JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 8, Number 3, 2001
Mary Ann Liebert, Inc.
Pp. 201–219

On Combinatorial DNA Word Design

AMIT MARATHE,[1] ANNE E. CONDON,[3] and ROBERT M. CORN[2]

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 (a) calculate the total number of words of length n whose free energy value, as approximated by a formula of Breslauer *et al.* (1986) falls in a given range, and (b) output a random such word. These algorithms are intended for use in heuristic algorithms for constructing DNA codes.

Key words: DNA computation, word design, molecular bar-codes, coding theory.

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 (MacWilliams and Sloane, 1997). 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 $5'$ and $3'$ 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 (Adleman, 1994). DNA code words are also used as molecular bar codes, or tags, for the purpose of manipulating and identifying individual molecules

[1]Computer Sciences Department, University of Wisconsin, Madison, WI 53706.
[2]Chemistry Department, University of Wisconsin, Madison, WI 53706.
[3]The Department of Computer Science, University of British Columbia, Vancouver, B.C. V6T 174.

in complex chemical libraries (Brenner, 1997; Brenner and Lerner, 1992; Shoemaker *et al.*, 1996; Smith and Schweitzer, 1995).

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 $5'$ and $3'$ ends. For example, the Watson–Crick complement of $5' - AACATG - 3'$ is $3' - TTGTAC - 5'$. 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 nonspecific 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 $5' - AACATG - 3'$ and $3' - TAATAC - 5'$, in the second and third positions from the $5'$ end of the first strand. Nonspecific 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 (Adleman, 1994; Baum, 1999; Brenner, 1997; Brenner and Lerner, 1992; Cuckrus *et al.*, 1998; Deaton *et al.*, 1999; Garzon *et al.*, 1997a; Deaton *et al.*, 1996; Frutos *et al.*, 1997; Garzon *et al.*, 1997b; Garzon *et al.*, 1998; Mir, 1999; Roweis *et al.*, 1999; Shoemaker *et al.*, 1996). 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 $5'$ end of the associated DNA code word. Thus, CCGAT represents $5' - CCGAT - 3'$, 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 $x = x_1x_2 \ldots x_n$ be a word. The *reverse* of x, denoted by x^R, is the word $x_nx_{n-1} \ldots x_1$. 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 $w = w_1w_2 \ldots w_n$ 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 Brenner, 1997; Cuckrus *et al.*, 1998; Deaton *et al.*, 1999; Frutos *et al.*, 1997; Roweis *et al.*, 1999; Shoemaker *et al.*, 1996; Smith and Schweitzer, 1995; Zhang and Shin, 1998), in order to limit nonspecific 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 (Cuckrus *et al.*, 1998; Deaton *et al.*, 1999; Frutos *et al.*, 1997; Roweis *et al.*, 1999; Smith and Schweitzer, 1995; Zhang and Shin, 1998). 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 over alphabet size q, with parameters n, d defined as for codes. For example, the code words ACG and CGG (representing strands $5' - ACG - 3'$ and $5' - CGG - 3'$, respectively) can be part of an $A_4^{RC}(3, 1)$ code but not an $A_4^{RC}(3, 2)$ code.
- The **reverse constraint** with parameter d is that, for all pairs of words w, x in the code (including the case when $w = x$), $H(W, x^R) \geq d$. 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 **free energy constraint,** which is the final constraint that we consider, has a somewhat different flavor. It is motivated by the goal that all code words in the set have similar melting temperatures, allowing hybridization of multiple words to proceed simultaneously (Shoemaker *et al.*, 1996). The melting temperature of a short DNA strand can be accurately estimated using a formula of Wetmur (Wetmur, 1991), which in turn uses estimates of two further parameters of a DNA strand, namely its free energy and enthalpy. We use ΔG to denote the Breslauer estimate of the free energy of a DNA word (Breslauer *et al.*, 1986). As explained in Section 5, the free energy estimate (measured in kcal/mol) is essentially the sum of "nearest neighbor" 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.

We are interested in constructing DNA codes that satisfy one or more of these constraints. It is desirable that the codes are as large as possible, for the following reasons. A code designer typically may choose a fixed n and d, based on chemical considerations. The designer would like as many words as possible (satisfying the Hamming, reverse complement, and possibly other constraints) because the number of bits that can be stored grows with the size of the code. For example, Frutos *et al.* (1997) designed a code of size 108 that satisfies the Hamming and reverse complement constraints with $n = 8$ and $d = 4$, plus one additional constraint (see Section 2). In their computations, information is organized into words of information, just as in a computer memory. Roughly, the code words are used to represent the bits stored in a memory word, and additional word labels represent the word address. With 108 words, less than 7 bits of information can be stored per word. If a code of size 256 could be designed that satisfies the same constraints, 8 bits could be stored in each word, increasing the density of information stored in a strand. Alternatively, a code designer may choose a fixed n and need a fixed number of words, but would like the word set to satisfy the Hamming and reverse complement constraints for as large a parameter d as possible. Intuitively, the larger d is, the greater the chance of avoiding errors.

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.
- **Cai's lower bound:** $A_q^R(2n, 2d) \geq \lfloor A_q(n,d)/2 \rfloor$. This result, which constructs a $(2n, 2d)$ reverse code from an optimal (n,d) code, is due to Jin-Yi Cai.
- **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,

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

- **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 $(2^n, 2^n, 2^{n-1})$ binary reverse codes, which are optimal, i.e., $A_2^R(2^n, 2^{n-1}) = 2^n$.

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.

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 estimated 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 can 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 (Gamal *et al.*, 1987).

2. RELATED WORK

Deaton *et al.* (1996, 1999) observe that the well-known sphere-packing bound (see Section 3) can be used to set an upper bound on the size of DNA codes (i.e., sets of words that satisfy the Hamming constraint). They describe genetic algorithms for finding DNA codes that satisfy several constraints, including the Hamming and reverse complement constraints. Heuristic methods for finding good DNA encodings have also been described by Garzon *et al.* (1999) and by Zhang and Shin (1998).

In his patent on methods for sorting polynucleotides using DNA tags, Brenner (1997) gives a greedy algorithm for generation of DNA codes. 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 (1999) proposed a word design for use in DNA computing over an alphabet representing just 3 of the 4 possible nucleotides; similarly Cukras *et al.* (1998) use a three-letter alphabet in designing a library of RNA strands for encoding bits.

For prototype experiments of the Wisconsin DNA computing project, a DNA word design with words of length 16 was developed (Frutos *et al.*, 1997). 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. Cukras *et al.* (1998) design a library of RNA strands which incorporate equi-length "bit encodings" that have pairwise high Hamming distance and have similar melting temperatures, along with other constraints. They use a computer program to find a library with the desired combinatorial properties.

Shoemaker *et al.* (1996) 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 ($61 \pm 5°C$). 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. a) The first arises, for example, in Brenner and Lerner's work (1992), 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. Roweis et al. (1999) propose such a frame shift constraint in their proposal for a sticker-based model for DNA computation and point to useful references in the applied mathematics literature on comma free codes and DeBruijn sequences. Smith and Schweitzer (1995) consider a "modified DeBruijn problem," namely, given k, design a DNA strand as long as possible, so that if S is any k-letter substrand of this DNA strand, then S occurs exactly once in the DNA strand and also the reverse of S and its complement do not occur in the strand. A greedy

algorithm for generating words satisfying a similar frame shift constraint is given by Garzon *et al.* (1997, 1998). b) Baum (1999) 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. c) In designing words for surface-based DNA computing, Frutos *et al.* (1997) 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. d) Seeman (1990) developed algorithms to design DNA sequences for use in construction of stable DNA structures with branched helix axes, "with the goal of minimizing sequence similarities between segments of molecules." The end product can be thought of as a set of words with similar GC content, such that if s is any string of some given length d (such strings are referred to as "critons") over $\{A, C, G, T\}$, then s and its reverse complement s^{RC} occur at most once in any word in the set. (Additional constraints are imposed on the sequences, for example at junction points in the branched structures.) e) The important *forbidden subwords constraint*, already mentioned in relation to the work of Shoemaker *et al.* (1996) 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 to be used as PCR primers, or strands that are recognized by restriction enzymes.

Finally, Smith and Schweitzer (1995) propose encoding methods that allow for error correction even when mismatches are assigned weights. A different measure of error potential in DNA code words has been proposed by Rose *et al.* (1999).

3. BOUNDS ON THE SIZE OF A CODE

In this section, we briefly review previous results on codes that will be extended or applied in Section 4 to obtain bounds on reverse codes. The text by MacWilliams and Sloane (1977) provides a good introduction to the subject.

Theorem 3.1 (sphere-packing upper bound).

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

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 $V(\lfloor (d-1)/2 \rfloor)$ for all words s in the code, cannot overlap.

Theorem 3.2 (Gilbert–Varshamov lower bound).

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

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).

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

To see why the Singleton bound holds, let k be the largest number such that $q^{k-1} < A_q(n,d)$. We will show that $k \le n-d+1$, which implies the bound. Let C be a code of size $A_q(n,d)$ over an alphabet of size q. Choose any $k-1$ coordinates of the code words in C. Since $q^{k-1} < A_q(n,d)$, there must be two code words in C, say x and y, which agree at the $k-1$ chosen coordinates. Hence, it must be that $d \le n-k+1$, and so $k \le n-d+1$.

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

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

We describe briefly why the Plotkin bound holds when $q = 2$ and $A_2(n,d)$ is even; the more general case is a straightforward extension of this special case. Let C be a code of (maximum) size $A_2(n,d)$. The proof proceeds by calculating $\sum_{w,x\in C} H(w,x)$ in two ways. First, since $H(w,x) \geq d$ if $w \neq x$, the sum is at least $A_2(n,d)(A_2(n,d)-1)d$. Next, let A be the $A_2(n,d) \times n$ matrix whose rows are the words of C in bits. Suppose that the ith column of A contains u_i 0's and $A_2(n,d) - u_i$ 1's. Then this column contributes $2u_i(A_2(n,d)-u_i)$ to the sum, so that the sum is equal to

$$\sum_{i=1}^{n} 2u_i(A_2(n,d)-u_i) \leq nA_2(n,d)^2/2.$$

The last inequality follows from the fact that $A_2(n,d)$ is even, and so the sum is maximized when $u_i = A_2(n,d)/2$ for all i. Comparing both sums, we have that

$$A_2(n,d)(A_2(n,d)-1)d \leq nA_2(n,d)^2/2.$$

Thus, $A_2(n,d) \leq 2d/(2d-n)$, as claimed.

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

Proof.

1. The code consisting of q words of length n, 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.
2. 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.
3. If we partition all of 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 (1992). 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.* (1999) 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 Hamming and reverse-complement constraints. Similarly, $A_q^R(n,d)$ denotes the maximum size of a code satisfying the reverse (and Hamming) constraint.

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

Theorem 4.1.

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

Proof. First, suppose that n is even. Let $\{x_i\}$ be a $(n, A^R(n,d), d)$ reverse code. Write each $x_i = a_i b_i$, where the lengths of a_i and b_i are equal. Then $\{y_i = a_i b_i^C\}$ is a $(n, A^R(n,d), d)$ reverse complement code. To see this, note that for any pair of strings r, s, $H(r,s) = H(r^C, s^C)$. Hence for all $1 \le i, j \le n$,

$$H(y_i^C, y_j^R) = H(a_i^C b_i, b_j^{RC} a_j^R) = h(a_i b_i, b_j^R a_j^R) = H(x_i, x_j^R) \ge d.$$

A similar argument shows that for $i \ne j$, $H(y_i, y_j) = H(x_i, x_j) \ge d$. Therefore, $A_4^R(n,d) \le A_4^{RC}(n,d)$. The proof that $A_4^R(n,d) \ge A_4^{RC}(n,d)$ is symmetric.

We next consider the case when n is odd. Note that if we truncate a $(n, A_4^R(n,d+1), d+1)$ reverse code by removing the middle letter of each code word, we obtain a $(n-1, A_4^R(n,d+1), d)$ reverse code. Therefore, $A_4^R(n,d+1) \le A_4^R(n-1,d)$. Since $n-1$ is even, we know from the theorem statement for even n that $A^R(n-1,d) = A_4^{RC}(n-1,d)$. Combining these two inequalities, we see that $A_4^R(n,d+1) \le A_4^{RC}(n-1,d)$. In addition, since $A_4^{RC}(n-1,d) \le A_4^{RC}(n,d)$ for odd n, we have that $A_4^R(n,d+1) \le A_4^{RC}(n,d)$, as stated.

Similarly, if we truncate a $(n, A_4^{RC}(n,d), d)$ reverse-complement code by removing the middle letter of each code word, we obtain a $(n-1, A_4^{RC}(n,d), d-1)$ reverse-complement code. Therefore, $A_4^{RC}(n,d) \le A_4^{RC}(n-1,d-1)$. Since $n-1$ is even, we know from the theorem statement for even n that $A^{RC}(n-1,d-1) = A_4^R(n-1,d-1)$. Combining these two inequalities, we see that $A_4^{RC}(n,d) \le A_4^R(n-1,d-1)$. In addition, since $A_4^R(n-1,d-1) \le A_4^R(n-1,d)$ for odd n, we have that $A_4^{RC}(n,d) \le A_4^R(n,d-1)$, completing the proof. ∎

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 $d = 3$. For the next theorem, we need one additional definition. Let S be the set of all words x (length n, alphabet size q) such that $H(x, x^R) \ge d$. Also, let $V(s,d')$ be the number of words of S that have distance at most d' from word $s \in S$. Then, we define $V^-(d')$ to be $\min\{V(s,d')\}$ where the min is taken over all s in S.

Theorem 4.2. *For $n \ge 4$,*

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

Proof. 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, $H(s, s^R) \ge d$ and hence $V(s, \lfloor (d-1)/2 \rfloor)$ is disjoint from $V(s^R, \lfloor (d-1)/2 \rfloor)$. Therefore,

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

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 $x_j \ne x_j^R$, i.e., if $x_j \ne x_{n-j+1}$. 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 in S of distance exactly 1 from x, and therefore $V^-(1) \geq 1 + 4(q-2) + (n-4)(q-1)$. 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. Let S be a set that satisfies the Hamming and reverse constraints. The reverse constraint implies that $H(x, x^R) \geq d$ for every word x in S. Also $x \in S \Rightarrow x^R \notin S$. Let S' be the set obtained by adding to S the reversals of all words in it. The set S' satisfies the Hamming constraint because the new words added are at least distance d apart from each other and from the words of in S (this follows from the fact that S satisfies the Hamming and reverse constraints). Also, the size of S' is twice that of S. ■

Theorem 4.5 (Cai's lower bound).

$$A_q^R(2n, 2d) \geq \left\lfloor \frac{A_q(n, d)}{2} \right\rfloor$$

Proof. Divide an optimal (n, d) code into two equal parts (after dropping one word if $A_q(n, d)$ is odd). Let these parts be $X = \{x_1, \ldots, x_t\}$ and $Y = \{y_1, \ldots, y_t\}$, where $t = \lfloor A_q(n, d)/2 \rfloor$. Then $\{x_i y_i^R | i = 1, \ldots, t\}$ is a $(2n, 2d)$ reverse code. ■

Theorem 4.6 ($d = 2$ construction).

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

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

Proof. The proof builds on the following claim: ■

Claim 4.1. *For even n and $q \in \{2, 4\}$, $\sum^n$ 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 4.1. 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 $\sum^n$, $i \in \{1, 2, 3, 4\}$ with the above properties and 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:

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

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 $\sum^{n+2}$ 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 4.6 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 $n > 1$ and $q \in \{2, 4\}$ 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 $(q^{n-1} - q^{\lfloor n/2 \rfloor})/2$. ■

Theorem 4.7 (product bound).

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

Proof. Let A be a reverse code and let B be a 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 appears in that position, while the corresponding bit in B makes a choice from the two remaining possibilities. Moreover, this map is one-to-one and so the set C of words obtained has size $|A||B|$.

We next show that each pair (w, x) of words in C satisfies the Hamming and reverse constraints. Let w be obtained from the pair $(a_w, b_w) \in A \times B$ and x from (a_x, b_x). Since A is a reverse code, $H(a_w, a_x^R) \geq d$ and so it must be the case that at d positions w has an A or T and x^R has C or G or vice versa. Hence, $H(w, x^R) \geq d$.

To show that $w \neq x \Rightarrow H(w, x) \geq d$ we consider two cases. First, suppose that $a_w \neq a_x$. Since A is a code, $H(a_w, a_x) \geq d$ and so it must be the case that at d positions w has A or T and x has C or G or vice versa. Second, suppose that $a_w = a_x$ but $b_w \neq b_x$. Then $H(b_w, b_x) \geq d$. At each position k where b_w and b_x differ, if the kth position of a_w is 0, then either w has A and x has T or vice versa. Also, if the kth position of a_w is 1, then either w has C and x has G or vice versa. Therefore $H(w, x) \geq d$.

Figure 1 contains an example of the product construction. ■

Theorem 4.8 (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,

$$H(x, y) \geq d, \qquad \text{if } x \neq y$$
$$H(x, y^R) \geq d,$$
$$H(x, y^C) \geq d, \text{ and}$$
$$H(x, y^{RC}) \geq d.$$

■

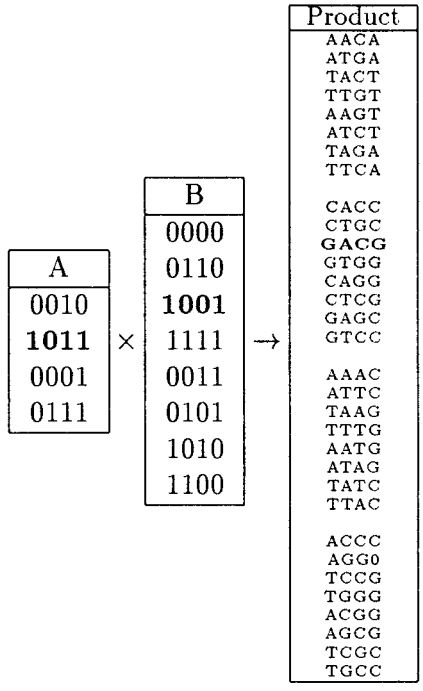

FIG. 1. Product set construction with $n = 4$ and $d = 2$. As indicated in bold, the element 1011×1001 in the Cartesian product $A \times B$ corresponds to the word GACG in the product set.

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 4.2. 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. ■

Getting back to the proof of Theorem 4.8, 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 MacWilliams and Sloane (1997), which by the halving bound (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 4.9.

1. $A_4^R(n,n) = \begin{cases} 2 & \text{if } n \geq 2 \text{ is even, and} \\ 0 & \text{otherwise,} \end{cases}$
2. $A_q^R(n,d) \leq A_q^R(n,d-1)$, *and*
3. $A_q^R(n,d)/q \leq A_q^R(n-1,d) \leq A_q^R(n,d)$, *for odd* n.

Proof.

1. 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.
2. This is trivially true because a (n, M, d) reverse code is also a $(n, M, d-1)$ reverse code.
3. The proof of the first inequality 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. To show the second inequality, let C be a $(n-1, A_q^R(n-1,d), d)$ code. Let C' be obtained from C by inserting an arbitrary symbol in the middle of each word in C. Then C' is a $(n, A_q^R(n-1,d), d)$ reverse code and so $A_q^R(n-1,d) \leq A_q^R(n,d)$. ■

TABLES OF CODES

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 an alphabet of size 4, via application of our halving bound (Theorem 4.4). Table 3 only contains lower bounds, which are needed to construct Table 2. 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 4.7). By extending known techniques for construction of codes to handle reversals, we obtain further bounds on the size of reverse codes.

In the tables, 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	4.7
d	Doubling	4.8
b	Basic	3.5, 4.9
x	$d = 2$ construction	4.6
k	Kschischang and Pasupathy	[22]
c	Cai	4.5

In Table 1 for $A_4(n,d)$, very few entries have matching upper and lower bounds. In fact, none of the entries for $d = 3$ match, and many of the upper and lower bounds differ by at least one order of magnitude. In Table 2, for $A_4^R(n,d)$, for odd n, and $d = 2$, the upper and lower bounds are not matching, and overall, upper and lower bounds can differ by 2–3 orders of magnitude. The tables show that there is much room for improvement of the methods in this paper.

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

TABLE 1. LOWER AND UPPER 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. LOWER AND UPPER 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	$(2^9)^{x,h}$	$16^p - 107^h$	$8^p - 107^h$	$2^p - 5^h$	2	0	0
7	$2016^x - (2^{11})^h$	$33^g - 372^h$	$16^p - 372^h$	$2^p - 34^s$	$2^p - 4^h$	0	0
8	$2(^{13})^{x,h}$	$160^p - 1310^h$	$128^p - 1310^h$	$8^b - 118^h$	$8^c - 118^h$	$2^p - 3^h$	2
9	$32640^x - (2^{15})^h$	$354^g - 4681^h$	$160^p - 4681^h$	$8^g - 372^h$	$8^b - 372^h$	$2^p - 14^h$	$2^p - 3^h$
10	$(2^{17})^{x,h}$	$1184^g - 16912^h$	$320^p - 16912^h$	$32^b - 1202^h$	$32^c - 1202^h$	$8^b - 142^h$	$8^c - 8^h$
11	$523776^x - (2^{19})^h$	$3903^g - 61680^h$	$576^p - 61680^h$	$60^g - 3964^h$	$32^b - 3964^h$	$8^b - 420^h$	$8^b - 420^h$
12	$(2^{21})^{x,h}$	$13233^g - 226719^h$	$2304^p - 226719^h$	$173^g - 13294^h$	$96^p - 13294^h$	$8^p - 1276^h$	$8^p - 1276^h$
13	$2095104^x - (2^{23})^h$	$45012^g - 838860^h$	$4096^p - 838860^h$	$487^g - 45221^h$	$128^p - 45221^h$	$18^g - 3964^h$	$8^p - 3964^h$
14	$(2^{25})^{x,h}$	$155496^g - 3121342^h$	$12539^g - 3121342^h$	$1444^g - 155705^h$	$512^p - 155705^h$	$64^p - 12555^h$	$32^p - 12555^h$
15	$134209536^x - (2^{27})^h$	$541020^g - 11671106^h$	$40385^g - 11671106^h$	$4280^g - 541746^h$	$1024^p - 541746^h$	$128^p - 40439^h$	$64^p - 40439^h$
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	2^b	2^c	1^g	1^g	0^g	0^g
7	2^b	2^b	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	2^b	2^c	1^g	1^g
11	13^g	8^b	2^b	2^b	1^g	1^g
12	24^g	16^c	4^b	4^c	2^b	2^c
13	40^g	16^b	4^b	4^b	2^b	2^b
14	73^g	32^c	8^b	8^c	4^b	4^c
15	127^g	32^b	8^b	8^b	4^b	4^b
16	231^g	64^c	16^b	16^b	16^b	16^d

$V(s, d')$ 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 $\binom{n}{i} = \frac{n(n-1)\ldots(n-i+1)}{i(i-1)\ldots 21}$ denotes the number of ways to choose i distinct items from a set of size n. Proofs of the following four bounds can be found in a survey article by Ericson (1989) or the text by MacWilliams and Sloane (1977).

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.* (1986) 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.* (1986):

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

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-1) * 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 1 = 2 to L
    for u = 1 to S
      for e = 0 to E
        begin
          N[1][u][e] = 0;
          for v = 1 to S
          if (e - w[u][v] >= 0) then
            N[1][u][e] += N[1 - 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 $M = 36$ because the free energies in the Breslauer table are all expressed as rational numbers of the form $a.b$ with the maximum being 3.6; multiplying these free energies by 10 produces integers with the maximum being 36. More generally, if the free energies are expressed using k degrees of accuracy, that is, as rational numbers of the form $a.b_1b_2 \ldots b_k$ with the maximum being $m_0.m_1m_2 \ldots m_k$, then M would be $m_0m_1 \ldots m_k$.

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 strand 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 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.

6. CONCLUSIONS

Design of good DNA codes is a hard and important problem for DNA computing and other biotechnology applications. Current approches focus on DNA codes that satisfy natural combinatorial constraints. Theoretical work on this problem can build on classical results from coding theory, thereby providing useful design tools, and also can provide limits on the attainable size of certain codes. The theoretical results of this paper are a first step in this direction, focusing on two widely used constraints: the Hamming and reverse complement constraints.

From a practical point of view, improvement in our constructions would be valuable in guiding code designs. As a case in point, we note that Frutos *et al.* (1997) designed a word set of size 108 that satisfies the Hamming and reverse complement constraints with $n = 8$ and $d = 4$. In addition, their word set satisfies the GC-content constraint with each word having four G/C positions and four A/T positions. It would be interesting to understand the effect of the GC-content constraint on word set size, and this requires tight bounds on $A^R(8, 4)$. The best construction we provide in this paper for a word set with $n = 8$ and $d = 4$ has size 128; if this is close to optimal, it would imply that adding the GC-content constraint does not significantly reduce the size of the word set. However, our upper bound on $A^R(n, d)$ is 1,310, higher by more than a factor of 10 than the lower bound.

More generally, there is a wide gap between our upper and lower bounds even for very small values of n. The halving bound appears to be a weak upper bound, and it would be interesting to know whether $A^R(n, d) = o(A(n, d))$.

Additional combinatorial constraints need to be considered, particularly the frame shift constraint (see Section 2). Codes that satisfy the frame shift constraint with Hamming distance equal to 1 have been extensively studied (see for example Eastman [1965], Litsyn and Vardy [1994]), but to our knowledge there is little known when the Hamming distance is greater than 1.

Perhaps more difficult theoretically, but well motivated from a practical point of view, would be to derive good methods for generating codes that satisfy the free energy constraint *in addition to* one ore more other constraints such as the Hamming, reverse complement, or frame shift constraints.

Finally, more experimental work is badly needed to validate the use of combinatorial constraints in design of DNA codes in the first place. Related to this, it is desirable to have a method for predicting the free energy of arbitrary duplexes—that is, a method that generalizes formulas based on "nearest neighbor" interactions in the special case of a duplex formed from a strand and its Watson–Crick complement (Breslauer *et al.*, 1986; Wetmus, 1991).

ACKNOWLEDGMENTS

We thank Jin-Yi Cai for allowing us to include his result, Theorem 4.5, in this paper. We also thank Lloyd Smith and the anonymous referees for their valuable comments.

REFERENCES

Adleman, L.M. 1994. Molecular computation of solutions to combinatorial problems. *Science* 266, 1021–1024.

Baum, E.B. 1999. DNA Sequences Useful for Computation. *Proc. DNA Based Computers II, DIMACS Workshop, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 44, 235–242.

Brenner, S. 1997. Methods for sorting polynucleotides using oligonucleotide tags. U.S. Patent Number 5,604,097.

Brenner, S., and Lerner, R.A. 1992. Encoded combinatorial chemistry. *Proc. Natl. Acad. Sci. USA* 89, 5381–5383.

Breslauer, K., Frank, R., Blocker, H., and Marky, L. 1986. Predicting DNA duplex stability from the base sequence. *Proc. Natl. Acad. Sci. USA* 83, 3746–3750.

Brouwer, A.E., Shearer, J.B., Sloane, N.J.A., and Smith, W.D. 1990. A new table of constant weight codes. *IEEE Trans. Inform. Theory* 36, 6, 1334–1380.

Cukras, A.R., Faulhammer, D., Lipton, R.J., and Landweber, L.F. 1998. Chest games: A model for RNA based computation. *Preliminary Proc. 4th Int. DIMACS Meeting on DNA Based Computers*, 27–37.

Deaton, R., Garzon, M., Murphy, R.C., Rose, J.A., Franceschetti, D.R., and Stevens Jr., S.E. 1996. Genetic search of reliable encodings for DNA-based computation. *Proc. 1st Ann. Conf. Genetic Programming 1996.*

Deaton, R., Murphy, R.C., Garzon, M., Franceschetti, D.R., and Stevens Jr., S.E. 1999. Good encodings for DNA-based solutions to combinatorial problems. *Proc. DNA Based Computers II, DIMACS Workshop, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 44, 247–258.

Garzon, M., Deaton, R., Neathery, P., Franceschetti, D.R., and Stevens Jr., S.E. 1997a. On the encoding problem for DNA computing. *Preliminary Proc. 3rd DIMACS Workshop on DNA Based Computers*, 230–237.

Eastman, W.L. 1965. On the construction of comma-free codes. *IEEE Trans. Inform. Theory*, 11, 263–267.

El Gamal, A.A., Hemachandra, L.A., Shperling, L., and Wei, V.K. 1987. Using simulated annealing to design good codes. *IEEE Trans. Inform. Theory*, IT-33, 1.

Ericson, T. 1989. Bounds on the size of a code. *Topics in Coding Theory*, Springer-Verlag.

Frutos, A.G., Liu, Q., Thiel, A.J., Sanner, A.M.W., Condon, A.E., Smith, L.M., and Corn, R.M. 1997. Demonstration of a word design strategy for DNA computing on surfaces. Nucleic Acids Res. 25, 23, 1997, 4748–4757.

Garzon, M., Deaton, R., Neathery, P., Franceschetti, D.R., and Murphy, R.C. 1997b. A new metric for DNA computing, *Proc. 2nd Genetic Programming Conf.,* 472–478, Morgan Kaufman.

Garzon, M., Deaton, R., Nino, L.F., Stevens Jr., S.E., and Wittner, M. 1998. Encoding genomes for DNA computing. *Proc. 3rd Genet. Prog. Conf.,* Madison, WI.

Garzon, M., Deaton, R.J., Rose, J.A., and Franceschetti, D.R. 1999. Soft molecular computing. *Preliminary Proc. 5th Int. Meeting on DNA Based Computers*, 89–98.

Golomb, S.W., Gordon, B., and Welch, L.R. 1958. Comma-free codes. *Can. J. Math.* 10, 202–209.

Koschnick, K.-U. 1991. Some new constant weight codes. *IEEE Trans. Inform. Theory* 37, 2, 370–371.

Klein, Y., Litsyn, S., and Vardy, A. 1995. Two new bounds on the size of binary codes with a minimum distance of three. *Designs, Codes and Cryptography* 6, 219–227.

Kschischang, F.R., and Pasupathy, S. 1992. Some ternary and quarternary codes and associated sphere packings. *IEEE Trans. Inform. Theory* 38, 2, 227–246.

Litsyn, S., and Vardy, A. 1994. The uniqueness of the best code. *IEEE Trans. Inform. Theory* 40, 5, 1693–1698.

MacWilliams, F.J., and Sloan, N.J.A. 1997. *The Theory of Error-Correcting Codes*, North-Holland.

Mir, K.U. 1999. A restricted genetic alphabet for DNA computing. *Proc. DNA Based Computers II, DIMACS Workshop, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 44, 243–246.

Rose, J.A., Deaton, R., Franceschetti, D.R., Garzon, M., and Stevens Jr., S.E. 1999. A statistical mechanical treatment of error in the annealing biostep of DNA computation. *Special Program in DNA and Molecular Computing at the Genetic and Evolutionary Computation Conference* (GECCO-99), Orlando, FL, Morgan Kaufmann.

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., and Adleman, L.M. 1999. A sticker-based model for DNA computation. *Proc. DNA Based Computers II, DIMACS Workshop, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 44, 1–29.

Seeman, N.C. 1990. *DeNovo designs of sequences for nucleic acid structural engineering.* J. Biomolecular Structure and Dynamics 8, 573–581.

Shoemaker, D.D., Lashkari, D.A., Morris, D., Mittman, M., and Davis, R.W. 1996. Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-cording strategy. *Nature Genetics* 16, 450–456.

Smith, W.D., and Schweitzer, A. 1995. DNA computers in vitro and in vivo. *NECI Technical Report*, March 20, 1995.

Wetmur, J.G. 1991. DNA probes: Applications of the principles of nucleic acid hybridization. *Critical Reviews in Biochemistry and Molecular Biology* 26(3/4), 227–259, 227–259.

Winfree, E., Liu, F., Wenzler, L.A., and Seeman, N. 1992. Design and self-assembly of two-dimensional DNA crystals. *Nature* 394, 6, 539–544.

Zhang, B.T., and Shin, S.-Y. 1998. Molecular algorithms for efficient and reliable DNA computing. *Proc. 3rd Ann. Genetic Programming Conf.,* Morgan Kaufmann, 735–742.

Address correspondence to:
Anne E. Condon
The Department of Computer Science
2366 Main Mall
University of British Columbia
Vancouver, B.C.
V6T 174, Canada

E-mail: condon@cs.ubc.ca