f’] and [‘a’, ‘a’], given the dictionary above. Since the NP-complete version of the pancake sorting problem concerns the use of strings and therefore concerns itself with repeated characters, the forward-seeking while loop accounted for repeated letters. The next while loop, a backward-seeking loop, looked for combinations such as [‘j’, ‘f’, ‘a’] and [‘f’, ‘a’, ‘a’], which are treated as backwards subarrays and thus are determined to help lower the cost of a chromosome. The dictionary is then returned from `find_sub_arrays_length_and_occ` and the `evaluate_cost` function then evaluates the fitness of the returned dictionary by summing all the subarray lengths by their occurences. This fitness is turned into the cost by dividing the fitness by one.

The second determinant for the fitness of each chromosome is the actual distance each character of the temporarily ordered string (as determined by applying all the flips to the original unordered input string given by the chromosome of indices.) This “distance cost” is found by comparing the actual locations of the temporarily ordered string relative to the indices that each character actually should be in when compared to the sorted string of characters. This distance approach is similar to the distance methods found in many approaches toward solving the notorious Traveling Salesman Problem, for it considers costs as determined by the relative distance each segment of the chromosome is from its optimal position.

The distance cost is found by simply adding the absolute value of the temporarily ordered string’s character index from where its character should be found in the sorted string. The final distance cost was found by dividing the summed absolute values of the distances by the maximum possible distance cost (i.e. the cost of a string that is completely in reverse sorted order.) This ensures that the cost is always a value between 0.0 and 1.0.

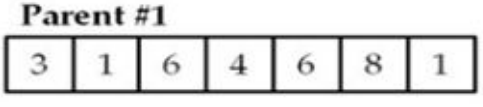
Differing weights for the subsequence and distance costs were assigned to be between 0.0 and 1.0. This allowed for experimenting with rather better results occurred with more emphasis on the string’s order or the number and length of occurrences found within the strings. The weights were then multiplied by their respective costs. After applying these weights, the distance and subsequence costs were multiplied together and returned for the roulette wheel selection to be applied to.

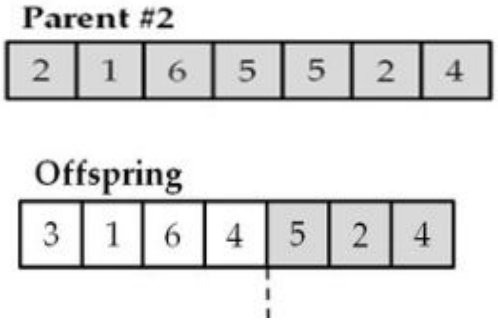
C. *Roulette Wheel Selection*

A roulette wheel selection approach was taken to find the parents of each crossover and mutation operator. The method of the `genetic_pancake_algorithm` class, `get_roulette_wheel`, works by first appending the costs of each chromosome in the given population to an array of costs. The costs are then divided by the summed total costs of every chromosome. Finally, each value in the roulette wheel array is set to be divided by the inverted sum of the costs so that each value inside the costs array sums to 1.0, as is expected behavior of the roulette wheel selection algorithm. Once the roulette wheel’s weights are calculated, Python's builtin in random.choices library then applies those weights to the chromosomes in the current population to select two of the more fit chromosomes in the generation. This helps ensure that the most perfect expert isn’t chosen every time and helps to ensure diversity in the populations by more randomly selecting the parent chromosomes to be selected for later generations.

D. *Crossover*

The crossover operator in genetic algorithms is used to combine the genetic information of two parents to create a better offspring. The single point crossover function is used in this project. “A crossover operation intercepts two parent chromosomes at the crossover point, and reconnects the dissected gene segments to create a new chromosome” [11]. The crossover function contains three elements in total: two parents and an offspring. The crossover operator selects an index in the list of indices (ones that have not been used already) and ‘flips’ the list at that index. The cost of the offspring is calculated using the fitness function.





E. Mutation

A simple mutation function was used to implement this project. The mutation probability is set to .01 for optimal results. In this function, an index that has not been visited yet is mutated to a random index that is in the list of indices.

4	0	2	5	1	3
4	0	2	4	1	3

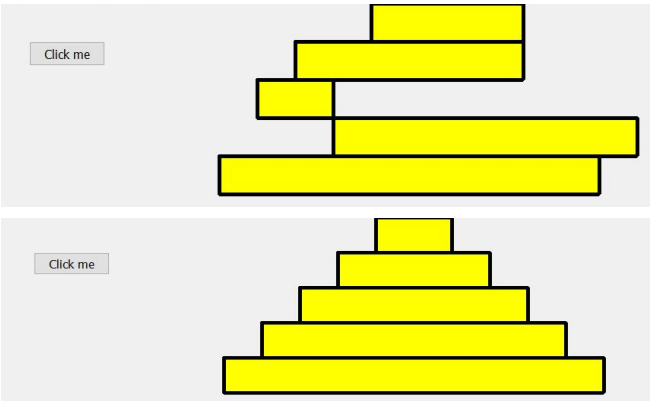
F. Wisdom of Crowds

The wisdom of crowds states that the opinion of group preferred over the opinion of a single expert. The wisdom of crowds chosen for this project was to select the smallest number of flips it takes to get a sorted pancake order, following the NP-complete problem of finding the minimum number of prefix reversals for a string to be sorted. The number of flips from each chromosome list is stored inside a list and the minimum number of flips is chosen at the end. The pancake flipping problem is NP-complete only if the number of flips it takes to sort the pancakes is found. The Wisdom of Crowds approach uses the data from the genetic algorithms to find the minimum number of flips it takes.

Similar to the simple Wisdom of Crowds approach found on the famous game show *"Who Wants to be a Millionaire?,"* the mode and mean of the answers yielded by running the genetic algorithm multiple times is found. After running the GA `number_of_woc_iterations` times, the minimum of all the number of flips required to find the strings for each GA is then found. This minimum is then used to determine the rough upper minimum bound required for *n* number of prefix reversals to occur for the strings to be sorted.

The figure above is an example of efficient flips to sort the order of pancakes where the numbers represent the diameter of pancakes [12]. The sequence of the left, [5,2,3,1,4] can be sorted in 4 flips, where as the sequence of the right, [5,2,3,4,1] cannot. If using an aggregate method to keep count of the most frequent flips and only flipping the stack at that position, it will not produce accurate results.

Once the wisdom of crowds is also completed, the data is ready to be viewed. The final results contain the original string, the number of flips required, and the indices to be flipped in order. The pancakes are flipped until all flips are performed and pancakes are sorted in ascending order starting from the top of the stack.



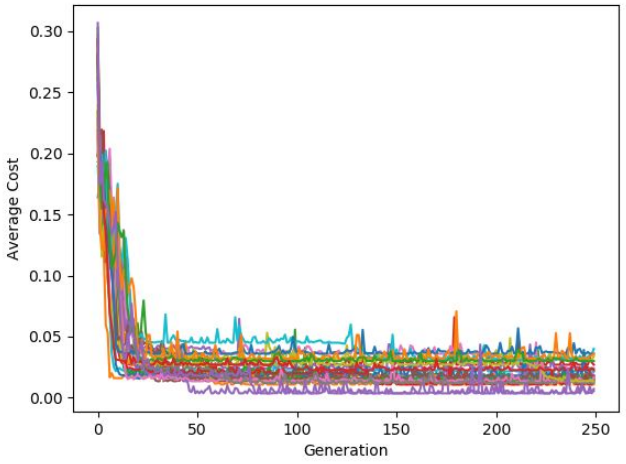
IV. Experimental Results

A. Data

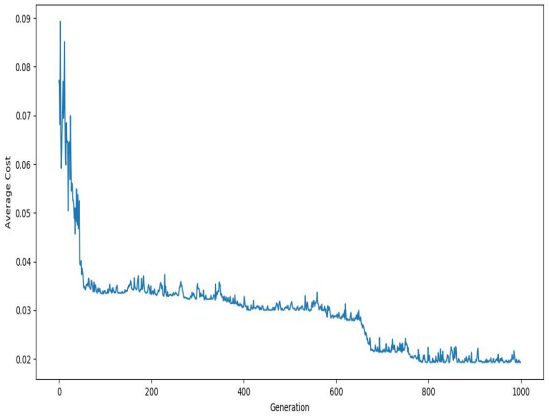
Bespoke .string files were created for this problem, with each .string file containing *n* amount of letters on each line and ending with an empty newline for processing. The read_string function was built to read in these strings and append them to a character array to represent the strings for the prefix reversals to be performed on.

B. *Results*

Below is a graph representing the results of running the Wisdom of Crowds approach over 25 iterations of a string of length 12. Each iteration and their respective costs are illustrated below over 250 generations each. The resulting minimum n number of flips found by this approach was 14 flips, which improves upon the maximum possible number of 17 flips by 30%.



The graph presented below shows the genetic algorithm that is run on a string of 50 characters with a population size of 50. As the number of generation increases, the cost decreases significantly. This results in more efficient solutions that require fewer flips to sort the string. As the solution gets more and more optimal, the results gets farther away from the upper bound on 18n/11.



The following table illustrates the results from comparing the unenhanced GAs with the approach combining Wisdom of Crowds and the GAs on an unsorted input string of 45 characters.

	Unenhanced GA	**WoC & GA (over 35 GAs)**
Average Time Taken	228.34 s	8156.78 s
Number of Flips Required for a String of 45 Characters (Cost)	Best: 86 Average: 101 Worst: 116 Standard Deviation: 8.978	Best: 84 Average: 92 Worst: 98 Standard Deviation: 4.107
Average Improvement over Unenhanced GAs	--	Best: Average: Worst:
Ratio of the percentage of increase of WoC & GAs vs simply Unenhanced GAs In Terms of Running Time	97.20% Decrease from WoC and GA Approach	3472.20% Increase from Unenhanced GA Approach
Ratio of the percentage of increase of WoC & GAs vs simply Unenhanced GAs In Terms of Average Cost	9.78% Cost Increase when compared to the WoC & GA Approach Combined	8.91% Cost Decrease over just the unenhanced GA Approach

The most important finding of the above chart concerns the roughly 9% increase that was found by using the Wisdom of Crowds approach as combined with the unenhanced Genetic Algorithms. The percentage cost as a unit of time is also drastically worse from the unenhanced GA approach, where a 3472% increase in running time cost is found for less than a 9% decrease in

the overall results. This large increase in initial required time, however, does not differ from the results found by other researchers who have also combined the Wisdom of Crowds approach with GAs to NP-hard problems such as Light Up, similar to how Yampolskiy et al. discovered the disadvantage of the large initial time required to produce the initial crowd using the Wisdom of Crowds approach [20].

The exponential growth of the size of the string did not entirely prohibit the ability to discover solutions, and runtime of the sorting would not take more than an hour and a half for a string of 100 characters. The larger the strings, the more time it took for each run of GAs to finish, with a nearly linear time of 250 extra seconds of runtime needed for every 50 extra characters in the input string.

The line graph below analyzed the optimal numbers for the distance and subarray weights for the fitness and cost functions:

The results show that a distance weight of 0.5 and a subarray weight of 1.0 produced the lowest overall cost, so these two parameters were used for the fitness and cost functions when testing the results of the Wisdom of Crowds approach combined with the genetic algorithm.

V. Conclusions

The unenhanced genetic algorithm for approximating the lower bound of the required number of prefix reversal flips for an unsorted string of length *n* was overall successful. The GA proved very successful in efficiently and accurately predicting the minimum number of flips required. The results can continually be improved by continued analysis of tweaking the parameters of the genetic algorithm, with more consideration to the fitness function being a great source for future research. Whether or not the subarray length and occurrences should be weighted more than the actual distance of the chromosome indices when applied to the unsorted strings in particular should lend itself particularly well to future research.

The research going into the Wisdom of Crowds approach as applied to the prefix reversal sorting problem has found a rather minimal increase in using Wisdom of Crowds over the unenhanced genetic algorithm, with a mere 8.91% decrease in the cost of reversals not warranting the nearly 3500% increase in running time over the GAs themselves. It is thus not recommended to use the Wisdom of Crowds approach as applied to finding the minimum bound for the prefix reversal sorting problem.

A further means of combining the genetic algorithms in a more effective way could still be researched in the future. A better percent decrease in the cost than the 8.91% percent decrease in finding the minimum cost in number of prefix reversals could be sought out with an attempt to combine the genetic algorithms in a more homogenous way rather than the heterogeneous approach taken in this project.

VI. Acknowledgements

The authors would like to thank Dr. Roman Yampolskiy and his Teaching Assistants at the University of Louisville for the opportunity to take his wonderful course on Artificial Intelligence that sparked much of the flames of our knowledge and passion concerning genetic algorithms combined with the Wisdom of Crowds approach.

VII. References

[1] P. Compeau, “Sorting by Signed Reversals,” unpublished.
[2] Hayes, Brian. “Sorting Out the Genome.” American Scientist, vol. 95, no. 5, 2007, pp. 386–387.
[3] Bafna, Vineet, and Pavel A Pevzner. “Sorting by Reversals: Genome Rearrangements in Plant Organelles and Evolutionary History of X Chromosome.” Molecular Biology and Evolution, 1995.
[4] Heyer, Laurie J., et al. “Bacterial Computing: Using E. Coli to Solve the Burnt Pancake Problem.” Math Horizons, vol. 17, no. 3, 2010, pp. 5–10.
[5] Gates, William H., and Christos H. Papadimitriou. “Bounds for Sorting by Prefix Reversal.” Discrete Mathematics, vol. 27, no. 1, 1979, pp. 47–57.
[6] B. Chitturi, et al. “An (18/11)n upper bound for sorting by prefix reversals.” Theoretical Computer Science, 2009, pp. 3372–3390.

[8] Papadimitriou C.H. (1997) NP-completeness: A retrospective. In: Degano P., Gorrieri R., Marchetti-Spaccamela A. (eds) Automata, Languages and Programming. ICALP 1997. Lecture Notes in Computer Science, vol 1256. Springer, Berlin, Heidelberg
[9] Cohen DS, Blum M, On the problem of sorting burnt pancakes, Discrete Appl Math 61 (2) :105–120, 1995.
[10] Bafna V, Pevzner PA, Genome rearrangements and sorting by reversals, SIAM J Comput 25 (2) :272–289, 1996.
[11] Zhang J, Kang M, Li X, Liu GY. Bio-Inspired Genetic Algorithms with Formalized Crossover Operators for Robotic Applications. Front Neurorobot. 2017;11:56. Published 2017 Oct 24. doi:10.3389/fnbot.2017.00056
[12]Bulteau, L., Fertin, G., Rusu, I.\ 2011.\ Pancake Flipping is Hard.\ arXiv e-prints arXiv:1111.0434.
[13] Deb, K. Sadhana (1999) 24: 293. https://doi.org/10.1007/BF02823145
[14] Fiori de Castro, H. and Lucchesi Cavalca, K. (2003), "Availability optimization with genetic algorithm", International Journal of Quality & Reliability Management, Vol. 20 No. 7, pp. 847-863.
[15] Wen-Yang Lin, Wen-Yung Lee and Tzung-Pei Hong(2003). "Adapting Crossover and Mutation Rates in Genetic Algorithms", Journal of Information Science and Engineering, Vol 19
[16] Hurkens, Cor, et al. "Prefix Reversals on Binary and Ternary Strings." SIAM Journal on Discrete Mathematics, vol. 21, no. 3, 2007, pp. 592–611.
[17] Chitturi, Bhadrachalam and Sudborough, Ivan. (2010). Prefix Reversals on Strings.. 591-598.
[18] Christie, David A., and Robert W. Irving. "Sorting Strings by Reversals and by Transpositions." SIAM Journal on Discrete Mathematics, vol. 14, no. 2, 2001, pp. 193–206.
[19] Garey, M. R., and D. S. Johnson. "Strong" NP-Completeness Results: Motivation, Examples, and Implications . 1978.
[20] Ashby, Leif H., and Roman V. Yampolskiy. "Genetic algorithm and wisdom of artificial crowds algorithm applied to light up." In 2011 16th International Conference on Computer Games (CGAMES), pp. 27-32. IEEE, 2011.