Testing Whether a Set of Code Words Satisfies a Given Set of Constraints

Hsin-Wen Wei[*], *Wan-Chen Lu*[*], *Pei-Chi Huang*[*], *Wei-Kuan Shih*[*],
Ming-Yang Kao[†], *and Tsan-sheng Hsu*[‡]
[*]Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan
{bertha, wanchen, peggy, wshih}@rtlab.cs.nthu.edu.tw
[†]Department of EECS, Northwestern University, U.S.A.
kao@northwestern.edu
[‡]Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan
tshsu@iis.sinica.edu.tw

Abstract

This paper investigates the problem of testing whether a set of code words satisfies certain biologically motivated Hamming distance constraints. The paper provides three general techniques to speed up the testing of the constraints, namely, the Enumeration, Table Lookup, and Encoding methods, with applications to the design of DNA words.

keywords: DNA verification, Hamming distance constraints, code word verification

1 Introduction

The scalable design of code words has been investigated by many researchers. Huffman [6] presented the well-known two-pass algorithm to encode variable-length source codes. A new one-pass algorithm for constructing dynamic Huffman codes is then proposed in [13]. The shortest common superstring problem, which is one of the simplest models for assembling the long string representing the whole molecule, is proved to be NP-hard [4]; an efficient code word design method for the shortest common superstring problem is introduced in [14]. In recent years, many code word design methods are applied to solve DNA-related problems [5, 9, 10, 11]. For instance, one is for detecting protein-DNA-binding sites [9, 10]. A number of different proteins bind into a genome. Some of these DNA-binding proteins only bind to specific locations where they have to execute certain functions. Another is for designing error-preventing DNA sequence [5, 11]. The potential errors in DNA sequence should be minimized for reliable DNA computing.

In this paper, we study the verification algorithm for designing DNA words. For the DNA word design problem, many algorithms have been developed. Marathe et al. [8] used a dynamic programming approach to design DNA sequences based on Hamming distance and free energy. Deaton et al. [3] used genetic search to find good encodings. Arita et al. [1] developed a DNA sequence design system using genetic algorithms and random generate-and-test algorithms. Tanaka et al. [12] listed sequence fitness criteria and sequences are generated using simulated annealing method. Recently, Kao et al. [7] considered more fitness criteria for DNA

sequence design.

Since some DNA word design algorithms are randomized, we need a fast verification algorithm to verify whether the output indeed satisfy the given set of constraints. Therefore, we focus on the development of a fast verification algorithm that can determine whether a set of words satisfy a given set of constraints. There are five constraints $C_1, \ldots, C_5$ considered in this paper and these constraints can be classified as basic Hamming distance constraint, reverse complement Hamming distance constraint, self-complementary Hamming distance constraint, and shifting Hamming distance constraint which are proposed in [2, 7, 8] for DNA word designs.

To verify these constraints for a set of DNA words designed, we propose the three following techniques: (1) Enumeration. It enumerates all possible combinations of selected $\ell - k + 1$ positions in a word, where ℓ is the length of a word and k is the given parameter of a constraint. This method determines whether the set of words satisfies the given constraints by checking whether any two sub-words selected from all possible combinations of $\ell - k + 1$ positions in any two words of the set are the same. (2) Table Lookup. It first enumerates all possible combinations of sub-words with a fixed length and then computes the Hamming distance between any two sub-words for constructing a data table. By dividing each word into some sub-words and looking up the constructed data table, the Hamming distance between two words can be obtained. (3) Encoding. It utilizes a table and a linked list for storing some necessary information. It first divides each word into some sub-words with a fixed length and puts them into different groups. If the sub-words of different words start at the same position in the words, then the sub-words are put into the same group. Each group of sub-words are assigned an unique group ID. By sorting the sub-words in a group, it determines whether any two sub-words are the same. If a group contains two different sub-words, then update the corresponding value in the table and store the group ID into the corresponding location of the linked list. After that, this method determines whether a set of words satisfies the given constraints by checking the table and linked list. The time complexities of these methods for given constraints are summarized as Table 1.

2 Preliminaries

In this paper we deal a set of words W with two alphabets, namely, the binary alphabet {0, 1} and the DNA alphabet {A, C, G, T}. A given set of words W is assumed to each has length ℓ and let $|W| = n$. Let $X = x_1x_2 \ldots x_\ell$ denote a word where x_i denotes the ith position in X, and $|X|$ denote the length of X, i.e., $|X| = \ell$. Let $X[i..j] = x_ix_{i+1} \ldots x_j$ denote a sub-word of X, $1 \leq i \leq j \leq \ell$. If $i = j$, then $X[i..i] = X[i] = x_i$. The reverse of X, denoted by X^R, is the word $x_\ell x_{\ell-1} \ldots x_1$, and the sub-word of the reverse of X is denoted by $X^R[i..j]$, e.g., $X^R[1..3]= x_\ell x_{\ell-1}x_{\ell-2}$. The complement of X, denoted by X^C, is the word $x_1^C x_2^C \ldots x_\ell^C$, and the sub-word of complement of X is denoted by $X^C[i..j]$, where x_i^C is defined as $0^C = 1$ and $1^C = 0$, if x_i is over the binary alphabet; $A^C = T$, $C^C = G$, $G^C = C$, $T^C = A$, if x_i is over the DNA alphabet.

The reverse complement of X is the complement of X^R, $X^{RC} = x_\ell^C x_{\ell-1}^C \ldots x_1^C$, and the sub-word of complement of X is denoted by $X^{RC}[i..j]$, e.g., $X^{RC}[1..2] = x_\ell^C x_{\ell-1}^C$. Note that, $X[i..j]^{RC}$, denotes the reverse complement of the sub-word of X, e.g., $X[1..2]^{RC} = x_2^C x_1^C$. Let "+" be the operation that concatenates any two words $X = x_1x_2 \ldots x_\ell$ and $Z = z_1z_2 \ldots z_i$, such that $X + Z = x_1x_2 \ldots x_\ell z_1z_2 \ldots z_i$, where $i > 0$, and $|X + Z| = \ell + i$. The Hamming distance between two words X

	Enumeration	Table Lookup	Encoding
C_1, C_2	$O(n*(\ell-k)*\ell^{\min\{k-1,\ell-k+1\}})$	$O(2^{2*\sqrt{\ell}}*\sqrt{\ell}+n^2*\sqrt{\ell})$	$O(n^2*\sqrt{\ell}*\sqrt{k})$
C_3, C_4	$O(n*(\ell-k)*\ell^{\min\{k-1,\ell-k+1\}}*k)$	$O(2^{2*\ell}*\ell+n^2*\ell)$	$O(n^2*\sqrt{\ell*k}*k)$
C_5	N.A.	$O(\ell*\log\ell+\frac{n*\ell*k}{\log\ell})$	N.A.

Table 1: Time complexities of each method for different constraints.

and $Y = y_1y_2\dots y_\ell$, which is defined as $d(X,Y) = |\{i|x_i \neq y_i, 1 \leq i \leq \ell\}|$, is the number of positions where X differs from Y. We are interested in verifying a given set of words W satisfies the following constraints [7], which is denoted as a function $C_f(k_f)$ with a given parameter k_f:

1. Basic Hamming Distance Constraint (k_1): $C_1(k_1) = \{d(Y,X) \geq k_1 | \forall X, Y \in W, X \neq Y\}$.
2. Reverse Complementary Constraint (k_2): $C_2(k_2) = \{d(Y, X^{RC}) \geq k_2 | \forall X, Y \in W, X \neq Y\}$.
3. Shifting Hamming Constraint (k_3): $C_3(k_3) = \{d(Y[1..i], X[(\ell-i+1)..\ell]) \geq k_3-(\ell-i) | 1 \leq i \leq \ell, \forall X, Y \in W, X \neq Y\}$.
4. Shifting Reverse Complementary Constraint (k_4): $C_4(k_4) = \{d(Y[1..i], X[1..i]^{RC}) \geq k_4-(\ell-i)$; and $d(Y[(\ell-i+1)..\ell], X[(\ell-i+1)..\ell]^{RC}) \geq k_4-(\ell-i) | 1 \leq i \leq \ell, \forall X, Y \in W, X \neq Y\}$
5. Shifting Self Complementary Constraint (k_5): $C_5(k_5) = \{d(Y[1..i], Y[1..i]^{RC}) \geq k_5-(\ell-i)$; and $d(Y[(\ell-i+1)..\ell], Y[(\ell-i+1)..\ell]^{RC}) \geq k_5-(\ell-i) | 1 \leq i \leq \ell, \forall Y \in W\}$.

Note a naive algorithm for testing C_1 and C_2 constraints takes $O(n^2\ell)$ time ; for testing C_3 and C_4 constraints takes $O(n^2*\ell*k)$ time; and for testing C_5 takes $O(n*\ell*k)$ time, where k is the parameter of the given constraint. In this paper, we propose three techniques for solving the verification of a given set of words. The proposed approaches determine whether a set of words satisfies a set of combinatorial constraints faster than the naive algorithms on several practical cases such as $\ell = O(\log n)$. The first is Enumeration method which can be applied to $C_1 - C_4$ constraints. While $\ell = O(\log n)$ and the parameters of given constraints are no greater than $O(\frac{\log n}{\log\log n})$, the time complexity of the Enumeration method is lower than a naive algorithm and the other methods which proposed in here. The second is Table Lookup method which can be applied to $C_1 - C_5$ constraints. The Table Lookup method is an efficient approach while $\ell = O(\log n)$ and its time complexity is lower than that of a naive algorithm for any constraint if ℓ is no greater than $O(\log n)$. Finally, the Encoding method can be applied to $C_1 - C_4$ and its time complexity is lower than that of naive algorithm while the order of k is less than ℓ. This method is better than the other methods while $\ell = \Omega(\log n)$.

The comparison of time complexities between naive algorithms and our techniques with assumption that $\ell = O(\log n)$ is summarized in Table 2.

3 Enumeration

Our discussion is based on C_1 constraint and using the binary alphabet. This method can be easily extend to $C_2 - C_4$ constraints on any fixed alphabet. For any two words X and Y in W, our problem is to test whether $d(X,Y) \geq k$, where k is the given parameter of the corresponding constraint. Our basic idea is that if

		k			
		$O(\log n)$	$O(\frac{\log n}{\log\log n})$	$O(\log\log n)$	$O(1)$
C_1, C_2	naive	$O(n^2 * \log n)$	$O(n^2 * \log n)$	$O(n^2 * \log n)$	$O(n^2 * \log n)$
	ours	$O(n^2 * \sqrt{\log n})$;#2	$O(n^2)$;#1	$O(n^{1+\epsilon})$;#1	$O(n * \log n)$;#1
C_3, C_4	naive	$O(n^2 * \log^2 n)$	$O(n^2 * \frac{\log^2 n}{\log\log n})$	$O(n^2 * \log n * \log\log n)$	$O(n^2 * \log n)$
	ours	$O(n^2 * \log n)$;#2	$O(n^2 * \frac{\log n}{\log\log n})$;#1	$O(n^{1+\epsilon} * \log\log n)$;#1	$O(n * \log n)$;#1
C_5	naive	$O(n * \log^2 n)$	$O(n * \frac{\log^2 n}{\log\log n})$	$O(n * \log n * \log\log n)$	$O(n * \log n)$
	ours	$O(\frac{n*\log^2 n}{\log\log n})$;#2	$O(\frac{n*\log^2 n}{(\log\log n)^2})$;#2	$O(n * \log n)$;#2	$O(\frac{n*\log n}{\log\log n})$;#2

Table 2: The comparison of time complexities between naive algorithms and our algorithms assuming $\ell = O(\log n)$ and ϵ is any constant less than 1. #1: Using Enumeration method, #2: Using Table Lookup method.

Algorithm 1 ENUMERATION Algorithm

```
1: procedure ENUMERATION(W) {* Output "Fail", if the constraint is not satisfied; "Success", if the constraint is satisfied *}
2:     for p from 1 to (ℓ choose ℓ−k+1) do
3:         Let W_p = {X_{1,p}, X_{2,p}, ..., X_{n,p}} contains n sub-words with length ℓ − k + 1 in W
               which are selected according to the p-th combination, where k = k_1, X_{i,p} is the
               sub-word of p-th combination of X_i, i = 1, 2, ..., n
4:         Perform radix sort on W_p;
5:         if there are any two sub-words X_{i,p} and X_{j,p}, i ≠ j in W_p are the same in the sorted
               result then
6:             return "Fail";
7:         end if
8:     end for
9:     return "Success";
10: end procedure
```

$d(X,Y) < k$, it means there are at least $\ell - k + 1$ positions that the same in X and Y. Hence, we select any two sub-words with length $\ell - k + 1$ from X and Y respectively, and then check if they are the same. If any two sub-words with length $\ell - k + 1$ of X and Y are not the same, then $d(X,Y) \geq k$; otherwise, $d(X,Y) < k$. There are totally $\binom{\ell}{\ell-k+1}$ combinations to select a sub-word with length $\ell - k + 1$ from a word with length ℓ. Without loss of generality, we assume that any sub-word with length $\ell - k + 1$ of a word with length ℓ is corresponding to an unique combination of $\ell - k + 1$ positions in a word. Each combination of $\ell - k + 1$ positions in a word is assigned an unique label "p", which is called p-th combination, where $p = 1, 2, \ldots, \binom{\ell}{\ell-k+1}$, e.g., 1-st combination is $1^{st}, 2^{nd}, \ldots, (\ell - k + 1)^{th}$ positions in a word and 2-nd combination is $1^{st}, 3^{rd}, \ldots, (\ell-k+1)^{th}$ positions in a word. The detail of the Enumeration method for C_1 constraint is shown in Algorithm 1.

Here is an example to show how to verify a set of words W satisfies $C_1(k_1)$ by Algorithm 1. Assume there are three words $X_1 = 10101010$, $X_2 = 11110000$, and $X_3 = 00110010$ in W. For $C_1(5)$, i.e., $k_1 = 5$, there are $\binom{8}{8-5+1}$=70 combinations to select 3 sub-words with 4, $\ell - k + 1 = 8 - 5 + 1 = 4$, bits from W. When the combination of positions in a word are $1^{st}, 3^{rd}, 6^{th}$ and 8^{th} positions, the corresponding sub-words of the 3 words in W are 1100, 1100, 0100 (from X_1, X_2, and X_3, respectively). After using radix sort at step 4, we find that two sub-words are

equal (i.e., 1100), and therefore W does not satisfy constraint $C_1(5)$.

Theorem 1 *Algorithm 1 is correct and its time complexity is* $O(n * (\ell - k) * \ell^{\min\{k-1,\ell-k+1\}})$*, where* k *is the given parameter of the corresponding constraint.*

Proof. For any two words X and Y, the sub-words of X and Y with length $\ell-k+1$ are compared in steps $2-8$. If there exists two sub-words with length $\ell-k+1$ that are equal, there must be at most $k-1$ different positions between the two words. Therefore, it does not satisfy the Hamming distance constraint; otherwise, it does. Hence the algorithm is correct.

In Algorithm 1, there are $\binom{\ell}{\ell-k+1}$ combinations to enumerate all possible combinations of sub-words with length $\ell - k + 1$ in W. Note that, $\binom{\ell}{\ell-k+1} \leq \ell^{\min\{k-1,\ell-k+1\}}$. The sub-words of a given p-th combination with length $\ell - k + 1$ in W are using radix sort at step 4 in $O(n * (\ell - k))$ time for each p from 1 to $\binom{\ell}{\ell-k+1}$. Hence, the total time complexity of Algorithm 1 is $O(n * (\ell - k) * \ell^{\min\{k-1,\ell-k+1\}})$. □

The Enumeration method can be used to verify C_3 and C_4 constraints. Note that, if the length i of any two sub-words is less than $\ell-k$, where $k = k_3$ or k_4, then any two sub-words with length i must satisfy the Hamming distance constraint, since, $k - (\ell - i) < 0$. Therefore, it only needs to deal with the case that the length of a sub-word is greater than $\ell-k$ and its time complexity is $O(n*(\ell-k)*\ell^{\min\{k-1,\ell-k+1\}}*k)$.

4 Table Lookup

This method, which determines Hamming distance between any two words by looking up a precomputed table, is called the Table Lookup method. There are two data structures needed in this method: (1) Information table, which records the Hamming distance between any two words in W, is called INF table. There are n^2 entries in the table, let (i, j) denote an entry in row i and column j in the INF table. Let $t_{i,j}$ denote the value of the entry i, j in the INF table which is the Hamming distance between the two words X_i and X_j. Note that, the initial value of $t_{i,j}$ is zero, for all $1 \leq i, j \leq n$. (2) Comparison table, which enumerates all possible combinations of sub-words of a given length, and provides the Hamming distance of any two sub-words in the enumeration, is called CMP table. For example, Figure 1 shows Hamming distance of any two sub-words with length 3 over the binary alphabet. It is obvious to see that there are 2^3 sub-words in the enumeration and $2^3 * 2^3$ entries in the CMP table, since the length of a sub-word is 3 and the size of the binary alphabet is 2.

The first step of Table Lookup method is to enumerate all possible combinations of sub-words over the given alphabet to construct the CMP table. Without loss of generality, we assume that the given alphabet is binary. Note that each entry in the CMP table records the Hamming distance between any two sub-words exhaustively enumerated. Second, divide each word $X_i \in W$ into $\sqrt{\ell}$ sub-words $X_i[a..b]$ with fixed length $\sqrt{\ell}$, where $a = (g-1)*\sqrt{\ell}+1, b = g*\sqrt{\ell}, 1 \leq g \leq \sqrt{\ell}$. Note that, $G = \{X_1[a..b], X_2[a..b], \ldots, X_n[a..b]\}$ denotes a group of sub-words of W, $X_i \in W$, $1 \leq i \leq n$.

Next, compute Hamming distance between any two sub-words $X_i[a..b]$ and $X_j[a..b]$, $i \neq j$, in the same group by looking up the CMP table, then update the corresponding entry in the INF table. Finally, after all comparison processes are finished, check the value of each entry in the INF table. If there exists a value lower than the parameter of the given constraint, then report that the given set of words does not satisfy the constraint. The detail of the Table Lookup method for verifying whether a given set of words

	000	001	010	011	100	101	110	111
000	0	1	1	2	1	2	2	3
001		0	2	1	2	1	3	2
010			0	1	2	3	1	2
011				0	3	2	2	1
100					0	1	1	2
101						0	2	1
110							0	1
111								0

Figure 1: Example of CMP table

W satisfies C_1 constraint is described in Algorithm 2. Note Algorithm 2 can be easily extended to verifying C_2 constraint on any fixed alphabet.

Theorem 2 *Algorithm 2 is correct and its time complexity is* $O(2^{2*\sqrt{\ell}} * \sqrt{\ell} + n^2 * \sqrt{\ell})$ *while* W *is over binary alphabet.*

Proof. Correctness: The CMP table records Hamming distance between any two possible enumerations of sub-words with a fixed length. Thus we can compare any two sub-words by examing the CMP table. Note that, at step 4, each word is divided into $\sqrt{\ell}$ sub-words. From steps $3-11$, it is obvious to see that the algorithm compares any two sub-words in the same group. For each group, the algorithm adds corresponding values to update INF table. After comparing all positions of the word, the cumulative values in INF table correspond to the Hamming distances of any two words. Therefore, Algorithm 2 is correct.

Time complexity: Step 2 costs $O(2^{2*\sqrt{\ell}} * \sqrt{\ell})$ time to construct CMP table, since (1) it takes $O(2^{\sqrt{\ell}})$ time to enumerate all possible combinations of two sub-words ; (2) to record any two pairs of all possible sub-words needs $2^{\sqrt{\ell}} * 2^{\sqrt{\ell}}$ entries; (3) to determine Hamming distance between two sub-words needs $O(\sqrt{\ell})$ time. In steps $3-11$, it is easy to see that it takes $O(n^2 * \sqrt{\ell})$ time to update INF table, and in steps $12-18$ it takes $O(n^2)$ time to check whether the value of each entry in INF table is less than the parameter of the given constraint. Therefore, it takes $O(2^{2*\sqrt{\ell}} * \sqrt{\ell} + n^2 * \sqrt{\ell})$ time in total. □

Corollary 3 *The time complexity of Algorithm 2 is* $O(\alpha^{2*\sqrt{\ell}} * \sqrt{\ell} + n^2 * \sqrt{\ell})$*, where* α *is the size of alphabet.*

Proof. Similarly to the proof of Theorem 2, the corollary is true. □

Here, we give an example for illustrating the steps in Algorithm 2. Assume that there nine words $X_1, \ldots, X_9$ in W with $\ell = 9$. The CMP table constructed at step 2 of Algorithm 2 is shown in Figure 1, and the first iteration of steps $3-11$ is depicted in Figure 2.

Next, we apply Table Lookup method to deal with the verification of C_3, C_4, and C_5 constraints. We first consider the cases of verifying C_3 and C_4 constraints. Unlike the cases of verifying C_1 and C_2 constraints, the algorithm proposed here does not divide a word X_p into sub-words, and then to compare them with sub-words of X_q. Therefore, INF table is not needed here. However, it needs to enumerate all possible combinations of words with length ℓ to construct the CMP table. In addition, the algorithm extracts two sub-words $X_p[1..i]$ and $X_q[(\ell-i+1)..\ell]$ from X_p and X_q, respectively, and concatenates a word Y with the sub-words such that $P = X_p[1..i]+Y$, $Q = X_q(\ell-i+1)..\ell]+Y$, and $|P| = |Q| = \ell$. By doing so, the Hamming distance between two extracted sub-words of X_p and X_q can be obtained from the CMP table. Then the algorithm determines whether the constraint is satisfied or not. The detail of the algorithm which is called TablewithShift is shown in Algorithm 3. Algorithm 3 focuses on verifying C_3 constraint and can be easily extended to verifying C_4 constraint.

Theorem 4 *Algorithm 3 is correct and its time complexity is* $O(2^{2*\ell} * \ell + n^2 * \ell)$.

Algorithm 2 TABLE LOOKUP Algorithm

1: **procedure** TABLE LOOKUP(W) {∗ Output "Fail", if the constraint is not satisfied; "Success", if the constraint is satisfied ∗}
2: Enumerate all possible combinations of sub-words with length $\sqrt{\ell}$ over the binary alphabet to construct the CMP table;
3: **for** g from 1 to $\sqrt{\ell}$ **do**
4: Let $X_{i,g} = X_i[((g-1)*\sqrt{\ell}+1)..g*\sqrt{\ell}]$, $i = 1, 2, \ldots, n$;
5: **for** i from 1 to n **do**
6: **for** j from 1 to n **do**
7: Lookup the value of $d(X_{i,g}, X_{j,g})$ from the CMP table;
8: Let $t_{i,j} = t_{i,j} + d(X_{i,g}, X_{j,g})$;
9: **end for**
10: **end for**
11: **end for**
12: **for** i from 1 to n **do**
13: **for** j from 1 to n **do**
14: **if** $t_{i,j} < k_1$, where $i \neq j$ **then**
15: **return** "Fail";
16: **end if**
17: **end for**
18: **end for**
19: **return** "Success";
20: **end procedure**

Proof. Similar to the proof of Theorem 2, this algorithm is correct. Time complexity of step 2 in Algorithm 3 is $O(2^{2*\ell} * \ell)$ and it takes $O(n^2 * \ell)$ in steps $3-14$. Therefore, the time complexity of Algorithm 3 is $O(2^{2*\ell} * \ell + n^2 * \ell)$. □

For the C_5 constraint, the sub-words $X_a[1..i]$ and $X_a[(\ell-i+1)..\ell]$ of a word X_a for i from $\ell-k_5+1$ to ℓ have to be compared with their reverse complement $X_a[1..i]^{RC}$ and $X_a[(\ell-i+1)..\ell]^{RC}$, respectively, where $a = 1, 2, \ldots, n$. As step 2 in Algorithm 2, it enumerates all possible combinations of sub-words with length $\frac{\log \ell}{2}$ over the binary alphabet to construct a $2^{\frac{\log \ell}{2}}$ by $2^{\frac{\log \ell}{2}}$ CMP table. Then it divides each word in W into $\frac{2\ell}{\log \ell}$ sub-words with length $\frac{\log \ell}{2}$ and looks up CMP table to verify each word in W. The detail of algorithm is described in Algorithm 4.

Theorem 5 *Algorithm 4 is correct and its time complexity is* $O(\ell * \log \ell + \frac{n*\ell*k}{\log \ell})$.

Proof. Similar to the proof of Theorem 2, this algorithm is correct. Time complexity of step 2 in Algorithm 4 is $O(\ell * \log \ell)$ and it takes $O(n * k * \frac{\ell}{\log \ell})$ for other steps, where k is the parameter of the given constraint. Therefore, the time complexity of this algorithm is $O(\ell * \log \ell + \frac{n*\ell*k}{\log \ell})$. □

5 Encoding

In this section, we propose the Encoding method to verify whether or not a set of words satisfies C_1 and C_2 constraints. We apply two data structures to speed up the checking. (1) INF table (Information table): the table records the Hamming distance between any two words in the given set of words as described in Sec 4. (2) NZO linked list: the main function of NZO linked-list is to store the group number of two compared sub-words while the two sub-words are different. NZO linked list consists of two kinds of nodes: Head nodes and Data nodes. Head node stores the label (i, j) of any two words X_i and X_j in data field and a pointer points to Data

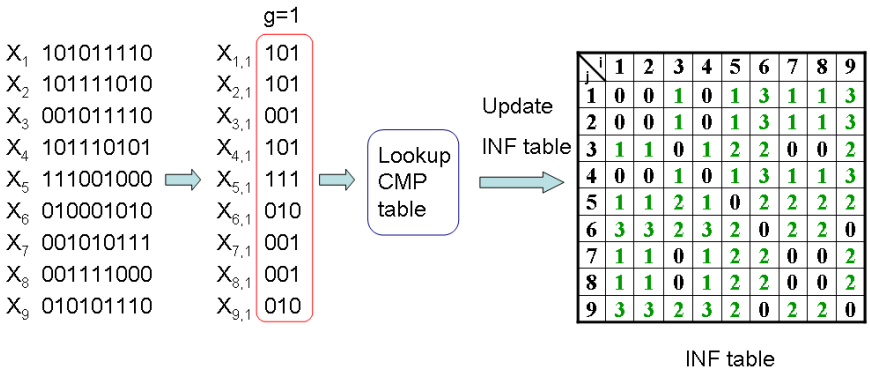

Figure 2: Illustration of the steps in TABLE LOOKUP Algorithm and results in INF table

Algorithm 3 TABLEWITHSHIFT Algorithm

```
1: procedure TableWithShift(W)  {* Output "Fail", if the constraint is not satisfied;
       "Success", if the constraint is satisfied *}
2:     Enumerate all possible combinations of words with length ℓ over the binary alphabet to
           construct the CMP table;
3:     for i from 1 to ℓ do
4:         Let Y = 11...1, where |Y| = ℓ − i;
5:         for p from 1 to n  do
6:             for q from 1 to n do
7:                 Let P = X_p[1..i] + Y and Q = X_q[(ℓ − i + 1)..ℓ] + Y;
8:                 Lookup the value of d(P, Q) from the CMP table;
9:                 if d(P, Q) < k_3 then
10:                    return "Fail";
11:                end if
12:            end for
13:        end for
14:    end for
15:    return "Success";
16: end procedure
```

nodes, where $X_i, X_j \in W$. Therefore, there are n^2 Head nodes in the list. Note that, the initial value in link the field of Head node is "NULL". Data node stores the group ID, which is defined later, of two compared sub-words if the sub-word of X_i and the sub-word of X_j are different. The NZO linked list is illustrated in Figure 3.

In this method, first, each word $X_i \in W$ is divided into $\sqrt{\ell * k}$ sub-words each with length $\sqrt{\frac{\ell}{k}}$. These sub-words are denoted as $X_{i,1}, X_{i,2}, \ldots, X_{i,\sqrt{\ell*k}}$, where $X_{i,g} = X_i[a..b]$, $a = (g-1) * \sqrt{\frac{\ell}{k}} + 1$, $b = g * \sqrt{\frac{\ell}{k}}$, $1 \le g \le \sqrt{\ell * k}$, and k is the parameter of the given constraint. Let $G_g = \{X_{1,g}, X_{2,g}, \ldots, X_{n,g}\}$ denote a set of n sub-words, where g is the group ID of G_g. Note that, $1 \le g \le \sqrt{\ell * k}$. and the total number of groups is $\sqrt{\ell * k}$. Second, using radix sort to sort the sub-words of each group, and then check if any two sub-words in same group are different. If $X_{i,g}$ and $X_{j,g}$ are different, then update the INF table, i.e., $t_{i,j} = t_{i,j} + 1$, and insert a new Data node with group ID g into the corresponding location of the NZO linked list. Note that the initial value of each entry in the INF table is 0. Finally, for each entry in the INF table, check if $t_{i,j} < k$. If the value $t_{i,j}$ of the entry in the INF table is less than k, then scan the corresponding

Algorithm 4 TABLEWITHSELFSHIFT Algorithm

procedure TABLEWITHSELFSHIFT(W) {∗ Output "Fail", if C_5 constraint is not satisfied; "Success", if C_5 constraint is satisfied ∗}
  Enumerate all possible combinations of words with length $\log \ell/2$ over the binary alphabet to construct the CMP table;
  for a from 1 to n **do**
    for i from $\ell - k_5 + 1$ to ℓ **do**
      Let $Y = 11 \ldots 1$, such that $|Y| = \ell - i$;
      Let $X'_p = X_a[1..i] + Y$;
      Let $X'_q = X_a[1..i]^{RC} + Y$;
      Let $Y_p = X_a[(\ell - i + 1)..\ell] + Y$;
      Let $Y_q = X_a[(\ell - i + 1)..\ell]^{RC} + Y$;
      Let $d' = 0$ and $d = 0$;
      for g from 1 to $\frac{2\ell}{\log \ell}$ **do**
        Let $P' = X'_p[((g-1) * \frac{\log \ell}{2} + 1)..(g * \frac{\log \ell}{2})]$;
        Let $Q' = X'_q[((g-1) * \frac{\log \ell}{2} + 1)..(g * \frac{\log \ell}{2})]$;
        Let $P = Y_p[((g-1) * \frac{\log \ell}{2} + 1)..(g * \frac{\log \ell}{2})]$;
        Let $Q = Y_q[((g-1) * \frac{\log \ell}{2} + 1)..(g * \frac{\log \ell}{2})]$;
        Lookup the value of $d(P', Q')$ and $d(P, Q)$ in the CMP table;
        Let $d' = d' + d(P', Q')$;
        Let $d = d + d(P, Q)$;
      end for
      if $d' < k_5$ or $d < k_5$ **then**
        return "Fail";
      end if
    end for
  end for
  return "Success";
end procedure

nodes in the NZO linked list to find the sub-words of X_i and X_j, such that these sub-words are compared again to determine whether $d(i, j) \geq k$. To simplify the discussion, let Head(i, j) denote a Head node in which the value of data field is (i, j) and let Data$(i, j)[g]$ denote a Data node, which is linked to Head(i, j) and stores a group ID g in its data field. The detail of the Encoding method is shown in Algorithm 5.

We give an example to illustrate the steps of Encoding method for verifying C_1 constraint. First, we assume that there are eight words, each word has eight bits, and the given parameter $k = 2$. In steps $3 - 4$, each word is divided into $\sqrt{8 * 2} = 4$ groups, i.e., $G_1, \ldots, G_4$, as shown in Figure 4(a) and Figure 4(b) shows the steps

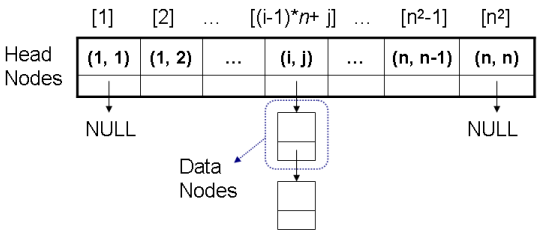

Figure 3: An illustration of the NZO linked list.

Algorithm 5 ENCODING Algorithm

1: **procedure** ENCODING(W) {∗ Output "Fail", if the constraint is not satisfied; "Success", if the constraint is satisfied ∗}
2: **for** g from 1 to $\sqrt{\ell * k_1}$ **do**
3: Let $X_{i,g} = X_i[((g-1) * \sqrt{\frac{\ell}{k_1}} + 1)..g * \sqrt{\frac{\ell}{k_1}}]$, $i = 1, 2, \ldots, n$;
4: Let $G_g = \{X_{1,g}, X_{2,g}, \ldots, X_{n,g}\}$;
5: Perform radix sort on G_g;
6: **for** i from 1 to n **do**
7: **for** j from 1 to n **do**
8: **if** $X_{i,g} \neq X_{j,g}$, where $i \neq j$ **then**
9: $t_{i,j} = t_{i,j} + 1$; {∗ Update INF table ∗}
10: Create a new Data node: Data$(i,j)[g]$ and insert the node into the location after Head(i,j) in the NZO linked list
11: **end if**
12: **end for**
13: **end for**
14: **end for**
15: **for** i from 1 to n **do**
16: **for** j from 1 to n **do**
17: **if** $t_{i,j} < k_1$ **then**
18: Find the Head node Head(i,j) and let $d = 0$;
19: **while** The link field of Head$(i,j) \neq$ NULL **do**
20: Let g' be the value that stored in the data field of a Data node, which is linked after Head(i,j), i.e., Data$(i,j)[g']$, then remove this Data node;
21: Compute Hamming distance between $X_{i,g'}$ and $X_{j,g'}$;
22: Let $d = d + d(X_{i,g'}, X_{j,g'})$;
23: **end while**
24: **if** $d < k_1$ **then**
25: **return** "Fail"
26: **end if**
27: **end if**
28: **end for**
29: **end for**
30: **return** "Success"
31: **end procedure**

$5 - 13$ in the Encoding method.

Theorem 6 *Algorithm 5 is correct and its time complexity is* $O(n^2 * \sqrt{\ell * k})$.

Proof. First, we prove the correctness of the algorithm. The value in the INF table corresponding to two words X_i and X_j only be added when the sub-words of X_i and X_j are different. Therefore, if this value is greater or equal to k, it means $d(X_i, X_j)$ must be greater than or equal to k, where k is the parameter of the given constraint. Our algorithm checks each entry of the INF table to make sure that any two words satisfy the constraint. However, there may be some entries with value less than k, and the sub-words of these entries need to be compared again. To compare those sub-words, the algorithm utilizes the NZO linked list to keep the information of those sub-words as step 10, such that the sub-words can be easily extracted from the words. Therefore, our algorithm is correct. Second, we analyze the time complexity of the algorithm. Each word in the set is partitioned into $\sqrt{\ell * k}$ sub-words each with length $\sqrt{\frac{\ell}{k}}$. After radix sort, it

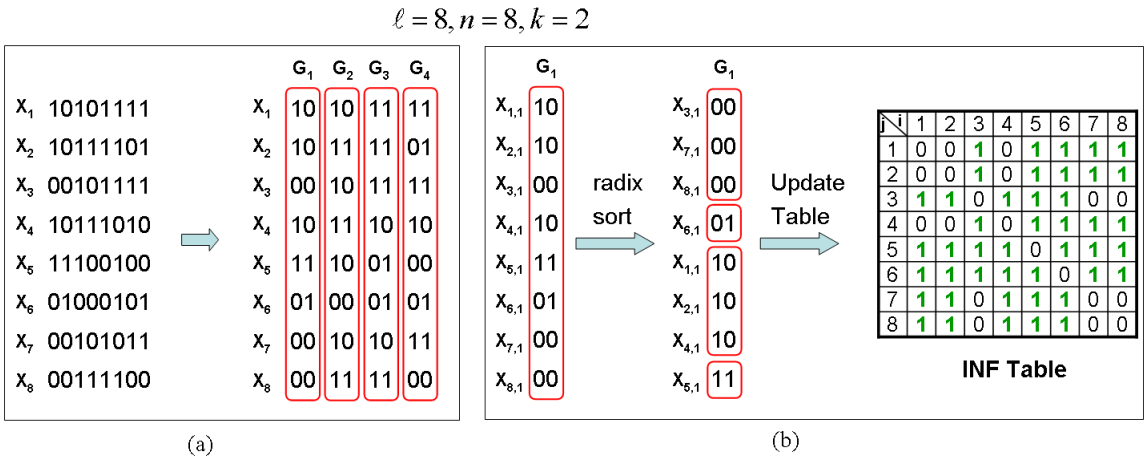

Figure 4: (a) Each word is divided into $\sqrt{\ell * k}$ sub-words each with length $\sqrt{\frac{\ell}{k}}$. (b) An illustration of steps for updating the INF table.

compares any two sub-words, and then updates the INF table and adds new node into the NZO linked list. Thus it takes $O(n^2 * \sqrt{\ell * k})$ in steps $2 - 14$. If there is any entry in the INF table with value less than k, then the number of groups which is stored in corresponding location of the NZO linked list must be less than k. Therefore, using information stored in the entries of the NZO linked list, the algorithm in steps $15 - 29$ compares the sub-words again to determine whether the $d(i, j) \geq k$ in $O(n^2 * \sqrt{\frac{\ell}{k}} * k)$ time. Hence, it takes $O(n^2 * \sqrt{\ell * k})$ time in total. □

Note that, the Encoding method can also be used to verify constraints C_3, C_4 and its time complexity is $O(n^2 * \sqrt{\ell * k} * k)$.

References

[1] M. Arita, A. Nishikawa, M. Hagiya, K. Komiya, H. Gouzu, and K. Sakamoto. Improving Sequence Design for DNA Computing. *Proceedings of the Genetic and Evolutionary Computation Conference*, 875–882, 2000.

[2] A. Brenneman and A. E. Condon. Strand Design for Bio-Molecular Computation. *Theoretical Computer Science*, 287:39–58, 2001.

[3] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens Jr. Reliability and Efficiency of a DNA-based Computation. *Physical Review Letters*, 8(2):417–420, 1998.

[4] J. K. Galllant. String Compression Algorithms. *Ph.D. dissertation, Dept. Elec. Eng. Comput. Sci, Princeton Univ., Princeton, NJ*, 1982.

[5] M. Garzon, R. Deaton, P. Neathery, R. C. Murphy, S. E. Stevens Jr., and D. R. Franceschetti. A New Metric for DNA Computing. *Proceedings of Genetic Programming*, 472–478, 1997.

[6] D. A. Huffman. A Method for the Construction of Minimum Redundancy Codes. *Proceedings of IRE*, 40:1098–1101, 1951.

[7] M. Y. Kao, M. Sanghi, and R. Schweller. Randomized Fast Design of Short DNA Words. *Lecture Notes Computer Science*, 3580:1275–1286, 2005.

[8] A. Marathe, A. Condon, and R. M. Corn. On Combinatorial DNA Word Design *Journal of Computational Biology*, 8:201–219, 2001.

[9] T. Martinetz, J. E. Gewehr and J. T. Kim. Statistical Learning for Detecting Protein-DNA-Binding Sites. *Proceedings*

of the International Joint Conference on Neural Networks, 2003.

[10] W. Shi and W. Zhou. Identifying Transcription Factor Binding Sites in Promoters of Human Genes. *International Multi-Conference on Computing in the Global Information Technology*, 2006.

[11] S. Y. Shin, D. Kim, I. H. Lee, and B. T. Zhang Evolutionary sequence generation for reliable DNA computing. *Proceedings of Congress on Evolutionary Computation*, 79–84, 2002.

[12] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, and A. Ohuchi. Developing support system for sequence design in DNA computing. *Proceedings of The 7th International Workshop on DNA Based Computers*, 340–349, 2002.

[13] J. Vitter. Design and Analysis of Dynamic Huffman Codes. *Journal of the Association for Computing Machinery*, 34(4): 825–845, 1987.

[14] E. H. Yang and Z. Zhang The Shortest Common Superstring Problem: Average Case Analysis for Both Exact and Approximate Matching. *IEEE Transactions on Information Theory*, 45(6), 1999.