BIOINFORMATICS

Vol. 18 no. 4 2002
Pages 529–535

Construction of optimal quality control for oligo arrays

Charles J. Colbourn[1], Alan C. H. Ling[2] and Martin Tompa[3,*]

[1]Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287-5406, USA, [2]Department of Computer Science, University of Vermont, Burlington, VT 05405, USA and [3]Department of Computer Science and Engineering, Box 352350, University of Washington, Seattle, WA 98195-2350, USA

Received on June 13, 2001; revised on August 22 and November 2, 2001; accepted on November 7, 2001

ABSTRACT

Motivation: Oligo arrays are important experimental tools for the high throughput measurement of gene expression levels. During production of oligo arrays, it is important to identify any faulty manufacturing step.

Results: We describe a practical algorithm for the construction of optimal quality control designs that identify any faulty manufacturing step. The algorithm uses hillclimbing, a search technique from combinatorial optimization. We also present the results of using this algorithm on all practical quality control design sizes.

Availability: On request from the authors.

Contact: tompa@cs.washington.edu

INTRODUCTION

An *oligo array* is a small chip containing tens of thousands of *spots*, to each of which is attached its own synthesized, short, single-stranded DNA molecule. Oligo arrays are important experimental tools for the high throughput measurement of gene expression levels by a given cell type under given conditions. For more information on oligo arrays see, for example, Lipshutz *et al.* (1999).

Our application is in the manufacture of oligo arrays rather than their subsequent use. Hubbell and Pevzner (1999) posed the quality control problem for oligo arrays, and formulated an approach to its solution that utilizes a small number of dedicated ‘quality control spots’ on the array. Sengupta and Tompa (2002) then reduced the problem to the design of good ‘balanced codes,’ but left open the general question of how to achieve such designs.

We resolve this open problem for all practical quality control design sizes, by producing not only *good* but *optimal* balanced codes. Our solution employs a very successful hillclimbing algorithm that can continue to be used in the future as design sizes increase.

As one concrete example of our results, suppose the manufacturer specifies that there are 100 manufacturing steps and synthesized DNA molecules of length 20. Our results then show, for any even $d \geqslant 4$, exactly how to design $10d$ quality control spots that will allow the array manufacturer or user to identify any failed manufacturing step among the 100 steps, even if $d-1$ of the quality control spots fail in arbitrary ways. This example is just one line from our Table 1.

The remainder of this section describes the quality control problem and solution in more detail.

The quality control problem for oligo arrays

An oligo array is manufactured in a series of steps labeled A, C, G, T, A, C, G, T, A, ... The number of manufacturing steps and the oligonucleotide lengths are fixed by the manufacturer. Initially every spot is empty. In preparation for any given step, an arbitrary subset of the spots can be *masked*. If the step is labeled with the nucleotide σ, only a spot that is unmasked will have σ appended to the end of its oligonucleotide. By appropriate construction of the masks, each spot can be designed to contain an arbitrary DNA sequence.

The manufacturing process is subject to two different sorts of faults: (1) several individual spots may be unreliable; and (2) an entire manufacturing step may fail, affecting all spots unmasked during that step. The goal of quality control is to identify any single failed step, even if e individual spots are unreliable, where e is a parameter of the manufacturing process. A small number of spots on the chip can be used for this quality control purpose, to be tested by hybridization with fluorescently tagged, complementary oligonucleotides.

Hubbell and Pevzner (1999) first investigated this problem. The clever idea underlying their approach is to manufacture identical oligonucleotides at multiple spots, using different schedules of steps. If no step fails, all such spots should show very similar fluorescent intensities. If some step fails, the spots with relatively low intensities hopefully provide a ‘signature’ that identifies the failed step.

The problem Hubbell and Pevzner left open was how

*To whom correspondence should be addressed.

Table 1. Existence of optimal balanced codes ‘Y,’ ‘+,’ ‘=’ denote existence, ‘.’ denotes nonexistence, and ‘o,’ ‘?’ denote open

v	k	Discrimination d			
		1 1234567890	1111111112 1234567890	2222222223 1234567890	3333333334 1234567890
9	8	+=========	==========	==========	==========
10	8	.++=======	==========	==========	==========
10	9	+=========	==========	==========	==========
11	8	.Y+Y======	==========	==========	==========
11	9	.+Y=======	==========	==========	==========
11	10	+=========	==========	==========	==========
12	8	.++=======	==========	==========	==========
12	9	.++=======	==========	==========	==========
12	10	.++=======	==========	==========	==========
13	8	.Y.Y+Y=Y==	==========	==========	==========
13	9	.YY+======	==========	==========	==========
13	10	.Y+=======	==========	==========	==========
14	8	...YY+YY+=	==========	==========	==========
14	9	.YY=+Y====	==========	==========	==========
14	10	.+Y=======	==========	==========	==========
15	8	....oY+YYY	Y=========	==========	==========
15	9	...+Y+Y===	==========	==========	==========
15	10	.+.=+=====	==========	==========	==========
16	8	.....+.+.+	.=.=+=?=+=	==========	==========
16	9	...YYY+YYY	==========	==========	==========
16	10	.Y.YY+Y=+=	==========	==========	==========
17	8	.....Yo+YY	Y=Y=Y=====	==========	==========
17	9	.....Yo+YY	Y=Y=Y=====	==========	==========
17	10	...YYY+=Y=	==========	==========	==========
18	8	...+YYo===	Y=========	==========	==========
18	9	.....+.+.+	.=.=.=+=?=	?=========	==========
18	10	...+YYo===	Y=========	==========	==========
19	8	...Y.YY+YY	==Y=======	==========	==========
19	9	.....ooY+Y	YYYYY=====	==========	==========
19	10	.....ooY+Y	YYYYY=====	==========	==========
20	8	...+Y+Y===	==========	==========	==========
20	9	....oYoY+Y	YYY=======	==========	==========
20	10	.....+.+.+	.=.=.=.=+=	?=?=======	==========
21	8	...YYYY+Y=	Y=========	==========	==========
21	9	...Yo+YY+=	Y=========	==========	==========
21	10	.....oooo+	YYYYYYYYY=	==========	==========
22	8	...+YYY===	==========	==========	==========
22	9	...YYYY=+=	Y=========	==========	==========
22	10	.....YoYo+	YYYY+=Y=Y=	==========	==========
23	8	...YYYY+YY	Y=========	==========	==========
23	9	...YoYYY+Y	=Y========	==========	==========
23	10	...YoYYYY+	Y=Y=======	==========	==========
24	8	...++++===	==========	==========	==========
24	9	...YY+Y=+=	==========	==========	==========
24	10	...Y.Yo=Y+	YY==+=Y===	==========	==========
25	8	...YYYY+==	==========	==========	==========
25	9	...YoYYY+Y	Y==Y======	==========	==========
25	10	...+o+Y=Y=	==========	==========	==========
26	8	...+YYY===	==========	==========	==========
26	9	...YYYY=+Y	YY========	==========	==========
26	10	...Y.YYYY+	Y=Y=+=====	==========	==========
27	8	...YYYY+==	==========	==========	==========
27	9	...++++===	==========	==========	==========
27	10	...YYYY==+	==========	==========	==========
28	8	...+Y+Y===	==========	==========	==========
28	9	...YYYYY+=	==========	==========	==========
28	10	...Y+YYY==	==========	==========	==========
29	8	.Y.YYYY+Y=	Y=========	==========	==========
29	9	...YYYY=+=	==========	==========	==========
29	10	...YYYY==+	YYY=======	==========	==========
30	8	.Y.+Y=====	==========	==========	==========
30	9	...YY+YY+=	==========	==========	==========
30	10	...++++===	==========	==========	==========
31	8	...YYYY+YY	Y=========	==========	==========
31	9	...YYYY=+=	Y=========	==========	==========
31	10	...YYYYYY+	==========	==========	==========
32	8	...++++===	==========	==========	==========
32	9	.Y.YYYYY+=	==========	==========	==========
32	10	...Y+YY===	==========	==========	==========
33	8	.Y.YYYY+==	==========	==========	==========
33	9	...YY+Y=+=	==========	==========	==========
33	10	...YYYYYY+	==Y=======	==========	==========
34	8	.Y.+Y=Y===	==========	==========	==========
34	9	...YYYYY+=	==========	==========	==========
34	10	...Y+YY===	==========	==========	==========

to design the quality control molecules and schedules to guarantee such signatures, even in the presence of e faulty spots. Sengupta and Tompa reduced this problem to the design of ‘balanced codes,’ defined below, and supplied an initial collection of good balanced codes (Sengupta and Tompa, 2002).

First Sengupta and Tompa abstracted the quality control problem as that of designing a *QC matrix* Q, which is a matrix of zeros and ones with a row for each quality control spot, a column for each manufacturing step, and $Q_{ij} = 1$ if and only if spot i is unmasked during step j. Given the spots with comparatively low fluorescent intensities as a column vector I, identifying the failed step corresponds roughly to finding the column of Q that resembles I, with up to e exceptions. Although this is similar to the familiar error-correcting code problem

A	C		
		G	T
A			T
	C	G	

A			T
	C	G	
A	C		
		G	T

Fig. 1. A pair of 4 × 4 QC blocks. For ease of visualization, the figure shows blanks instead of zeros, and the manufacturing step's label instead of a one.

(MacWilliams and Sloane, 1977), what makes it more complicated is that (1) one cannot always reliably compare the intensities of spots with different DNA sequences, and (2) even for the spots with identical sequences, one cannot always reliably distinguish between all such spots having high intensity and all such spots having low intensity.

In terms that are beyond the scope of the present discussion, but are detailed by Sengupta and Tompa (2002), the resulting properties of a good QC matrix Q are as follows:

(1) the set of DNA molecules manufactured at the quality control spots hybridize poorly to themselves and each other;

(2) Q has high 'separation' sep(Q), which ensures sufficient coverage of each step, and sufficient difference between steps to identify the failed step. Sengupta and Tompa proved that $\text{sep}(Q) \geqslant 2e + 1$ is sufficient to identify any single failed step, even in the presence of e arbitrarily faulty spots.

Sengupta and Tompa showed how to design QC matrices with these properties using a product construction. First they hand-crafted some *QC blocks*, which are small QC matrices. An example of a pair of 4 × 4 QC blocks from their paper is given in Figure 1. They then showed that a certain cross product of any good balanced code and these QC blocks yields a QC matrix with the desired properties above. To understand this, we turn to a discussion of balanced codes.

Balanced codes

A (v, b, k)-*set system* is a $b \times v$ matrix whose entries are zeros and ones, with exactly k ones per row. For each column j, we define the *replication number* of j to be the number of ones in column j. The set system is said to be r-*equireplicate* if every column has replication number r. The set system is said to have *discrimination* d if:

(1) for every column j, the replication number r_j satisfies $d \leqslant r_j \leqslant b - d$, and

(2) for every two distinct columns i and j, the number of rows in which columns i and j differ (called the *Hamming distance* between columns i and j) is at least d.

0	1	0	1	1	1	0	1	1	0	0	1	1	1	0
0	0	1	0	1	1	1	0	1	1	0	1	0	1	1
1	0	0	1	0	1	1	1	0	1	0	0	1	1	1
1	1	0	0	1	0	1	1	1	0	1	0	1	0	1
0	1	1	0	0	1	0	1	1	1	1	0	0	1	1
1	0	1	1	0	0	1	0	1	1	1	1	0	1	0
1	1	0	1	1	0	0	1	0	1	1	1	0	0	1
1	1	1	0	1	1	0	0	1	0	0	1	1	0	1
0	1	1	1	0	1	1	0	0	1	1	1	1	0	0
1	0	1	1	1	0	1	1	0	0	1	0	1	1	0

Fig. 2. A (15, 10, 9, 4)-bbc.

A (v, b, k)-set system with discrimination d is henceforth called a (v, b, k, d)-*balanced binary code*, or (v, b, k, d)-bbc for short. An example of a $(15, 10, 9, 4)$-bbc from Alon *et al.* (2001) is shown in Figure 2.

Returning now to the product construction for QC matrices, suppose B is a (v, b, k, d)-bbc, and one alternately replaces the ones in each row of B by the two 4 × 4 QC blocks of Figure 1, and replaces the zeros in B by 4×4 matrices of zeros. Sengupta and Tompa proved that the result is a $4b \times 4v$ QC matrix Q for which each DNA molecule has length $2k$, the set of DNA molecules hybridizes poorly, and $\text{sep}(Q) = 2d$ (Sengupta and Tompa, 2002).

An example of this product construction is shown in Figure 3, where the balanced code is the one from Figure 2. The 40 rows in this example correspond to 40 quality control spots, and the 60 columns correspond to 60 manufacturing steps. The oligonucleotide for each quality control spot can be obtained by reading across the corresponding row. For instance, the first spot contains the oligo ACATACATACATACATAC, and this same oligo is manufactured at nine other spots, each using a different schedule of manufacturing steps. Analogous statements are true for the oligos at spots two, three, and four.

Since the manufacturer specifies the number of steps ($4v$) and the molecule lengths ($2k$), and the goal is to minimize the number of quality control spots ($4b$) and maximize separation ($2d$), the resulting balanced code design problem is to minimize b and maximize d for a given v and k. To make this precise, given v, k, and d, we seek the smallest value of b for which a (v, b, k, d)-bbc exists. (The dual problem of finding the greatest value of d, given values of $v, k,$ and b, is equivalent.) Alon *et al.* established the following lower bound on b: if a (v, b, k, d)-bbc exists, then $b \geqslant \max\left(\left\lceil \frac{vd}{k} \right\rceil, \left\lceil \frac{vd}{v-k} \right\rceil\right)$ (Alon *et al.*, 2001). We call a (v, b, k, d)-bbc *optimal* when b has this minimum

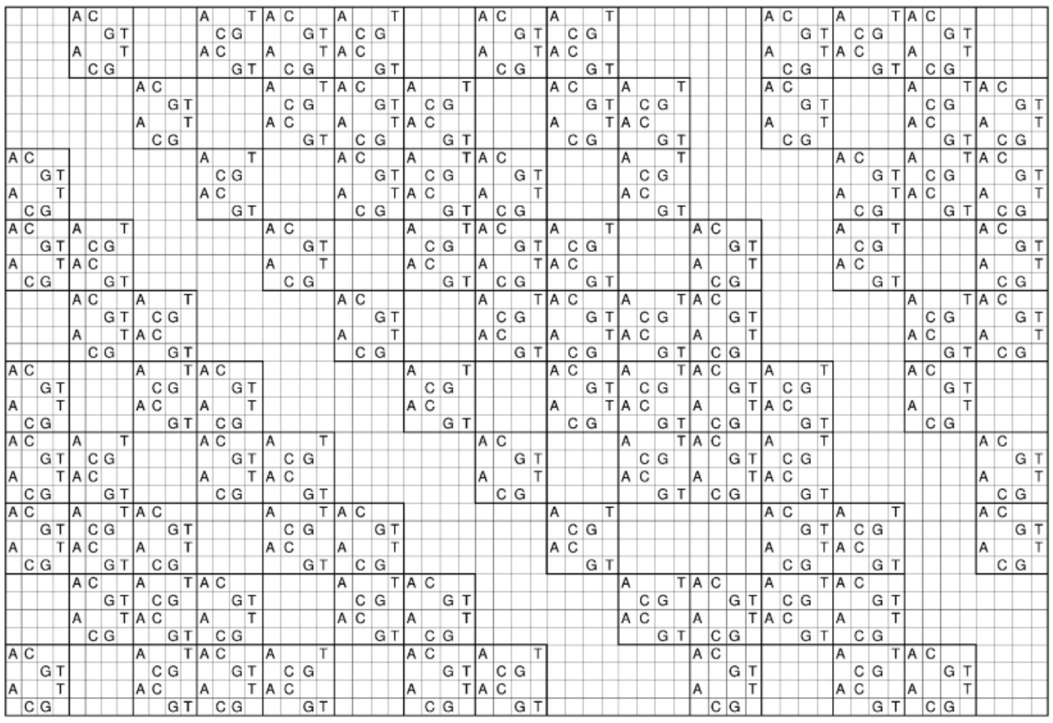

Fig. 3. The product of the (15, 10, 9, 4)-bbc of Figure 2 and the pair of 4×4 QC blocks of Figure 1, resulting in a 40×60 QC matrix Q with minimum separation $\text{sep}(Q) = 8$.

possible value $\max\left(\left\lceil \frac{vd}{k} \right\rceil, \left\lceil \frac{vd}{v-k} \right\rceil\right)$, because in this case the manufacturer is devoting as few spots to quality control as possible.

For the current photolithographic process, reasonable ranges for the parameters are $16 \leqslant 2k \leqslant 20$, $60 \leqslant 4v \leqslant 136$, and $4b$ up to a few hundred. Although Sengupta and Tompa supplied an initial collection of balanced codes, the major open problem of their paper was the construction of optimal balanced codes for arbitrary choices of v, k, and d. The current paper and its companion (Alon *et al.*, 2001) resolve this open problem for the relevant parameter ranges given above. The resulting constructions are summarized in Table 1. This table can be used to construct a spectrum of optimal quality control designs for any choices of oligo length and number of manufacturing steps, as in the example of 100 manufacturing steps presented earlier.

One useful tool for combining balanced codes is the following. Alon *et al.* (2001) proved that, if B_1 is a (v, b_1, k, d_1)-bbc and B_2 is a (v, b_2, k, d_2)-bbc, then $\left[\frac{B_1}{B_2}\right]$, the union of the rows of B_1 and B_2, is a $(v, b_1 + b_2, k, d_1 + d_2)$-bbc. We call this operation *addition*. Unfortunately, the addition of two optimal balanced codes need not be optimal. However, Alon *et al.* (2001) proved that, when either balanced code is equireplicate, the addition is optimal: if B_1 is an optimal equireplicate (v, b_1, k, d_1)-bbc and B_2 is an optimal (v, b_2, k, d_2)-bbc, then $\left[\frac{B_1}{B_2}\right]$ is an optimal $(v, b_1 + b_2, k, d_1 + d_2)$-bbc. For this reason, critical ingredients in producing larger optimal balanced codes are small ones that are equireplicate. In a companion paper, Alon *et al.* (2001) used techniques from combinatorial design theory to establish, for the practical parameter ranges $k \in \{8, 9, 10\}$ and $k < v \leqslant 34$, the existence or nonexistence of optimal equireplicate balanced codes in all but five cases, marked '?' in Table 1. In this paper, we therefore develop a practical technique that is well suited to producing the companion optimal nonequireplicate balanced codes. These are far more common than optimal equireplicate balanced codes: by a result of Alon *et al.* (2001), an optimal equireplicate code can only exist under the stringent conditions that $v \geqslant 2k$ and vd is a multiple of k, or $v < 2k$ and vd is a multiple of $v - k$. All other optimal balanced codes must be nonequireplicate, rendering the construction technique of this paper the more practical.

ALGORITHM

We now describe the algorithm to produce an optimal (v, b, k, d)-bbc. If $v < 2k$, we instead produce a $(v, b, v - k, d)$-bbc, and then change all zeros to ones and ones to zeros to obtain the desired balanced code. We can therefore assume that $v \geqslant 2k$.

The algorithm employs *hillclimbing*, a randomized search technique that constrains the search to move closer to a solution at each stage; see Gibbons (1996) and Mathon (1991) for a discussion of the basic technique. The method has been successful in a number of combinatorial problems; see Dinitz and Stinson (1981, 1987); Elliott and Gibbons (1992); Gibbons and Mathon (1993); Griggs and Murphy (1993) and Stinson (1985) for examples.

Given v, k, and d, we first compute $b = \lceil \frac{vd}{k} \rceil$. We next compute target replication numbers $r_1, r_2, \ldots, r_v$, satisfying $\sum_{j=1}^{v} r_j = bk$ and $d \leqslant r_j \leqslant b - d$ for $1 \leqslant j \leqslant v$. In general, there are numerous ways to satisfy these constraints, so we employ one of three simple methods:

(1) the first $bk - vd$ replication numbers are $d + 1$ and the remainder are d, or;

(2) if $bk - vd$ is even, the first $\frac{1}{2}(bk - vd)$ replication numbers are $d + 2$ and the remainder are d, or;

(3) if $bk - vd$ is odd, the first $\frac{1}{2}(bk - vd - 1)$ replication numbers are $d + 2$, the next is $d + 1$, and the remainder are d.

We form an initial $b \times v$ matrix M of zeros and ones, whose row sums are all k, and whose v column sums are $r_1, \ldots, r_v$. The matrix M can be constructed in many ways in general. The simple method we use is as follows. Each new row is constructed so that the position of the k one entries are those whose current column sums are farthest from their target values r_j. When M is constructed in this manner, it has the correct row and column sums to be a (v, b, k, d)-bbc, but likely it is not, because any two columns need not have Hamming distance at least d. We call a matrix with the correct dimensions and correct row and column sums a *putative balanced code*.

Consider two columns i and j. Let d_{ij} be the Hamming distance between these two columns. Since the desired distance is at least d, define the (i, j)-*defect*, def_{ij}, to be $\max(0, d - d_{ij})$. The defect is 0 only when these two columns have Hamming distance at least d. Then a measure of the poorness of a putative balanced code M is the *defect* of M, defined as $\mathrm{def}(M) = \sum_{1 \leqslant i < j \leqslant v} \mathrm{def}_{ij}$. This provides a means to compare two putative balanced codes for goodness: we define a putative balanced code M to be *closer* to a solution than another putative balanced code M' when the defect of M is less than that of M'. When the defect of M is 0, it is a valid solution.

The basic hillclimbing strategy follows. First we define a set $\mathcal{T}$ of easily computed transformations that map putative balanced codes to putative balanced codes. Our goal is to select transformations from $\mathcal{T}$ that reduce the defect, eventually to 0. A prototype hillclimbing algorithm is as follows:

```
Select an initial putative balanced code $M$.
**repeat**
    choose $T \in \mathcal{T}$, and calculate $M' = T(M)$.
    **if** $\mathrm{def}(M') \leqslant \mathrm{def}(M)$ **then** set $M = M'$.
**until** $\mathrm{def}(M) = 0$.
```

A strict hillclimbing algorithm might replace the test $\mathrm{def}(M') \leqslant \mathrm{def}(M)$ by $\mathrm{def}(M') < \mathrm{def}(M)$, so that a replacement is made only when actual progress towards a balanced code occurs. However, we allow (indeed, require) 'lateral' moves that leave $\mathrm{def}(M)$ unchanged. Before filling in the details of this algorithm, we introduce the transformations.

Ryser (1957) examined the following set of transformations. Let A be a $b \times v$ matrix of zeros and ones. Let $A[r_1, r_2; c_1, c_2]$ be the 2×2 submatrix indexed by rows r_1 and r_2, and columns c_1 and c_2. (We do not assume here that $r_1 < r_2$ or $c_1 < c_2$.) When $A[r_1, r_2; c_1, c_2]$ is the identity matrix

$$\begin{array}{c|cc} & c_1 & c_2 \\ \hline r_1 & 1 & 0 \\ r_2 & 0 & 1 \end{array}$$

we define the transformation $T[r_1, r_2; c_1, c_2]$ to be the replacement of this submatrix by its complement

$$\begin{array}{c|cc} & c_1 & c_2 \\ \hline r_1 & 0 & 1 \\ r_2 & 1 & 0 \end{array}.$$

The set $\mathcal{T}_M$ of allowed transformations is the set of transformations of type $T[r_1, r_2; c_1, c_2]$. Ryser proved that if A and B are binary matrices of the same dimension and the same row and column sums, then there is a sequence of transformations of this type producing B from A (Ryser, 1957).

Therefore, if the desired balanced code exists, it can be obtained by some sequence of transformations from any putative balanced code. Shaver (1973) used Ryser's transformations as the basis for a hillclimbing technique for balanced incomplete block designs, which are equireplicate balanced codes with *equal* Hamming distance between every pair of columns. The success of Shaver's method was somewhat limited: it succeeded typically only when exhaustive techniques such as backtracking (Gibbons, 1996) were also feasible. The first very successful hillclimbing method for constructing combinatorial designs was that of Dinitz and Stinson (1981) for 'strong starters;' it and its successors adopted a different set of transformations that are well suited to such set systems. The limited success of Shaver's approach can be attributed in part to the very severe constraints of balanced incomplete block designs on the Hamming distance between pairs of columns. In the case of balanced codes, the constraint is much less severe. Indeed, Ryser's

```
iterationcount = 0;
success = false;

while (iterationcount < $I$) and not success
    choose an initial putative balanced code M;
    lateralcount = 0;
    while lateralcount < $L$ and def (M) > 0

        /* find suitable 2 × 2 submatrix */
        choose r1 in {1, . . . , $b$} at random;
        repeat
            choose c1 in {1, . . . , $v$} at random
        until M[r1,c1] = 1;
        repeat
            choose c2 in {1, . . . , $v$} at random
        until M[r1,c2] = 0;
        /* sequentially search for suitable second row */
        r2 = (r1 mod $b$) + 1;
        while r2 ≠ r1 and (M[r2,c1] ≠ 0 or M[r2,c2] ≠ 1)
            r2 = (r2 mod $b$) + 1
        endwhile;

        /* accept transformation if it improves M */
        if r2 ≠ r1 then
            apply $T$[r1,r2;c1,c2] to M to produce N;
            if def (N) < def (M) then
                lateralcount = 0
            else lateralcount = lateralcount + 1
            endif;
            if def (N) ≤ def (M) then
                M = N
            endif
        endif

    endwhile;
    if def (M) = 0 then success = true endif
endwhile;

if success then output M else output 'failure' endif;
```

Fig. 4. Hillclimbing algorithm to produce optimal balanced codes.

set $\mathcal{T}_M$ of transformations performed remarkably well for balanced codes.

Given a putative balanced code M with nonzero defect, there is no guarantee that any transformation in $\mathcal{T}_M$ reduces its defect. Hence we must incorporate a stopping rule in the method other than waiting for success, if we are to avoid searching forever. Since we may abandon a particular search without success, we revise the basic method to allow multiple restarts. From the standpoint of efficiency, it is costly to generate all transformations in $\mathcal{T}_M$ and then select one to apply. Hence we also modify the method to generate a transformation more efficiently. The complete hillclimbing algorithm is given in Figure 4.

Two limits I and L are specified in the algorithm. The first limits the number of restarts from an initial configuration. The second limits the number of lateral moves, that is, transformations that do not improve the defect.

IMPLEMENTATION

For the quality control application described in the Section **Introduction**, we are interested in the construction of optimal (v, b, k, d)-bbcs with $k \in \{8, 9, 10\}$ and $k < v \leqslant 34$. We examined all parameter sets in this range. Initially we set the iteration limit I to 1, and the lateral limit L to 10 000. Remarkably, this quickly settled existence affirmatively for 372 different parameter sets. We do not list them all here. (These parameter sets included some later found to be implied by addition.)

We next examined the parameter sets for which hillclimbing with the specified limits failed to produce the desired balanced code. While any explanation is speculative, it appears that hillclimbing failed more frequently when the discrimination d was large (and hence the number b of blocks was also large), and also failed more frequently when the balanced code was equireplicate, or close to equireplicate. The former suggests that the lateral limit was too low for the parameters chosen, causing searches to be abandoned prematurely. We therefore attempted further searches using iteration limit $I = 10$, and raising the lateral limit to $L = 750\,000$. Searches with lateral limit 10 000 completed within 3–5 s, but searches with lateral limit 750 000 consumed 10–15 h on a 500 MHz Pentium system. Nevertheless, with these extended limits, an additional 25 parameter sets were settled affirmatively.

The failures on many equireplicate balanced codes are more problematic. The explanation appears to be that the success of hillclimbing depends critically on the presence of a large number of different balanced codes, but when the balanced code is equireplicate and v is close to $2k$, the number of solutions is very small. Indeed, this range of parameters is related to certain block designs called Hadamard designs (Beth *et al.*, 1986). Shaver (1973) encountered difficulties with hillclimbing on these types of designs, so our more frequent failures here are not surprising. Fortunately, Alon *et al.* (2001) settled almost all equireplicate cases by using combinatorial constructions, and so the effective use of hillclimbing in the nonequireplicate cases is sufficient.

DISCUSSION

In Table 1, we summarize our results for all parameter sets with $k \in \{8, 9, 10\}$ and $k < v \leqslant 34$. The encoding in this table is as follows:

(1) 'Y' denotes the existence of an optimal nonequireplicate balanced code found using the hillclimbing algorithm described in this paper;

'+' denotes the existence of an optimal equireplicate balanced code described in the companion paper (Alon *et al.*, 2001);

'=' denotes the existence of an optimal balanced code by addition of balanced codes with smaller discrimination;

(2) '`.`' denotes a parameter set for which Alon *et al.* (2001) proved that no optimal balanced code can exist;

(3) '`o`' denotes an unsettled nonequireplicate case;

'`?`' denotes an unsettled equireplicate case.

To illustrate the application of this table, let us derive the concrete example given in the Section **Introduction**. Suppose the manufacturer is interested in designs with 100 steps and molecule length 20. Since the number of steps is $4v$ and molecule length is $2k$ (see the Section **Introduction**), this example corresponds to the row $v = 25$, $k = 10$ in Table 1. From this row, we see that optimal balanced codes exist for every $d \geqslant 4$, with the possible exception of $d = 5$. By definition of optimality, $b = \lceil vd/k \rceil = \lceil 25d/10 \rceil = \lceil 5d/2 \rceil$. If d is even, $b = 5d/2$. The number of spots for such a design is $4b = 10d$, and its separation is $\text{sep}(Q) = 2d \geqslant 2e + 1$, where e is the maximum number of faulty spots that can be tolerated. From this inequality, $e = d - 1$. Thus, for any even $d \geqslant 4$, there is a design using $10d$ quality control spots, up to $d-1$ of which may be faulty.

Table 1 presents the status only for $1 \leqslant d \leqslant 40$, but it is easily established that existence is implied for all $d > 40$ for all parameter sets in our range, using addition.

In addition to the five equireplicate cases left unsettled (Alon *et al.*, 2001), only 23 nonequireplicate cases were not settled by hillclimbing, marked '`o`' in Table 1. Given the 397 cases that were settled by hillclimbing, this is a remarkably good success rate. One of the frustrations of using nonexhaustive search techniques such as hillclimbing is that, if the search is unsuccessful, there is no way to know whether the object being sought exists or not. While it is tempting to speculate that the 23 unsettled parameters correspond to nonexistent balanced codes, it is perhaps more likely that these instead demonstrate the limitations of the hillclimbing approach.

Despite these unsettled cases, we pursued hillclimbing because of its simplicity and its remarkable success, settling nearly 400 cases of practical interest. It is possible that a more sophisticated optimization method such as simulated annealing might succeed in resolving some of the 28 unsettled cases.

We conclude by noting that hillclimbing provides a useful method for producing not just one, but typically many different balanced codes for a given setting of the parameters. Thus we expect the method to be practical for the construction of additional balanced codes as needed in the quality control application.

ACKNOWLEDGEMENTS

We thank Noga Alon, Jeff Dinitz, Vic Klee, Rimli Sengupta, and the anonymous referees for helpful suggestions. This research was supported in part by ARO grant DAAG55-98-1-0272, DOE grant DE-FG02-00ER45828, and NSF grant DBI-9974498.

REFERENCES

Alon,N., Colbourn,C., Ling,A. and Tompa,M. (2001) Equireplicate balanced binary codes for oligo arrays. *SIAM J. Discrete Math.*, **14**, 481–497.

Beth,T., Jungnickel,D. and Lenz,H. (1986) *Design Theory*. Cambridge University Press, Cambridge.

Dinitz,J.H. and Stinson,D.R. (1981) A fast algorithm for finding strong starters. *SIAM J. Algebr. Discrete Meth.*, **2**, 50–56.

Dinitz,J.H. and Stinson,D.R. (1987) A hill-climbing algorithm for the construction of 1-factorizations and Room squares. *SIAM J. Algebr. Discrete Meth.*, **8**, 430–438.

Elliott,J.R. and Gibbons,P.B. (1992) The construction of subsquare free Latin squares by simulated annealing. *Aust. J. Combinatorics*, **5**, 209–228.

Gibbons,P. (1996) Computational methods in design theory. In Colbourn,C.J. and Dinitz,J.H. (eds), *The CRC Handbook of Combinatorial Designs*. CRC Press, Boca Raton, FL, pp. 718–740.

Gibbons,P.B. and Mathon,R. (1993) The use of hill-climbing to construct orthogonal Steiner triple systems. *J. Comb. Des.*, **1**, 27–50.

Griggs,T.S. and Murphy,J.P. (1993) 101 anti-Pasch Steiner triple systems of order 19 (at least). *J. Comb. Math. Comb. Comput.*, **13**, 129–141.

Hubbell,E. and Pevzner,P.A. (1999) Fidelity probes for DNA arrays. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, Heidelberg, Germany, pp. 113–117.

Lipshutz,R.J., Fodor,S.P.A., Gingeras,T.R. and Lockhart,D.J. (1999) High density synthetic oligonucleotide arrays. *Nature Genet.*, **21** (Suppl.), 20–24.

MacWilliams,F.J. and Sloane,N.J.A. (1977) *The Theory of Error-correcting Codes*. North-Holland, Amsterdam.

Mathon,R. (1991) Computational methods in design theory. In *Proceedings of the Thirteenth British Combinatorial Conference,* (*London Math. Soc., Lecture Note* 166). Cambridge University Press, Cambridge.

Ryser,H.J. (1957) Combinatorial properties of matrices of zeroes and ones. *Canad. J. Math.*, **9**, 371–377.

Sengupta,R. and Tompa,M. (2002) Quality control in manufacturing oligo arrays: a combinatorial design approach. *J. Comput. Biol.*, in press.

Shaver,D.P. (1973) *Construction of* (v, k, λ)*-designs Using a Nonenumerative Search Technique*, PhD Thesis, Syracuse University.

Stinson,D.R. (1985) Hill-climbing algorithms for the construction of combinatorial designs. *Ann. Discrete Math.*, **26**, 321–334.