Received 14 October 2022, accepted 25 October 2022, date of publication 28 October 2022, date of current version 8 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3218003

RESEARCH ARTICLE

A Heuristic Solution to the Closest String Problem Using Wave Function Collapse Techniques

SHIRLEY XU[1] AND DAVID PERKINS[2]
[1]The Bishop's School, La Jolla, CA 92037, USA
[2]Department of Computer Science, Hamilton College, Clinton, NY 13323, USA
Corresponding author: Shirley Xu (shirleytxu@gmail.com)

ABSTRACT The Closest String Problem (CSP) 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 a non-negative integer d, construct a solution string t, if it exists, such that the Hamming distance between t and each input string is no larger than d. This paper proposes WFC-CSP, a novel heuristic algorithm inspired by Wave Function Collapse (WFC) techniques to solve CSP. Experimental results show that WFC-CSP is highly reliable and efficient in solving CSP across different configurations and instance sizes. Using extensive test data sets, WFC-CSP's performance was compared with multiple state-of-the-art algorithms including Gramm et al.'s Fixed-parameter algorithm (FP-CSP), the Ant-CSP algorithm by Faro and Pappalardo using metaheuristic techniques, the third IP formation algorithm by Meneses et al., the LDDA_LSS algorithm by Liu et al., and a sequential version of the heuristic algorithm (Heuris_Seq) by Gomes et al. We observe that WFC-CSP outperforms the other algorithms in solution quality or run time or both metrics. The WFC-CSP algorithm has wide applications in solving CSP in the fields of computational biology and coding theory.

INDEX TERMS WaveFunctionCollapse, closest string problem, NP-complete, NP-hard, heuristic.

I. INTRODUCTION

The Closest String Problem (CSP), introduced in [1], is known to be NP-complete. The problem is also known as the Center String Problem, Hamming Center Problem or Minimum Radius Problem, and has diverse applications in computational biology and coding theory fields. Given a set of strings of the same length L, the Closest String Problem tries to find a solution string of length L that is as close as possible to the input strings. The quality of the solution is evaluated by its distance from the farthest input string, where the distance between two strings is defined by their Hamming distance. Hamming distance and a more formal formation of the Closest String Problem is described in Section II.

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]. It 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]. A common task in these applications is to design a new DNA or protein sequence that is very similar to each given input sequence.

Due to the NP-completeness of the problem, it is unlikely that CSP can be solved with an exact algorithm that has a polynomial time complexity. Researchers have developed different algorithms trying to solve CSP. These algorithms can be characterized in the following categories: approximation algorithms, fixed parameter tractable (FPT) exact solutions, and heuristic algorithms.

The category of approximation algorithms guarantees a bound on the ratio of the solution returned by the algorithm to the objective value of the minimum solution. Lanctot et al. [1] developed the first non-trivial approximation algorithm with a $4/3 + \epsilon$ approximation based on randomized rounding. Li et al. [17] presented a Polynomial Time Approximation Scheme (PTAS) solution to the closest string problem with time complexity $O(L \cdot n^{O(\epsilon^{-5})})$. In [3] and [4], PTAS's time complexity was improved further, achieving $O(L \cdot n^{O(\epsilon^{-2})})$

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Da Lin.

with approximation ratio $1+\epsilon$ with the improvements made in [4]. These algorithms all involve solving integer linear programming, making them not practical in computational complexity in solving large-size instances of the CSP. Meneses et al. [19] have empirically shown the practical use of integer programming techniques to solve moderate-size instances with 10-30 strings of length of 300-800 characters.

Some researchers also approached CSP by studying its parameterized complexity, developing fixed parameter tractable (FPT) exact algorithms. In [5] *Fixed-parameter algorithms for closest string and related problems, Gramm et al.* proposed an exact solution that is fixed-parameter tractable with respect to the maximum Hamming distance parameter d and has a time complexity of $O(kL + kd \cdot d^d)$. The FPT algorithms find merits in solving CSP with a small parameters, but becomes prohibitive in applications when the parameters are large. In [22], Liu et al. presented an exact algorithm called the Distance First Algorithm (DFA), which is specifically for solving CSP with three strings and of alphabet size of two.

Another approach to NP-hard problems is to use heuristic algorithms. In practice, heuristic algorithms are of more interest because of their relative high accuracy and low run time. In [10], Liu et al. proposed heuristic algorithms to solve CSP based on Simulated Annealing (SA) and Genetic Algorithm (GA). In [11], Liu et al. presented a hybrid algorithm that combined the genetic and the simulated annealing approaches, although results were limited to a binary alphabet. In [12], Faro et al. proposed the Ant-CSP algorithm, which is based on the meta-heuristic Ant Colony Optimisation (ACO) in [13]. Ant-CSP showed positive results compared to the GA and SA algorithms in [10]. In [20], Gomes et al. proposed a heuristic (Algorithm 1 in [20]) and implemented the sequential and parallel versions of the algorithm. Later in [21], a slightly different version of the heuristic was proposed and run on a parallel machine with 28 nodes for larger instances of the CSP. In [23], Liu et al. presented a polynomial time approximation algorithm, Largest Distance Decreasing Algorithm (LDDA), based on the idea of the Largest Processing Time algorithm for solving the Job Shop Scheduling Problem. It was improved in [22], where Liu et al. designed a polynomial heuristic LDDA_LSS which is a combination of LDDA in [23] and local search strategies.

This paper proposes WFC-CSP, a new, efficient and reliable heuristic solution to CSP inspired by the WaveFunction-Collapse (WFC) algorithm. Inspired by Paul Merrell's Model Synthesis algorithm [24], WFC was developed by game developer Maxim Gumin [6] and generates procedural content patterns from a sample image. The application of WFC to other NP-complete problems includes the preprint [25], which uses concepts from WFC to heuristically attack the Vertex Color problem.

Using extensive test data sets, we compare WFC-CSP's performance with a variety of algorithms including Gramm et al. Fixed-parameter algorithm (FP-CSP) in [5], the Ant-CSP algorithm using the (meta)heuristic techniques proposed in [12], the third IP formation algorithm in [19], the LDDA_LSS algorithm in [22] and a sequential version of the heuristic algorithm in [21] referred to as Heuris_Seq.

This paper is organized as follows: Section II provides the notations used in the paper and a mathematical description of the Closest String Problem (CSP); Section III introduces concepts of the Wave Function Collapse (WFC) technique, describes the WFC-CSP algorithm in detail, and provides experimental results running WFC-CSP; Section IV compares performance of WFC-CSP with other CSP algorithms; Section V summarizes the conclusion of this paper and describes future research plans.

II. NOTATIONS AND CLOSEST STRING PROBLEM FORMULATION

Let s be a string of length L over alphabet set Σ. Let $s[p]$ indicate the p^{th} character of s, given that p is an integer.

Definition 1 (Hamming Distance): The Hamming distance between two strings s_1 and s_2 of the same length L over alphabet set Σ is denoted by $d_H(s_1, s_2)$. It is defined as the number of positions at which the corresponding characters differ:

$$d_H(s_1, s_2) = \sum_{p=1}^{L} \delta(s_1[p], s_2[p]), \, where:$$
$$\delta(s_1[p], s_2[p]) = \begin{cases} 1 & \text{if } s_1[p] \neq s_2[p] \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

For example, Strings s_1 and s_2 over alphabet $\{A, C, G, T\}$ in Figure 1 have a Hamming distance of 3, or $d_H(s_1, s_2) = 3$.

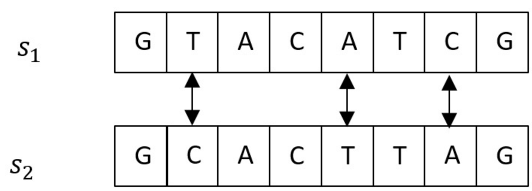


FIGURE 1. Illustration of hamming distance between strings.

Definition 2 (Closest String Problem): The Closest String Problem (CSP) is formulated as:

Input: Given k strings in input set $S = \{s_1, s_2, \cdots, s_k\}$ 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 $i = 1, \cdots, k$?

The notations used in this section also apply to the rest of this paper, in which:

- k: Number of input strings
- L: Length of each input string
- $S = \{s_1, s_2, \ldots, s_k\}$: Set of k strings s_i ($1 \leq i \leq k$), each of length L
- $s_i[p]$: Character at location $p (1 \leq p \leq L)$ of string s_i

- Σ: Alphabet set that the characters of the strings belong to, $\Sigma = \{A_1, \ldots, A_N\}$, where A_j denotes the j^{th} character in Σ, $1 \leq j \leq N$
- $\mid \Sigma \mid$: Size of the alphabet; $\mid \Sigma \mid = N$
- d: Target maximum Hamming distance, or Hamming distance between a solution string and the "farthest" input string as defined in definition 1

In addition to the decision problem version of CSP as defined in Definition 2, another flavor of the CSP does not supply d as an input to target for. Instead, it requires the algorithms to come up with a solution string to minimizing the maximum Hamming distance between the solution string and the input strings.

III. PROPOSED WFC-CSP ALGORITHM

A. WAVE FUNCTION COLLAPSE

WaveFunctionCollapse (WFC) is a constraint-based algorithm that was 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 authors of [8] summarize the ideology of WFC:

> *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.*

In case that a conflict is reached, *Gumin*'s algorithm globally restarts instead of backtracking locally. Key concepts and ideas of WFC can be explained using a Sudoku game (Figure 2) as an example. The objective of Sudoku is to fill each cell in the 9×9 grid with one number from 1 to 9, such that they satisfy the following "**constraints**": each column, each row, and each of the nine 3×3 "boxes" within the grid must contain all numbers from 1 to 9, and no number may appear more than once 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; the cells are in a "**superposition**" occupying all nine possible states at once. When a Sudoku puzzle is initialized with some cells filled, those cells' superpositions have been "**collapsed**" to a single possibility. As we try to solve the game, the logical thing to do is to look for the cell with the lowest number of remaining possible states, or the cell with the lowest "**entropy**," and collapse it to a single value. The knowledge of the newly collapsed cell then "**propagates**" to its surrounded cells, affecting the possible values that those cells could take. We continue this process of iterating over the puzzle, collapsing and propagating until all cells have been collapsed to a single value and the puzzle is solved.

B. WFC-CSP ALGORITHM

The proposed WFC-CSP algorithm utilizes the aforementioned ideology of WaveFunctionCollapse to solve the Closest String Problem as defined in 2. WFC-CSP constructs and returns solution string t if it satisfies $d_H(t, s_i) \leq d$ for all $i = 1, \ldots, k$. If no such solution string can be found, WFC-CSP returns *"t not found"*.

The algorithm begins by initializing a solution string t with L undetermined positions. Each position has an initial **superposition** of all the characters in Σ. The WFC-CSP algorithm proceeds through multiple passes. In each pass, a decision is made for one position of the solution string. In other words, a certain position in the solution string will be **collapsed** to a single character in Σ. One iteration of WFC-CSP is completed after L passes and when all positions of the solution string have been determined.

In the Closest String Problem, the success of an algorithm and the quality of the solution string is measured by the Hamming distance between the solution string t and the string in the input set that is farthest from t. If the Hamming distance between the solution string and one or more input strings is larger than d, the algorithm has failed. Therefore, the goal of the algorithm is to minimize the worst or maximum Hamming distance from the solution string to the input strings.

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

To determine which position will be collapsed to which character, the WFC-CSP algorithm utilizes the WaveFunctionCollapse idea of entropy. The WFC-CSP algorithm associates the entropy of an input string s_i at pass l with its current Hamming distance to t^{l-1}, the current partial solution string. The input string s_i with the largest current Hamming distance $d_H^{l-1}(t^{l-1}, s_i)$ has the lowest entropy. During pass l, WFC-CSP aims to find a position in t and collapses it with a character such that it will reduce s_i's partial Hamming distance by 1: $d_H^l(t^l, s_i) = d_H^{l-1}(t^{l-1}, s_i) - 1$.

While the algorithm aims to help s_i, the "worst" string with the highest partial Hamming distance, when making the collapsing decision for each pass, WFC-CSP also tries to help as many other strings in set S as possible to reduce their Hamming distances. With s_i identified as the worst string in the current pass, WFC-CSP finds an "uncollapsed" (i.e. undetermined) position p and character A_j pair $\{p, A_j\}$, 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 other input strings across all undetermined positions. In Algorithm 1, the appearance frequency of position-character pair is denoted with *CharFreq*, while *scoreboard* denotes the array with *CharFreq* values sorted from highest to lowest. This is how the algorithm completes one pass: position p of the solution string t is collapsed to the character at position p of the worst string s_i. The Hamming distance

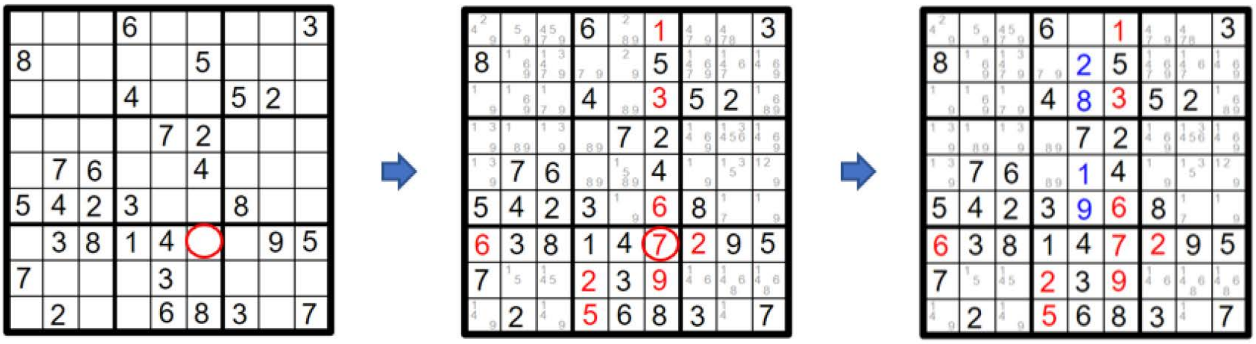


FIGURE 2. WFC concepts explained with sudoku game.

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 solution string t.

In the case that multiple input strings are tied for having the worst current Hamming distance, a randomly selected string among them will be declared as the worst string. In the case that there are 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 characters. These randomization variations encourage solution exploration and bring diversity in WFC-CSP's solutions. They allow the algorithm to produce alternate solutions in different iterations and benefit from running multiple iterations in case of failure within initial attempts.

The WFC-CSP algorithm completes one iteration when all positions in t have been determined. If WFC-CSP is unsuccessful after an iteration, it will globally restart with new randomization until it either finds a solution string that satisfies the requirements, or the maximum iteration parameter (max_iter) set by the user is reached (whichever comes first). If an application requires multiple solution strings that each satisfy the maximum Hamming distance constraints, WFC-CSP can also be run multiple times, even if it succeeds in the first iteration and obtains different results each time.

The pseudocode of one iteration of the WFC-CSP algorithm is described in *Algorithm* 1.

Table 1 illustrates the step-by-step procedure of WFC-CSP solving an example CSP problem with 3 input strings ($k = 3$). The string length L is 5, and the alphabet size N is 4. At each pass, the input strings' partial Hamming distances are calculated (d_1, d_2, d_3). The strings with the largest partial Hamming distances are candidates for the worst string. In case of ties, a random string is chosen among the tied strings. A scoreboard march is then performed to locate the entries with the highest score among the worst string's uncollapsed $\{p, A_j\}$ entries. In case of ties, a random $\{p, A_j\}$ entry is chosen, and location p of the solution string is collapsed to character A_j, completing one pass of WFC-CSP. The complete solution string is constructed after 5 passes. In this example, WFC-CSP constructed a solution string with a worst Hamming distance of 2.

TABLE 1. WFC-CSP example.

(a) Input Strings

Input Strings	Position in String				
	0	1	2	3	4
s_1	A	A	C	G	T
s_2	A	A	G	G	T
s_3	C	A	G	A	A

(b) Character Frequency Scoreboard

WFC-CSP CharFreq Scoreboard		Position in string p				
		0	1	2	3	4
Characters A_j	**A**	2	3	0	1	1
	C	1	0	1	0	0
	G	0	0	2	2	0
	T	0	0	0	0	2

(c) WFC-CSP Procedures

pass #	d_1, d_2, d_3	Worst string ties	Worst string decision	Worst string highest score $\{p, A_j\}$ ties	Collapse decision	Solution
1	5,5,5	s_1, s_2, s_3	s_1	{1,A}	{1,A}	?A???
2	4,4,4	s_1, s_2, s_3	s_3	{2,G}	{2,G}	?AG??
3	4,3,3	s_1	s_1	{0,A}, {3,G}, {4,T}	{0,A}	AAG??
4	3,2,3	s_1, s_3	s_3	{3,A}, {4,A}	{4,A}	AAG?A
5	3,2,2	s_1	s_1	{3,G}	{3,G}	AAGGA
Result	**2,1,2**					**AAGGA**

Algorithm 1 WFC-CSP(S, Σ, d)

Input:
S: Set of k strings $S = \{s_1, s_2, \ldots, s_k\}$, each length L
Σ: Alphabet, $\Sigma = \{A_1, A_2, \ldots, A_N\}$, where $N =\mid \Sigma \mid$
d: Target maximum Hamming distance
Output:
Solution string t with $max(d_H(t, s_i)) \le d, i = 1, \cdots, k$, if exists;
otherwise "not found."

W1: Initialize $t \leftarrow [A_0, A_0, \ldots, A_0]$,$A_0 \notin \Sigma$.
W2: $CharFreq[l][n] \leftarrow \sum_{i=1}^{k} \delta(s_i[l], A_n)$, $(1 \le l \le L, 1 \le n \le N)$

where: $\delta(s_i[l], A_n) = \begin{cases} 1 \text{ if } s_i[l] = A_n \\ 0 \text{ otherwise} \end{cases}$

W3: Sort triplet $\{l, n, CharFreq[l][n]\}$ by $CharFreq[l][n]$ from highest $\rightarrow$ lowest
$scoreboard[m] \leftarrow sorted\{l, n, CharFreq[l][n]\}$
W4: Initialize set of undecided positions $P := \{p \mid t[p] == A_0\}$
Initialize partial Hamming distance $d_i = d_H(t, s_i)$
- **W4.1:** $P \leftarrow \{1, 2, \ldots, L\}$
- **W4.2:** $\{d_1, d_2, \ldots, d_k\} \leftarrow \{L, L, \ldots, L\}$

W5: while P not empty:
- **W5.1:** Choose "worst string" s_i such that its $d_i = max\{d_1, d_2, \ldots, d_k\}$
In the case of ties, randomly choose one of the worst strings as s_i.
- **W5.2:** March along $scoreboard[m]$ to find entries with the biggest $CharFreq[p][j]$ value such that $s_i[p] == A_j$, $p \in P$. In the case of ties, randomly choose one of the tied $\{p, j\}$ pairs.
- **W5.3:** $t[p] \leftarrow A_j$
- **W5.4:** Remove p from P: $P = P - \{p\}$
- **W5.5:** Update $\{d_1, d_2, \ldots, d_k\}$

W6: Return t if $max\{d_1, d_2, \ldots, d_k\} \le d$.
Otherwise, return "t not found."

C. EXPERIMENTAL RESULTS

In this section, we describe the experimental procedures and examine the performance of the WFC-CSP algorithm. WFC-CSP algorithm is implemented in Python 3.8.3, the tests are executed on a PC with a 4.00GHz processor and 16GB main memory.

1) TEST CASE GENERATION AND TEST SETUP

Test cases were created in order to test the performance of the algorithms. The following steps describe the procedure to generate one test case for a given configuration of k strings of length L over Σ, with a specified target maximum Hamming distance of d:

- **T1:** Randomly generate an "answer string" s of length L over Σ.
- **T2:** Initialize input strings: Copy s into each string of $S : s \rightarrow \{s_1, s_2, \ldots, s_k\}$
- **T3:** For each string $s_i \in S = \{s_1, s_2, \ldots, s_k\}$, $(i = 1, 2, \ldots, k)$, randomly choose d different locations and overwrite each chosen location with a randomly selected, different character in Σ

The resulting set of input strings $S = \{s_1, s_2, \ldots, s_k\}$, along with Σ and d are provided as one test case. With this test case generation procedure, it is guaranteed that there exists at least one string s (the answer string from the test case generation procedure) that satisfies $d_H(s, s_i) \le d$ for all $i = 1, \ldots, k$. Upon running the algorithm with the test case, success is declared if the algorithm finds a solution string t that satisfies $d_H(t, s_i) \le d$. This test case generation procedure and the criteria for declared success is consistent with [5].

Test cases of the following various configurations and parameters were generated:

- Alphabet size $N = \{4, 20\}$
- String length $L = \{10, 15, 20, 30, 40, 60, 80, 120, 160, 180, 240, 320\}$
- Number of strings $k = \{10, 20, 40, 80, 120\}$

The alphabet sizes 4 and 20 are important in CSP's practical applications, as they are the number of DNA bases and the number of amino acids, respectively. For each configuration of the parameter set, 1000 test cases were generated.

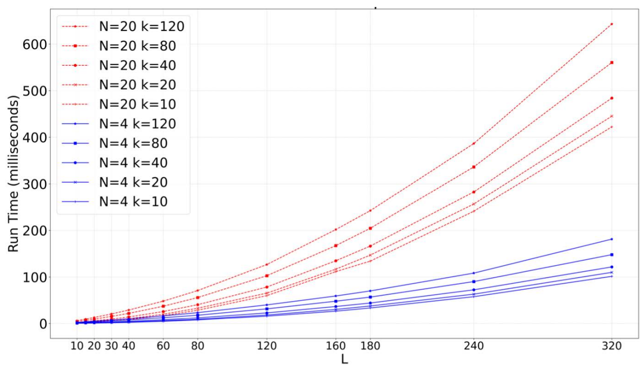


FIGURE 3. WFC-CSP run time per iteration.

2) WFC-CSP SINGLE ITERATION COMPUTATIONAL COMPLEXITY

In order to study the complexity of the WFC-CSP algorithm with respect to values of N, k and L, the run time of one iteration of WFC-CSP was examined. The complexity of one iteration of WFC-CSP does not depend on the target maximum Hamming distance d. For each configuration of the parameter set, 1000 test cases were run. Their average run times (in milliseconds) were recorded and graphed in Figure 3. As shown in Figure 3, the complexity of each iteration of WFC-CSP is tractable with respect to k, L and N, and does not have dependency on d.

3) WFC-CSP PERFORMANCE

The performance of the algorithm under each configuration is measured by:

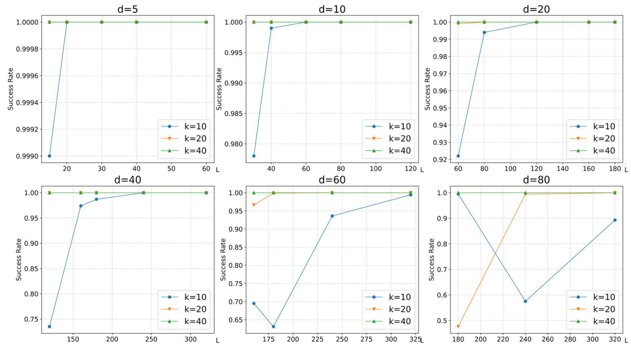


FIGURE 4. **WFC-CSP success rate ($N = 4$).**

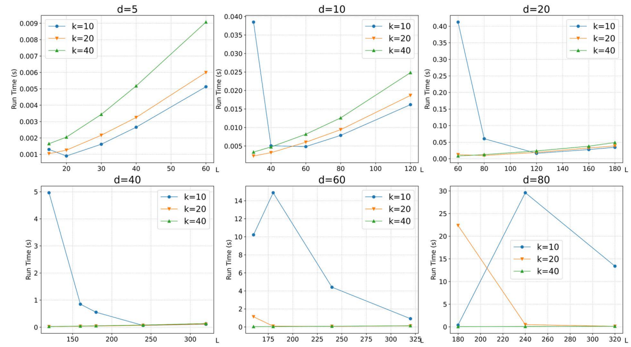


FIGURE 5. **WFC-CSP run time ($N = 4$).**

- Success rate: The percentage of test cases in which the algorithm can find a solution string t for that satisfies $d_H(t, s_i) \leq d$ out of all test cases run
- Run time: Averaged among all test cases to obtain run time per test case measurement

1000 test cases were run for each configuration and performance results were collected.

The overall run time of the WFC-CSP algorithm is dependent on the number of iterations that are performed. If a successful solution was not found after max_iter was reached, the run time of that test case was recorded as the time it took to run all max_iter iterations. Therefore, the run time performance of an algorithm is affected by the max_iter parameter and the number of iterations it actually takes to succeed.

Figure 4 and 5 show WFC-CSP's success rate and run time under different configurations when $N = 4$. max_iter is set as 1000 in this experiment. Please note that experimental results of some configurations ($k > 40$, $L > 320$) were not plotted as their success rates are at or close to 100% in conjuncture with having low run times. As shown in Figure 4 and Figure 5, the "difficulty" level of a test case varies with different configurations. "Easier" cases have success rates either close to or equal to 100%. The small number of iterations needed to solve "easier" cases lead to shorter run times. More "difficult" cases have lower success rates and longer run times as a result of a larger number of iterations.

It is observed by running WFC-CSP that the ratio of L/d has a large impact on the difficulty level of solving CSP: under the same d parameter, the larger L is, the higher the success rate that the algorithm can generally achieve. In other words, problems with a larger L/d ratio are easier to solve. Configurations with larger L/d ratios are of more importance in CSP's practical applications. The same observation has been made in [5] by Gramm et al.

It is also notable that WFC-CSP succeeds with one iteration in all test cases with $L/d \geq 6$. In addition, WFC-CSP is more efficient in finding solutions with larger k values: for all cases with $k = 40$ and $L/d \geq 3$, WFC-CSP succeeds with only one iteration.

When the algorithm fails to find a solution string that meets the target Hamming distance requirement, the actual maximum Hamming distance that the algorithm's achieved solution achieves is examined to study the quality of the solution. This is shown in Table 2 for selected (more "difficult") configurations ($N = 4$). As shown, when WFC-CSP's success rate is not 100%, the average of the maximal Hamming Distance it achieves is very close (with difference being < 1) to the target Hamming distance d.

Figure 6 and Figure 7 show WFC-CSP's success rate and run time under different configurations, with $N = 20$ and $max\_iter = 1000$. WFC-CSP has a generally easier time (higher success rate) solving configurations with a higher alphabet size. All configurations with $d \leq 40$ have a success rate equal to or close to 100%. Please note that experimental results of some configurations ($k > 20$, $L > 320$) were not plotted as their success rates are consistently at or close to 100% with low run times.

Table 3 examines the actual maximum Hamming distance of the algorithm's achieved solutions for selected (more "difficult") configurations ($N = 20$). As shown, when WFC-CSP's success rate is not 100%, the average of the maximum Hamming distance is very close (with difference being <1) to the target Hamming distance d.

4) WFC-CSP MAX_ITER PARAMETER CASE STUDY

For "difficult" configurations, WFC-CSP needs more iterations to achieve a high success rate. The most difficult configuration, or the configuration with the lowest success rate ($N = 4, k = 10, d = 60, L = 180$) in III-C is further examined in Figures 8. Figure 8a graphs WFC-CSP's success rate and run time for max_iter values varying between {200, 300, 500, 1000, 1500, 2000, 3000}. WFC-CSP's success rate improves from 52.6% to 67.4% as max_iter increases from 200 to 3000. This, in turn, results in a longer run time. Figure 8b shows a histogram that depicts the actual number of iterations WFC-CSP takes to solve a test case. The histogram contains 100 bins, each represents 30 iterations. Out of the 1000 test cases, 402 cases succeeded within 30 iterations; 634 cases succeeded within 1000 iterations; in 326 cases, WFC-CSP was not able to find a solution satisfying the d requirement after 3000 iterations. Even for a scenario such a low success rate, it will be shown in section III-C5 that

TABLE 2. **WFC-CSP solution max HD (N = 4).**

d	k	L	Success Rate	Solution Max HD (Average)
40	10	120	0.735	40.307
		160	0.974	40.028
		180	0.987	40.014
		240	1	40
		320	1	40
	20	120	1	40
		160	1	40
		180	1	40
		240	1	40
		320	1	40
	40	120	1	40
		160	1	40
		180	1	40
		240	1	40
		320	1	40
60	10	160	0.695	60.407
		180	0.631	60.446
		240	0.936	60.068
		320	0.994	60.007
	20	160	0.966	60.04
		180	0.999	60.002
		240	1	60
		320	1	60
	40	160	1	60
		180	1	60
		240	1	60
		320	1	60
80	10	180	0.995	79.811
		240	0.575	80.536
		320	0.893	80.112
	20	180	0.476	80.89
		240	0.994	80.006
		320	1	80
	40	180	1	80
		240	1	80
		320	1	80

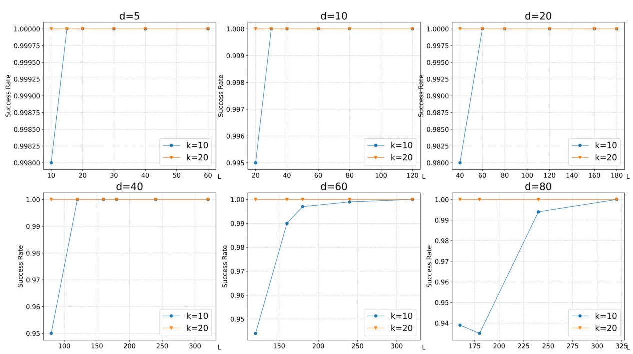


FIGURE 6. **WFC-CSP Success Rate ($N = 20$).**

the worst case solution WFC-CSP finds is not far from the target d.

5) MORE STATISTICAL EXPERIMENTAL RESULTS

Next, we examine the consistency of WFC-CSP's solution quality by running the algorithm on large number of test cases

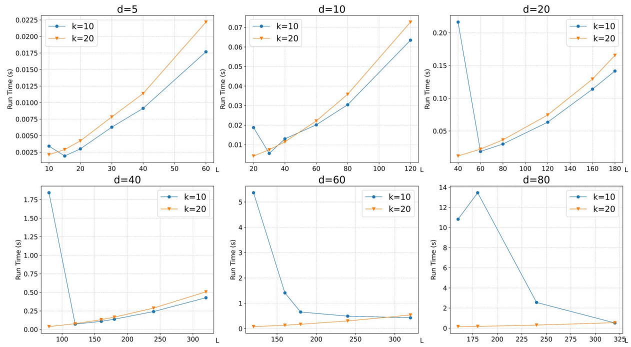


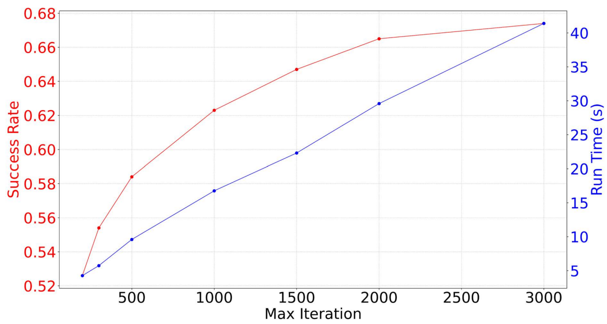
FIGURE 7. **WFC-CSP run time ($N = 20$).**

TABLE 3. **WFC-CSP solution max HD (N = 20).**

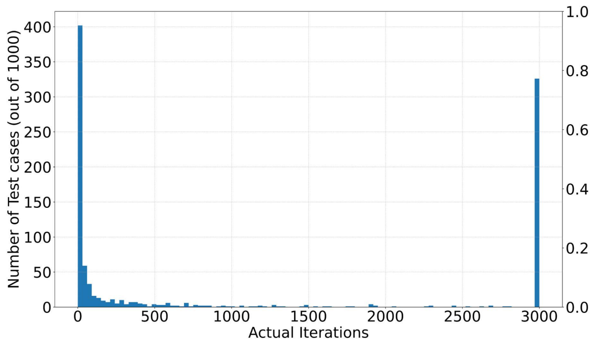
d	k	L	Success Rate	Solution Max HD (Average)
40	10	80	0.95	40.05
		120	1	40
		160	1	40
		180	1	40
		240	1	40
		320	1	40
	20	80	1	40
		120	1	40
		160	1	40
		180	1	40
		240	1	40
		320	1	40
60	10	120	0.944	60.056
		160	0.99	60.01
		180	0.997	60.003
		240	0.999	60.001
		320	1	60
	20	120	1	60
		160	1	60
		180	1	60
		240	1	60
		320	1	60
80	10	160	0.939	80.061
		180	0.935	80.065
		240	0.994	80.006
		320	1	80
	20	160	1	80
		180	1	80
		240	1	80
		320	1	80

and collecting statistics of the solutions obtained. Test cases generated in III-C1 are used, however instead of supplying d as input for the algorithm to target for, in this experiment, WFC-CSP was run 20 iterations on each test case, and the minimal max HD obtained from those runs was taken as WFC-CSP's solution. Results for selected configurations (same as the ones in Table 2 and 3) were collected.

For each configuration, 1000 test cases were run and the minimum, average, maximum and standard deviation statistics of the solutions were analyzed. The smaller WFC-CSP number of iterations (20) and the large number of test



(a) WFC-CSP Success Rate vs. Iterations



(b) WFC-CSP Iterations Histogram

FIGURE 8. **WFC-CSP iterations case study ($N = 4, k = 10, d = 60, L = 180$).**

TABLE 4. **WFC-CSP solution statistics (N = 4).**

Instance			WFC-CSP (max_iter=20)			
d	k	L	Min	Avg.	Max	Std.
40	10	120	40	40.5	42	0.50
		160	40	40.1	41	0.22
		180	40	40.0	42	0.16
		240	40	40	40	0
		320	40	40	40	0
	20	120	40	40	40	0
		160	40	40	40	0
		180	40	40	40	0
		240	40	40	40	0
		320	40	40	40	0
	40	120	40	40	40	0
		160	40	40	40	0
		180	40	40	40	0
		240	40	40	40	0
		320	40	40	40	0
60	10	120	57	58.9	61	0.69
		160	59	60.6	62	0.50
		180	60	60.6	62	0.48
		240	60	60.1	61	0.30
		320	60	60.0	61	0.11
	20	120	60	61.2	62	0.47
		160	60	60.0	62	0.21
		180	60	60.0	61	0.04
		240	60	60	60	0
		320	60	60	60	0
	40	120	60	60.0	62	0.23
		160	60	60	60	0
		180	60	60	60	0
		240	60	60	60	0
		320	60	60	60	0
80	10	160	75	78.1	80	0.79
		180	77	79.5	81	0.66
		240	79	80.7	82	0.47
		320	80	80.2	81	0.40
	20	160	80	81.3	83	0.46
		180	80	80.8	82	0.63
		240	80	80.0	81	0.09
		320	80	80	80	0
	40	160	80	80.1	82	0.34
		180	80	80	80	0
		240	80	80	80	0
		320	80	80	80	0

cases (1000) per configuration were chosen in order to observe the robustness and consistency of WFC-CSP's solution quality.

Table 4 and 5 show test results for $N = 4$ and $N = 20$ cases respectively, Out of the 1000 test cases run for each configuration, minimum, average, maximum values of the solutions found by WFC-CSP for those test cases, as well as the standard derivation among solutions are provided in "Min", "Avg.", "Max" and "Std." columns. The worst case standard deviation is 0.79, which is with the configuration ($d = 80, N = 4, k = 10, L = 160$), the difference between the minimum and maximum solution values is 5 in this case, which is 6.25% of the target d (80). The small differences between minimum and maximum solutions and the small standard deviation values in Table 4 and 5 demonstrated that WFC-CSP is robust and consistent when tested on large amount of test cases.

6) DISCUSSIONS

This section further discusses some of the findings from the WFC-CSP experiments.

The computation complexity of each iteration of WFC-CSP is tractable with respect to k, L, N, and does not have dependency on d. This is because that during each iteration, WFC-CSP constructs a solution string without the knowledge of d to target for. Performance of WFC-CSP can be improved by running more iterations of the algorithm. Each iteration could potentially generate a different solution due to tie-breaking randomization process of the algorithm. Running more iterations in turn increases the run time of the algorithm.

It can be observed that configurations with smaller L/d ratio, smaller alphabet size, and/or fewer number of strings are more difficult to solve. An intuitive way to understand this

TABLE 5. WFC-CSP solution statistics (N = 20).

Instance			WFC-CSP (max_iter=20)			
d	k	L	Min	Avg.	Max	Std.
40	10	80	40	40.2	41	0.37
		120	40	40	40	0
		160	40	40	40	0
		180	40	40	40	0
		240	40	40	40	0
		320	40	40	40	0
	20	80	40	40	40	0
		120	40	40	40	0
		160	40	40	40	0
		180	40	40	40	0
		240	40	40	40	0
		320	40	40	40	0
60	10	120	60	60.3	61	0.45
		160	60	60.0	61	0.13
		180	60	60.0	61	0.06
		240	60	60.0	61	0.04
		320	60	60	60	0
	20	120	60	60	60	0
		160	60	60	60	0
		180	60	60	60	0
		240	60	60	60	0
		320	60	60	60	0
80	10	160	80	80.3	81	0.46
		180	80	80.2	81	0.42
		240	80	80.0	81	0.11
		320	80	80	80	0
	20	160	80	80	80	0
		180	80	80	80	0
		240	80	80	80	0
		320	80	80	80	0

is that, under these cases the entropy and scoreboard values among different partial solutions may be more "clustered", i.e. more choices have similar scores. Therefore it is more likely that the algorithm makes a locally optimal decision that turns out to not be the best decision globally.

The WFC-CSP algorithm is robust in generating solutions with consistency in solution quality. It also provides a simple knob (the *max_iter* parameter) for trade-off between solution quality and run time.

Being a heuristic algorithm, WFC-CSP can not guarantee the optimal solution can be found. The randomness in solution exploration comes from breaking the ties as described in algorithm 1. This randomization criteria and process could be an area for future algorithm improvement to allow more solution exploration.

One limitation with test cases generated using the procedure in section III-C1 is: although d is used as target distance, it is potentially possible that the actual maximum Hamming distance among strings in S is less than d. As we use the same test cases to compare performance of different algorithms, this limitation does not affect the fairness as the same success criteria is used across algorithms under comparison.

IV. COMPARING WFC-CSP WITH OTHER ALGORITHMS

A. WFC-CSP VS. FP-CSP

The Fixed-parameter algorithm for CSP (abbreviated FP-CSP in this paper) is proposed by Gramm et al. in [5], the strategy is described in [5] as:

> *Start with the one of the given strings, e.g., s_1, as a "candidate string." If there is a string s_i, $i = 2, \ldots, k$, that differs from the candidate string in more than d positions, we recursively try several ways to move the candidate string "towards" s_i; moving closer here means that we select a position in which the candidate string and s_i differ and set this position in the candidate string to the character of s_i at this position. We stop either if we moved the candidate "too far away" from s_1 or if we found a solution. By a careful selection of subcases of this recursion we can limit the size of this search tree to $O(d^d)$.*

The pseudocode of *Gramm* et al.'s recursive algorithm (referred to as FP-CSP) is described in *Algorithm* 2.

Algorithm 2 FP-CSP Recursive Procedure $CSd(s, \Delta d)$

Global variables: Set of strings $S = \{s_1, s_2, \cdots, s_k\}$, integer d.
Input: Candidate string s and integer Δd.
Output: A string $\hat{s}$ with $max_{i=1,\cdots,k}, d_H(\hat{s}, s_i) \leq \Delta d$ if it exists;
"not found" otherwise.
(D0) If $(\Delta d < 0)$, then return "not found";
(D1) If $(d_H(s, s_i) > d + \Delta d)$ for some $i \in \{1, \cdots, k\}$ then return "not found";
(D2) If $(d_H(s, s_i) \leq d)$ for all $i = 1, \cdots, k$ then return s;
(D3) Choose some $i \in \{1, \cdots, k\}$ such that $(d_H(s, s_i) > d$:
  $P := \{p \mid s[p] \neq s_i[p]\}$;
  Choose any $P' \subseteq P$ with $| P' | = d - 1$;
  For all $p \in P'$ do
    $s' := s$;
    $s'[p] := s_i[p]$;
    $s_{ret} := CSd(s', \Delta d - 1)$;
    $If s_{ret} \neq$ "not found" then return s_{ret};
(D4) Return "not found"

[5] proves the correctness of the algorithm, determines its time complexity of $O(kL + kd \cdot d^d)$, and shows empirical results with different k, d, and L parameters over $\Sigma = \{A, C, G, T\}$. Aside from d, the ratio L/d also has a major impact on the difficulty of the Closest String Problem. Smaller L/d ratios significantly increase the running time of the algorithm.

Both WFC-CSP and FP-CSP are implemented in the same language and tested on the same machine as described

			N=4				N=20			
			WFC-CSP	Run time(s)			WFC-CSP	Run time(s)		
d	k	L	Success Rate	WFC-CSP	FP-CSP	FP/WFC Ratio	Success Rate	WFC-CSP	FP-CSP	FP/WFC Ratio
10	10	30	0.98	0.012	1.209	104.8	1.00	0.007	0.372	55.0
		40	1.00	0.003	0.152	46.9	1.00	0.011	0.102	9.7
	20	30	1.00	0.003	0.120	47.3	1.00	0.007	0.092	13.0
		40	1.00	0.004	0.017	4.7	1.00	0.011	0.013	1.2
12	10	36	0.97	0.033	4.942	149.0	1.00	0.008	13.059	1724.4
		48	1.00	0.006	5.066	898.0	1.00	0.012	0.038	3.1
	20	36	1.00	0.003	1.819	553.7	1.00	0.010	1.131	109.1
		48	1.00	0.005	0.003	0.5	1.00	0.016	0.022	1.4
14	10	36	0.85	0.085	TO	-	1.00	0.011	TO	-
		48	0.99	0.012	29.695	2577.2	1.00	0.013	35.113	2705.9
		60	1.00	0.006	0.002	0.3	1.00	0.019	1.207	62.6
	20	36	1.00	0.004	210.811	51496.9	1.00	0.010	453.115	44682.2
		48	1.00	0.005	0.229	47.4	1.00	0.017	60.321	3626.5
16	10	48	0.93	0.061	152.139	2492.2	1.00	0.013	TO	-
		60	0.99	0.014	TO	-	1.00	0.020	TO	-
	20	48	1.00	0.005	TO	-	1.00	0.016	TO	-
		60	1.00	0.007	TO	-	1.00	0.024	173.315	7358.5
20	10/20	36-72	varies	varies	TO	-	1.00	varies	TO	-

TABLE 6. Comparison of WFC-CSP vs. FP-CSP algorithm.

in III-C. The same test cases generated with the procedure in III-C1 were used in the experiments. The configurations of the test cases include:

- $N = \{4, 20\}$
- $k = \{10, 20\}$
- $d = \{10, 12, 14, 16, 20\}$
- $L = \{30, 36, 40, 48, 60, 72\}$

FP-CSP is an exact algorithm, and runs recursively until it successfully finds a solution. Therefore, its success rate is always 100%. In this experiment, WFC-CSP's max_iter is set to 200. A test case "times out" (TO) if the run time per test case exceeds 1000 seconds. As shown in Table 6, WFC-CSP consistently has a higher run time than FP-CSP while maintaining high success rates. When $d \geq 16$, FP-CSP times out in all configurations except one for both N = 4 and N = 20.

B. WFC-CSP VS. ANT-CSP

In [12], Faro et al. proposed an Ant Colony Optimisation (ACO) metaheuristic-based algorithm to solve CSP. ACO was first proposed by Dorgio et al. in [13] and [15] as an innovative approach to the Traveling Salesman Problem (TSP). Inspired by the foraging behavior of social ants in a colony, ACO focuses on the indirect communication among ants with chemical pheromone trails. In nature, these pheromone trails help ants find the shortest path between a food source and their colony. When ants walk from a food source to the colony and vice versa, they deposit a pheromone substance. Ants can smell pheromones, and are more likely to traverse paths with strong pheromone concentrations. The pheromone evaporates over time. Thus, the pheromone concentrations on shorter paths are higher. They get walked over faster, resulting in more pheromone deposits. There are several different variants of the ACO. The algorithm this paper uses as a comparison to the proposed WFC-CSP is based on and leveraged from [12]. The pseudocode is provided in [12] as follows:

Algorithm 3 Ant-CSP(S)

initialize table T
for $i \leftarrow 1$ to L **do**
  for $j \leftarrow 1$ to $\mid \Sigma \mid$ **do**
    $T_{ij} \leftarrow 1/\mid \Sigma \mid$
  end for
end for
initialize *COLONY*
while not (TERMINATION_CRITERION) **do**
  for $i \leftarrow 1$ to u **do**
    $COLONY_i \leftarrow new\_ant()$
    $COLONY_i.find\_solution()$
    $COLONY_i.evaluate\_solution()$
  end for
  for $i \leftarrow 1$ to L **do** ▷ start pheromone evaporation
    for $j \leftarrow 1$ to $\mid \Sigma \mid$ **do**
      $T_{ij} \leftarrow (1 - \rho) \cdot T_{ij}$;
    end for
  end for ▷ end pheromone evaporation
  $COLONY_{best}.update\_trails()$
end while

The Ant Colony System's (ACS) state transition rule [15] is described in *find_solution()* and applied when the ant makes a decision, namely:

$$s = \begin{cases} argmax_{u \in J_k(r)}\{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}\} \\ \quad \text{if } q \leq q_0 \text{ (exploitation)} \\ S, \quad \text{otherwise (biased exploration)} \end{cases} \tag{2}$$

where τ is the pheromone, η is a local performance metric, β is a parameter which determines the relative importance of τ versus η ($\beta > 0$), q is a random number uniformly distributed in $[0 \ldots 1]$, q_0 is the exploitation probability parameter ($0 \leq q_0 \leq 1$), and S is a random variable selected according to the probability distribution given in

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,u)] \cdot [\eta(r,u)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

An elitist strategy is adopted– only the ant that has produced the best solution, $COLONY_{best}$, is allowed to update the pheromone trails. The amount of pheromone deposited is proportional to the quality of the solution built. In particular:

$$\tau^{(t+1)}[i,j] = \tau^{(t)}[i,j] + (1 - \frac{HD}{L}) \tag{4}$$

where HD is the maximum Hamming distance of the current string from all strings in S.

Both WFC-CSP and Ant-CSP are implemented in the same language and tested on the same machine as described in III-C. The same test cases generated with the procedure in III-C1 were used in the experiments. The configurations of the test cases include:

- Alphabet size $N = \{4, 20\}$
- Target maximum Hamming distance $d = \{5, 10, 20, 40, 60, 80\}$
- String length $L = \{10, 15, 20, 30, 40, 60, 80, 120, 160, 180, 240, 320\}$
- Number of strings $k = \{10, 20, 40\}$

In this experiment, WFC-CSP's max_iter is set to 200. The maximum iteration (max_iter) of the Ant-CSP is set to 1000 with 10 agents (ants) constructing solutions in each iteration. Early termination is allowed in both algorithms (the algorithm terminates if a solution is found before reaching max_iter). The β, ρ and q_0 parameters of the Ant-CSP algorithm were optimized by running CSP experiments using different parameter values and choosing the parameter combination that yielded the highest success rate. The following parameter values were tested in the process of parameter optimization:

- $\beta = \{1, 2, 3\}$
- $\rho = \{0, 0.0001, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005, 0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}]$
- $q_0 = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

Figure 9 and 10 graph comparisons between the success rates and run times of WFC-CSP and Ant-CSP when $N = 4$. Figure 11 and 12 compare the success rates and run times of WFC-CSP and Ant-CSP when $N = 20$.

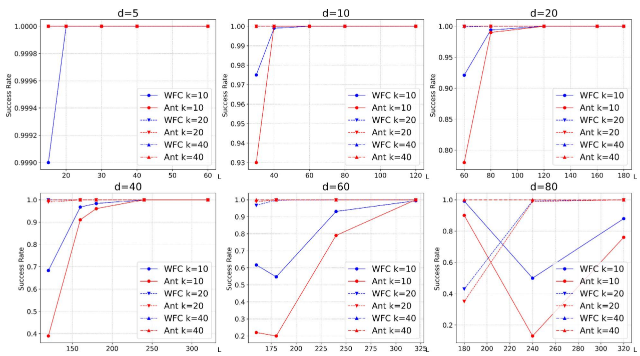


FIGURE 9. WFC-CSP vs. Ant-CSP success rate ($N = 4$).

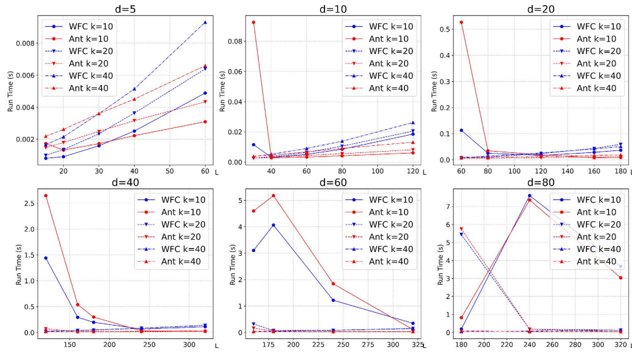


FIGURE 10. WFC-CSP vs. Ant-CSP run time ($N = 4$).

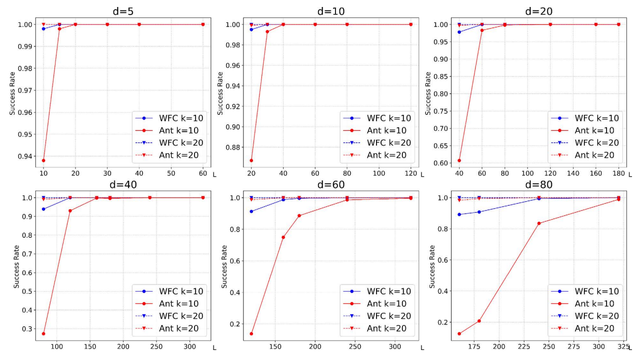


FIGURE 11. WFC-CSP vs. Ant-CSP success rate ($N = 20$).

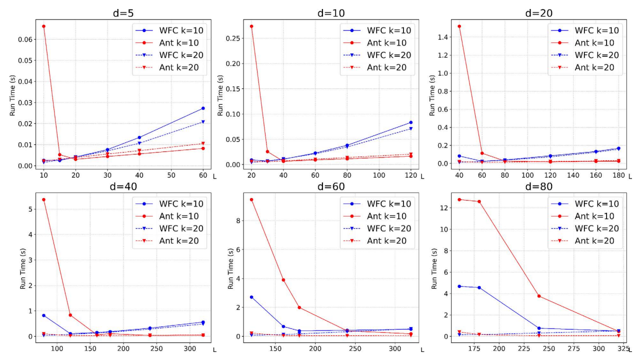


FIGURE 12. WFC-CSP vs. Ant-CSP run time ($N = 20$).

In the case that the algorithm fails on a test case, the difference between the target d and the maximum Hamming Distance achieved by the algorithm is examined. Table 7 contains select comparisons of WFC-CSP and Ant-CSP on success rate, average maximum Hamming distance, and run time.

As shown in Figure 9, 10, 11, and Table 7, WFC-CSP performs consistently better than Ant-CSP in achieving higher success rates, while having similar run times. In many cases, WFC-CSP offers both higher success rates and lower run times than Ant-CSP. The average maximum Hamming distance achieved by WFC-CSP is also consistently lower than that of Ant-CSP.

C. WFC-CSP VS. MORE CSP ALGORITHMS

In [23], Liu et al. designed a polynomial time approximation algorithm: Largest Distance Decreasing Algorithm (or LDDA in short). This algorithm, however, frequently encountered local optima. In [22], Liu et al. improved the efficiency of LDDA by combining it with local search strategies, and named the new proposed heuristic LDDA_LSS. The authors of [22] compared the performance of LDDA_LSS with the third IP formulation algorithms in [19] (or "Exact" as it was called in [22]) and with a sequential version of the heuristic algorithm in [21] (referred to as Heuris_Seq).

The authors of [22] use real and simulated biological data to test their algorithms. The McClure instances in [18] are

TABLE 7. Comparison of WFC-CSP vs. Ant-CSP algorithm.

			N=4							N=20						
			Success rate		Ave max HD		Run time(s)			Success rate		Ave max HD		Run time(s)		
d	k	L	WFC-CSP	Ant-CSP	WFC-CSP	Ant-CSP	WFC-CSP	Ant-CSP	Ant/WFC Ratio	WFC-CSP	Ant-CSP	WFC-CSP	Ant-CSP	WFC-CSP	Ant-CSP	Ant/WFC Ratio
10	10	30	**0.98**	0.93	**10.03**	10.07	**0.012**	0.093	8.0	**1.00**	0.99	**10.00**	10.01	**0.007**	0.026	3.8
10	10	40	**1.00**	1.00	**10.00**	10.00	**0.003**	0.003	0.9	**1.00**	1.00	**10.00**	10.00	**0.011**	0.006	0.6
40	10	120	**0.68**	0.39	**40.36**	40.62	**1.441**	2.647	1.8	**1.00**	0.93	**40.00**	40.07	**0.105**	0.836	7.9
40	10	160	**0.97**	0.91	**40.04**	40.09	**0.295**	0.541	1.8	**1.00**	1.00	**40.00**	40.00	**0.147**	0.062	0.4
40	10	180	**0.98**	0.96	**40.02**	40.04	**0.198**	0.299	1.5	**1.00**	1.00	**40.00**	40.00	**0.182**	0.103	0.6
60	10	160	**0.62**	0.22	**60.49**	60.79	**3.103**	4.597	1.5	**0.99**	0.75	**60.01**	60.26	**0.675**	3.899	5.8
60	10	180	**0.55**	0.20	**60.54**	60.81	**4.063**	5.178	1.3	**1.00**	0.89	**60.00**	60.11	**0.374**	1.989	5.3
60	10	240	**0.93**	0.79	**60.07**	60.21	**1.218**	1.843	1.5	**1.00**	0.99	**60.00**	60.01	**0.419**	0.374	0.9

TABLE 8. Results for McClure dataset with an alphabet of 20 characters.

Instance			Exact	Parallel	LDDA_LSS	Heuris_Seq	WFC-CSP
Name	k	L	Val.	Val.	Val.	Val.	Val.
Mc582.12.seq	12	141	97	100	100.7	99.6	98
Mc582.10.seq	10	141	97	101	100.3	99.6	98
Mc582.6.seq	6	141	95	101	-	-	95
Mc586.12.seq	12	98	77	79	81	80.3	77
Mc586.10.seq	10	98	75	78	79.3	78.3	76
Mc586.6.seq	6	100	70	77	71.3	74.3	72

TABLE 9. Results for small-size simulated data with an alphabet of 2 characters.

Instance		Exact	LDDA_LSS				Heuris_Seq				WFC-CSP (max_iter=10)				WFC-CSP (max_iter=200)			
k	L	Avg.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.
10	100	39	40	40.3	41	0.47	39	41	43	1.63	40	40.7	41	0.58	39	**39.3**	40	0.58
10	200	75	76	76.7	78	0.94	75	81.3	86	4.64	73	77.3	81	4.04	73	**76.3**	80	3.51
10	300	113	116	116.7	118	0.94	117	123.3	128	4.64	115	116	117	1	115	**115.3**	116	0.58
20	100	43.3	44	44.7	45	0.47	44	**44.3**	45	0.47	45	45.3	46	0.58	44	44.7	45	0.58
20	200	84	86	86.3	87	0.47	88	89.7	92	1.7	86	87	89	1.73	85	**86.3**	88	1.53
20	300	125.7	127	129	131	1.63	132	135	139	2.94	128	128.7	129	0.58	127	**128**	129	1

TABLE 10. Results for small-size simulated data with an alphabet of 4 characters.

Instance		Exact	LDDA_LSS				Heuris_Seq				WFC-CSP (max_iter=10)				WFC-CSP (max_iter=200)			
k	L	Avg.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.
10	100	59.3	60	60.7	61	0.47	63	61.3	60	1.25	58	59	60	1	57	**58.3**	60	1.53
10	200	117.7	119	120	121	0.82	121	120.3	120	0.47	116	117.7	119	1.53	115	**116.7**	118	1.53
10	300	173	177	177.7	179	0.94	177	176.3	176	0.47	174	175.3	177	1.53	174	**174.7**	176	1.15
20	100	64.7	67	67.3	68	0.47	67	66.3	65	0.94	65	66	67	1	65	**65.7**	66	0.58
20	200	127.3	131	131.3	132	0.47	132	130.7	129	1.25	131	131	131	0	130	**130.3**	131	0.58
20	300	191.3	196	196.3	197	0.47	196	195.7	195	0.47	192	193.3	194	1.15	192	**192.3**	193	0.58

protein sequences frequently used to test string comparison algorithms. In order to create a set of inputs strings of equal length, [22] let the length of the strings to be equal to the length of the shortest string in the set, and removed the last characters for strings with length greater than the minimum. Six instances with alphabet size 20 were chosen from the McClure dataset [18]. For the simulated dataset, small-sized and large-sized instances were generated with the following procedure: with a given k (number of strings), L (string length) and an alphabet Σ, randomly choose a character from Σ for each position in the resulting string. Three different alphabet sizes are tested, including instances with alphabet size of two representing binary strings with applications in Coding Theory, and instances with alphabet size of four and twenty which appear in applications involving DNA and amino acid sequences, respectively. For the small-size instances, each algorithm was executed over a set of 54 instances, with 18 instances for each of the alphabets.

TABLE 11. Results for small-size simulated data with an alphabet of 20 characters.

Instance		Exact	LDDA_LSS				Heuris_Seq				WFC-CSP (max_iter=10)				WFC-CSP (max_iter=200)			
k	L	Avg.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.
10	100	79.3	82	83	84	0.82	82	83.7	85	1.25	78	**78.3**	79	0.58	78	78.3	79	0.58
10	200	157	161	161.7	163	0.94	164	165.7	167	1.25	157	**157.3**	158	0.58	157	157.3	158	0.58
10	300	234.7	240	240	240	0	245	247.3	249	1.7	235	**236**	237	1	235	236	237	1
20	100	84.3	87	87.7	88	0.47	90	90.3	91	0.47	85	**85**	85	0	85	85	85	0
20	200	168	172	173.3	174	0.94	178	179	180	0.82	169	**169.3**	170	0.58	169	169.3	170	0.58
20	300	253.3	256	257	258	0.82	266	266.7	267	0.47	252	**252.7**	254	1.15	252	252.7	254	1.15

TABLE 12. Results for large-size simulated data with an alphabet of 2 characters.

Instance		LDDA_LSS				Heuris_Seq				WFC-CSP (max_iter=10)				WFC-CSP (max_iter=200)			
k	L	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.
10	1000	379	380	382	1.41	382	393.7	406	9.81	376	380.3	384	4.04	376	**379.3**	383	3.51
10	2000	755	758	760	2.16	760	770	781	8.6	751	756	764	7	749	**754**	763	7.81
10	3000	1131	1135.7	1141	4.11	1131	**1134.3**	1140	4.03	1132	1135.7	1141	4.73	1132	1135.3	1141	4.93
10	4000	1516	1524	1531	6.16	1522	1533.7	1555	15.11	1505	1508	1511	3	1505	**1507.7**	1510	2.52
10	5000	1885	1888.3	1891	2.49	1884	1901.3	1914	12.68	1872	1881.7	1890	9.07	1871	**1880.7**	1889	9.07
20	1000	421	**421.7**	423	0.94	430	453.7	481	20.98	423	423.7	424	0.58	421	422.3	423	1.15
20	2000	837	838.3	840	1.25	849	852.3	858	4.03	830	834	839	4.58	829	**833**	838	4.58
20	3000	1249	1251.3	1254	2.05	1305	1330.3	1355	20.42	1236	1242	1248	6	1235	**1241.7**	1248	6.51
20	4000	1661	1665.7	1670	3.68	1704	1760.3	1843	59.72	1651	1654.3	1660	4.93	1648	**1652.3**	1657	4.51
20	5000	2075	2077	2081	2.83	2103	2138	2171	27.8	2067	2068.3	2070	1.53	2064	**2065.7**	2067	1.53
30	1000	440	441	443	1.41	447	457.7	465	7.72	435	437.7	439	2.31	435	**437.3**	439	2.08
30	2000	870	871	872	0.82	917	943.3	964	19.6	869	870	871	1	868	**868.7**	869	0.58
30	3000	1302	1304	1305	1.41	1344	1371	1402	23.85	1297	1299.3	1301	2.08	1295	**1297**	1298	1.73
30	4000	1726	1730.7	1733	3.3	1791	1893.3	1946	72.37	1725	1727.3	1731	3.21	1724	**1726**	1730	3.46
30	5000	2159	2160.7	2163	1.7	2239	2270	2298	24.18	2141	2152.3	2163	11.02	2139	**2149.7**	2158	9.71

With large-size simulated instances, each algorithm was exceuted over a set of 135 instances, with 45 instances for each of the alphabets.

In this paper, we used the same six McClure instances in [22], and generated simulated small-size and large-size instances with the same procedure as described in [22] to test WFC-CSP. Tables 8 through 14 shows the testing results of different instances or test configurations. Results for Exact (third IP formation algorithm in [19]), LDDA_LSS (from [22]), and Heuris_Seq (from [21]) are from [22]. WFC-CSP algorithm was run with different max_iter configurations. In this experiment, maximum HD d is not provided as an input to WFC-CSP, WFC-CSP runs for max_iter iterations, and the solution from the iteration resulting in the smallest maximum HD is chosen as WFC-CSP's solution.

In the comparisons, we only compare the quality of solutions among algorithms without comparing algorithms' run times. Being heuristic, all algorithms except for the "Exact" algorithm are able to complete the tests in short times. The "Exact" algorithm has high computational complexity, and is not viable for large-size simulated instances; therefore, "Exact" is not included in the comparisons that use large simulated instances. The parameters for Exact, LDDA_LSS and Heuris_Seq algorithms are as described in [22]: To LDDA_LSS, B was set to $0.1n$, *b_rep* is set to 0.5 for large-size instances with twenty characters, and *b_rep* is set to 2 for all of the other test instances. For Heuris_Seq, N is set to 10,000. For WFC-CSP, our testing varies the max_iter parameter.

In Table 8, columns labeled "k" and "L" are the number of strings and string length parameters, and columns labeled "Exact", "Parallel", "LDDA_LSS", "Heuris_Seq" and "WFC-CSP" contains the max HD results running the algorithms on the test case. Note that in addition to algorithms included in [22], results from the "Parallel" algorithm came from [21]. WFC-CSP is configured to run with maximum of 100 iterations. It is shown that WFC-CSP finds solutions at most two away from the "Exact" algorithm, and is the best performing algorithm among the heuristic algorithms.

In Table 9 to 11, minimum, average, maximum values of the solutions as well as the standard derivation among solutions are shown for LDDA_LSS, Heuris_Seq and WFC-CSP algorithms in columns "Min", "Avg.", "Max" and "Std.". The best average solution among algorithms (other than "Exact" algorithm) are in bold. For WFC-CSP, max_iter configurations of 10 and 200 were tested. It is shown that for small-size alphabet size of 2 cases, running WFC-CSP with 200 iterations yield better results except for the ($k = 20$, $L = 100$) case, where Heuris_Seq performed better. In cases where alphabet size is 4, running WFC-CSP with 200 iterations always yield best solution. When alphabet size is 20, running WFC-CSP with 10 iterations already achieved

TABLE 13. Results for large-size simulated data with an alphabet of 4 characters.

Instance		LDDA_LSS				Heuris_Seq				WFC-CSP (max_iter=10)				WFC-CSP (max_iter=200)			
k	L	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.
10	1000	588	588.7	589	0.47	585	587	589	1.63	582	583.3	585	1.53	582	**583**	585	1.73
10	2000	1167	1172.3	1181	6.18	1168	1173	1181	5.72	1162	1166.7	1171	4.51	1162	**1166.3**	1170	4.04
10	3000	1752	1753.3	1755	1.25	1754	1756	1758	1.63	1733	1735.7	1739	3.06	1732	**1735**	1738	3
10	4000	2346	2348	2349	1.41	2346	2353	2358	5.1	2313	2316	2321	4.36	2313	**2315.7**	2321	4.62
10	5000	2908	2915.3	2921	5.44	2918	2920.7	2925	3.09	2897	**2899**	2902	2.65	2897	2899	2902	2.65
20	1000	644	646.7	649	2.05	644	648	651	2.94	635	637.3	641	3.21	634	**636**	639	2.65
20	2000	1280	1282.3	1287	3.3	1292	1293.7	1295	1.25	1267	1269	1270	1.73	1265	**1267**	1269	2
20	3000	1918	1922	1926	3.27	1934	1940	1947	5.35	1902	1903.3	1904	1.15	1901	**1902**	1903	1
20	4000	2544	2551.7	2562	7.59	2571	2574.7	2578	2.87	2529	2530	2531	1	2528	**2528.7**	2530	1.15
20	5000	3180	3185.3	3193	5.56	3219	3221.3	3224	2.05	3158	3162.3	3170	6.66	3158	**3161.7**	3168	5.51
30	1000	672	673.3	675	1.25	674	674.7	675	0.47	660	661.3	662	1.15	660	**660.3**	661	0.58
30	2000	1324	1328	1332	3.27	1342	1343.7	1345	1.25	1317	1319	1320	1.73	1316	**1318**	1319	1.73
30	3000	1984	1987.3	1992	3.4	2013	2013.7	2014	0.47	1968	1972.7	1976	4.16	1967	**1971**	1975	4
30	4000	2628	2631.7	2637	3.86	2665	2669.7	2676	4.64	2625	2628.7	2632	3.51	2626	**2627.3**	2630	2.31
30	5000	3296	3298.7	3301	2.05	3349	3353.7	3357	3.4	3270	3277.7	3286	8.02	3270	**3276.3**	3283	6.51

TABLE 14. Results for large-size simulated data with an alphabet of 20 characters.

Instance		LDDA_LSS				Heuris_Seq				WFC-CSP (max_iter=10)				WFC-CSP (max_iter=40)			
k	L	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.	Min	Avg.	Max	Std.
10	1000	787	790	794	2.94	812	815.3	819	2.87	782	**784**	787	2.65	782	784	787	2.65
10	2000	1570	1573.7	1578	3.3	1601	1606.7	1612	4.5	1563	1563.7	1564	0.58	1563	**1563.3**	1564	0.58
10	3000	2352	2353.3	2355	1.25	2400	2401.3	2403	1.25	2347	**2349.3**	2353	3.21	2347	2349.3	2353	3.21
10	4000	3132	3138.3	3143	4.64	3193	3195.7	3199	2.49	3124	3132	3136	6.93	3124	**3131.7**	3136	6.66
10	5000	3918	3923	3927	3.74	3980	3984.7	3989	3.68	3905	**3907.3**	3910	2.52	3905	3907.3	3910	2.52
20	1000	846	848	851	2.16	887	888	889	0.82	838	**839.3**	841	1.53	838	839.3	841	1.53
20	2000	1689	1690.3	1692	1.25	1763	1769	1773	4.32	1678	**1678**	1678	0	1678	1678	1678	0
20	3000	2531	2534	2537	2.45	2633	2642.3	2650	7.04	2515	**2515.7**	2516	0.58	2515	2515.7	2516	0.58
20	4000	3367	3369.7	3374	3.09	3499	3502.7	3510	5.19	3353	3354	3355	1	3353	**3353.7**	3355	1.15
20	5000	4208	4212.3	4218	4.19	4373	4380.3	4394	9.67	4190	4193.7	4198	4.04	4190	**4193.3**	4197	3.51
30	1000	874	874.7	875	0.47	911	912	913	0.82	863	864.3	865	1.15	863	**863.7**	864	0.58
30	2000	1742	1742	1742	0	1819	1820.7	1823	1.7	1723	1725	1727	2	1723	1725	1727	2
30	3000	2607	2608.3	2610	1.25	2722	2728.7	2736	5.73	2586	2587.3	2588	1.15	2586	**2587**	2588	1
30	4000	3472	3473.7	3475	1.25	3626	3630.7	3633	3.3	3444	3445.7	3447	1.53	3444	**3445.3**	3446	1.15
30	5000	4343	4344.3	4347	1.89	4530	4534.3	4537	3.09	4308	4309.3	4311	1.53	4307	**4309**	4311	2

best solutions, increasing the iterations to 200 did not result in better solutions.

Similar observations are made in Table 12 to 14 for large-size simulated data. In ($k = 10$, $L = 3000$) and ($k = 20$, $L = 1000$) cases when alphabet size is 2, LDDA_LSS and Heuristic_Seq performed marginally better (difference in solution values being within 1) respectively than WFC-CSP; in all other cases, WFC-CSP performed best.

Similar observations are made as previously noted: WFC-CSP has an easier time solving instances with larger alphabet size, it does not need to be run with large number of iterations to achieve optimal solution, and its performance advantage is bigger compared with other algorithms in large alphabet size instances.

V. CONCLUSION AND FUTURE RESEARCH

This paper proposes the novel WFC-CSP algorithm to solve the Closest String Problem by leveraging WaveFunctionCollapse techniques, and demonstrates its merits in algorithm complexity and performance compared to multiple previous CSP algorithms.

Compared to previous CSP algorithms, WFC-CSP is significantly simpler to implement. WFC-CSP is a non-backtracking algorithm. Constructing a solution string with WFC-CSP involves only simple operations of scoreboard sorting, tie-breaker randomization, and bookkeeping of intermediate results. WFC-CSP is also easier to use than many other CSP algorithms. WFC-CSP provides a simple knob (the max_iter parameter) for solution quality and run time trade off, and does not have other parameters that need to be tuned for performance optimization.

The complexity of each iteration of WFC-CSP is tractable with respect to number of strings k, string length L, and alphabet size $|\Sigma|$. The target maximum Hamming distance d does not affect the algorithm's complexity within an iteration. WFC-CSP's success rate or quality of solution increases as more iteration is allowed, at the cost of increased run time. It has also been shown that the WFC-CSP algorithm is robust in generating consistent high quality results on different test cases.

Performance comparison between WFC-CSP and other CSP algorithms are provided in this paper. Comparing with exact and approximation algorithms such as the Fixed-parameter algorithm (FP-CSP) in [5] and the third IP formation algorithm in [19], WFC-CSP, being heuristic, of course does not have an advantage in success rate or solution quality, but it enjoys high performance and short run time while solving large instances that make the FP-CSP and the IP formation algorithms unviable. Comparing with other heuristic algorithms Ant-CSP, LDDA_LSS and Heuris_Seq algorithms, WFC-CSP generally has higher success rate or higher solution quality.

Future research plans include enhancing the WFC-CSP algorithm to improve its performance in solving challenging instances, such as configurations with smaller L/d ratio, smaller alphabet size, and fewer number of strings. For example, we plan to optimize the randomization portion of the algorithm, where currently randomization only happens when there are ties among worst strings or scores in the algorithm's scoreboard. We suspect that it may be beneficial to allow more randomization to encourage exploration of the solution space.

Another future research area is to customize WFC-CSP for specific applications; for example, in some bioinformatics applications, it may be desired to calculate "weighted" scores in the scoreboard to accommodate the fact that certain DNA mutation are more common than others. This customization possibility is a unique advantage of score based algorithms such as WFC-CSP.

REFERENCES

[1] K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang, "Distinguish string search problems," in *Proc. 10th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1999, pp. 633–642.

[2] M. Li, B. Ma, and L. Wang, "On the closest string and substring problems," *J. ACM*, vol. 49, no. 2, pp. 157–171, Mar. 2002.

[3] D. Marx, "The closest substring problem with small distances," in *Proc. 46th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, 2005, pp. 63–72.

[4] B. Ma and X. Sun, "More efficient algorithms for closest string and substring problems," in *Proc. 12th Annu. Int. Conf. Comput. Biol. (RECOMB)*, 2008, pp. 396–409.

[5] J. Gramm, R. Niedermeier, and P. Rossmanith, "Fixed-parameter algorithms for CLOSEST STRING and related problems," *Algorithmica*, vol. 37, no. 1, pp. 25–42, Sep. 2003.

[6] M. Gumin. (2016). *Wave Function Collapse*. GitHub. [Online]. Available: https://github.com/mxgmn/WaveFunctionCollapse

[7] I. Karth and A. M. Smith, "WaveFunctionCollapse is constraint solving in the wild," in *Proc. 12th Int. Conf. Found. Digit. Games*, Aug. 2017, pp. 1–10.

[8] A. Newgas, "Tessera: A practical system for extended WaveFunctionCollapse," in *Proc. 16th Int. Conf. Found. Digit. Games (FDG)*, Aug. 2021, pp. 1–7.

[9] L. Gasieniec, J. Jansson, and A. Lingas, "Efficient approximation algorithms for the Hamming center problem," in *Proc. 10th Annu. ACM SIAM Symp. Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999, pp. 905–906.

[10] X. Liu, H. He, and O. Sýkora, "Parallel genetic algorithm and parallel simulated annealing algorithm for the closest string problem," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2005, pp. 591–597.

[11] X. Liu, M. Holger, Z. Hao, and G. Wu, "A compounded genetic and simulated annealing algorithm for the closest string problem," in *Proc. 2nd Int. Conf. Bioinf. Biomed. Eng.*, May 2008, pp. 702–705.

[12] S. Faro and E. Pappalardo, "Ant-CSP: An ant colony optimization algorithm for the closest string problem," in *SOFSEM, Theory and Practice of Computer Science* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2010, pp. 370–381.

[13] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theor. Comput. Sci.*, vol. 344, nos. 2–3, pp. 243–278, Aug. 2015.

[14] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992.

[15] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Aug. 1997.

[16] Boris. *Wave Function Collapse Explained*. BorisTheBrave.Com. Accessed: Nov. 12, 2021. [Online]. Available: https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/

[17] M. Li, B. Ma, and L. Wang, "Finding similar regions in many strings," in *Proc. 31st Annu. ACM Symp. Theory Comput. (STOC)*, 1999, pp. 473–482.

[18] M. A. McClure, T. K. Vasi, and W. M. Fitch, "Comparative analysis of multiple protein-sequence alignment methods," *Mol. Biol. Evol.*, vol. 11, no. 4, pp. 571–592, Jul. 1994.

[19] C. N. Meneses, Z. Lu, C. A. S. Oliveira, and P. M. Pardalos, "Optimal solutions for the closest-string problem via integer programming," *INFORMS J. Comput.*, vol. 16, no. 4, pp. 419–429, Nov. 2004.

[20] F. C. Gomes, C. N. Meneses, P. M. Pardalos, and G. V. Viana, "Parallel algorithm for the closest string problem," in *Proc. 1st Int. Symp. Math. Comput. Biol.*, vol. 2, 2005, pp. 326–332.

[21] F. C. Gomes, C. N. Meneses, P. M. Pardalos, and G. V. R. Viana, "A parallel multistart algorithm for the closest string problem," *Comput. Oper. Res.*, vol. 35, no. 11, pp. 3636–3643, Nov. 2008.

[22] X. Liu, S. Liu, Z. Hao, and H. Mauch, "Exact algorithm and heuristic for the closest string problem," *Comput. Oper. Res.*, vol. 38, no. 11, pp. 1513–1520, Nov. 2011.

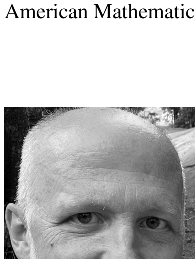
[23] X. Liu, K. Fu, and R. Shao, "Largest distance decreasing algorithm for the closest string problem," *J. Inf. Comput. Sci.*, vol. 1, pp. 287–292, Dec. 2004.

[24] P. Merrell, "Example-based model synthesis," in *Proc. Symp. Interact. 3D Graph. Games (ID)*, 2007, pp. 105–112.

[25] A. Mac and D. Perkins, "Wave function collapse coloring: A new heuristic for fast vertex coloring," 2021, *arXiv:2108.09329*.

SHIRLEY XU is currently with The Bishop's School, La Jolla, CA, USA. Her research interests include exploring efficient solutions to NP-complete problems, machine learning, image classification and segmentation, computer vision, bioinformatics, and biophysics.

Her awards and honors include the Second Place Grand Award in the Mathematics Category of the 2022 Regeneron International Science and Engineering Fair (ISEF), ISEF First Place Special Award from Mu Alpha Theta, and ISEF Third Place Special Award from the American Mathematical Society.



DAVID PERKINS received the Ph.D. degree in mathematics from the University of Montana, Missoula, MT, USA, in 2005.

He teaches computer science at the Hamilton College, Clinton, New York, and collaborated with Ms. Xu through Pioneer Academics. He has written two books that were published in the *Mathematical Association of America* (Spectrum Series). His research interest includes unusual blending of algorithms in the hopes of discovering novel heuristic solutions to NP-complete problems.