A Heuristic solution to Closest String Problem Using WaveFunctionCollapse Techniques

Shirley Xu
The Bishop's School

David Perkins
Department of Computer Science
Hamilton College

Abstract

The Closest String problem is an NP-Complete problem which seeks to find the geometrical center of a set of input strings– given k strings of length L and non-negative integer d, find string t, if it exists, such that each input string has Hamming distance $\leq d$ to string t. This research paper proposes an algorithm for the Closest String problem inspired by Wave Function Collapse called WFC-CSP algorithm. Experiment results show that the Heuristic WFC-CSP algorithm is highly reliable, and it has polynomial complexity with respect to number of strings k, string length L, and alphabet size $|\Sigma|$. Target Hamming distance d has negligible effect on the algorithm's complexity. In comparison with Fixed-parameter algorithm (FP) *Grammet.al* proposed in [5], WFC-CSP is significantly faster when $d \geq 10$. The WFC-CSP algorithm for Closest String problem has many applications, such as computational biology and coding theory.

1 Introduction

The Closest String Problem (CSP) is introduced in [1], and is proven to be NP-hard. The problem is also known as the Center String Problem, Hamming Center Problem or Minimum Radius Problem; it tries to find a string that is "close" to all strings in a given set. The "close-ness" of two strings is defined by their Hamming distance, which is further explained later. The Closest String Problem *"comes from coding theory when we are looking for a code not too far away from a given set of codes"* [2], and also has a variety of applications in computational biology, *"such as discovering potential drug targets, creating diagnostic probes, universal primers or unbiased consensus sequences."* [1]

Since the Closest String Problem is NP-hard, it is unlikely that it can be solved with exact algorithms that have a polynomial time complexity. In early attempts, researchers have developed approximation algorithms to solve CSP. *Lanctot et.al* [1] developed the first non-trivial approximation algorithm with a 4/3 approximation ratio, and *Li et. al* [2] presented a Polynomial Time Approximation Scheme (PTAS). In [3] and [4], the time complexity of PTAS was further improved. However, both the approximation nature and time complexity remain concerns of using the PTAS algorithm for the NP-hard CSP.

In [5] *Fixed-parameter algorithms for closest string and related problems, Gramm et. al* proposed an exact solution to the CSP. *Gramm et. al*'s algorithm is fixed-parameter tractable with respect to distance parameter d and has a time complexity of $O(kL + kd \cdot d^d)$. This algorithm finds merits in solving CSP with a small maximum Hamming distance d.

This paper proposes an efficient and reliable Heuristic solution called WFC-CSP, which is inspired by the WaveFunctionCollapse algorithm. WaveFunctionCollapse is a recent algorithm developed by game developer *Maxim Gumin* in 2016 [6] that generates procedural content patterns from a sample image.

The remaining of this paper is organized as follows: section 2 provides a mathematical description of the Closest String Problem (CSP); Section 3 introduces key concepts and ideas of the Wave Function

Collapse algorithm which inspired the WFC-CSP algorithm this paper proposes to solve the Closest String Problem; Section 4 describes in detail the WFC-CSP algorithm; and Section 5 provides experimental results of WFC-CSP algorithm and compares its performance with an algorithm proposed in [2]. Section 6 draws the conclusion of this paper.

2 Closest String Problem Formulation

For any two strings s_1 and s_2 of same length L over alphabet set Σ, we use $d_H(s_1, s_2)$ to denote the Hamming distance between them, which is defined as the number of mismatched positions in the two strings. For example, String s_1 and s_2 over alphabet $\{A, C, G, T\}$ in Figure 1 has a Hamming distance of 3, $d_H(s_1, s_2) = 3$.

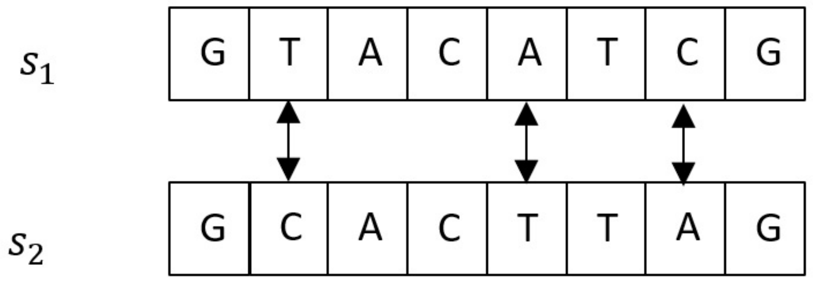

Figure 1: Illustration of Hamming Distance Between Strings

The Closest String Problem (CSP) is formulated as:

Input: Given k strings in input set $S = \{s_1, s_2, \ldots, s_k\}$ over alphabet Σ of length L each, and a non-negative integer d
Question: Is there a string t such that $d_H(t, s_i) \leq d$ for all $i = 1, \ldots, k$?

3 Wave Function Collapse

WaveFunctionCollapse (WFC) is a constraint-based algorithm developed by game developer *Maxim Gumin* in 2016 [6] for procedural content generation. WaveFunctionCollapse is Constraint Solving in the Wild [7] examines WFC as an instance of a constraint solving method and describes the algorithm in detail: *"The key idea is an extension of standard constraint solvers with a "minimal entropy heuristic" that randomly directs the solver's search in a way that follows a user-specified frequency distribution without compromising the efficiency of the search procedure."* [8] In the rare case that a conflict is reached, Gumin's algorithm globally restarts instead of backtracking locally.

In this section, key concepts and ideas of WFC are explained with the example of a Sudoku game. The algorithm for solving the Closest String Problem (CSP) is inspired by these ideas.

As shown in Figure 2, the objective of the Sudoku game is to fill each cell in its 9×9 grids with one number between 1 to 9, such that they satisfy the following **"constraints"**: each column, each row, and each of the nine 3×3 sub grids "boxes" must contain all numbers 1 to 9, and no number may appear twice or more within the same column, row or box. With a blank Sudoku puzzle, every cell has the potential to be any of the nine possible numbers, being in a **"superposition"** occupying all nine possible states at once. When a Sudoku puzzle is initialized with some cells filled, those cells' superpositions are **"collapsed"** to a single possibility, and these collapsed cells will affect the possible values the surrounding cells could take. For example, if a cell is filled with 2, we would know that none of the other cells in the same column, row or 3×3 box may contain 2. We then **"propagate"** this knowledge to those affected cells removing 2 from their superpositions. After propagating all the consequences imposed by the initial condition constraints, the next logical thing to do is to look for the cell with the lowest number of remaining possible states, or to find the cell with the lowest **"entropy"**, and collapse it to a single possibility, followed by propagating the consequence to affected cells. The reason we prioritize the low entropy cells is that there are fewer states left to choose from, so deciding them early will reduce the possibilities that we run into conflicts later. We will continue this process of iterating over the puzzle, collapsing and propagating until all the cells are collapsed to a single possibility and the puzzle is solved.

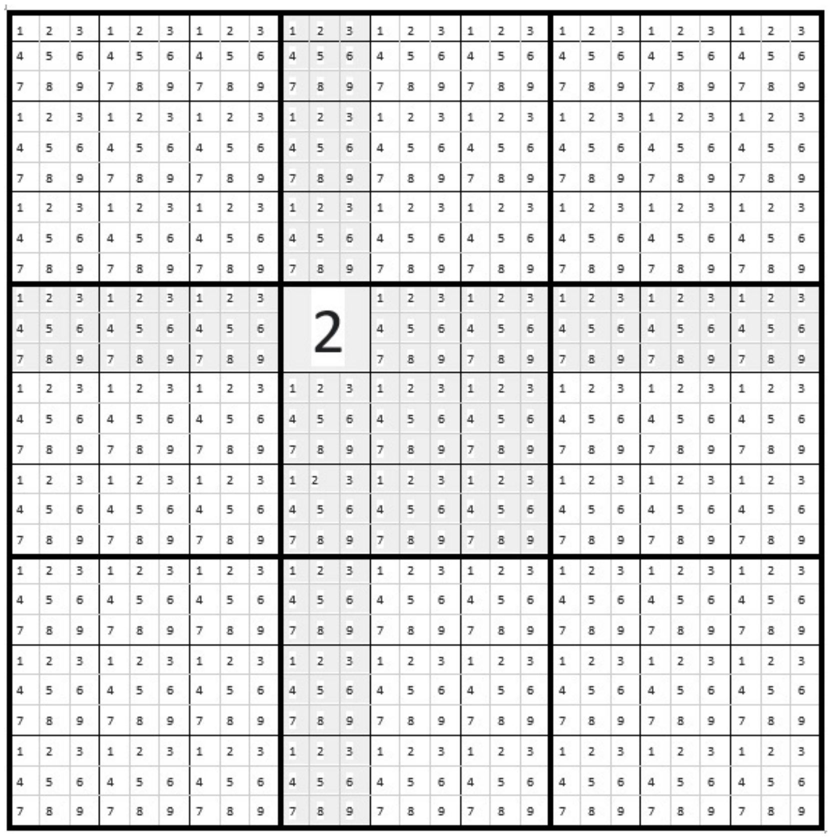

Figure 2: WFC Concepts Explained With Sudoku Game

4 WFC-CSP Algorithm

The Wave Function Collapse Closest String Problem Algorithm (abbreviated as WFC-CSP Algorithm) utilizes the aforementioned characteristics of WaveFunctionCollapse. As described in Section 2, our algorithm's input includes:

- A set of strings $S = \{s_1, s_2, \ldots, s_k\}$ each with length L over the alphabet set $\Sigma = \{A_1, A_2, \ldots, A_N\}$ where $N =$ the number of characters in the alphabet Σ
- Hamming distance parameter d.

Our algorithm checks if the algorithm's output answer string t satisfies $d_H(t, s_i) \leq d$ for all $i = 1, \ldots, k$ If so, our algorithm returns t, which is a string with Hamming distance $\leq d$ to all strings in S, and *"t found"*. If no such answer string can be found with our algorithm, *"t not found"* will be returned.

The algorithm begins by initializing an answer string with L empty positions. The positions each have initial **superpositions** of all characters in Σ. The WFC-CSP algorithm proceeds through multiple passes. In each pass, a decision is made for one position of the answer string. In other words, a certain position in the answer string will be **collapsed** to a single character in Σ.

To determine which position will be collapsed to which character, the WFC-CSP Algorithm utilizes the WaveFunctionCollapse idea of entropy. In the Closest String Problem, the success of an algorithm and the quality of the answer string is measured by the Hamming distance between the answer string and the string in the input set that is farthest from it in Hamming distance. If the maximum Hamming distance is $> d$, the algorithm has failed. Therefore, the goal of the algorithm is to minimize the maximum Hamming distance from the answer string.

At pass l of the WFC-CSP algorithm, we denote the current partial Hamming distance between the partially formed answer string at this point t^l and input string s_i as $d_H^l(t^l, s_i)$, and its value is defined as $d_H(t, s_i)$ with the assumption that all undecided positions in t at this point (pass l) will eventually not match s_i at those positions. Note that $d_H^0(t^0, s_i) = L$ for all $1 \leq i \leq k$ as none of the L positions has been decided at initialization; mismatches would be assumed at all positions.

The WFC-CSP algorithm associates the **"entropy"** of an input string s_i at pass l to its current Hamming distance from the current partial answer string. The input string s_i with the largest current Hamming distance $d_H^l(t^l, s_i)$ has the lowest entropy after pass l. During pass $l+1$, WFC-CSP will target to find a position in t and collapse it with a character such that it will reduce s_i's partial Hamming distance by 1: $d_H^{l+1}(t^{l+1}, s_i) = d_H^l(t^l, s_i) - 1$.

While targeting to help s_i, the "worst" string with the highest partial Hamming distance, to reduce Hamming distance in next pass, the WFC-CSP algorithm also tries to help as many other strings in set S as possible to reduce their Hamming distances in the next pass. With s_i being identified as the worst string in pass l, WFC-CSP finds an "uncollapsed" (i.e. undetermined) position p and character A_j pair at pass $l+1$ such that s_i contains A_j at position p and $\{p, A_j\}$ is the position-character pair with the highest appearance frequency in the remaining input strings across all undetermined positions. This is how the algorithm completes one pass- position p of the answer string is collapsed to the character at index p of the worst string s_i. The Hamming distance of each input string is then updated (or **propagated**) according to the newly collapsed position by either reducing the previous Hamming distance value by 1 or remaining at the previous value. After L passes, the WFC-CSP algorithm will have collapsed all positions in the answer string t.

In the case of ties when determining which string in set S has the worst current Hamming distance, a worst string choice is randomized from the subset of strings that share the worst Hamming distance. In the case of ties when determining which position has the highest frequency of the same character across all strings, a random choice is similarly made from the subset of positions that share the highest frequency of a character. These randomization variations allow our algorithm to obtain different results each time it is run and allows it to viably repeat from the beginning of the WFC-CSP algorithm if it fails the first time.

In the event that the algorithm fails after one run, the WFC-CSP algorithm will globally restart with new randomizations and will keep restarting until either it finds an answer satisfying the requirements, or the maximum repeat count parameter set by the user is reached (whichever comes first). If an application desires to find multiple answer strings satisfying the maximum Hamming distance constraints, WFC-CSP can also be run multiple times even if it succeeds in the first run and obtains different results.

The pseudocode of one run of the WFC-CSP is described below.

WFC-CSP Pseudocode

```
Algorithm 1: findClosestString($S, L, \Sigma, d$)
Input variables:
$S$: Set of strings $S = \{s_1, s_2, \ldots, s_k\}$, each with length $L$
$L$: Length of $s_1, s_2, \ldots, s_k$
$\Sigma$: Alphabet, $\Sigma = \{A_1, A_2, \ldots, A_N\}$, where N=the number of characters in the alphabet
$d$: Maximum Hamming distance
–
Output:
$t$: string with maximum Hamming distance $\leq$ d to all strings in $S$. Otherwise, "t not found".
–
W1: Initialize $t$ as $[A_0, A_0, ..., A_0]$ where $A_0 \notin \Sigma$.
W2: Create a 2D table $LetterFreqTable[L][N]$ where $LetterFreqTable[l][n]$ is the total number
 of occurrence of character $A_n$ at position $l$ of $s_1, s_2, \ldots, s_k$.
W3: Sort $LetterFreqTable$ from highest to lowest.
W4: Initialization
  • W4.1: Initialize $P$ as $1, 2, ..., L$, where $P$ is the set of undecided positions: For all $p \in P$
    $t[p] == A_0$
  • W4.2: initialize set of Hamming distance $d_1, d_2, \ldots, d_k$ as $L, L, ...L$ which is current Hamming
    distance between $t$ and each string in $S$between $t$ and each string in $S$
W5: Loop while P is not empty:
  • W5.1: Find $s_i$ in $\{s_1, s_2, ..., s_k\}$ such that $s_i$ has highest Hamming distance to $t$.
  • W5.2: Find the $1^{st}$ entry in the sorted $LetterFreqTable$ denoted by $LetterFreqTable[p][j]$
    satisfying $s_i[p] == A_j$.
  • W5.3: Change $t[p]$ to $A_j : t[p] = A_j$.
  • W5.4: Remove $p$ from $P$.
  • W5.5: Update Hamming Distance of $t$ to $\{s_1, s_2, \ldots, s_k\}$
W6: Return $t$ if the maximum Hamming distance is $\leq d$. Otherwise, return "$t$ not found".
```

5 Experimental Results

Test cases for experiments in this section are generated as the following:

- A "solution string" s of length L over Σ is generated by randomly deciding character at each position.
- Copy s into each string of $S = \{s_1, s_2, \ldots, s_k\}$
- For each $s_i \in$S=$\{$s$_1, s_2, \ldots, s_k\}$, where $i = 1, 2, \ldots, k$, randomly select d positions and overwrite each position with a randomly selected character in Σ

$S = \{s_1, s_2, \ldots, s_k\}$ and d are provided as inputs to the experiments. With the test case generation procedure above, it is guaranteed that there is at least one string s that satisfies $d_H(s, s_i) \leq d$ for all $i = 1, \ldots, k$.

The WFC-CSP algorithm is implemented in Python 3.8.3, run on PC with 4.00GHz processor and 16GB main memory.

Figure 3 and Figure 4 show the results of the following experiments: WFC-CSP algorithm was run on input test cases under different parameter settings, varying k (number of strings) among $\{10, 20, 40, 80\}$, L (string length) among $\{20, 40, 60, 80, 100, 120, 140, 160, 180, 200\}$, and d (Hamming distance) among $\{5, 10, 20, 40, 80\}$, size of alphabet set Σ is 4 ($|\Sigma| = 4$) in Figure 3, and 8 ($|\Sigma| = 8$) in Figure 4. For each configuration of parameter set, 1000 test cases were run and their average run time (in milliseconds) was recorded and graphed.

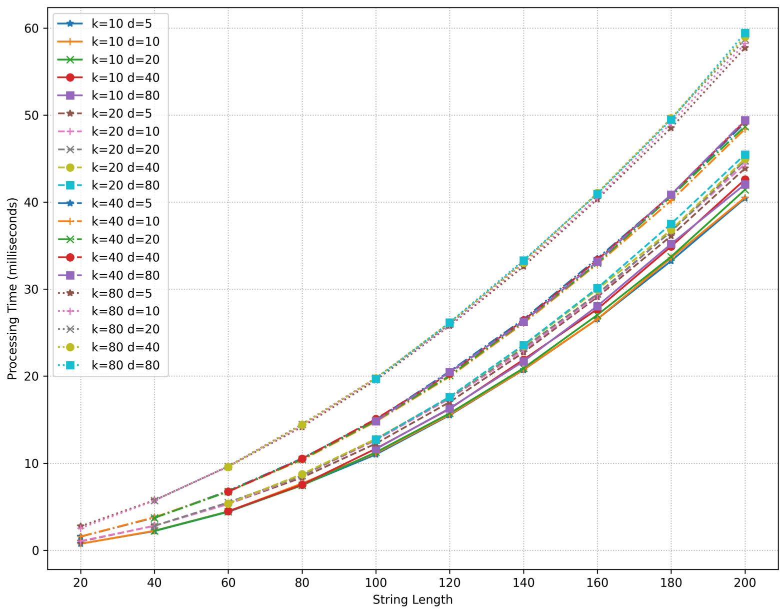

Figure 3: WFC-CSP Algorithm Average Run Time Per Test Case ($|\Sigma| = 4$)

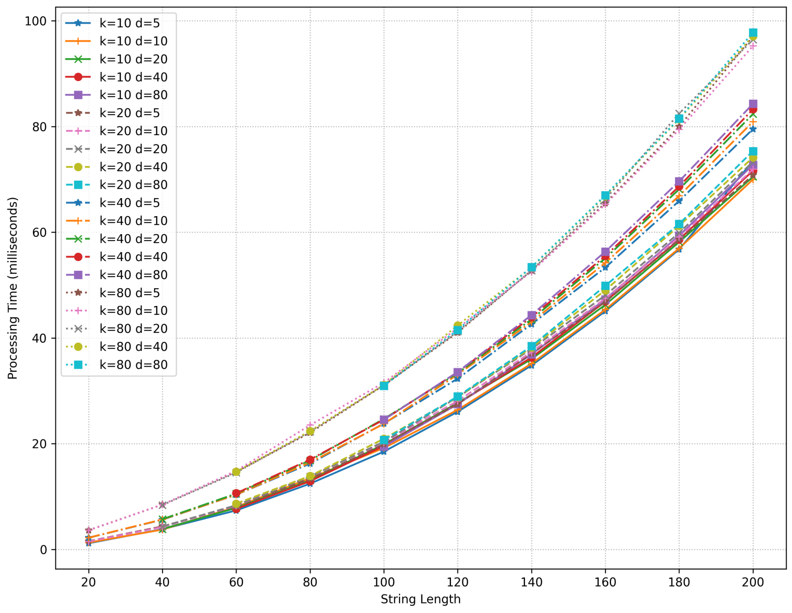

Figure 4: WFC-CSP Algorithm Average Run Time ($|\Sigma| = 8$)

As shown in Figure 3 and Figure 4, the WFC-CSP algorithm is very run time efficient. Its run time has polynomial complexity with respect to k, L and $|\Sigma|$, and has negligible dependency on d: With the same k, L and $|\Sigma|$ parameters, the curves for different d parameters are clustered together.

Among all the tests graphed in Figure 3, there is 1 test case that failed the first iteration of WFC-CSP run, i.e. t is not found such that $d_H(t, s_i) \leq d$ for all $i = 1, \ldots, k$. Among all the tests graphed in Figure 4, there are 26 test cases that failed the first iteration of WFC-CSP run. After global restart and reruns (allowing up to 10 reruns), answer strings t satisfying $d_H(t, s_i) \leq d$ are eventually found for all the test cases failing the first iteration of WFC-CSP run. This shows that WFC-CSP algorithm is highly reliable in finding answer string satisfying Hamming distance requirement.

To compare performance with the Fixed-parameter algorithm (FP) *Grammet.al* proposed in [5], *Grammet.al*'s algorithm is also implemented in Python 3.8.3 and run on the same PC.

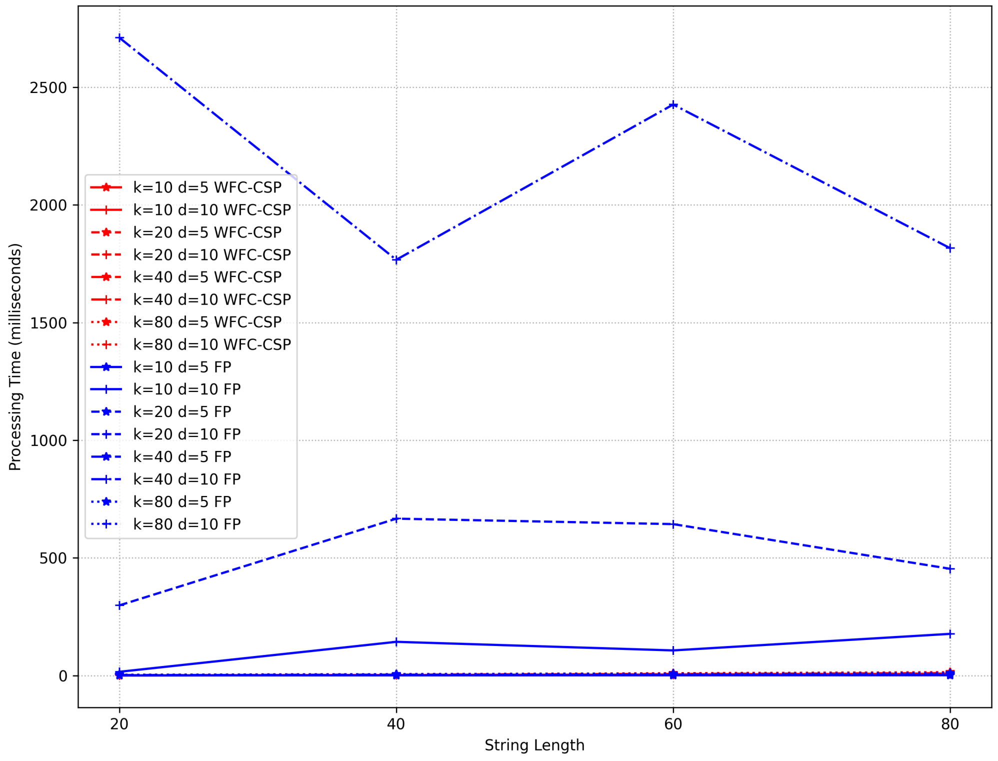

Figure 5: Run Time of WFC-CSP v.s. Fixed-Parameter (FP) Algorithm

Figure 5 shows the comparison results between the two algorithms for $k = \{10, 20, 40, 80\}$, $L = \{20, 40, 60, 80\}$, and $d = \{5, 10\}$, with average run time of 1000 test cases for each set of parameters recorded and graphed. $d = 15$ run was also attempted, however result was not generated for the FP algorithm after overnight run. As shown: when $d = 5$, both algorithms run very efficiently; when $d = 10$, the run time of WFC-CSP is significant faster than FP algorithm. The WFC-CSP algorithm is still very efficient at d values not feasible for the FP algorithm.

6 Conclusion

In conclusion, this paper proposes WFC-CSP algorithm to solve Closest String Problem leveraging WaveFunctioncollapse techniques. It has been shown that the Heuristic WFC-CSP algorithm is highly reliable, and it has polynomial complexity with respect to number of strings k, string length L, and alphabet size $|\Sigma|$, and target Hamming distance d has negligible effect on the algorithm's complexity. In comparison with Fixed-parameter algorithm (FP) *Grammet.al* proposed in [5], WFC-CSP is significant faster when $d \geq 10$.

References

[1] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, Distinguish string search problems, Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 633-642, San Francisco, (1999)

[2] M. Li, B. Ma, and L. Wang. On the Closest String and Substring Problems. Journal of the ACM, 49(2):157–171, 2002.

[3] D. Marx. The closest substring problem with small distances. In FOCS 2005, pages 63–72, 2005

[4] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. In Proceedings of the 12th Annual International Conference on Computational Biology (RECOMB), pages 396–409, 2008.

[5] J. Gramm, R. Niedermeier, and P. Rossmanith, Fixed-parameter algorithms for closest string and related problems. Algorithmica, (2003) 25–42

[6] Maxim Gumin. 2016. WaveFunctionCollapse. https://github.com/mxgmn/ WaveFunctionCollapse. GitHub repository (2016).

[7] Isaac Karth and Adam M. Smith. 2017. WaveFunctionCollapse is Constraint Solving in the Wild. In Proceedings of the 12th International Conference on the Foundations of Digital Games (Hyannis, Massachusetts) (FDG '17). Association for Computing Machinery, New York, NY, USA, Article 68, 10 pages. https: //doi.org/10.1145/3102071.3110566

[8] Adam Newgas, Tessera: A Practical System for Extended WaveFunctionCollapse, https://www.boristhebrave.com/permanent/21/08/Tessera_A_Practical_System_for_WFC.pdf