On Combinatorial DNA Word Design*

Amit Marathe
Computer Sciences Department
1210 West Dayton Street
University of Wisconsin
Madison, WI 53706

Anne E. Condon
Computer Sciences Department
1210 West Dayton Street
University of Wisconsin
Madison, WI 53706

Robert M. Corn
Chemistry Department
1101 University Avenue
University of Wisconsin
Madison, WI 53706

February 25, 1999

*Research supported by NSF and DARPA via grants CCR-96-13799 and CCR-97-25021. The authors' email addresses are: marathe@cs.wisc.edu, condon@cs.wisc.edu, corn@chem.wisc.edu.

Abstract

We consider the problem of designing *DNA codes*, namely sets of equi-length words over the alphabet $\{A, C, G, T\}$ that satisfy certain combinatorial constraints. This problem is motivated by the task of reliably storing and retrieving information in synthetic DNA strands, for use in DNA computing or as molecular bar codes in chemical libraries. The primary constraints that we consider, defined with respect to a parameter d, are as follows: for every pair of words w, x in a code, there are at least d mismatches between w and x if $w \neq x$; and also between the reverse of w and the Watson-Crick complement of x. Extending classical results from coding theory, we present several upper and lower bounds on the maximum size of such DNA codes and give methods for constructing such codes.

An additional constraint that is relevant to the design of DNA codes is that the free energies and enthalpies of the code words, and thus the melting temperatures, be similar. We describe dynamic programming algorithms that can (i) calculate the total number of words of length n whose free energy value (as approximated by a formula of Breslauer et al.) falls in a given range, and (ii) output a random such word. These algorithms are intended for use in heuristic algorithms for constructing DNA codes.

1 Introduction

The design of codes that satisfy combinatorial constraints has long been studied, motivated by the problem of sending information reliably over a noisy channel [18]. In this paper, we study code design problems that are motivated by the task of storing and retrieving information in short DNA strands, which we refer to as DNA *code words*. A (single) DNA strand is a sequence of nucleotides; there are four possible nucleotides, denoted A, C, G, and T, at each position of the sequence, which has chemically distinct ends known as the $3'$ and $5'$ ends. Since a DNA strand of length n can be used to represent one of up to 4^n possible values, and since short DNA strands can be quickly and cheaply synthesized, DNA code words can be used to store information at the molecular level, thus providing a basis for biomolecular computation [1]. DNA code words are also used as molecular bar codes, or tags, for the purpose of manipulating and identifying individual molecules in complex chemical libraries [3, 4, 20].

These applications require success in achieving specific hybridization between a DNA code word and its Watson-Crick complement, while minimizing false positive and false negative signals, as we now explain. The Watson-Crick complement of a DNA strand is the strand obtained by replacing each A by a T and vice versa, each C by a G and vice versa, and switching the $3'$ and $5'$ ends. For example, the Watson-Crick complement of $3' - AACATG - 5'$ is $5' - TTGTAC - 3'$. Specific hybridization is the process whereby a strand and its Watson-Crick complement bond to form a double helix. Specific hybridization can be used (along with other methods) to identify and retrieve target DNA code words from a set of such code words. A false positive results when non-specific hybridization occurs, such as between a DNA strand and the Watson-Crick complement of a distinct DNA strand, in which case there are *mismatches*. For example, there are two mismatches between $3' - AACATG - 5'$ and $5' - TAATAC - 3'$, in the second and third positions from the $3'$ end of the first strand. Non-specific hybridization may also occur between a DNA strand and the reverse of a distinct strand. A false negative occurs when hybridization between a DNA strand and its complement does not take place as intended.

Several papers have proposed the use of combinatorial constraints on the composition of a set of DNA code words, in order to limit false positives and false negatives in specific applications [1, 2, 3, 4, 7, 8, 11, 12, 13, 19, 20]. Our premise is that a theoretical framework for designing sets of DNA code words should be useful for scalable use of DNA code words.

We focus on sets of words satisfying one or more of four constraints, which we next define and motivate. In our study, we represent a DNA code word simply as a string over the alphabet $\{A, C, G, T\}$ and assume that the leftmost (or low order) end of the string corresponds to the $3'$ end of the associated DNA code word. Thus, CCGAT represents $3' - CCGAT - 5'$, for example. It is useful to define a *word* to be a string over a finite alphabet, where the alphabets of most interest to us are of size 2 or 4 (such as $\{A, C, G, T\}$). Let $x = x_1 x_2 \ldots x_n$ be a word. The *reverse* of x, denoted by x^R, is the word $x_n x_{n-1} \ldots x_1$. If x is over the alphabet $\{A, C, G, T\}$ then the *complement* of x, denoted by x^C, is the word obtained by replacing each A in x by T and vice versa, and by replacing each C in x by G and vice versa. If x is over the binary alphabet $\{0, 1\}$ then x^C is obtained by replacing each 0 in x by 1 and vice versa. Finally, the Hamming distance $H(x, w)$ between x and word $w = w_1 w_2 \ldots w_n$ is the number of indices i for which $w_i \neq x_i$. The following constraints pertain to a set of words, each of length n.

- The **Hamming constraint** with distance parameter d is that for all pairs of distinct words w, x in the set, $H(w, x) \geq d$. A set of words of size M satisfying the Hamming constraint is called a (n, M, d) *code*, or when the parameters are implied, simply a *code*. We let $A_q(n, d)$ denote the maximum size of a code with words of length n over alphabet size q. In most reports on the use of DNA codes, a high Hamming distance is enforced between pairs of code words (see for example [3, 7, 11, 20]), in order to limit non-specific hybridization whereby the Watson-Crick complement of a code word x anneals to a distinct word w.

- The **reverse-complement constraint** with parameter d is that for all pairs of words w, x in the set (where w may equal x), $H(w^C, x^R) \geq d$. In DNA code applications, the reverse-complement constraint is intended to limit hybridization between a code word and the reverse of another code word [7, 11]. We call a code that also satisfies the reverse-complement constraint a reverse-complement code. We let $A_q^{RC}(n, d)$ denote the maximum size of a reverse-complement code, with parameters n, d defined as for codes.

- The **reverse constraint** with parameter d is that, for all pairs of words w, x in the code, $H(w, x^R) \geq d$. We call a code that also satisfies the reverse constraint a reverse code. We let $A_q^R(n, d)$ denote the maximum size of a reverse code. Our study of this constraint is not motivated directly by the goal of limiting false positives or false negatives in the use of DNA code words, but indirectly by a close relationship (presented in Section 4) between $A_q^{RC}(n, d)$ and $A_q^R(n, d)$. For example, when n is even, one can obtain a reverse complement code from a reverse code simply by complementing the symbols in the second half of each word in the code. Thus, constructions of reverse codes can easily be adapted to obtain constructions of reverse complement codes. We consider the reverse constraint to be simpler than the reverse-complement constraint and so focus on this.

- The final constraint that we consider have a somewhat different flavor. It is motivated by the goal that all code words in the set have similar melting temperature, allowing hybridization of multiple words to proceed simultaneously [20]. The melting temperature of a short DNA strand can be accurately estimated using a formula of Wetmur [21], which in turn uses estimates of two further parameters of a DNA strand, namely its free energy and enthalpy due to Breslauer et al [5]. We use ΔG to denote the Breslauer estimate of the free energy of a DNA word. As explained in Section 5, the free energy constraint is essentially the sum of weights associated with substrings of length 2 in a word. The **free energy constraint** with parameters g, ϵ is that for all words in the code, $g - \epsilon \leq \Delta G \leq g + \epsilon$. A closely related constraint, also defined precisely in Section 5, is that the enthalpy of all words in the code lies within a small range.

Section 3 summarizes some well known results on codes. Our new results, presented in Section 4, focus on reverse and reverse-complement codes. Following is a summary of these results.

We first show a close relationship between the maximum sizes of reverse codes and reverse-complement codes. Specifically,

$$A_4^{RC}(n, d) = A_4^R(n, d) \qquad \text{when } n \text{ is even, and}$$

$$A_4^R(n, d+1) \leq A_4^{RC}(n, d) \leq A_4^R(n, d-1) \qquad \text{when } n \text{ is odd.}$$

We then show several methods for obtaining upper and lower bounds on the size of reverse codes with alphabet sizes 2 and 4, including the following.

- **Halving bound:** $A_q^R(n,d) \leq A_q(n,d)/2$. This bound follows from the fact that if S is a $(n, |S|, d)$ reverse code then $S \cup S^R$ is a $(n, 2|S|, d)$ code.
- **Construction for $d = 2$:** We give a simple inductive construction of a reverse code that is optimal for even n, and close to optimal for odd n. For $q = 2$ or 4,

$$\begin{aligned} &A_q^R(n,2) = q^{n-1}/2 && \text{when } n \text{ is even, and} \\ &(q^{n-1} - q^{\lfloor n/2 \rfloor})/2 \leq A_q^R(n,2) \leq q^{n-1}/2 && \text{when } n \text{ is odd.} \end{aligned}$$

- **Product bound:** $A_4^R(n,d) \geq A_2^R(n,d) A_2(n,d)$. In particular, reverse codes over an alphabet of size 4 can be obtained by taking the "product" of a reverse code and a code, both over an alphabet of size 2.
- **Doubling construction:** We show how to construct $(2^n, 2^{n-1}, 2^{n-1})$ binary reverse codes, which are optimal, i.e. $A(2^n, 2^{n-1}) = 2^{n-1}$.

We apply these results to obtain explicit bounds for $A_2^R(n,d)$ and $A_4^R(n,d)$ for $2 \leq n \leq 16$ and $2 \leq d \leq 8$. These are presented in Tables 2 and 3.

In Section 5, we turn to the free energy constraint. Since this is more complex combinatorially than the Hamming or reverse constraints, we are interested in an efficient algorithm for generating code words that satisfy the free energy constraint. Our main contribution in Section 5 is a dynamic programming algorithm that calculates the total number of words of a specified length n whose free energy value (as approximated by a formula of Breslauer) equals a given specified value. The running time of the algorithm is $O(n^2)$, (where the hidden constant depends on the values in the Breslauer formula; details are given in Section 5). Variations of the algorithm can calculate the total number of words of length n whose free energy value or enthalpy falls in a given range, or output a random such word. These algorithms could be used by a program for generating DNA codes, based for example on simulated annealing, which has proved valuable in the construction of binary codes [9].

2 Related work

Deaton et al. [7, 8] observe that the well-known sphere-packing bound (see Section 3) can be used to upper bound the size of DNA codes satisfying the Hamming constraint. They describe genetic algorithms for finding DNA codes that satisfy several constraints, including the Hamming and reverse complement constraints.

In his patent on methods for sorting polynucleotides using DNA tags, Brenner [3] gives a greedy algorithm for generation of DNA codes satisfying the Hamming distance constraint. Brenner also considers sets of words over an alphabet of size 3, where one of the 4 possible nucleotides A, C, G,

or T is absent. Mir [19] also proposed a word design for use in DNA computing over an alphabet representing just 3 of the 4 possible nucleotides.

For prototype experiments of the Wisconsin DNA computing project, a DNA word design with words of length 16 was developed [11]. The internal 8 bases of a word are constrained to satisfy exactly the Hamming and reverse complement constraints with $d = 4$. In addition, the words also satisfy the constraint that 4 out of the 8 bases are from the set $\{C, G\}$. A set of words of size 108 satisfying these constraints was found.

Shoemaker et al. used an algorithm to generate a set of 9,105 20-mers that satisfy the Hamming constraint with $d = 5$ and are predicted to have similar melting temperatures $(61 \pm 5^o C)$. In addition, the words in the set are predicted to have no secondary structure. They give no details on their algorithm.

Finally, we list other combinatorial constraints that are relevant to DNA code design but which we do not study in this paper. (i) The first arises, for example, in Brenner and Lerner's work [4], where DNA tags and the polymers to be tagged are chemically synthesized in an alternating parallel fashion. Thus each code word (or tag) is the concatenation of "units," one per monomeric chemical unit in the polymer. The units are designed to have the *comma free* or frame-shift constraint: no unit x occurs as a substring in the concatenation of two other distinct units yz. The purpose of imposing this constraint is to limit the possibility of "frame shift errors" in the hybridization process. A greedy algorithm for generating words satisfying a similar frame shift constraint is given by Garzon et al. [12, 13]. (ii) Baum [2] developed bounds on the size of DNA word sets satisfying several combinatorial constraints, key among them being the *subword constraint* that for some parameter d, for any pair of distinct words w and x, no subword of w of length d equals a subword of x of length d. (iii) In designing words for surface-based DNA computing, Frutos et al. [11] enforced the *GC content constraint* that the fraction of G's and C's in each code word be $1/2$. The motivation for this is to limit the range of melting temperatures of the code words. (iv) Finally, the important *forbidden subwords constraint*, already mentioned in relation to the work of Shoemaker et al. [20] above, is that no word in the code set contains as a subword a specified set of undesirable words, such as DNA strands with secondary structure, strands that are be used as PCR primers, or strands that are recognized by restriction enzymes.

3 Bounds on the size of a code

In this section, we briefly review previous results on codes that will later be extended or applied in obtaining bounds on reverse codes. The text by MacWilliams and Sloane [18] provides a good introduction to the subject. Table 1 gives some upper and lower bounds on $A_4(n, d)$, or equivalently, on the maximum size of a DNA code with code words of length n and distance parameter d. We use several of these upper bounds on $A_4(n, d)$ to obtain the upper bounds of Table 2 for reverse codes over alphabet of size 4, via application of our halving bound (Theorem 4.4). In addition, we use known lower bounds on $A_2(n, d)$ to construct reverse codes over alphabet of size 4, via application of our product bound (Theorem 4.6). By extending known techniques for construction of codes to handle reversals, we obtain further bounds on the size of reverse codes.

The following two bounds on $A_q(n, d)$ are described in terms of two quantities. Let S be the set from which words in the code are drawn. Then the first quantity we need is $|S|$, which is clearly

q^n. Let $V(s, d')$ be the number of words of S that have distance at most d' from word s, where $s \in S$. $V(s, d')$ is independent of s; denote it by $V(d')$. Here, $V(d') = \sum_{i=0}^{d'} \binom{n}{i} (q-1)^i$, where $\binom{n}{i} = \frac{n(n-1)...(n-i+1)}{i(i-1)...21}$ denotes the number of ways to choose i distinct items from a set of size n. Proofs of following four bounds can be found in a survey article by Ericson [10] or the text by MacWilliams and Sloane [18].

Theorem 3.1 (Sphere-Packing upper bound)

$$A_q(n, d) \leq \frac{|S|}{V(\lfloor (d-1)/2 \rfloor)} = \frac{q^n}{\sum_{i=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{i} (q-1)^i}.$$

The sphere-packing bound holds because the "spheres" of radius $\lfloor (d-1)/2 \rfloor$ around each code word s in a code, namely the sets of size $V(\lfloor (d-1)/2 \rfloor)$ for all words s in the code, cannot overlap.

Theorem 3.2 (Gilbert-Varshamov lower bound)

$$A_q(n, d) \geq \frac{|S|}{V(d-1)} = \frac{q^n}{\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i}.$$

A simple greedy algorithm for constructing a code yields the Gilbert-Varshamov lower bound: repeatedly select a word w from S to be in the code, and remove from S all words that are of distance less than d from w. At each selection, at most $V(d-1)$ words are removed from S and so the selection step can be repeated at least $\frac{|S|}{V(d-1)}$ times.

Theorem 3.3 (Singleton upper bound)

$$A_q(n, d) \leq q^{n-d+1}.$$

Theorem 3.4 (Plotkin upper bound) *For* $d > \frac{q-1}{q} n$,

$$A_q(n, d) \leq \frac{qd}{qd - (q-1)n}.$$

The following basic relationships are also useful:

Theorem 3.5 *1.* $A_q(n,n) = q$,

2. $A_q(n,d) \geq A_q(n+1,d+1)$, *and*

3. $A_q(n,d) \geq A_q(n+1,d)/q$.

Part 1 of Theorem 3.5 is true because the code consisting of q words, each containing a different letter repeated n times, is an example of a (n,q,n)-code. The Singleton upper bound shows that the size of this code is the best possible.

To see part 2, note that an $(n, A_q(n+1,d+1), d)$-code can be obtained from a $(n+1, A_q(n+1,d+1), d+1)$-code by removing the first letter of each code word.

To see part 3, note that if we partition all the words in a $(n+1, A_q(n+1,d), d)$ code into q subsets according to the starting letter, one of the subsets has size at least $A_q(n+1,d)/q$ and thus is a $(n+1, A_q(n+1,d)/q, d)$ code. By removing the (common) starting letter from all words in this largest subset, a $(n, A_q(n+1,d)/q, d)$ code is obtained.

Most of the lower bounds on $A_4(n,q)$ listed in Table 1 are obtained from tables of cyclic codes of Kschischang and Pasupathy [17]. We note that some of the underlying cyclic codes contain palindromic code words. If the coefficient vector for the generator polynomial for a cyclic code is palindromic, then the code contains palindromic words. For example, the codes of Kschischang and Pasupathy with $n = 8, 12, 16$, or 30 and $d = 3$, or $n = 10$ and $d = 4$, are generated by palindromic generator polynomials.

Brouwer et al. [6] give a table of upper and lower bounds for $A_2(n,d)$ which we use in obtaining some of our bounds on reverse codes. These bounds are obtained using a wide range of methods, and we do not comment on them further here.

4 Bounds on the Size of Reverse and Reverse-Complement Codes

We now present new bounds on the maximum size of reverse codes and reverse complement codes, as defined in the introduction. Recall that for $q \in \{2,4\}$, $A_q^{RC}(n,d)$ denotes the maximum size of a code of length n over an alphabet of size q that satisfies the reverse and reverse-complement constraints. Similarly, $A_q^R(n,d)$ denotes the maximum size of a code satisfying the Hamming and reverse constraints.

There is a close relationship between the size of reverse and reverse-complement codes for the alphabet $\{A,C,G,T\}$.

Theorem 4.1

$$A_4^{RC}(n,d) = A_4^R(n,d), \text{ for } n \text{ even, and}$$

$$A_4^R(n,d+1) \leq A_4^{RC}(n,d) \leq A_4^R(n,d-1), \text{ for } n \text{ odd.}$$

Proof First, suppose that n is even. Let $\{x_i\}$ be a (n, M, d) reverse code. Write each $x_i = a_i b_i$, where the lengths of a_i and b_i are equal. Then $\{a_i b_i^C\}$ is a (n, M, d) reverse complement code, and so $A_4^R(n,d) \leq A_4^{RC}(n,d)$. We can prove the other inequality in the same manner.

When n is odd, truncating a $(n, A_4^R(n, d+1), d+1)$ code by removing the middle letter of each code word and then applying the previous result for n even gives $A_4^R(n, d+1) \leq A_4^{RC}(n-1, d)$. Since $A_4^{RC}(n-1,d) \leq A_4^{RC}(n,d)$ for odd n, the first inequality in the case that n is odd follows. The proof of the second inequality is similar. □

The remaining results in this section pertain to reverse codes. Extending the proof of the sphere-packing and Gilbert-Varshamov bounds, i.e. Theorems 3.1 and 3.2, we obtain the following bound for reverse codes with $d = 3$.

Theorem 4.2 *For $n \geq 4$,*

$$A_q^R(n,3) \leq \frac{q^{\lceil n/2 \rceil} \sum_{i=2}^{\lfloor n/2 \rfloor} \binom{\lfloor n/2 \rfloor}{i} (q-1)^i}{2(1 + 4(q-2) + (n-4)(q-1))}.$$

Proof Let S be the set of all words x (length n, alphabet size q) such that $H(x, x^R) \geq d$. Also, let $V(s, d')$ be the number of words of S that have distance at most d' from word $s \in S$ and let $V^-(d') = \min\{V(s,d')\}$ where the min is taken over all s in S.

Following the proof of the sphere-packing bound for codes, to obtain an upper bound on the size of a code drawn from S we consider the set D_x, consisting of words which are disqualified when a word x from S is chosen to belong to the code. A lower bound on D_x is given by $2V^-(\lfloor (d-1)/2 \rfloor)$. This is because for any word s in S, $H(s, s^R) \geq d$ and hence $V(s, \lfloor (d-1)/2 \rfloor)$ is disjoint from $V(s^R, \lfloor (d-1)/2 \rfloor)$. Therefore,

$$A_q^R(n,d) \leq \frac{|S|}{2V^-(\lfloor (d-1)/2 \rfloor)}.$$

We first calculate the size of S. Note that if $x = x_1 x_2 \ldots x_n$ then $x_j^R = x_{n-j+1}$. We say a mismatch occurs at j if $x_j \neq x_j^R$, i.e. if $x_j \neq x_{n-j+1}$. If a mismatch occurs at j then by symmetry a mismatch also occurs at x_{n-j+1}. In fact, $H(x, x^R)$ is always even.

How many words x have $H(x, x^R) = 2i$? The number of such words is the number of words that have i mismatches at indices $j \leq \lfloor n/2 \rfloor$. There are $\binom{\lfloor n/2 \rfloor}{i}$ choices for these i indices. At each chosen index j, there are q choices for the letter x_j and once this is chosen, there are $q-1$

choices for the letter x_{n-j+1}. Also, there are $\lceil n/2 \rceil - i$ indices j of x at which $x_j = x_{n-j+1}$. There are q choices for the value of x_j at each of these indices. In total, there are

$$\binom{\lfloor n/2 \rfloor}{i} (q(q-1))^i q^{\lceil n/2 \rceil - i}$$

words x such that $H(x, x^R) = 2i$. Therefore,

$$|S| = \sum_{i=\lceil d/2 \rceil}^{\lfloor n/2 \rfloor} \binom{\lfloor n/2 \rfloor}{i} (q(q-1))^i q^{\lceil n/2 \rceil - i} = q^{\lceil n/2 \rceil} \sum_{i=\lceil d/2 \rceil}^{\lfloor n/2 \rfloor} \binom{\lfloor n/2 \rfloor}{i} (q-1)^i.$$

Next, consider a word $x = x_1 x_2 \ldots x_n$ in S. We claim that for $d = 3$ there are at least $4(q-2) + (n-4)(q-1)$ words of distance exactly 1 from x, and therefore $V^-(1) = 1 + 4(q-2) + (n-4)(q-1)$. To show the claim, let j and j' be such that $1 \leq j < j' \leq n/2$, $x_j \neq x_{n-j+1}$ and $x'_j \neq x_{n-j'+1}$ (j, j' exist because $H(x, x^R)$ is even and at least 3). For each of the four possible indices i with $i \in \{j, j', n-j+1, n-j'+1\}$, there are $q-2$ ways to change x_i to obtain a word x' of distance 1 from x, such that $H(x', (x')^R) = H(x, x^R) \geq 3$. For each of the $n-4$ remaining indices i, there are $q-1$ ways to change x_i to obtain a word x' of distance 1 from x, such that $H(x', (x')^R) \geq 3$. Thus, there are at least $4(q-2) + (n-4)(q-1)$ words of distance exactly 1 from x, as required. □

The upper bound of Theorem 4.2 can easily be generalized to $d > 3$ by lower bounding $V^-(\lfloor (d-1)/2 \rfloor)$. Generalizing the argument above, it is not hard to show that $V^-(d') \geq \sum_{i=0}^{d'} \binom{n}{i} (q-2)^i$. However, this generalization was not useful in obtaining Tables 2 and 3.

Theorem 4.3

$$A_q^R(n, d) \geq \frac{|S|}{2V^+(d-1)},$$

where $V^+(d) = max\ \{V(s, d) | s \in S\}$ *and* S *is as in Theorem 4.2.*

Proof As in Theorem 3.2, a greedy algorithm provides a reverse code of size $|S|/2V^+(d-1)$. □

Theorem 4.4 (Halving bound)

$$A_q^R(n, d) \leq \frac{A_q(n, d)}{2}.$$

Proof The reverse constraint implies that $H(x, x^R) \geq d$ for every word x in S. Also $x \in S \Rightarrow x^R \notin S$. Thus, starting with a set S satisfying the Hamming and reverse constraints we can get a new set by adding to S the reversals of all words in it. This new set satisfies the Hamming constraint because the new words added are at least distance d apart from each other and from the original words in S (this follows from the fact that S satisfies the Hamming and reverse constraints). The new set is twice the size of the original. □

Theorem 4.5 (d=2 Construction)

$$A_q^R(n, 2) = \frac{q^{n-1}}{2}, \textit{ for even } n \textit{ and } q \in \{2, 4\}, \textit{ and}$$

$$A_q^R(n, 2) \geq \frac{q^{n-1} - q^{\lfloor n/2 \rfloor}}{2}, \textit{for odd } n \textit{ and } q \in \{2, 4\}.$$

Proof The proof builds on the following claim:

Claim 4.1 *For even n, Σ^n can be partitioned into subsets, each containing q^{n-1} words, such that*

1. *any two words from the same subset differ in at least two positions,*
2. *if a word belongs to a subset, its reversal is also in the same subset, and*
3. *all the $q^{n/2}$ palindromes are in the same subset.*

Proof (of Claim) The partitions for the base case ($n = 2$) can be $S_1^2 = \{AA, CC, GG, TT\}$, $S_2^2 = \{AC, CA, GT, TG\}, S_3^2 = \{AG, GA, CT, TC\}, S_4^2 = \{AT, TA, CG, GC\}$ for $q = 4$ and $\{00, 11\}, \{01, 10\}$ for $q = 2$.

For the induction case, when $q = 4$, assume that we have a partition S_i^n of Σ^n, $i \in \{1, 2, 3, 4\}$, with the above properties, with S_1^n containing all of the palindromes. Then S_i^{n+2} for $i \in \{1, 2, 3, 4\}$ can be defined as follows.

$$S_1^{n+2} = S_1^n.S_1^2 \cup S_2^n.S_2^2 \cup S_3^n.S_3^2 \cup S_4^n.S_4^2,$$
$$S_2^{n+2} = S_1^n.S_2^2 \cup S_2^n.S_3^2 \cup S_3^n.S_4^2 \cup S_4^n.S_1^2,$$
$$S_3^{n+2} = S_1^n.S_3^2 \cup S_2^n.S_4^2 \cup S_3^n.S_1^2 \cup S_4^n.S_2^2,$$
$$S_4^{n+2} = S_1^n.S_4^2 \cup S_2^n.S_1^2 \cup S_3^n.S_2^2 \cup S_4^n.S_3^2,$$

where $A.B = \{pwq | w \in A, pq \in B, |p| = |q| = 1\}$.

It is not difficult to verify that this is a partition of Σ^{n+2} having all the three properties, with S_1^{n+2} containing the palindromes. The induction step for $q = 2$ utilizes a similar "product-of-sets" construction. As an example, when $q = 2$, the two subsets S_1^4 and S_2^4 obtained by the above construction are as follows:

S_1^4	S_2^4
0000	0010
0110	0100
1001	1011
1111	1101
0011	0001
0101	0111
1010	1000
1100	1110

□

Now, to complete the proof of Theorem 4.5 when n is even, note that if we take any of the subsets not containing any palindromes and drop half of the words from it (either a word or its reversal), we get a set satisfying the Hamming and reverse constraints (for $d = 2$). The identical upper bound follows from Theorem 4.4 combined with Theorem 3.3.

The construction for odd $n > 1$ uses the sets S_i^{n-1} obtained above as follows:

$$S_1^n = S_1^{n-1}.A \cup S_2^{n-1}.C \cup S_3^{n-1}.G \cup S_4^{n-1}.T,$$

$$S_2^n = S_1^{n-1}.C \cup S_2^{n-1}.G \cup S_3^{n-1}.T \cup S_4^{n-1}.A,$$

$$S_3^n = S_1^{n-1}.G \cup S_2^{n-1}.T \cup S_3^{n-1}.A \cup S_4^{n-1}.C,$$

$$S_4^n = S_1^{n-1}.T \cup S_2^{n-1}.A \cup S_3^{n-1}.C \cup S_4^{n-1}.G,$$

where $B.X = \{w_1 X w_2 | w_1 w_2 \in B, |w_1| = |w_2|\}$. With this construction, each of the four subsets S_i^n has $q^{\lfloor n/2 \rfloor}$ palindromes. By first removing these palindromes from each subset and then dropping half of the remaining words (either a word or its reversal), we obtain four reverse codes with parameter $d = 2$, each of size $(1/2)(q^{n-1} - q^{\lfloor n/2 \rfloor})$. □

Theorem 4.6 (Product Bound)

$$A_4^R(n, d) \geq A_2^R(n, d) A_2(n, d).$$

Proof Let B be a code and let A be a reverse code, both over alphabet $\{0, 1\}$, with words of length n. Then each element of the Cartesian product $A \times B$ corresponds to a word over an alphabet of size 4, where a bit in A determines whether A/T or G/C appear in that position, while the corresponding bit in B makes a choice from the 2 remaining possibilities. Moreover, this map is one-to-one and the set of $|A||B|$ words so obtained satisfies the Hamming and reverse constraints. □

Theorem 4.7 *(***Doubling Construction***) For* $n \geq 2$

$$A_2^R(2^n, 2^{n-1}) = 2^n.$$

Proof Let C be a code with words of length n. Call C a $H^{RC}(n,d)$ code if it has the following property: for all words x, y in C,

$$\begin{array}{ll} H(x,y) \geq d, & \text{if } x \neq y \\ H(x,y^R) \geq d, & \\ H(x,y^C) \geq d, \text{ and} & \\ H(x,y^{RC}) \geq d. & \end{array}$$

Claim 4.2 *Let C be a $H^{RC}(n,d)$ code with $d \leq n/2$. Then the code $C' = CC \cup CC^C$ is a $H^{RC}(2n,2d)$ code, where $CC = \{xx | x \in C\}$ and $CC^C = \{xx^C | x \in C\}$. Moreover, the size of C' is twice that of C.*

Proof (of Claim) It is straightforward to check that for all words $x', y' \in C'$, the four conditions of a $H^{RC}(2n,2d)$ code are met. (In some cases, a condition is met because there is a Hamming distance of d between both halves of the words being compared; in other cases it is met because half of one string is x, the corresponding half of the other is x^C which differs in n positions, and $d \leq n/2$). The size of C' is twice that of C because $H(x,y^C) \geq d$ for all x, y in C, and so the words in CC and CC^C are disjoint. □

We now show that for $n \geq 2$, there is a $H^{RC}(2^n, 2^{n-1})$ code of size at least 2^{n-1}. If C is such a code, then $C \cup C^C$ satisfies the Hamming and reverse constraints and has twice the size of C, and from this the lower bound of the theorem follows. For the upper bound, we have that $A_2(4r,2r) = 8r$ from [18], which by the halving bounds (Theorem 4.4) implies that $A_2^R(2^n, 2^{n-1}) \leq 2^n$ for $n \geq 2$.

Let $n = 2$. It is straightforward to verify that $\{0111, 0010\}$ is a $H^{RC}(4,2)$ code of size 2. The construction of the claim then inductively yields a $H^{RC}(2^n, 2^{n-1})$ code of size 2^{n-1} for all $n > 2$, as required. □

Note: It is possible to carry out a "quadrupling" construction in the case $q = 4$ (similar to the doubling construction for the binary case) and thus get a direct lower bound on $A_4^R(n,d)$ for special values of n, d. However, this does not lead to improved results in our tables.

Finally, some useful basic relationships between the sizes of reverse codes are summarized in the following theorem:

Theorem 4.8 *1.* $A_4^R(n,n) = \begin{cases} 2 & \text{if } n \geq 2 \text{ is even, and} \\ 0 & \text{otherwise,} \end{cases}$

2. $A_q^R(n-1,d) \leq A_q^R(n,d) \leq A_q^R(n,d-1)$, *and*

3. $A_q^R(n-1,d) \geq A_q^R(n,d)/q$, *for odd* n.

To see part 1, note that when $n = 2k, k \geq 1$ the code $\{A^kT^k, C^kG^k\}$ clearly satisfies both constraints. Also if w is a word in such a code then the first letter of w must be different from the

first and last letters of all other words in the code. Therefore the code can contain at most two words. For odd n, the middle letter always results in a match when any word is compared with its reversal. Hence no word can belong to the code.

The proof of part 2 is straightforward. The proof of part 3 is similar to part 3 of Theorem 3.5, the only change being that we now partition based on the middle letter (instead of the first).

5 The Free Energy Constraint

In this section, we present an algorithm to calculate the number of DNA strands (of a certain length) whose free energy equals a given value. The algorithm relies on a heuristic proposed by Breslauer et. al. [5] to approximate the free energy of any DNA strand. Using another heuristic from the same paper it is possible to modify the algorithm for the calculation of the number of DNA strands having a particular enthalpy using the formula of Breslauer et al.

The data produced by the above algorithms can be used to efficiently generate a random strand with free energy/enthalpy close to a given value. This data could be used, for example, by a simulated annealing algorithm for finding a set of DNA strands with similar melting temperatures.

5.1 Algorithm Outline

The free energy of the DNA strand $u_1u_2 \ldots u_l$ is approximated by the following formula from Breslauer et. al. [5]

$$\Delta G_{total} = correction\ factor + \sum_{i=1}^{l-1} w(u_i, u_{i+1})$$

where $w(x, y)$ is the observed free energy of the 2-mer xy. Thus, the free energy is hypothesized to depend only on the nearest-neighbor interactions of nucleotides in the strand. The enthalpy ΔH_{total} is approximated similarly.

Let $N(l, u, e)$ be the number of DNA strands of length l, beginning with nucleotide u which have free energy e. Then $N(l, u, e)$ can be calculated for $l > 1$ from "previous" values by the following equation.

$$N(l, u, e) = \sum_{v \in \{A,C,G,T\}} N(l-1, v, e - w(u, v))$$

with the convention that strands of length 1 have free energy 0 and that $N(l, u, e)$ equals 0 when $e < 0$.

The correctness of the above equation can be proved by making a case analysis on the second nucleotide in the DNA strand; this nucleotide has to be $A/C/G/T$ and accordingly the free energy of the tail (the strand comprising of the last $l-1$ nucleotides) is $e - w(u, A/C/G/T)$.

5.2 Pseudocode

The following pseudocode elaborates on the algorithm outlined in the previous section. It sets entries in the N-array (which is passed to the *free_energy* procedure as a reference parameter) to their correct values.

```
procedure free_energy(integer L, integer S, array integer w[S][S],
        reference array integer N[L][S][E])
local integer E;
// L = word length
// S = alphabet size
// w[S][S] = 10 * free energies of all 2-mers
// E = 10 * upper bound on the free energy of any strand of length L

begin
  // first calculate M, the maximum free energy of a 2-mer, and
  // use it to initialize E
  M = -1;
  for u = 1 to S
    for v = 1 to S
      if (M < w[u][v])
  then M = w[u][v];
  E = L * M;

  for u = 1 to S
    N[1][u][0] = 1;      // base case for the dynamic programming algorithm
                         // all other entries are assumed initialized to 0

  for l = 2 to L
    for u = 1 to S
      for e = 0 to E
      begin
        N[l][u][e] = 0;
        for v = 1 to S
          if (e - w[u][v] >= 0)
          then N[l][u][e] += N[l - 1][v][e - w[u][v]];
      end
end
```

The running time of the above algorithm is $O(L^2S^2M)$ where M is the maximum entry in the w-array (in this case $M = 36$).

Once the N-array is initialized by this procedure we can find the number of strands whose free energies lie within the range $[P, Q]$ in $O(S(Q - P))$ time. A plot showing the number of strands corresponding to a free energy value/range can also be produced in $O(LSM)$ time.

5.3 Random Generation Algorithm

This section shows how the data obtained from the dynamic programming algorithm above can be used to randomly select a strand from all strands of a given length and free energy. Let $S = \{w_1, \ldots, w_N\}$ be the set of all strands of length L and free energy E. To randomly select a strands from S we generate a random number r in the range $[1, N]$. By the dynamic programming algorithm,

$$N = N_A + N_C + N_G + N_T$$

where N_j is the size of S_j, the set of strands of S which begin with nucleotide j. Since N_A, N_C, N_G, N_T are known (they are entries in the N-array) we can fix the first nucleotide in the random strand to be generated, depending on whether r is in the range $[1, N_A], [N_A + 1, N_A + N_C], [N_A + N_C + 1, N_A + N_C + N_G]$ or $[N_A + N_C + N_G + 1, N]$.

Applying this process iteratively we can generate the entire strand. i.e. if r is in the range $[N_A + 1, N_A + N_C]$ (say), we fix the first nucleotide to be C. We then consider the set S_C and then choose a strand at random from this set by using the random number $r - N_A$, which will be uniformly distributed over the range $[1, N_C]$.

This algorithm basically orders the strands in S using the N-array and then uniformly selects one by generating a random number between 1 and N.

References

[1] L. M. Adleman, *Molecular Computation of Solutions to Combinatorial Problems*, Science, **266**, 11 November, 1994, pp. 1021–1024.

[2] E. B. Baum, "DNA Sequences Useful for Computation," Proc. 2nd DIMACS Workshop on DNA Based Computers, June 1996.

[3] S. Brenner, "Methods for Sorting Polynucleotides using Oligonucleotide Tags," U.S. Patent Number 5,604,097, Feb 18, 1997.

[4] S. Brenner and R. A. Lerner, "Encoded combinatorial chemistry," Proc. Natl. Acad. Sci. USA, Vol 89, pp 5381-5383, June 1992.

[5] K. Breslauer,R. Frank, H. Blocker, L. Marky, "Predicting DNA duplex stability from the base sequence", Proc. Natl. Acad. Sci. USA 83 (1986), pages 3746-3750.

[6] A.E.Brouwer, J.B. Shearer, N.J.A. Sloane, W.D. Smith, "A New Table of Constant Weight Codes", IEEE Transactions on Information Theory, Vol. 36, No. 6, November 1990, pages 1334-1380.

[7] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr., "Good Encodings for DNA-Based Solutions to Combinatorial Problems," Proc. of DNA Based Computers II, DIMACS Workshop June 10-12, 1996, L. F. Landweber and E. B. Baum, Editors, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999, pages 247-258.

[8] R. Deaton, M. Garzon, R. C. Murphy, J. A. Rose, D. R. Franceschetti, and S. E. Stevens, Jr., "Genetic Search of Reliable Encodings for DNA-Based Computation," Koza, John R., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors), Proceedings of the First Annual Conference on Genetic Programming 1996.

[9] A. A. El Gamal, L. A. Hemachandra, I. Shperling, and V. K. Wei, "Using Simulated Annealing to Design Good Codes," IEEE Transactions on Information Theory, Vol. IT-33, No. 1, January 1987.

[10] T. Ericson, "Bounds on the Size of a Code", Topics in Coding Theory, Springer-Verlag, 1989.

[11] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn, "Demonstration of a Word Design Strategy for DNA Computing on Surfaces," Nucleic Acids Research, Vol. 25, No. 23, December 1997, pages 4748-4757.

[12] M. Garzon, R. Deaton, P. Neathery, D.R. Franceschetti, and R.C. Murphy, "A New Metric for DNA Computing," Proc. 2nd Genetic Programming Conference, Morgan Kaufman, 1997, pages 472-478.

[13] M. Garzon, R. Deaton, L.F. Nino, S.E. Stevens Jr., and M. Wittner, "Encoding Genomes for DNA Computing," Proc. 3rd Genetic Programming Conference, Madison, WI, 1998.

[14] S. Litsyn, A. Vardy, "The Uniqueness of the Best Code", IEEE Transactions on Information Theory, Vol. 40, No. 5, November 1994, pages 1693-1698.

[15] Y. Klein, S. Litsyn, A. Vardy, "Two New Bounds on the Size of Binary Codes with a Minimum Distance of Three", Designs, Codes and Cryptography, 6, 1995, pages 219-227.

[16] Klaus-Uwe Koschnick, "Some New Constant Weight Codes", IEEE Transactions on Information Theory, Vol. 37, No. 2, November 1991, pages 370-371.

[17] F.R. Kschischang, S. Pasupathy, "Some ternary and Quaternary Codes and Associated Sphere Packings", IEEE Transactions on Information Theory, Vol. 38, No. 2, March 1992, pages 227-246.

[18] F. J. MacWilliams and N. J. A. Sloane, "The Theory of Error-Correcting Codes," North-Holland, 1977.

[19] K. U. Mir, "A Restricted Genetic Alphabet for DNA Computing," Proc. of DNA Based Computers II, DIMACS Workshop June 10-12, 1996, L. F. Landweber and E. B. Baum, Editors, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 44, 1999, pages 243-246.

[20] D. D. Shoemaker, D. A. Lashkari, D. Morris, M. Mittman, and R. W. Davis, "Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-coring strategy," Nature Genetics, Vol 16, December 1996, pages 450–456.

[21] J. G. Wetmur. "DNA Probes: Applications of the Principles of Nucleic Acid Hybridization," Critical Reviews in Biochemistry and Molecular Biology 26(3/4):227-259, 1991, pages 227–259.

Tables

This section contains tables of the functions $A_4(n,d), A_4^R(n,d)$, and $A_2^R(n,d)$ for various values of the parameters. Tables 1 and 2 provide both a lower bound and an upper bound on the true value; whenever the two bounds are equal the entry is a single integer. The superscripts on entries indicate the method by which the bound was obtained. The following chart gives an overview of the superscripts used in the tables.

Superscript	Name of Bound	Relevant Theorem or Reference
s	Sphere-Packing	3.1
pl	Plotkin	3.4
h	Halving	4.4
g	Gilbert-Varshamov	3.2
p	Product	4.6
d	Doubling	4.7
b	Basic	3.5, 4.8
x	$d = 2$ construction	4.5
k	Kschischang and Pasupathy	[17]

Table 1: Bounds on $A_4(n,d)$

n,d	3	4	5	6	7	8
4	16^k-19^s	4^{pl}	1	1	1	1
5	64^k-64^s	16^k-16^{pl}	4^{pl}	1	1	1
6	64^p-215^s	16^p-$(2^6)^b$	4-10^{pl}	4^{pl}	1	1
7	256^p-744^s	64^p-$(2^8)^b$	16^k-$(5\times 2^3)^b$	4-8^{pl}	4^{pl}	1
8	$(2^{10})^k$-2621^s	256^p-$(2^{10})^b$	64^k-$(5\times 2^5)^b$	16^k-$(2^5)^b$	4-7^{pl}	4^{pl}
9	$(2^{12})^k$-9362^s	$(2^{10})^k$-$(2^{12})^b$	$(2^8)^k$-$(5\times 2^7)^b$	$(2^6)^k$-$(2^7)^b$	16^k-28^{pl}	4-6^{pl}
10	$(2^{14})^k$-33825^s	$(2^{12})^k$-$(2^{14})^b$	$(2^{10})^k$-$(5\times 2^9)^b$	$(2^8)^k$-$(2^9)^b$	16^k-$(7\times 2^4)^b$	16^k-16^{pl}
11	$(2^{16})^k$-123361^s	$(2^{14})^k$-$(2^{16})^b$	$(2^{12})^k$-$(5\times 2^{11})^b$	$(2^{10})^k$-$(2^{11})^b$	64^k-$(7\times 2^6)^b$	16^k-$(2^6)^b$
12	$(2^{18})^k$-453438^s	$(2^{16})^k$-$(2^{18})^b$	$(2^{12})^k$-$(5\times 2^{13})^b$	$(2^{10})^k$-$(2^{13})^b$	$(2^8)^k$-$(7\times 2^8)^b$	64^k-$(2^8)^b$
13	$(2^{20})^k$-1677721^s	$(2^{18})^k$-$(2^{20})^b$	$(2^{14})^k$-$(5\times 2^{15})^b$	$(2^{12})^k$-$(2^{15})^b$	$(2^{10})^k$-$(7\times 2^{10})^b$	$(2^8)^k$-$(2^{10})^b$
14	$(2^{22})^k$-6242685^s	$(2^{20})^k$-$(2^{22})^b$	$(2^{16})^k$-$(5\times 2^{17})^b$	$(2^{14})^k$-$(2^{17})^b$	$(2^{12})^k$-25110^s	$(2^{10})^k$-$(2^{12})^b$
15	$(2^{24})^k$-23342213^s	$(2^{22})^k$-$(2^{24})^b$	$(2^{18})^k$-$(5\times 2^{19})^b$	$(2^{16})^k$-$(2^{19})^b$	$(2^{14})^k$-80878^s	$(2^{12})^k$-$(2^{14})^b$
16	$(2^{26})^k$-87652393^s	$(2^{22})^k$-$(2^{26})^b$	$(2^{18})^k$-$(5\times 2^{21})^b$	$(2^{18})^k$-$(2^{21})^b$	$(2^{16})^k$-264321^s	$(2^{12})^k$-$(2^{16})^b$

Table 2: Bounds on $A_4^R(n,d)$

n,d	2	3	4	5	6	7	8
2	2	0	0	0	0	0	0
3	6^x-$(2^3)^h$	0	0	0	0	0	0
4	$(2^5)^{x,h}$	2^p-8^s	2	0	0	0	0
5	120^x-$(2^7)^h$	4^p-26^s	2^p-8^h	0	0	0	0
6	$(2^9)^{x,h}$	12^g-107^h	4^p-107^h	2^p-5^h	2	0	0
7	2016^x-$(2^{11})^h$	33^g-372^h	8^p-372^h	2^p-34^s	2^p-4^h	0	0
8	$(2^{13})^{x,h}$	160^p-1310^h	128^p-1310^h	4^p-118^h	2^p-118^h	2^p-3^h	2
9	32640^x-$(2^{15})^h$	354^g-4681^h	160^p-4681^h	8^g-372^h	4^p-372^h	2^p-14^h	2^p-3^h
10	$(2^{17})^{x,h}$	1184^g-16912^h	320^p-16912^h	23^g-1202^h	6^p-1202^h	2^p-142^h	2^p-8^h
11	523776^x-$(2^{19})^h$	3903^g-61680^h	576^p-61680^h	60^g-3964^h	14^g-3964^h	4^p-420^h	2^p-420^h
12	$(2^{21})^{x,h}$	13233^g-226719^h	1271^g-226719^h	173^g-13294^h	34^g-13294^h	8^g-1276^h	3^g-1276^h
13	2095104^x-$(2^{23})^h$	45012^g-838860^h	3946^g-838860^h	487^g-45221^h	86^g-45221^h	18^g-3964^h	6^g-3964^h
14	$(2^{25})^{x,h}$	155496^g-3121342^h	12539^g-3121342^h	1444^g-155705^h	230^g-155705^h	46^g-12555^h	13^g-12555^h
15	134209536^x-$(2^{27})^h$	541020^g-11671106^h	40385^g-11671106^h	4280^g-541746^h	621^g-541746^h	111^g-40439^h	27^g-40439^h
16	$(2^{29})^{x,h}$	1901386^g-43826196^h	132111^g-43826196^h	13066^g-1902111^h	4096^p-1902111^h	576^p-132160^h	512^p-132160^h

Table 3: Lower bounds on $A_2^R(n,d)$

n,d	3	4	5	6	7	8
4	1^g	1^g	0^g	0^g	0^g	0^g
5	1^g	1^g	0^g	0^g	0^g	0^g
6	1^g	1^g	1^g	1^g	0^g	0^g
7	2^g	1^g	1^g	1^g	0^g	0^g
8	8^b	8^d	1^g	1^g	1^g	1^g
9	8^b	8^b	1^g	1^g	1^g	1^g
10	8^b	8^b	1^g	1^g	1^g	1^g
11	13^g	8^b	1^g	1^g	1^g	1^g
12	24^g	8^b	2^g	1^g	1^g	1^g
13	40^g	10^g	3^g	2^g	1^g	1^g
14	73^g	17^g	5^g	2^g	1^g	1^g
15	127^g	27^g	7^g	3^g	1^g	1^g
16	231^g	46^g	16^b	16^b	16^b	16^d