JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 7, Numbers 3/4, 2000
Mary Ann Liebert, Inc.
Pp. 503–519

Universal DNA Tag Systems: A Combinatorial Design Scheme

AMIR BEN-DOR,[1,2] RICHARD KARP,[3] BENNO SCHWIKOWSKI,[2,4]
and ZOHAR YAKHINI[1,5]

ABSTRACT

Custom-designed DNA arrays offer the possibility of simultaneously monitoring thousands of hybridization reactions. These arrays show great potential for many medical and scientific applications, such as polymorphism analysis and genotyping. Relatively high costs are associated with the need to specifically design and synthesize problem-specific arrays. Recently, an alternative approach was suggested that utilizes fixed, universal arrays. This approach presents an interesting design problem—the arrays should contain as many probes as possible, while minimizing experimental errors caused by cross-hybridization. We use a simple thermodynamic model to cast this design problem in a formal mathematical framework. Employing new combinatorial ideas, we derive an efficient construction for the design problem and prove that our construction is near-optimal.

Key words: universal DNA arrays, zipcodes arrays, combinatorial design, De Bruijn sequences, SNP genotyping.

1. INTRODUCTION

Oligonucleotides are short single-stranded pieces of DNA (typically 15–50 nucleotides) made by chemical synthesis. In solution, oligonucleotides tend to specifically *hybridize* (bind) with their Watson–Crick complements (Watson *et al.*, 1996) and form a stable DNA duplex. This specificity is exploited in molecular hybridization assays, in which oligonucleotides are used as probes to identify any complementary (or near-complementary) DNA from a complex mixture of target DNA.

Array-based hybridization assays, introduced in the late 1980s (Drmanac *et al.*, 1991; Khrapo *et al.*, 1991; Lin *et al.*, 1996; Solas *et al.*, 1994; Blanchard and Hood, 1996; De Risi *et al.*, 1997), offer the possibility of simultaneously monitoring a multitude (currently up to tens of thousands) of hybridization reactions. These assays show great potential for many different applications such as SNP genotyping (Hacia, 1999), gene expression profiling (Alon *et al.*, 1999), and resequencing DNA (Kozal *et al.*, 1996; Hacia, 1999). A *DNA array* (or *array* for short) consists of a set of oligonucleotides that is bound to a solid support surface (e.g., silicon or glass). A fluorescently labeled target sample mixture of DNA or RNA fragments is brought in contact with the array and allowed to hybridize with the synthesized oligonucleotides. Theoretically,

[1] Agilent Laboratories, 3500 Deer Creek Road, Palo Alto, CA 94304.
[2] Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195.
[3] International Computer Science Institute, University of California at Berkeley, Berkeley, CA 94704.
[4] Institute for Systems Biology, Seattle, WA 98105.
[5] Computer Science Department, Technion, Haifa 32000, Israel.

the assay conditions are such that hybridization occurs only in sites on the surface that are Watson–Crick complements to some substring in the target. Therefore, scanning the resulting array fluorescence pattern reveals information about the content of the sample mixture. In practice, cross-hybridization is a main source of cross-signal contamination in any array-based hybridization assay.

In typical array-based hybridization assays, the set of oligonucleotides that is bound to the surface is target-specific, that is, the array consists of oligonucleotides that are subsequences of the target DNA sample. While this approach enables direct measurements of the target content, it suffers from cross-hybridization and from the high cost of manufacturing target-specific arrays. An alternative approach was recently suggested by S. Brenner and others (Brenner, 1997; Morris *et al.*, 1997). In the new approach, the same fixed *(universal)* array is utilized, instead of the custom-made arrays used in the conventional approach. A crucial design step of the new approach (detailed below) is the selection of a large set of oligonucleotides, called *tags*, such that each tag hybridizes well to its the Watson–Crick complement (the *antitag*), while exhibiting very poor hybridization to other antitags.

Before describing a possible application of the universal array approach in detail, we describe its general operation. In contrast to conventional microarrays, the analysis of a DNA sample now consists of three steps. In the first step, a set of *reporter molecules*, DNA sequences consisting of a target-specific *probe part* and a unique *tag part*, are prepared. In the second step, a solution-phase hybridization takes place between each piece of target DNA present in the sample and the probe part of the corresponding reporter molecules. In some cases, a reaction is performed that labels the reporter molecule fluorescently or otherwise. In the third step, a solid-phase hybridization on the universal array takes place between the tag parts of the reporter molecules that were involved into probe–target hybridizations in the previous step and the corresponding antitags on the array. Due to the complementarity between the tag part of each reporter molecule and the corresponding antitag on the array, each reporter molecule is sorted into the location on the array where the corresponding antitag is present. Therefore, in this approach, features of target DNA are indirectly mediated by the reporter molecules and are detectable on the appropriate locations on the array.

Note that the tag parts of the reporter molecules, and their counterparts on the universal array, can be designed independently of the particular DNA target. In this paper, we propose formal criteria and a method for the practical construction of such a set of universal tags, which we call a *Universal DNA Tag System.*

Compared to conventional microarrays, universal arrays have several advantages:

- Complicated array manufacturing processes are required only for the fixed, universal component of the assay. These universal components can therefore be mass-produced, significantly reducing manufacturing costs.
- The assay components that need to be designed for a specific target are involved in solution-phase processes. The underlying nucleic acid chemistry and thermodynamics are better understood than the same aspects of surface-based processes. Therefore, a more efficient and effective design process is facilitated.

Notice that, similar to conventional oligonucleotide microarrays, universal arrays can be employed for detecting various features of DNA targets. Before we address the specific design questions associated with a Universal DNA Tag System, we describe one example in which the universal array serves as a multiplexed SNP genotyping assay. SNPs (Single Nucleotide Polymorphisms) are single base differences, across the population, within an otherwise conserved genomic sequence (Wang *et al.*, 1998). Genotyping is a process that determines the variants present in a given genomic DNA. Consider a set of SNPs to be genotyped. The assay is performed as follows (see Figure 1):

1. A set of reporter molecules (one for each SNP) is synthesized in solution. Each reporter molecule consists of two parts that are ligated together. The first part is the Watson–Crick complement of the upstream sequence that immediately precedes the polymorphic site of the SNP. The second part of each reporter molecule is a unique tag from the universal set of tags.
2. When an individual is to be genotyped, a sample is prepared that contains the sequences flanking each of the SNP loci. The sample is mixed with the reporter molecules. Solution-phase hybridization then takes place. Assuming that specificity is perfect, this results in the flanking sequences of the SNPs paired only with the appropriate reporter molecule.
3. Single nucleotides, `A`,`C`,`T`,`G`, fluorescently labeled with four distinct colors, are added to the mixture. These labeled nucleotides hybridize to the polymorphic site of each SNP and are ligated to the

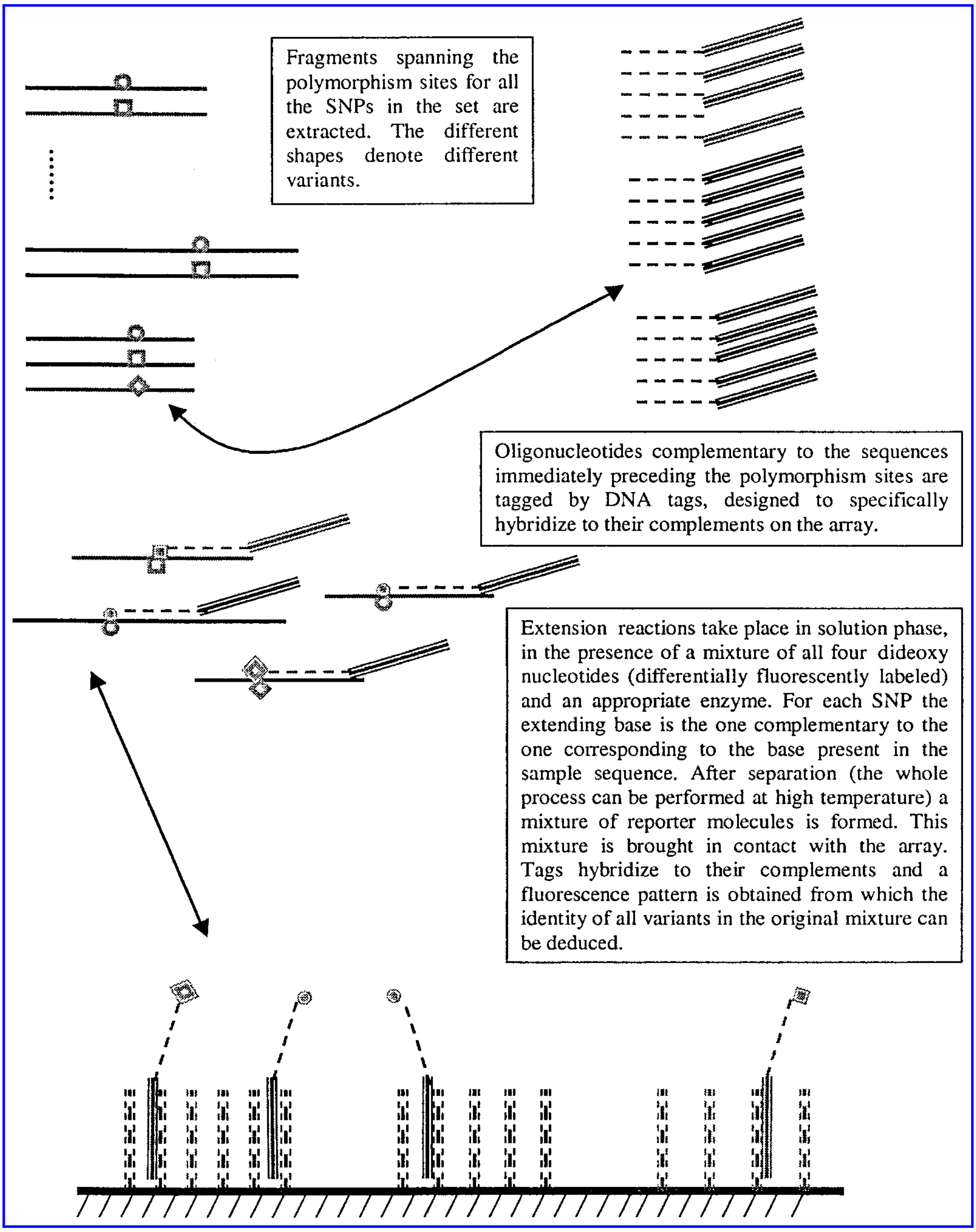

Fragments spanning the polymorphism sites for all the SNPs in the set are extracted. The different shapes denote different variants.
Oligonucleotides complementary to the sequences immediately preceding the polymorphism sites are tagged by DNA tags, designed to specifically hybridize to their complements on the array.
Extension reactions take place in solution phase, in the presence of a mixture of all four dideoxy nucleotides (differentially fluorescently labeled) and an appropriate enzyme. For each SNP the extending base is the one complementary to the one corresponding to the base present in the sample sequence. After separation (the whole process can be performed at high temperature) a mixture of reporter molecules is formed. This mixture is brought in contact with the array. Tags hybridize to their complements and a fluorescence pattern is obtained from which the identity of all variants in the original mixture can be deduced.

FIG. 1. Schematic for SNP genotyping using a DNA tag system.

corresponding reporter molecule. That is, each reporter molecule is extended by exactly one labeled nucleotide.

4. The extended reporter molecules are separated from the sample fragments and brought into contact with the universal array. Assuming that specificity is perfect, the tag part of each reporter molecule will hybridize only to its complementary antitag on the array. Thus, the extended reporter molecules sort into the array sites where the corresponding antitag is present.
5. For each site of the array, the fluorescent colors present at that site are detected. The colors indicate which bases were used for the extension at the corresponding SNP site and thus reveal the SNP variations present in the individual.

The problem of designing a DNA tag system presents a trade-off. Clearly, it is desirable to have as many tags as possible, in order to maximize the number of SNPs that can be genotyped in parallel. On the other hand, if too many tags are used, similar tags will necessarily entail cross-hybridization events (where tags hybridize to foreign antitags), reducing the accuracy and fidelity of the assay.

This design problem was identified in previous work and several formulations and solutions were proposed (Frutos *et al.*, 1997; Brenner, 1997; Morris, 1997; Shoemaker *et al.*, 1996; Garry *et al.*, 1999). These papers differ both in the way hybridization is modeled and in the algorithmic approach employed to find high-quality DNA tag systems. In Frutos *et al.* (1997) a tag system is described as a component of a surface-based DNA computing strategy. The authors take a coding theory approach and choose to model cross-hybridization constraints as general Hamming distance conditions. A set of 108 8-mers, with a 50% G/C content, which differ in at least 4 bases from each other, is constructed and experimentally tested for cross-hybridization.

In Garry *et al.* (1999) the method of using a DNA tag system to sort target DNA is presented, together with several examples of applications. The model assumption is that two oligonucleotides of length n need to have perfectly complementary substrings of length more than λ in order to form a reasonably stable duplex. A set of n-mers is said to be a λ*-free code* if no two elements of the set have a common substring of length more than λ. Given n, the design problem implied in Brenner (1997) is to construct the largest possible λ-free code.

A *De Bruijn sequence of order* λ is a cyclic sequence in which each possible λ-mer occurs exactly once (de Bruijn, 1946). In Morris *et al.* (1997) the authors observe that by parsing a De Bruijn sequence of order λ, an optimal λ-free code of size $4^\lambda/(n - \lambda + 1)$ can be obtained. However, the authors also recognize the shortcoming of their highly simplified mathematical model. To capture cross-hybridization, a model has to embody thermodynamic properties of DNA duplexes. The work we present here improves on this aspect by using De Bruijn sequences in a different way.

Additionally, Morris *et al.* (1997) suggest a greedy approach to designing DNA tag systems—starting with an empty set and iteratively adding a new tag to it, provided it does not hybridize with any of the complements of the tags already included. This approach clearly allows the use of arbitrarily complex thermodynamic models. However, there is no analysis for the performance of such greedy heuristics.

Recently, a small prototype universal array was used to detect K-ras mutations in tumor and cell line DNA (Garry *et al.*, 1999). The results reported suggest that universal arrays may be used to rapidly detect low-abundance mutations in any gene of interest.

In the context of hybridization arrays, the DNA tag system is used in order to facilitate sorting molecules into defined physical locations. The same idea is also applicable for other addressing systems, such as coded beads.

In this paper we use a simple thermodynamic model of hybridization to give a precise formulation of the tag system design problem. We employ new combinatorial ideas to provide an efficient construction and prove that our solution is near-optimal.

1.1. Thermodynamic model

In this section we describe a simple thermodynamic model of DNA duplex formation and use this model to derive a formal tag design problem. DNA duplexes are held together by weak hydrogen bonds formed between Watson–Crick complementary nucleotides. Denaturation (or melting) of a DNA duplex is generally achieved by heating the solution to a temperature which disrupts the hydrogen bonds. The energy required to separate complementary DNA strands is dependent on a number of factors, notably: strand length—longer duplexes contain a large number of hydrogen bonds and require more energy to separate them—and base composition, because C–G pairs have one more hydrogen bond than A–T pairs, strands with higher C–G content are more difficult to separate than those with low C–G content (Strachan, 1997).

As the assay temperature increases, the probability that a duplex will be separated increases. A useful measure of the hybridization behavior of two oligonucleotides U and V is their *melting temperature* $t_M(U, V)$ (in degrees Celsius). This is the temperature at which, under stated experimental conditions, half of the U and V oligonucleotides will be in a single-stranded form and half will occur in duplexes.

Let f denote the fraction of duplexes formed at a specific array site between an immobilized oligonucleotide and a fluorescently labeled tag. We assume that we are given two design parameters $0 \leq \alpha \leq \beta \leq 1$ such that:

- If $f > \beta$, the resulting fluorescent level is interpreted as a positive result.
- If $f < \alpha$, the resulting fluorescent level is reported as a negative result.

Moreover, we assume that the experiment is performed at temperature t. The parameters α and β can be translated into two temperature parameters, C and H ($C < t < H$) with the following property: If a duplex has a melting temperature of at most C, in the experiment (done at temperature t), at most a fraction ϵ of the duplexes will form. Similarly, if a duplex has a melting temperature of at least H, then at temperature t, at least a fraction δ of the duplexes will form.

The design goal is to construct a large DNA tag system such that

1. for each tag U, $t_{\mathrm{M}}(U, \overline{U}) \geq H$, and
2. for any two distinct tags U and V, $t_{\mathrm{M}}(U, \overline{V}) < C$.

Such a system, if used with temperature t, would allow each tag–antitag hybridization to be detected, without any cross-hybridization errors. Note that, in a practical application, C can be lowered and H can be increased to provide a buffer against parameter inaccuracies and limitations in the melting temperature model used, imperfect temperature control, and further experimental biases.

For estimating the melting temperature of a duplex between a tag and its antitag, we employ the *2–4 rule* that is commonly used for short oligonucleotides (Strachan, 1997): The melting temperature of a sequence and its complement is approximately twice the number of A–T base pairs plus four times the number of C–G base pairs. This model also enables us to state a condition that is sufficient to exclude cross-hybridization errors. The sufficient condition is derived from a characterization of the duplex formation process by Southern *et al.* (1999).

> The process begins by the formation of a transient nucleation complex from the interaction of very few base pairs. Duplex formation proceeds, one base pair at a time, through a zippering process. At any point the reaction may go in one of two directions, pairing or separation: if bases are complementary and freely available for pairing, duplex formation is more likely to proceed; if bases are non-complementary or a stable structure inhibits base pair formation, the block to the zippering process may drive the nucleation complex to fall apart. Duplex formation, and hence duplex yield, will be determined by the stability of the nucleation complex and of intermediates up to the point in the zippering process where the likelihood of strand separation is negligible.

Based on the above characterization and the 2–4 rule, we make the following assumptions.

1. Stable hybridization is always initiated by the formation of a nucleation complex.
2. A nucleation complex is a region of perfect base pairs between the tag and the foreign antitag.
3. The melting temperature of the nucleation complex can be computed according to the 2–4 rule.

Following these assumptions, we aim at avoiding the situation where a tag contains a substring x and a noncomplementary antitag contains $\overline{x}$, where $t_{\mathrm{M}}(x, \overline{x}) \geq C$. Since tags and antitags are complementary to each other, the above requirement can be stated in terms of solely the tag sequences.

2. TAG SYSTEM DESIGN PROBLEM

We can now formally define the tag system design problem. Our goal is to construct a tag system with a maximum number of tag–antitag pairs such that the following properties are satisfied:

(1) For each tag–antitag pair $(U, \overline{U})$, the melting temperature (using the 2–4 rule) satisfies $t_{\mathrm{M}}(U, \overline{U}) \geq H$.

(2) For any two distinct tags U and V, and for each oligonucleotide x that occurs as a substring in both U and V, $t_M(x, \overline{x}) < C$.

We now present a conservative formalization of the above tag design problem. Specifically, we not only forbid that a string x with $t_M(x, \overline{x}) \geq C$ occurs in any two distinct tags, but we also forbid that it occurs twice in the same tag. This mild restriction has also been imposed in previous work (Morris *et al.*, 1997) and allows a rigorous analysis and a near-optimal constructive solution.

As usual, we model oligonucleotides as strings over the alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$. We assume that the parameters C and H are fixed. To keep our exposition simple, we assign to each string the number that corresponds to half of its melting temperature in degrees Celsius.

GACCAAT	CAGCTAT	GTCGATA	CTGGTTA
CATTATCA	GAAATTCT	CTTAATGA	GTATTTGT
ATATAGTG	TAAAACTC	AATAAGAG	TTTTACAC

FIG. 2. A valid 4–10 code with 12 tags.

Definition 1. *The weight $w(s)$ of a string $s = a_1a_2 \cdots a_k$ is $\sum_{i=1}^{k} w(a_i)$, where $w(\mathtt{A}) = w(\mathtt{T}) = 1$ and $w(\mathtt{C}) = w(\mathtt{G}) = 2$. Given two parameters c and h, we call a set $\mathcal{T}$ of strings or "tags" a* valid *c-h* code *if the following two conditions are satisfied:*

Condition 1. *Each tag has a weight of h or more.*

Condition 2. *Any substring of weight c or more occurs at most once.*

Note that a valid c–h code corresponds to a solution of the tag design problem in which the lower melting temperature C is $2c$ and the upper melting temperature H is $2h$. We call the problem of finding a maximum valid c–h code the **Combinatorial Tag Design Problem**. See Figure 2 for an example of a valid code.

In the next section we derive an upper bound that, in particular, implies that the code in Figure 2 is optimal. That is, there exists no valid 4–10 code with more than 12 tags.

3. UPPER BOUND

In this section we derive a tight upper bound for the number of tags in a valid c–h code. The idea is to associate a numerical resource with each tag. We show that any tag has to use a certain minimum amount of resource and the global resource usage over all tags cannot exceed a certain maximum. The upper bound on the number of possible tags then follows from dividing the global upper bound by the minimum amount of resource used by each tag.

Roughly speaking, the limited resource we consider consists of those substrings in a tag with a weight of c or more that can occur only once in a valid c–h code. The following definition captures the minimal *suffixes* that can occur only once in a valid c–h code.

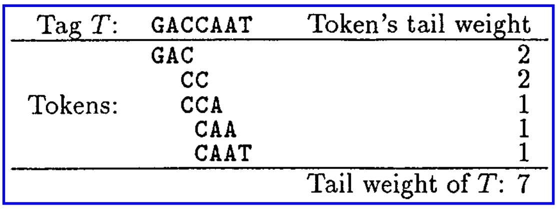
Definition 2. *We call a string t a c-*token *if $w(t) \geq c$, but t does not properly contain a suffix of weight $\geq c$.*

It is straightforward to see that a substring of weight $\geq c$ occurs twice if and only if some c-token occurs twice. We can therefore replace Condition 2 in our problem by the following equivalent condition.

Condition 2'. *Any c-token occurs at most once.*

Hereafter, we refer to a c-token simply as a *token*. With each token, we associate its *tail weight*, the weight of its terminal character. The tail weight of a tag T is the sum of the tail weights of all the tokens it contains as substrings. Figure 3 gives an example for $T =$ GACCAAT and $c = 4$.

Notice that all characters of T except the first two terminate a token and thus contribute their weight to the tail weight of T. The first two characters do not terminate a token because they do not terminate



Tag T:	GACCAAT	Token's tail weight
Tokens:	GAC	2
	CC	2
	CCA	1
	CAA	1
	CAAT	1
		Tail weight of T: 7

FIG. 3. Tokens and tail weight.

a suffix of weight $\geq c$. In the general case of a tag T in a valid c–h code, the maximal prefix that does not contain a suffix of weight $\geq c$ has a total weight of at most $c - 1$. Since the weight of a tag in a valid code is at least h (Condition 1), we have the following lower bound.

Lemma 1. *Any tag in a valid c–h code has a tail weight of at least $h - c + 1$.*

Based on Conditions 1 and 2', we now derive the upper bound on the total tail weight of a valid c–h code. We use $<n>$ to denote the set of strings with weight $n \in \mathcal{N}$, and G_n to denote the number of such strings. It is straightforward to derive the recurrence $G_1 = 2$, $G_2 = 6$, and $G_n = 2 \cdot G_{n-2} + 2 \cdot G_{n-1}$ for $n \geq 3$, and for the sake of simplicity we define $G_0 := 1$. Using standard techniques for solving recurrences, it can be shown that, for all $n \in \mathcal{N}$, G_n is the nearest integer to

$$\left(\frac{3+\sqrt{3}}{6}\right) \cdot \left(1+\sqrt{3}\right)^n.$$

To compute the maximal total tail weight of the tokens in a valid code, we partition tokens into four classes. In our symbolic representation of these classes, we denote any character with a weight of 1 (`A` or `T`) by `W` ("weak") and a character with a weight of 2 (`C` or `G`) by `S` ("strong"). To see how the set of tokens is partitioned, observe that any token is terminated by either a strong or a weak character, and it has a weight of either c or $c + 1$. Tokens of weight $c + 1$ begin with a strong character, since a string of weight $c + 1$ that begins with a weak character properly contains a suffix of weight c. Table 1 lists the corresponding four classes of tokens, their maximal cardinalities, and the maximal total tail weight they can contribute in a valid code.

The total tail weight of each class is computed by multiplying its size by the weight of the terminal character of its members. Only the last row requires additional explanation: Observe that any token in the class `S`$<c - 2>$`W` contains a token of the form `S`$<c - 2>$ as a substring, and thus only $2 \cdot G_{c-2}$ tokens of this form can exist in a valid code. Therefore the number of tokens of the form `S`$<c - 2>$`W` cannot exceed $2 \cdot G_{c-2}$. Summing up the rightmost column proves the following lemma.

Lemma 2. *The total tail weight of all tags contained in a valid c–h code is at most*

$$2 \cdot G_{c-1} + 6 \cdot G_{c-2} + 8 \cdot G_{c-3}.$$

Combining this with Lemma 1 yields the following upper bound.

Theorem 1. *Any valid c–h code contains at most*

$$\frac{2 \cdot G_{c-1} + 6 \cdot G_{c-2} + 8 \cdot G_{c-3}}{h - c + 1} \quad \textit{tags.}$$

For $h = 10$ and $c = 4$, the upper bound is $\frac{2 \cdot 16 + 6 \cdot 6 + 8 \cdot 2}{10 - 4 + 1} = 12$, which proves:

Corollary 1. *The 4–10 code in Figure 2 is optimal.*

TABLE 1. BOUNDS ON THE NUMBER OF TOKENS AND THEIR TAIL WEIGHT IN A VALID c–h CODE

Token class	*Max. occurrences in valid code*	*Max. tail weight*
$<c - 2>$`S`	$2 \cdot G_{c-2}$	$4 \cdot G_{c-2}$
`S`$<c - 3>$`S`	$4 \cdot G_{c-3}$	$8 \cdot G_{c-3}$
$<c - 1>$`W`	$2 \cdot G_{c-1}$	$2 \cdot G_{c-1}$
`S`$<c - 2>$`W`	$2 \cdot G_{c-2}$	$2 \cdot G_{c-2}$

4. OUR CONSTRUCTION USING CIRCULAR STRINGS

In this section we describe a method of constructing a nearly optimal c–h code for arbitrary values of c and h. Specifically, our code comprises at least

$$\frac{2 \cdot G_{c-1} + 6 \cdot G_{c-2} + 4 \cdot G_{c-3}}{h - c + 3} - 1 \quad \text{tags.}$$

Comparing our code with the upper bound, and using the recurrence for G_n, one finds that our method at least achieves a factor of approximately $0.89 \cdot (h - c + 1)/(h - c + 3)$ relative to the upper bound. For the values $c = 12$ and $h = 30$ that can be seen as relevant in practice, our construction yields 12119 tags, which corresponds to 87.6% of the upper bound of 13840 one gets from Theorem 1.

Throughout our exposition we will assume that the parameters c and $h \geq c$ are fixed. In addition we will assume that c is even. The case of an odd value of c requires only small modifications.

4.1. Construction overview

Our construction proceeds in two stages. In the first stage we construct a set of *circular strings* in which each token occurs at most once. The characters of a circular string are arranged in a cyclic order, and when convenient, we will assume that a specific character is designated as its origin.

In the second stage of our construction, the tags of our design are extracted as substrings from the circular strings, as illustrated in Figure 4. To satisfy Condition 1, each of the extracted substrings has a weight of h or more. To satisfy Condition 2', the overlap between two tags has a weight of at most $c - 1$.

Tags can be extracted from a circular string by a straightforward greedy algorithm that iterates the following operation. Starting at some position, the algorithm collects characters until their cumulative weight reaches or exceeds h, forming one tag, and then tracks back over as many characters as possible without collecting a weight of c or more. This operation is repeated until some overlap of weight $\geq c$ with the first extracted tag occurs, and the last retrieved tag is discarded. Given the best start position, this algorithm produces the largest number of tags that are substrings of a given circular string and can be included in a valid code. Observe that, since each character has a weight of 1 or 2, each tag extracted in this manner has a weight of at most $h + 1$ and the overlap between two tags is at least $c - 2$. Therefore, each circular string C leads to at least $\frac{w(C)}{h-c+3} - 1$ tags. This lower bound for the number of tags extracted from each cycle motivates us to consider the following formal problem.

Definition 3 (Circular String Problem). *Given the parameters $c > 0$ and $h > c$, construct a set $\mathcal{C}$ of circular strings that contain any substring of weight $\geq c$ at most once, and maximize*

$$\sum_{C \in \mathcal{C}} \left(\frac{w(C)}{h - c + 3} - 1 \right). \tag{1}$$

The construction we will describe optimally solves this problem. Specifically, our construction will yield a single cycle with the maximal possible weight among all set of circular strings that contain each substring of weight $\geq c$ at most once.

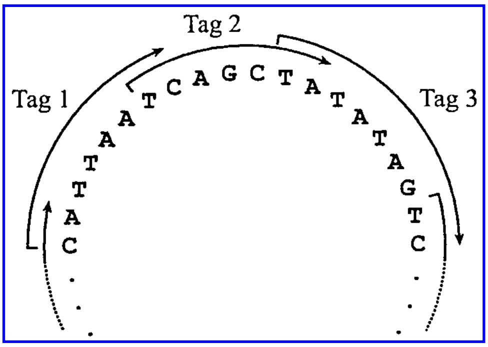


FIG. 4. Second stage—extracting tags from circular strings.

```
A = (W,0)
T = (W,1)
C = (S,0)
G = (S,1)
```

FIG. 5. Encoding of each character into one meta-character and one bit.

CCTGCAGGACGT	*String s=(μ(s),β(s))*
SSWSSWSSWSSW	*Meta-string μ(s)*
001100110011	*Bit string β(s)*

FIG. 6. A string corresponds to its meta-string bit string pair.

4.2. Meta-Strings and De Bruijn sequences

Our construction is based on the encoding of the nucleotides as given in Figure 5. Each character $a \in \Sigma$ is identified with a pair (μ, β), where $\mu \in \{\mathtt{W}, \mathtt{S}\}$ and $\beta \in \{\mathtt{0}, \mathtt{1}\}$. Extending this to strings over $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$, we identify each string s with its pair of *meta-strings* $\mu(s)$ and *bit string* $\beta(s)$. In this context, we will also call the string s an *instance* of the meta-string μ. Figure 6 gives an example.

Each circular string (or "cycle") in our construction will be an instance of a long circular meta-string that arises from repeating a shorter meta-string. If s is a string, we will denote k repetitions of s by s^k.

De Bruijn sequences. To avoid generating several identical tokens from repetitions of a meta-string μ, our construction will ensure that each instance of μ is paired with a different pattern in the bit-string.

For $k \in \mathcal{N}$, a binary De Bruijn sequence of order k is a cyclic binary sequence of length 2^k in which each possible substring of length k occurs exactly once (de Bruijn, 1946). Such sequences exist for all $k \in \mathcal{N}$ and can be constructed in linear time. We assume that a fixed De Bruijn sequence of order k is given for each $k \in \mathcal{N}$. Reading this sequence once, starting from a specific offset i relative to a fixed origin position, we obtain a linear string, a *linearization* that we denote by $\mathcal{D}_k^i$.

4.3. Cycle construction

Each cycle in our construction is based on a meta-string μ of weight c. Before describing the case of general meta-strings μ, we illustrate the construction principle for a special case.

4.3.1. Simple case. We consider meta-strings μ with the following two properties.

(P1) $\gcd(|\mu| + 1, 2^{|\mu|}) = 1$, i.e., the greatest common divisor of $|\mu| + 1$ and $2^{|\mu|}$ is 1.[1]

(P2) $\mu\mathtt{W}$ cannot be represented as a concatenation of two or more identical substrings.

For meta-strings μ that satisfy the two conditions above, our construction contains the cycle

$$C_0(\mu) = \left((\mu\mathtt{W})^{2^{|\mu|}}, (\mathcal{D}^0_{|\mu|})^{|\mu|+1}\right).$$

Figure 7 shows $C_0(\mu)$ for $c = 4$ and $\mu = \mathtt{SS}$. Notice that the meta-string $(\mu\mathtt{W})^{2^{|\mu|}}$ not only contains the meta-string $\mu = \mathtt{SS}$ four times as a substring, but also contains the meta-strings $\mathtt{SSW}$ and $\mathtt{SWS}$ four times.

4.3.2. General case. The construction for general meta-strings μ is a generalization of the above construction. Let $\alpha = \alpha(\mu)$ be the shortest *period* of $\mu\mathtt{W}$, i.e., the shortest substring such that $\mu\mathtt{W}$ can be

[1] Note that this condition is equivalent to the simpler "$|\mu|$ is even," but we prefer the above form in this context.

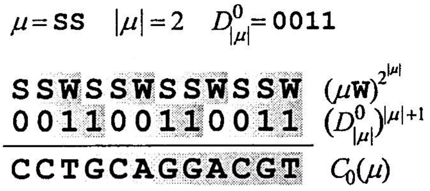

μ=SS $|\mu|=2$ $D^0_{|\mu|}$=0011
SSWSSWSSWSSW $(\mu W)^{2^{|\mu|}}$
001100110011 $(D^0_{|\mu|})^{|\mu|+1}$
CCTGCAGGACGT $C_0(\mu)$

FIG. 7. Construction of the cycle $C_0(\texttt{SS})$.

written as $\mu\texttt{W} = (\alpha)^p$, and set $k = k(\mu) = \gcd(|\alpha|, 2^{|\mu|})$. Define the meta-cycle $MC(\mu)$ and the bit cycles $BC_i(\mu)$ as follows:

$$MC(\mu) := (\alpha)^{2^{|\mu|}/k},$$

$$BC_i(\mu) := \left(\mathcal{D}^i_{|\mu|}\right)^{|\alpha|/k}, \quad i = 0, \dots, k-1.$$

For every meta-string μ with $w(\mu) = c$, our code contains the k cycles

$$C_i(\mu) = \Big(MC(\mu), BC_i(\mu)\Big), \quad i = 0, \dots, k-1.$$

Using the above notation, the set of cycles we construct is

$$\mathcal{C} := \bigcup_{w(\mu)=c} \bigcup_{i=0}^{k(\mu)-1} C_i(\mu).$$

We illustrate our construction using the parameters $c = 4$ and $h = 10$. Notice that all cycles in $\mathcal{C}$ are generated by the three meta-tokens $\mu = \texttt{SS}$, $\texttt{SWW}$, and $\texttt{WWWW}$. Using the De Bruijn sequences $\mathcal{D}_2 = \texttt{0011}$, $\mathcal{D}_3 = \texttt{00011101}$, and $\mathcal{D}_4 = \texttt{0000111101100101}$, Table 2 displays the set of cycles we construct.

4.4. Validity of our construction

In this section we prove that our construction yields a valid set of cycles, i.e., that any token of weight c or $c + 1$ occurs at most once. We will first state the proof for the case of tokens of weight c and then extend the proof to tokens of weight $c + 1$.

Let t be a token of weight c, and denote by (μ, β) the corresponding pair of meta-string and bit string. Clearly, t occurs if and only if both μ and β occur together, i.e., in the same position of a meta-cycle MC and an associated bit cycle BC_i. To show that t occurs at most once, we first show that μ can only occur in the meta-cycle $MC(\mu)$. Then we show that μ and β occur together at most in one cycle $C_i(\mu) = \big(MC(\mu), BC_i(\mu)\big)$, and, in such a cycle, they can occur together at most once.

We need some notations and two technical lemmas. For a string x and an integer b, denote by $x_{|b}$ the string obtained by cyclically rotating x to the left by b characters. That is, if $x = x_0, \dots, x_{\ell-1}$,

$$x_{|b} := x_{b \bmod \ell}, x_{(b+1) \bmod \ell}, \dots, x_{(b+\ell-1) \bmod \ell}.$$

TABLE 2. THE SET OF CYCLES $C_i(\mu)$ WE CONSTRUCT FOR $c = 4$ AND $h = 10$

	$\mu = \texttt{SS}$ $\alpha(\mu) = \texttt{SSW}$ $k(\mu) = 1$	$\mu = \texttt{SWW}$ $\alpha(\mu) = \texttt{SWWW}$ $k(\mu) = 4$				$\mu = \texttt{WWWW}$ $\alpha(\mu) = \texttt{W}$ $k(\mu) = 1$
i	*0*	*0*	*1*	*2*	*3*	*0*
$MC(\mu)$ $BC_i(\mu)$	SSWSSWSSWSSW 001100110011	SWWWSWWW 00011101	SWWWSWWW 00111010	SWWWSWWW 01110100	SWWWSWWW 11101000	WWWWWWWWWWWWWWWW 0000111101100101
$C_i(\mu)$	CCTGCAGGACGT	CAATGTAT	CATTGATA	CTTTCTAA	GTTAGAAA	AAAATTTTATTAATAT

Lemma 3. *If y is a cyclic rotation of x, the shortest period of y is a cyclic rotation of the shortest period of x.*

Proof. Let b be an integer such that $y = x_{|b}$. If $x = (\alpha)^p$, for some string α and some positive integer p, then $y = (\alpha_{|b})^p$. ■

Lemma 4. *Let x be a meta-string of weight c that occurs in a meta-cycle MC. Then x is followed by a weak character in MC.*

Proof. By our construction, MC is formed by repeating a meta-string α where $(\alpha)^p$ is of weight $c+1$. As x is a meta-string of weight c that occurs in MC, there exists a cyclic rotation α' of α such that x is a prefix of $(\alpha')^p$. Moreover, as $(\alpha')^p$ is of weight $c+1$, we conclude that $x^{\mathtt{W}} = (\alpha')^p$ occurs in MC. ■

We can now complete the first part of the validity proof.

Theorem 2. *The meta-string μ can occur in no meta-cycle except for $MC(\mu)$.*

Proof. Assume that μ occurs in a meta-cycle $MC(\tau)$ for some meta-string τ of weight c. By Lemma 4, $\mu^{\mathtt{W}}$ then also occurs in $MC(\tau)$. As $w(\mu^{\mathtt{W}}) = w(\tau^{\mathtt{W}})$, we get that $\mu^{\mathtt{W}}$ is a cyclic rotation of $\tau^{\mathtt{W}}$. Applying Lemma 3, we conclude that the shortest period of $\mu^{\mathtt{W}}$ is a cyclic rotation of the shortest period of $\tau^{\mathtt{W}}$, and thus $MC(\mu) = MC(\tau)$. ■

We now prove that μ and β can occur together at most once in in one of the cycles $\{C_i(\mu)\}_{i=0,\ldots,k-1}$. Denote by α the shortest period of $\mu^{\mathtt{W}}$, and set $k = \gcd(|\alpha|, 2^{|\mu|})$ as before. In the next lemma we compute the positions in which μ and β occur in the meta-cycles and bit cycles. In Lemma 6 we show that μ and β occur in the same position only in one cycle of the form $C_i(\mu)$. Finally, in Lemma 7 we show that, in such a cycle, μ and β can occur together only at most once.

Lemma 5. *The meta-string μ occurs in the meta-cycle $MC(\mu)$ in positions*

$$p(\mu) := \left\{ j \cdot |\alpha| \;\middle|\; j = 0, \ldots, \frac{2^{|\mu|}}{k} - 1 \right\}.$$

The bit-string β occurs in the i-th bit cycle $BC_i(\mu)$ in positions

$$p_i(\beta) := \left\{ b_0 + \ell \cdot 2^{|\mu|} - i \;\middle|\; \ell = 0, \ldots, \frac{|\alpha|}{k} - 1 \right\},$$

where $0 \le b_0 < 2^{|\mu|}$ is some fixed constant.

Proof. By construction, α has exactly $2^{|\mu|}/k$ occurrences in $MC(\mu)$, spaced $|\alpha|$ apart, as defined by $p(\mu)$. Since occurrences of α and μ start in the same positions, $p(\mu)$ also describes where occurrences of μ start. Consider now the bit string β. Since it has a length of $|\mu|$, it occurs exactly once in the De Bruijn sequence $\mathcal{D}^0_{|\mu|}$, in position b_0 (for some $0 \le b_0 < 2^{|\mu|}$). Therefore, in $BC_0(\mu)$, it occurs in positions $b_0 + \ell \cdot 2^{|\mu|}$. As the i-th bit-cycle $BC_i(\mu)$ is a cyclic rotation of BC_0 by i characters, we obtain the above expression for $p_i(\beta)$. ■

Lemma 6. *The meta-string β occurs together with μ in at most one of the cycles $C_i(\mu)$.*

Proof. As $k = \gcd(|\alpha|, 2^{|\mu|})$, for every position $p \in p(\mu)$, we have

$$p \bmod k = 0.$$

Similarly, for every position $q \in p_i(\beta)$, we have

$$q \bmod k = (b_0 - i) \bmod k.$$

Thus, p and q can be equal only if $i = b_0 \pmod{k}$, i.e., $i = b_0$, since $0 \le i < k$. ■

Lemma 7. *In any cycle $C_i(\mu)$, μ and β occur together at most once.*

Proof. The distance between any two consecutive occurrences of μ in $MC(\mu)$ is $|\alpha|$. The distance between two consecutive occurrences of β in $BC_i(\mu)$ is $2^{|\mu|}$. The least common multiple of these two distances is identical to $|\alpha| \cdot 2^{|\mu|}/k$, the length of $C_i(\mu)$, which proves our claim. ■

From Theorem 2 and Lemmas 6 and 7 we immediately obtain that μ and β occur together at most once over all cycles in our set $\mathcal{C}$ of cycles, and therefore the following holds.

Theorem 3. *The set $\mathcal{C}$ of cycles does not use any token of weight c twice.*

It remains to be shown that also tokens of weight $c + 1$ do not occur more than once. To see why this holds, assume that t is a token of weight $c + 1$ and denote by (μ, β) the corresponding pair of meta-string and bit string. As we assume here that c is even, μ contains at least one weak character. Let i be the position of the first weak character in μ. Set $\mu' = \mu_{|i+1}$, i.e., rotate μ left by $i + 1$ characters. This brings the weak character of μ to the last position of μ'. Denoting the c-weight prefix of μ' by τ, we have $\mu' = \tau\mathtt{W}$. By Theorem 2, we conclude that μ', and thus μ, can occur only in the meta-cycle $MC(\tau)$. Using Lemmas 6 and 7, it follows that t occurs at most once in our construction. Thus we have the following.

Theorem 4. *The set $\mathcal{C}$ of cycles is valid, i.e., it uses no token twice.*

4.5. *Token content of our cycles*

We now examine how many tokens of weight c and $c + 1$ the set of cycles $\mathcal{C}$ contains.

Lemma 8. *$\mathcal{C}$ contains each token of weight c exactly once.*

Proof. Let μ be an arbitrary meta-string of weight c. Observe that, in each cycle $C_i(\mu)$, $i = 0, \dots,$ $k-1$, μ occurs $2^{|\mu|}/k$ times, each time paired with a different μ-bit substring of $\mathcal{D}^i_{|\mu|}$. Therefore $\mathcal{C}$ contains all $k \cdot 2^{|\mu|}/k = 2^{|\mu|}$ distinct instances of each meta-string of weight c, which means that each token of weight c occurs exactly once. ■

Lemma 9. *$\mathcal{C}$ contains exactly half of the tokens of weight $c+1$, and each of these occurs exactly once.*

Proof. Let μ be an arbitrary meta-string of weight $c + 1$. Since we have assumed that c is even, μ contains at least one weak character. Thus, μ can be represented as a rotation of some meta-string $\mu'\mathtt{W}$, where μ' is a meta-string of weight c. Therefore, all instances of μ occur in the cycles of the type $C_i(\mu')$, alternating with instances of μ'. Due to the alternation, the numbers of instances of μ' and μ contained in any $C_i(\mu')$ are identical. Instances of μ in these cycles are also distinct, because each time μ occurs, it occurs with a distinct bit string. Since we have seen in Lemma 8 that all instances of μ' occur exactly once and the maximally possible number of instances for μ is twice as high, we can conclude that exactly half of the instances of μ occur in $\mathcal{C}$. ■

Together with Table 1, Lemmas 8 and 9 yield:

Corollary 2. *The total tail weight of $\mathcal{C}$ is $2 \cdot G_{c-1} + 6 \cdot G_{c-2} + 4 \cdot G_{c-3}$.*

4.6. *Pasting the cycles together*

Before extracting tags from the cycles in $\mathcal{C}$, we combine all cycles into a single cycle, without modifying the set of tokens that occur. This avoids the "end effect" that occurs when we extract tags from the cycles: Recall that every cycle can possibly leave all characters of a nearly complete tag unused. By pasting all cycles into a single cycle before the extraction, this situation occurs only once.

With the basic operation we present here, one can paste any two cycles A and B together if they share a common substring s with a weight of $c - 1$. If this is the case, A can be written as A_0s, where A_0 is some string, and, analogously, B can be written as B_0s. Then paste$(A, B) := A_0sB_0s$ defines a new cyclic string. The following lemma guarantees that the tail weight and the validity of the set of tokens contained in the circular code is preserved.

Lemma 10. *For any two cycles A and B that share a common substring of weight $c - 1$,* paste(A, B) *contains exactly the union of the tokens contained in A and the tokens contained in B.*

Proof. Observe that a token t cannot have the form $x{<}c-1{>}y$, with $x, y \in \Sigma$, since ${<}c-1{>}y$ already has a weight of c or more. Therefore, assuming that A and B share a common substring s, any token t contained in A_0s is contained in at least one of its linearizations A_0s and sA_0. Analogously, any token contained in B_0s is contained in at least one of its linearizations B_0s and sB_0. Since all above linearizations are substrings of paste$(A, B) = A_0sB_0s$, all tokens in A and B are also contained in paste(A, B). Conversely, any token contained in A_0sB_0s is contained in one of the above four linearizations, and thus also present in A or B. ■

Lemma 11. *There exists a sequence of* paste *operations that merges all cycles of $\mathcal{C}$ into a single cycle.*

Proof. The central observation is that each cycle $C_i(\mu)$, where μ is a meta-string containing $k \geq 1$ strong characters, can be pasted with a cycle of the form $C_j(\nu)$, where ν contains only $k - 1$ strong characters. To see how this works, observe that the circular meta-string of $C_i(\mu)$ can be expressed as a repetition of $\mu'\mathtt{S}$ for some meta-string μ' of weight $c - 1$. Consider an instance s of μ' in $C_i(\mu)$. The string $s\mathtt{T}$ is a token of weight c and, according to Lemma 8, does occur in some cycle $C_j(\nu)$ with $\nu = \mu'\mathtt{W}$. Observe that the meta-string ν contains only $k - 1$ strong characters. Since both cycles $C_i(\mu)$ and $C_j(\nu)$ contain s as a substring, they can be pasted together.

Iterating the above paste operation leads from any given cycle $C_i(\mu)$ to the one cycle $C_i(\mathtt{W}^c)$ that contains no strong characters. Since substrings of weight $c - 1$ are preserved by the paste operation, all cycles of $\mathcal{C}$ can indeed be pasted into a single cycle. ■

Together with Corollary 2, Lemmas 10 and 11 prove our earlier claim.

Theorem 5. *The above construction yields at least*

$$\frac{2 \cdot G_{c-1} + 6 \cdot G_{c-2} + 4 \cdot G_{c-3}}{h - c + 3} - 1 \quad \textit{tags.}$$

As mentioned before, this means that, asymptotically, our code achieves 89.8% of the upper bound from Theorem 1. For the values of $c = 12$ and $h = 30$, our code achieves 87.6% of the upper bound.

For our example with the parameters $c = 4$ and $h = 10$, Figure 8 shows how the set of cycles we constructed in Table 2 can be pasted together. Each cycle $C_i(\mu)$ appears in layer k, where k is the number of strong characters in μ.

According to the constructive proof of Lemma 11, each cycle in any layer $k \geq 1$ shares a common substring of weight $c - 1 = 3$ with a cycle in the layer $k - 1$ below it. In Figure 8, the common substrings are indicated by edges between the cycles. Viewing the figure as a graph, the paste operation can be viewed

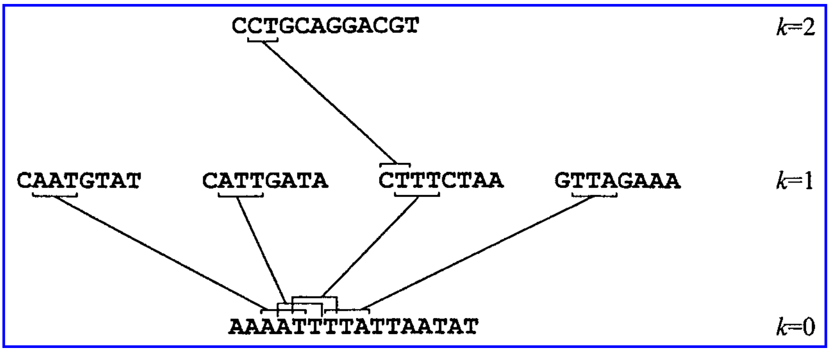
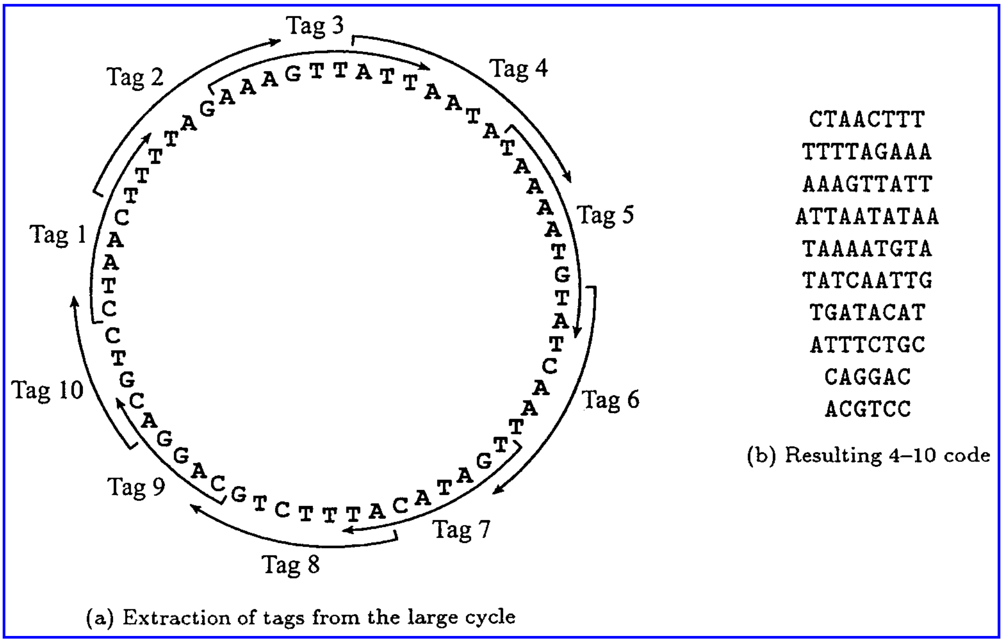


FIG. 8. Merging all cycles into one, for $c = 4$, $h = 10$.



(a) Extraction of tags from the large cycle

CTAACTTT
TTTTAGAAA
AAAGTTATT
ATTAATATAA
TAAAATGTA
TATCAATTG
TGATACAT
ATTTCTGC
CAGGAC
ACGTCC

(b) Resulting 4–10 code

FIG. 9. Final step in our construction of a 4–10 code.

as a graph operation. Replacing two cycles A and B by paste(A, B) corresponds to contracting an edge between A and B to form a new node with the cycle paste(A, B). As proven in Lemma 10, the set of tokens, and therefore shared substrings among cycles, which are edges in the graph, are preserved by this operation. In terms of the graph, Lemma 11 was proved by showing that the edges always form a spanning tree, and therefore any sequence of edge contractions can be used to merge all cycles into a single cycle.

Contracting the edges in Figure 8 in the order from bottom to top and from left to right, we get a single cycle, and can extract tags using the greedy algorithm described in Section 4.1. The resulting cycle and the final tag set are depicted in Figure 9.

For the parameters $c = 4$ and $h = 10$, our construction yields 10 tags. Notice that, in this case, our construction does not achieve the optimum of 12 tags (as does the optimal code given in Figure 2). However, compared to the optimum (that here coincides with the upper bound), the achieved ratio of 83.3% is already close to the asymptotic ratio of 89.8%.

5. OPTIMALITY OF OUR CONSTRUCTION

In this section we show that our construction is optimal within the limits of the circular approach to tag construction, i.e., our construction solves the Circular String Problem optimally. To this end, we prove that any set of cycles that is valid (i.e., no token is repeated) has a weight of at most $2 \cdot G_{c-1} + 6 \cdot G_{c-2} + 4 \cdot G_{c-3}$. Note that, using the recursion for G_n, this can be written as $G_c + 2 \cdot G_{c-1}$.

Let $\mathcal{C}'$ denote a valid set of cycles. We continue to discuss even values of c and use r to denote $c/2$. To bound the total weight of $\mathcal{C}'$, we will bound the number of weak characters and the number of strong characters in $\mathcal{C}'$. We need the following technical lemma.

Lemma 12. *For $k \in \{0, \ldots, r-1\}$, the number of instances of the meta-string $\mathtt{S}^k\mathtt{W}$ in $\mathcal{C}'$ is at most $2^k G_{2(r-k)}$.*

Proof. To bound the number of instances of $\mathtt{S}^k\mathtt{W}$, we bound the number of tokens that have an instance of $\mathtt{S}^k\mathtt{W}$ as a suffix. There are two cases to consider:

- Tokens of weight $2r$, which have the form $<2r-2k-1>\mathtt{S}^k\mathtt{W}$. There are $2^{k+1}G_{2(r-k)-1}$ tokens of this type.
- Tokens of weight $2r+1$, which have the form $\mathtt{S}<2r-2k-2>\mathtt{S}^k\mathtt{W}$. Since the prefix $\mathtt{S}<2r-2k-2>\mathtt{S}^k$ has a weight of $2r$, these tokens cannot occur more than $2^{k+1}G_{2(r-k)-2}$ times in $\mathcal{C}'$.

Therefore, the total number of tokens that have an instance of $\mathtt{S}^k\mathtt{W}$ as a suffix is

$$2^{k+1}G_{2(r-k)-1} + 2^{k+1}G_{2(r-k)-2} = 2^k G_{2(r-k)}.$$

As each instance of $\mathtt{S}^k\mathtt{W}$ in $\mathcal{C}'$ is the suffix of exactly one token, our claim follows. ■

The following lemma is straightforward to prove.

Lemma 13. *$\mathcal{C}'$ contains at most 2^r instances of the meta-string $\mathtt{S}^r$.*

The case of $k = 0$ in Lemma 12 implies that $\mathcal{C}'$ contains at most G_{2r} weak characters. To bound the number of strong characters in $\mathcal{C}'$, we need the following lemma.

Lemma 14. *The number of strong characters in $\mathcal{C}'$ equals the number of instances of the meta-string $\mathtt{S}^k\mathtt{W}$ (over $k = 1, \dots, r-1$), plus the number of instances of $\mathtt{S}^r$.*

Proof. We prove this lemma by mapping each character s in $\mathcal{C}'$ to instances of $\mathtt{S}^k\mathtt{W}$ or $\mathtt{S}^r$ in $\mathcal{C}'$. If the $r-1$ characters $\sigma_1, \dots, \sigma_{r-1}$ that follow s in $\mathcal{C}'$ are all strong, we map s to the string $s, \sigma_1, \dots, \sigma_{r-1}$, an instance of $\mathtt{S}^r$ in $\mathcal{C}'$. Otherwise, let i be the minimal index such that σ_i is a weak character. In this case, s is mapped to $s, \sigma_1, \dots, \sigma_i$, an instance of $\mathtt{S}^k\mathtt{W}$ in $\mathcal{C}$. It is easy to verify that this mapping is one-to-one and onto. ■

To establish a bound on the number of strong characters in $\mathcal{C}'$, we can now sum up the above bounds on the total number of instances of $\mathtt{S}^k\mathtt{W}$ ($k = 1, \dots, r-1$) and the number of instances of $\mathtt{S}^r$ in $\mathcal{C}$. In the following lemma we show that this sum (and thus the upper bound on the number of strong characters in $\mathcal{C}'$) equals G_{2r-1}.

Lemma 15.

$$\sum_{k=1}^{r} 2^k G_{2(r-k)} = G_{2r-1}$$

Proof. Using $i = r - k$, the above equality is equivalent to

$$\sum_{i=0}^{r-1} 2^{r-i} \cdot G_{2i} = G_{2r-1}$$

which is straightforward to prove by induction on r, using the above recursion for G_n. ■

Lemma 15 yields an upper bound of G_{c-1} for the number of strong characters in $\mathcal{C}'$. Together with the upper bound of G_c on the number of weak characters, we get the following.

Theorem 6. *The total weight of any valid set of cycles is at most $G_c + 2 \cdot G_{c-1}$.*

This proves that our construction solves the Circular String Problem optimally.

6. DISCUSSION

This paper formulates the design of a universal set of tags as a combinatorial problem and achieves a provably near-optimal solution. Our formulation rests on two assumptions:

1. If a sequence has weight greater than or equal to h (corresponding to melting temperature greater than or equal to $2h$ in the 2–4 model) then the sequence will hybridize to its complement.
2. Sequence x will fail to hybridize to sequence y provided that there is no string z of weight greater than or equal to c such that z is a substring of x and $\overline{z}^C$ is a substring of y.

In practice, the choice of the parameters h and c will depend on the concentrations of the reagents involved in the hybridization process and on other hybridization conditions.

The present combinatorial formulation of the actual design problem is conservative in the following sense: we require strings that may form a nucleation complex and initiate cross hybridization not only not be common to two different tags but also not repeat within a given tag. All our results, including the upper bound, are attained under this requirement.

Since our model of hybridization is only an approximate rule of thumb, it is inevitable that our design will include tag–antitag pairs that violate the two assumptions. Secondary structure in a tag may cause it to fail to hybridize to its antitag, even though its weight is greater than or equal to h. A tag and a foreign antitag may hybridize together because of long substrings that are nearly complementary but not exactly complementary.

Such violations depend on very specific properties of sequences, such as unusual dinucleotide composition, high-weight near-perfect matches or specific structural motifs. For example, a principal type of secondary structure within a DNA sequence is a hairpin, which can occur when the sequence contains two high-weight complementary substrings separated by at least three nucleotides. Such features occur infrequently in random sequences, and our method of tag construction does not appear to be strongly biased toward their occurrence. The 4–10 code in Figure 2 contains no complementary substrings of weight 4 in a single tag. The 12–30 code our method yields contains no tag containing complementary substrings of weight higher than 12. Only approximately 0.1% (14 tags out of a total of 12,119) contain a pair of complementary substrings of weight 12.

Moreover, we can guard against possible bias by randomizing the choice of De Bruijn sequences in the cycle formation stage of our construction. Thus, we believe that violations will not occur frequently in our set of tags and antitags if the parameters h and c are chosen conservatively.

In order to make our design useful, it will be necessary to verify our belief that violations are infrequent, and then get rid of the violations that do exist by deleting some tags. We can perform these tasks by a combination of computational and experimental approaches.

The computational approach depends on the availability of refined models of DNA secondary structure and of duplex formation between (not necessarily complementary) DNA sequences. For special types of duplexes, such refined models are already available (Santa Lucia, 1998; Peyret *et al.*, 1999). Given such models, we can computationally screen our tags for secondary structure interfering with hybridization, and screen the tag–antitag pairs for undesired hybridization. Because of the huge number of tag–antitag pairs, an exhaustive approach to the latter task may be infeasible. Instead, it may be possible to use mathematical properties of our design and of the model of duplex formation to limit the search. For example, it may be possible to show that a tag x and a foreign antitag y are likely to form a duplex only if they have been constructed from highly similar meta-tokens.

Another problem lies in the experimental validation of tag–antitag systems. Once a set of tags has been screened computationally, one can perform further screening by building universal arrays and exposing them to sets of antitags. Choosing these sets of antitags for this screening procedures poses another new design problem.

ACKNOWLEDGMENTS

Amir Ben-Dor was supported by the Program for Mathematics and Molecular Biology (PMMB). Benno Schwikowski was supported by a fellowship from the German Academic Exchange Service (DAAO). We thank Deborah Nickerson for introducing us to the problem of designing universal arrays, and we gratefully acknowledge the helpful remarks of the anonymous referees.

REFERENCES

Alon, U., Barkai, N., Notterman, D.A., Gish, K., Ybarra, S., Mack, D., and Levine, A.J. 1999. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues by oligonucletide arrays. *PNAS* 96, 6745–6750.

Blanchard, A.P., and Hood, L. 1996. Sequence to array: probing the genome's secrets. *Nature Biotechnology* 14, 1649.

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

De Bruijn, N.G. 1946. A combinatorial problem. *Proc. Kon. Ned. Akad. v. Wetensch.* 49, 758–764.

DeRisi, J., Iyer, V., and Brown, P. 1997. Exploring the metabolic genetic control of gene expression on a genomic scale. *Science* 278, 680–686.

Drmanac, R., Lennon, G., Drmanac, S., Labat, I., Crkvenjakov, R., and Lehrach, H. 1991. Partial sequencing by oligo-hybridization: Concept and applications in genome analysis. In *Proceedings of the First International Conference on Electrophoresis Supercomputing and the Human Genome. Edited by C. Cantor and H. Lim*, pages 60–75, Singapore, World Scientific.

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

Garry, N., Wotiwski, N., Day, J., Hammer, R., Barany, G., and Barany, F. 1990. Universal DNA microarray method for multiplex detection of low abundance point mutations. *J. Mol. Bio.* 292, 251–262.

Hacia, J.G. 1999. Resequencing and mutational analysis using oligonucleotide micro arrays. *Nature Genetics* 21(1), 42–47.

Khrapko, K.R., Khorlin, A., Ivanov, I.B., Chernov, B.K., Lysov, Y.P., Vasilenko, S., Floreny'ev, V., and Mirzabekov. 1991. Hybridization of DNA with oligonucleotides immobilized in gel: A convenient method for detecting single base substitutions. *Molecular Biology* 25(3), 581–591.

Kozal, M., Shah, N., Shen, N., Fucini, R., Yang, R., Merigan, T., Richman, D.D., Morris, M.S., Hubbell, E., Chee, M., and Gingeras, T.R. 1996. Extensive polymorphisms observed in HIV-1 clade B protease gene using high density oligonucleotide arrays: implications for therapy. *Nature Medicine* 7, 753–759.

Lin, C.Y., Hahnenberger, K.H., Cronin, M.T., Lee, D., Sampas, N.M., and Kanemoto, R. 1996. A method for genotyping cyp2d6 and cyp2c19 using genechip probe array hybridization. In *ISSX Meeting.*

Morris, M.S., Shoemaker, D.D., Davis, R.W., and Mittmann, M.P. 1999. *Methods and compositions for selecting tag nucleic acids and probe arrays*, European Patent Application 97302313.

Peyret, N., Seneviratne, P.A., Allawi, H.T., and Santalucia, J. Jr. 1999. Nearest-neighbor thermodynamics and NMR of DNA sequences with internal A.A, C.C, G.G, and T.T mismatches. *Biochemistry* 38(12), 3468–3477.

Santa Lucia, J. 1998. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci.* 95(4), 1460–1465.

Shoemaker, D.D., Lashkari, D.A., Morris, D., Mittmann, M., and Davis, R.W. 1996. Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-coding strategy. *Nature Genetics* 4(14), 450–456.

Solas, D., Pease, A.C., Sullivan, E.J., Cronin, M.T., Holmes, C.P., and Fodor, S.P.A. 1994. Oligonucleotide arrays for rapid DNA sequence analysis. *Proc. Natl. Acad. Sci. USA* 91, 5022–5026.

Southern, E., Mir, K., and Shchepinov, M. 1999. Molecular interactions on microarrays. *Nature Genetics* 21(1), 5–9.

Strachan, T., and Read, A.P. 1997. *Human Molecular Genetics.* John Wiley & Sons, New York.

Wang, D.G. et al. 1998. Large-scale identification, mapping and genotyping of single nucleotide polymorphisms in the human genome. *Science* 280, 1077–1082.

Watson, J.D., Gilman, M., Witkowski, J., and Zoller, M. 1996. *Recombinant DNA.* Scientific American Books, New York.

Address correspondence to:
Amir Ben-Dor
Chemical and Biological Systems Department
Agilent Laboratories
3500 Deer Creek Road
Palo Alto, CA 94304

E-mail: amirbd@cs.washington.edu

This article has been cited by:

1. Nina Svensen, Juan José Díaz-Mochón, Mark Bradley. 2011. Decoding a PNA Encoded Peptide Library by PCR: The Discovery of New Cell Surface Receptor Ligands. *Chemistry & Biology* **18**:10, 1284-1289. [CrossRef]
2. Anthony A. Philippakis , Aaron M. Qureshi , Michael F. Berger , Martha L. Bulyk . 2008. Design of Compact, Universal DNA Microarrays for Protein Binding Microarray Experiments. *Journal of Computational Biology* **15**:7, 655-665. [Abstract] [PDF] [PDF Plus]
3. Ion I. Mandoiu, Claudia Prajescu. 2007. High-Throughput SNP Genotyping by SBE/SBH. *IEEE Transactions on Nanobioscience* **6**:1, 28-35. [CrossRef]
4. Ion I. M#ndoiu , Drago# Trinc# . 2006. Exact and Approximation Algorithms for DNA Tag Set Design. *Journal of Computational Biology* **13**:3, 732-744. [Abstract] [PDF] [PDF Plus]
5. Jing-Guang Li, Ulrika Liljedahl, Chew-Kiat Heng. 2006. Tag/anti-tag liquid-phase primer extension array: A flexible and versatile genotyping platform. *Genomics* **87**:1, 151-157. [CrossRef]
6. B. DasGupta, K. M. Konwar, I. I. Mandoiu, A. A. Shvartsman. 2005. DNA-BAR: distinguisher selection for DNA barcoding. *Bioinformatics* **21**:16, 3424-3426. [CrossRef]
7. John A. Rose, Akira Suyama. 2004. Physical modeling of biomolecular computers: Models, limitations, and experimental validation. *Natural Computing* **3**:4, 411-426. [CrossRef]
8. John A. Rose, Russell J. Deaton, Akira Suyama. 2004. Statistical thermodynamic analysis and designof DNA-based computers. *Natural Computing* **3**:4, 443-459. [CrossRef]
9. Kaleigh Smith , Mike Hallett . 2004. Towards Quality Control for DNA Microarrays. *Journal of Computational Biology* **11**:5, 945-970. [Abstract] [PDF] [PDF Plus]
10. Amir Ben-Dor , Tzvika Hartman , Richard M. Karp , Benno Schwikowski , Roded Sharan , Zohar Yakhini . 2004. Towards Optimally Multiplexed Applications of Universal Arrays. *Journal of Computational Biology* **11**:2-3, 476-492. [Abstract] [PDF] [PDF Plus]
11. Suzanne Jenkins, Neil Gibson. 2002. High-Throughput SNP Genotyping. *Comparative and Functional Genomics* **3**:1, 57-66. [CrossRef]