ACS SyntheticBiology

Research Article

pubs.acs.org/synthbio

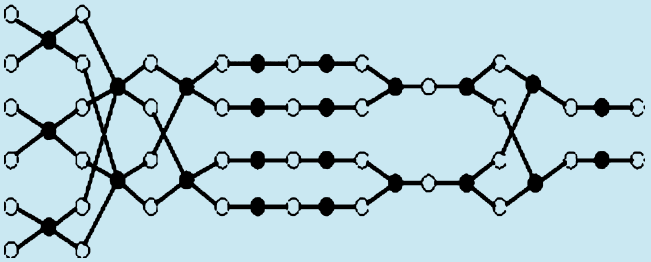
Rational Design of Orthogonal Libraries of Protein Coding Genes

Daniel Ryan[†] and Dimitris Papamichail*[,‡]

[†]National Institute for Mathematical and Biological Synthesis, University of Tennessee, Knoxville, Tennessee 37996, United States

[‡]Computer Science Department, University of Miami, Coral Gables, Florida 33146, United States

ABSTRACT: Array-based oligonucleotide synthesis technologies provide access to thousands of custom-designed sequence variants at low cost. Large-scale synthesis and high-throughput assays have become valuable experimental tools to study in detail the interplay between sequence and function. We have developed a methodology and corresponding algorithms for the design of diverse protein coding gene libraries, to exploit the potential of multiplex synthesis and help elucidate the effects of codon utilization and other factors in gene expression. Using our algorithm, we have computationally designed gene libraries with hundreds to thousands of orthogonal codon usage variants, uniformly exploring the design space of codon utilization, while demanding only a small fraction of the synthesis cost that would be required if these variants were synthesized independently.



KEYWORDS:

The emerging field of synthetic biology moves beyond conventional genetic manipulation to construct novel genes, cells, and eventually life forms that do not originate in nature. To achieve this goal, it utilizes large-scale DNA synthesis, DNA sequencing, and design and assembly of building blocks for biological system engineering. DNA synthesis and sequencing have enjoyed enormous advances in the past decade, with orders of magnitude improvements in both speed and cost.[1,2] It is worth noting that the cost of synthesizing a 2,500-bp long DNA sequence today, coding for an average sized protein, is less than $1000, and newly explored microarray-based synthesis techniques promise to reduce the cost of DNA synthesis to a fraction of a cent per base.[3] However, the ability to rationally design genes, enzymes, and pathways has not kept pace, to a large extent due to the inherent complexity of biological systems. Algorithmically driven methods can be employed to design the next generation of large-scale experiments, which will quantitatively characterize biological components and bridge the large gap of knowledge between sequence and function, enabling the rational engineering of genomic sequences.

Traditionally, due to the complexity of designing protein coding genes with well controlled attributes and the large gap of knowledge on the effect of these attributes, large-scale gene design experiments have relied on random synonymous mutations to generate the gene libraries that are then studied in well characterized organisms and regulatory contexts. Welsh et al.[4] have performed experiments with genes encoding commercially valuable proteins, by synthesizing 72 variants and chimeric combinations. They showed that variation in expression is highly correlated to codon usage, although preferred codons were not those used most frequently by *E. coli*, the organism where the genes were expressed. In particular, they pinpointed 5−6 codons as most critical for expression. In contrast, a paper from the Plotkin lab[5] claimed that most of the variance in expression results from the amount of secondary structure in the 5′ end of the gene, after testing 154 variants of the GFP protein, carrying random synonymous mutations, also in *E. coli*. Further analysis of Plotkin's data set by Supec and Mac,[6] using support vector machines and a M5′ regression tree model, identified 5 specific codons from 4 amino acids to contribute almost all of the variation in expression levels attributable to codon usage. These codons were different than the ones identified by Welsh et al. Additional findings from the Plotkin lab[7] indicate complex relationships between codon selection, translation initiation and elongation, misfolding of proteins, and autocorrelation.

These results beg the question: which rules should one follow to design genes for optimized gene expression? Currently it is hard to tell, even in a well studied model organism such as *E. coli*. A common belief, emphasized by Plotkin's group in ref 5 and by Super and Mac in ref 6, is that the mechanisms that determine gene expression can be established only by further large-scale experimentation. In this paper we present algorithms that allow the design of synthetic gene libraries on the scale and diversity necessary to address the needs of high-throughput in vitro experimentation.

Combinatorial Gene Libraries. Gene synthesis is the process during which oligonucleotides (oligos) are combined into larger DNA fragments, several hundred or thousand bases in length. Numerous protocols have been developed for

Special Issue: IWBDA 2012

Received: September 16, 2012
Published: February 27, 2013

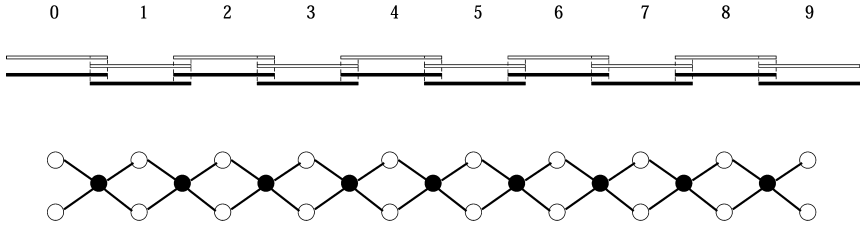


Figure 1. Combinatorial design of library that varies single codon usage and utilizes two variant oligos per position. All neighboring oligos share overlaps. The graph representation uses white nodes for internal oligo regions and black nodes for overlaps. Edges connect nodes (regions) that are part of the same oligo.

performing gene synthesis, three being most representative: the polymerase chain assembly (PCA),[8] ligase chain reaction (LCR),[9,10] and thermodynamically balanced inside out synthesis (TBIO).[11] In 2009, Gibson et al. published a paper describing a new technique for DNA assembly,[12] which is since commonly referred to as *in vitro isothermal assembly* or *Gibson isothermal assembly*. This technique is capable of simultaneously assembling large batches of overlapping DNA fragments, resulting in assemblies of size up to the mega-base level. Each assembly stage comprises a single isothermal reaction, but for large assemblies multiple stages are required. This technique was used to assemble an entire 16.3 kilo-base mouse mitochondrial genome from 600 overlapping 60-mers using 3 assembly stages.[13] Later, an entire 1.08 mega-base *Mycoplasma genitalium* genome was assembled from approximately 1000 cassettes of 1-kilo-base each, using 3 assembly stages.[14]

There are several advantages of the in vitro isothermal assembly over other methods, such as being largely sequence-independent, fast, and mostly labor-free. In ref 15 the in vitro isothermal assembly was used to create a combinatorial library of biochemical pathways. This synthetic library, containing 144 combinations of 3 promoters and 4 gene variants of the acetate utilization pathway in *E. coli*, was introduced into an *E. coli* mutant, and a high percentage (81%) of the colonies screened contained the complete functional pathway. This feat demonstrates the effective use of assembly methods to accurately and efficiently construct combinatorial libraries and, more importantly, rationally designed sets. These same techniques that were used to construct combinatorial pathways can also be used to synthesize libraries of gene variants. Using any assembly protocol, there are cost-effective ways to create a very large number of systematically varied versions of specified genes, pathways, or even genomes, by combining oligos with carefully chosen attributes, such as varied codon or codon pair distributions. Systematically varied data sets have the potential to enable experimental and statistical inquiries into the correlations between deciding characteristics, such as codon usage, codon context and secondary structure, and determinant features, such as gene expression, degradation, and function.

RESULTS AND DISCUSSION

Our goal is to develop a method that minimizes the number of DNA sequence oligos required to construct libraries of gene variants, by allowing the constructs to share common oligos. Let us examine a gene that can be assembled by putting together 10 overlapping DNA oligos. If we define as 1x coverage the total worth of sequence required to synthesize one gene variant (which consists of a larger number of base pairs than the length of the gene due to the overlapping regions), one can observe that 2x coverage is sufficient to synthesize 1024 gene variants, as depicted in Figure 1, where all neighboring oligos share overlapping regions. In such a design, we have 2 variations for each of the 10 oligos, with one maximizing the target codon's occurrences, and the other minimizing it; in Figure 1 these are depicted as light and dark colored oligos, respectively. Below the oligo design, we use a graph model to depict the oligo connectivity by overlap sharing, where white nodes indicate internal oligo regions, black nodes indicate overlaps, and edges connect nodes (regions) that belong to the same oligo.

Such a methodology though will not produce all unique designs from a codon frequency perspective, since the codon frequency values follow a binomial distribution, as shown in Figure 2 (from a real example of a 10-oligo synthesizable gene,

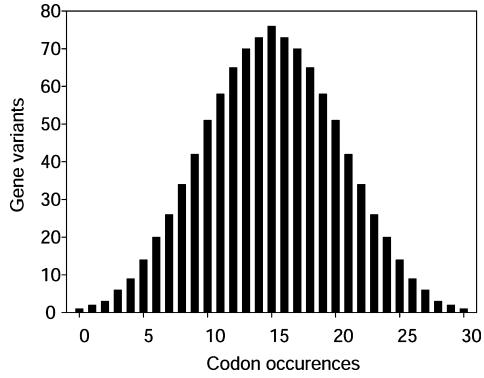


Figure 2. Binomial single codon usage distribution of gene variants of naive 10-oligo library design.

varying the codon usage of an amino acid that appears in 30 locations throughout the coding sequence). Out of 1024 gene variants, one contains the maximum number of occurrences of the target codon and another one the minimum, with the vast majority of constructs clustered midway. In most experiments today, this effect is undesirable, independently of the high-throughput methodology used to assay the properties of the gene variants, since there always exist limits in the number of sequences that can be realistically sampled and tested, thus restricting the scope of the experiment with respect to codon frequency modulation or any other desirable varied measure.

Our algorithm, for each codon whose frequency we intend to alter, examines consecutive DNA oligos to identify groups (intervals) containing enough corresponding amino acids to allow a 'step' between the frequency levels we wish to examine. As an example, assuming that we would like to create constructs that vary the usage of a particular codon, call it 'C', according to the frequencies (0.05, 0.30, 0.55, 0.80), we would identify consecutive oligos containing at least 25% (the step) of the corresponding amino acid's (call it 'A') occurrences in the gene. Then the algorithm identifies disjoint groups of consecutive

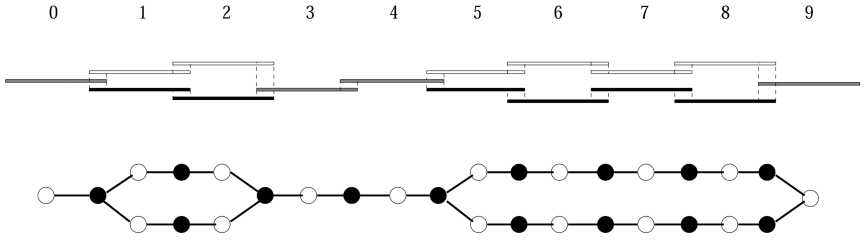


Figure 3. Single codon usage variation in GFP, 4 codon frequency values, 4 gene variants. In the graph representation, the variants are depicted as paths from the left-most to the right-most node.

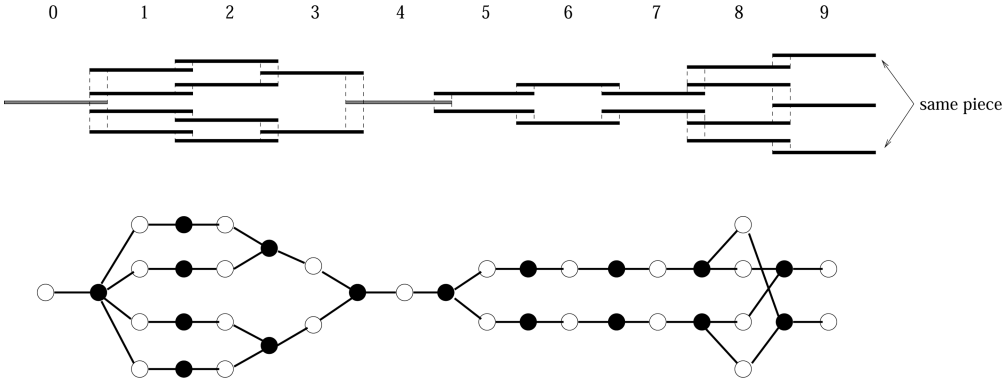


Figure 4. Two codon usage variation in GFP, 4 frequency values per codon, 16 gene variants.

oligos that, when combined, can produce the desired constructs, each with a unique frequency of the codon under consideration. An example of a design that can be used to construct gene variants utilizing a codon at 4 frequencies, differing by 25%, is shown in Figure 3, where interval (1,2) contains at least 25% and interval (5,8) at least 50% of amino acid A. The problem as described reduces to prime factorization of the number of target codon frequencies and construction of intervals with minimized sum of lengths, which is similar to the Frobenius coin problem,[16] but with bounded coefficients. An example varying the occurrences of two codons, with a second codon whose amino acid 'B' appears at least 50% in interval (1,3) and at least 25% in interval (8,9) is shown in Figure 4. In this example we can see that, instead of ordering 160 oligos to assemble the 16 desired gene variants, 24 oligos suffice, with 24 being also the minimum number to realize the design.

We have investigated the problem of minimizing the number of genomic oligos needed to synthesize a library of gene variants, all coding for the same amino acid sequence, but each having a unique codon distribution. For example, assuming the aim is to test the contributions of a specific codon to the expression of the gene, one can alter the frequency of the codon in the mRNA encoding. For testing 4 different frequency levels (such as low, medium low, medium high, and high) of the usage of a codon, 4 designs would be needed, each utilizing the codon at one of the 4 levels. For examining the effects of 5 different codons at 4 levels each, one would need to synthesize 1024 (4^5) different genes, to account for all possible combinations. The number of individual gene variants increases exponentially with the number of codons that we wish to investigate, as does the cost of synthesizing all of the different genes.

Problem Formulation and Notation. DNA synthesis is performed by assembling synthesized oligos, with typical lengths ranging from 50 to 400 base pairs. These oligos share common overlapping regions at their ends, ranging typically in length from 15 to 80 base pairs. For our problem, we are trying to minimize the number of oligos that are used to create a library of protein coding gene variants, such that each variant is uniquely representing a combination of selected codon frequencies.

More specifically, the input of our problem is the following:

1. An amino acid sequence A_1, A_2, ..., A_m, with m amino acids.
2. The number of DNA base pairs (bp) per synthesized oligo l and the desired size of the overlaps between oligos v.
3. A list of amino acids (the "amino acids of interest"), a_1, ..., a_k, and corresponding codons (the "codons of interest"), c_1, ..., c_k that will have their frequencies systematically varied.
4. For each codon of interest c_i: a minimum frequency $\min_i$, a maximum frequency $\max_i$, and total number of desired frequencies n_i (which will be equally spaced between the min and max values).

To put these definitions in perspective, let us consider an example, where we want to design a library of the jellyfish Green Fluorescent Protein (GFP) gene, with a length of m = 238 amino acids. Setting the values l = 90 and v = 18, for the oligo and overlap lengths, respectively, would require the use of N = 10 oligos in order to assemble one single variant of the gene, of length 714 bp. If we want to study the effect of the codons c_1 = 'CTA', coding for amino acid a_1 = 'S' (serine), and c_2 = 'ACC', coding for amino acid a_2 = 'T' (threonine), we could vary the frequencies of the two codons at 4 different equidistant values, with $\min_i$ = 0.05(5%), $\max_i$ = 0.80(80%), and n_i = 4, $i \in (1,2)$, which would require the synthesis of 16 different gene variants, for all combinations of the (0.05, 0.30, 0.55, 0.80) frequencies of the two codons.

We will use the term *coverage* to indicate the amount of sequence necessary to synthesize the gene variants. We define as 1x coverage as the amount of DNA sequence necessary to synthesize one gene variant. In the example above, 1x coverage for GFP would be 900 base pairs, since 10 overlapping oligos of 90 bp each are necessary to assemble one variant of the gene.

dx.doi.org/10.1021/sb300086d | *ACS Synth. Biol.* 2013, 2, 237−244

The problem has been presented in terms of a single codon being systematically varied per amino acid of interest; however, since the sum of all codon frequencies for any amino acid must be 1, a single codon's frequency cannot be varied in isolation. A simple way to specify the frequencies for an amino acid's remaining codons is to assume a fixed relative frequency between all codons other than the codon of interest (e.g., use either the original gene distribution or target host codon distribution). The algorithm presented herein is capable of handling a more general situation than this, where the occurrences of all codons for an amino acid of interest are either increasing or decreasing by a specified frequency step size (or remain fixed), such that the sum of their frequencies remains one. Using Proline as an example, let c_1, c_2, c_3, and c_4 denote its four codons and let $n = 3$. We could specify three frequency levels for c_1, for example, (0.1,0.4,0.7), and in addition allow all remaining codons to have equal relative frequencies to each other, resulting in frequencies ((0.1, 0.3, 0.3, 0.3), (0.4, 0.2, 0.2, 0.2), (0.7, 0.1, 0.1, 0.1)) for (c_1, c_2, c_3, c_4). Alternatively, we could specify a starting frequency distribution of (0.1, 0, 0.7, 0.2) for (c_1,c_2,c_3,c_4) and let c_1's frequency increase by 0.2, c_2's by 0.1, c_3's frequency decrease by 0.3, and c_4 remain fixed between each of the three levels, resulting in frequencies ((0.1, 0, 0.7, 0.2), (0.3, 0.1, 0.4, 0.2), (0.5, 0.2, 0.1, 0.2)). Specifying a fixed relative frequency between the remaining codons is actually a special case of the second method, where all remaining codon frequencies are changing in the opposite direction of the single codon of interest. We present the algorithm in terms of a single codon per amino acid of interest to keep the notation simpler and not confuse matters more than necessary. Implementing the more general case requires applying the allocation algorithm described for the single codon of interest case to a virtual codon, with step size between frequencies equal to the sum of all increasing codons' step size.

We will refer to the desired equally spaced frequencies for each amino acid of interest as the design space for that amino acid. When we discuss the design space without mentioning one particular amino acid, we will imply all possible combinations arising from choosing one point from the design space of each amino acid of interest. We can view the design space of each amino acid a_i as a subset of n_i equally spaced points in $[0,1]$ and the general design space as an equally spaced grid of points in $[0,1]^k$ consisting of $\prod_{i=1}^{k} n_i$ points in total.

Let Δ_i denote the space between two points of the design space of the amino acid a_i. This can easily be computed from the minimum, maximum, and number of points in the design space for the amino acid:

$$\Delta_i = \frac{\max_i - \min_i}{n_i - 1}$$

We can view the design space for amino acid a_i as a vector of n_i points:

$$\langle \min_i , \min_i + \Delta_i, \min_i + 2\Delta_i, ..., \min_i + (n-1)\Delta_i \rangle \quad (1)$$

In our previous example, both amino acids a_1 and a_2 have a Δ of 0.25.

The precise problem would be defined as: Given the user inputs 1−4 above, find and specify the minimal number of oligos that will yield uniquely the gene variations in the design space defined by the input, when assembled in every combination of possible overlaps. It is expected that the resulting library will cover the design space uniformly when assembled via a method such as Gibson isothermal assembly, assuming annealing of all overlaps is equally likely.

One final note before describing the algorithm for solving this problem: the design space refers to an overall codon usage profile and thus is completely "non-local" in nature. This implies that any two amino acid sequences with the same codon usage profile will be considered the same gene design within this framework. This assumption is supported by experimental results that have indicated that codon usage affects translation rates in a linear aggregate fashion.[17] The same is true for codon pair usage,[18] which is another protein coding mRNA feature that can be assayed using synthetic libraries designed with the methodologies described in this paper. It has been shown though that codon usage alone cannot account for all of the variation seen in gene or protein expression levels.[7] Other mechanisms known to affect expression are mRNA secondary structure (especially near the 5′ end),[5] codon pair bias,[18] and Shine-Dalgarno sequences.[19] Secondary structure at the 5′ end of the gene can be preserved with minimal interference to the overall designs, since structures that significantly affect expression levels are local and located within the initial 40−60 nucleotides of the coding region, therefore affecting only the first oligo of a synthetic design. By disallowing modifications in this oligo, we preserve the original local secondary structures, minimally affecting library construction. Shine-Dalgarno binding sites occur infrequently, once every 4096 nucleotides by chance. Even in a large library of designs, such sites will rarely appear and can be altered or preserved with minimal or no impact to the codon distribution. Codon pair bias is another factor that should be accounted for. Although it is difficult to alter codon usage without affecting codon pair bias, in practice it is not expected that libraries that do not specifically alter the codon pairs toward any particular end would generate significantly codon pair biased designs.

Green Fluorescent Protein Coding Libraries. To demonstrate the optimization potential of our algorithm, we experimented in-silico with library designs involving the jellyfish GFP protein, which has a length of 238 amino acids. We set the oligo size to 90 bp and the overlap length to 18 bp, with 10 overlapping oligos covering the whole coding sequence, values compatible with current synthesis and assembling technologies. Our initial designs involved 3 amino acids, threonine (T), valine (V), and glutamic acid (E). We varied a single codon of threonine and valine at 4 frequencies, (0.1, 0.35, 0.60, 0.85) and glutamic acid at 6 frequencies (0.1, 0.25, 0.4, 0.55, 0.7, 0.85). This library would require the synthesis of 96 individual gene variants if each was synthesized independently. Our algorithm generated a multiplex library with 3.8x coverage, requiring the synthesis of 25 times fewer oligos. The design is shown in Table 1, where the graph depicting the oligo overlaps is displayed in Figure 5.

Expanding the range of frequency values for all 3 codons, by reducing the minimum frequency from 10% to 5%, and increasing the maximum from 85% to 90%, while preserving the values of all other parameters, produces a library design that requires 7.8x coverage to be synthesized. This library realizes savings in excess of 12-fold over the individual gene variant design. Interestingly, limiting the frequency range by decreasing the maximum codon frequency from 85% to 70%, while keeping the minimum frequency at 10%, leads to a design with

Table 1. GFP Design with 10 Overlapping Oligos, Varying 3 Codons (of Amino Acids T, V, and E) at 4 Frequency Levels for T and V and Six Frequency Levels for E[a]

AA	fragments utilized	Design
T	[(1, 2), (7, 9)]	{⟨0, 2⟩, ⟨0, 1⟩}
V	[(0, 0), (2, 7)]	{⟨0, 1⟩, ⟨0, 2⟩}
E	[(0, 1), (3, 5)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}

[a]The first column identifies the amino acid whose codon frequency is varied, the second column depicts contiguous oligo sets that contain the appropriate Δ codon content multiple, and the third shows the resulting designs that minimize coverage.

a required coverage of 3.5x, thus achieving only marginally better savings.

Next we consider the realistic case involving the 4 codons and their corresponding amino acids (S, T, V, A) that Supek and Muc[6] identified as contributing most of the variation in expression in Plotkin's[5] experiments. Varying the occurrences of each codon at 4 frequency levels (0.05, 0.30, 0.55, 0.80) would require synthesizing a library of 256 gene variants, in order to quantify the effect of these codons in the expression of a gene. Our algorithm produces the multiplex design depicted in Table 2.

This particular library design can be synthesized with 5.2x coverage, for which we only need to order 2% as much sequence as we would for the 256 separate gene variants, reducing the synthesis cost of such an experiment from $70,000 to about $1,500 (based on 2012 synthesis costs), including a modest cost of labor and material necessary to assemble the oligos.

Similarly, varying single codon frequencies of 8 amino acids (P, T, V, E, D, G, Q, K) in the GFP protein encoding at (0.1, 0.3, 0.5, 0.7) and with all other parameters remaining the same requires in a library with 65,536x coverage to be synthesized when no optimization is applied, where our algorithm allows the exploration of the same design space with only 12x coverage or in other words achieves in excess of 5000-fold savings. The resulting design is displayed in Table 3.

Polio Virus Capsid Protein Coding Libraries. We evaluated our algorithm performance against another protein coding gene, significantly larger than GFP, that codes for the P1 capsid protein of Polio virus. The RNA encoding of the gene has a length of 2643 bases, and several computer generated designs of this gene have been synthesized and evaluated in studies that aimed to characterize codon and codon pair distribution effects in gene expression.[17,18] We randomly selected four amino acids, isoleusine (I), asparagine (N),

Table 2. GFP Design with 10 Overlapping Oligos, Varying 4 Codons (of 4 Different Amino Acids) at 4 Frequency Levels Each[a]

AA	fragments utilized	design
S	[(0, 3), (8, 8)]	{⟨0, 2⟩, ⟨0, 1⟩}
T	[(1, 2), (7, 9)]	{⟨0, 2⟩, ⟨0, 1⟩}
V	[(0, 0), (2, 7)]	{⟨0, 1⟩, ⟨0, 2⟩}
A	[(3, 4), (7, 9)]	{⟨0, 1⟩, ⟨0, 2⟩}

[a]The first column identifies the amino acid whose codon frequency we are varying, the second column depicts contiguous oligo sets that contain the appropriate Δ codon content multiple, and the third shows the resulting designs that minimize coverage.

Table 3. GFP Design Varying 8 Codons at 4 Frequency Levels Each

AA	fragments utilized	design
P	[(2, 3), (7, 8)]	{⟨0, 2⟩, ⟨0, 1⟩}
T	[(1, 2), (8, 9)]	{⟨0, 2⟩, ⟨0, 1⟩}
V	[(0, 2), (7, 9)]	{⟨0, 2⟩, ⟨0, 1⟩}
E	[(0, 0), (3, 5)]	{⟨0, 1⟩, ⟨0, 2⟩}
D	[(4, 5), (6, 9)]	{⟨0, 1⟩, ⟨0, 2⟩}
G	[(0, 2), (5, 6)]	{⟨0, 2⟩, ⟨0, 1⟩}
Q	[(3, 3), (7, 7)]	{⟨0, 1⟩, ⟨0, 2⟩}
K	[(3, 5), (6, 6)]	{⟨0, 2⟩, ⟨0, 1⟩}

proline (P), and threonine (T), and varied their codon frequencies. We set the oligo and overlap lengths to the same values as the GFP experiments, 90 bp and 18 bp, respectively. In the case of the P1 protein, 37 overlapping oligos are necessary to synthesize a single variant.

We performed four experiments, varying the range and steps of the four codon frequencies, using the same frequency pattern for all codons in each experiment:

(0,0.2, 0.4, 0.6, 0.8, 1)
(0.05, 0.23, 0.41, 0.59, 0.77, 0.95)
(0.1, 0.26, 0.42, 0.58, 0.074, 0.9)
(0.05, 0.35, 0.65, 0.95)

The resulting library designs are shown in Tables 4, 5, 6, and 7. If these libraries were constructed with independent synthesis of each gene variant, they would require 1296x coverage, which is equivalent to the synthesis of 47,952 oligos of 90 bp length, when sampling the four codons at six different frequency values. Libraries which sample the codons at four frequency values each would require 256x coverage, or equivalently 9,472 oligos.

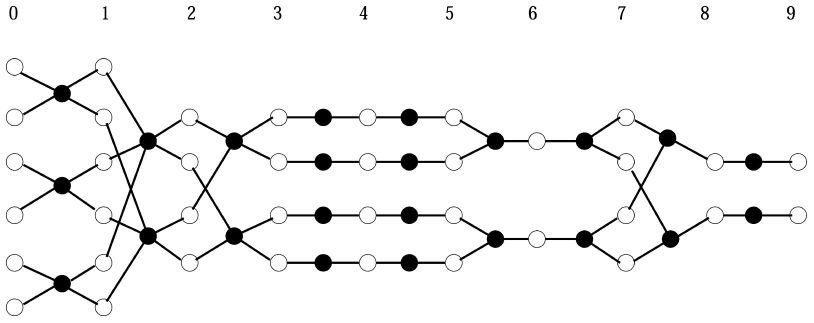


Figure 5. Three codon usage variation in GFP, amino acids T, V, and E. Four frequency values per codon of T and V, six frequency values for codon of E. 96 total gene variants.

dx.doi.org/10.1021/sb300086d | ACS Synth. Biol. 2013, 2, 237−244

Table 4. Polio P1 Design with 37 Overlapping Oligos, Varying 4 Codons at Frequency Levels (0, 0.2, 0.4, 0.6, 0.8, 1)

AA	fragments utilized	design
I	[(0, 21), (22, 34)]	{⟨0, 3⟩, ⟨0, 1, 2⟩}
N	[(0, 11), (12, 34)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}
P	[(1, 14), (15, 36)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}
T	[(0, 22), (24, 36)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}

Table 5. Polio P1 Design Varying 4 Codons at Frequency Levels (0.05, 0.23, 0.41, 0.59, 0.77, 0.95)

AA	fragments utilized	design
I	[(0, 13), (15, 32)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}
N	[(0, 10), (10, 30)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}
P	[(4, 20), (22, 36)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}
T	[(2, 22), (24, 36)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}

Table 6. Polio P1 Design Varying 4 Codons at Frequency Levels (0.1, 0.26, 0.42, 0.58, 0.074, 0.9)

AA	fragments utilized	design
I	[(11, 18), (20, 35)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}
N	[(0, 15), (29, 34)]	{⟨0, 2, 4⟩, ⟨0, 1⟩}
P	[(9, 22), (25, 36)]	{⟨0, 3⟩, ⟨0, 1, 2⟩}
T	[(0, 11), (17, 31)]	{⟨0, 1, 2⟩, ⟨0, 3⟩}

Table 7. Polio P1 Design Varying 4 Codons at Frequency Levels (0.05, 0.35, 0.65, 0.95)

AA	fragments utilized	design
I	[(2, 13), (15, 34)]	{⟨0, 1⟩, ⟨0, 2⟩}
N	[(0, 14), (23, 34)]	{⟨0, 2⟩, ⟨0, 1⟩}
P	[(4, 22), (25, 36)]	{⟨0, 2⟩, ⟨0, 1⟩}
T	[(0, 11), (13, 31)]	{⟨0, 1⟩, ⟨0, 2⟩}

The coverage necessary to synthesize the optimized combinatorial libraries produced by our method for the four experiments are 29.84x, 23.95x, 12.81x, and 10.92x, respectively, while the corresponding savings over the nonoptimized libraries are 43.4-fold, 54.11-fold, 101.17-fold, and 23.44-fold.

Conclusions. We have developed a methodology and implemented an algorithm to design protein coding gene libraries that uniformly vary codon usage of multiple amino acids. Our novel methods enable the cost-effective exploration of the codon utilization design space and take advantage of modern synthesis technologies to assemble DNA libraries for use in high-throughput in vitro and in vivo experimentation. Although primarily described as a codon usage exploration tool, our methodology can be readily applied to vary other globally acting coding and noncoding sequence features, such as nucleotide composition, CG and CpG content, codon context (such as codon pair and codon next base biases), and pattern incorporation and/or elimination.

The algorithm has been implemented in python and all computational experiments were performed on a laptop machine with an i3 Intel processor. The code is available and can be obtained from the authors.

METHODS

Algorithm Description. Our optimization method aims to create combinatorial designs that cover the design space of a protein coding gene library in a one-to-one manner, requiring the synthesis of as few oligos as possible. The one-to-one covering condition is necessary to guarantee that the resulting library of gene variants from the assembly process will not be heavily skewed toward any particular design.

For each codon of interest c_i, we need to allocate a subset of oligos that contain at least $(n_i - 1)\Delta_i$'s worth of occurrences of a_i to create the variation needed for the design space of a_i. We can then assign $\min_i$ worth of c_i's to the a_i's found outside of the allocated subset and then systematically vary the usage of c_i in Δ_i-sized steps in the region we have allocated to create variation in the use of c_i. Define an *oligo interval* to be a contiguous set of oligos. The allocated region can be made of one large oligo interval or multiple smaller oligo intervals. However, in the case of multiple oligo intervals, we will need to examine which of the possible allocations of the Δ_i's to each oligo interval maintains a one-to-one covering of the amino acid's design space. Since we are only interested in creating the Δ_i-sized steps of variation in the allocated region, we can think of this space as

$$\langle 0, 1, ..., n_i - 1 \rangle \quad (2)$$

where each integer represents the number of Δ_i's.

Now, consider what happens when multiple oligo intervals are utilized for a single codon's variation. Each oligo interval will be assigned multiple levels of c_i usage in a way that each level of variation is equally likely to occur under simultaneous assembly. This is achieved by ensuring that the interior overlap regions of the oligo interval are distinct between each version of the interval but the outer overlap regions are the same in each version. See Figure 3 for a diagram depicting two oligo intervals, (1,2) and (5,8), each with two distinct versions. To maintain a one-to-one covering of the design space, the size of a_i's design space should be equal to the product of the number of levels of variation in each oligo interval. Hence, the number of options in each oligo interval must be a factorization of n_i. For example, if $n_i = 6$, we could:

1. put all Δ_i's into a single contiguous region with options ⟨0, 1, 2, 3, 4, 5⟩ or,
2. split them among two oligo intervals with one having options ⟨0, 1, 2⟩ and the other having options ⟨0, 3⟩ or,
3. split them among two oligo intervals with one having options ⟨0, 1⟩ and the other having options ⟨0, 2, 4⟩.

No other way of allocating the Δ_i's to disjoint oligo intervals will yield a one-to-one covering of the amino acid's design space. We will refer to each collection of options as a *component* and a set of components that cover the design space by choosing one option from each component as a *design*. We showed above that {⟨0, 1, 2, 3, 4, 5⟩}, {⟨0, 1, 2⟩,⟨0, 3⟩}, and {⟨0, 1⟩,⟨0, 2, 4⟩} are the possible designs for $n_i = 6$.

With this terminology, allocating variation for a given design is equivalent to assigning nonoverlapping oligo intervals to each component of the design such that each oligo interval has sufficient occurrences of a_i for the corresponding component. We will call such an assignment an *oligo-design assignment*.

Let us compare options 1 and 2 above. For the purpose of minimizing coverage, option 2 is superior, because the total number of oligos allocated to the design will be smaller than or equal to the number of oligos allocated by option 1. This follows the reasoning that, as long as the codon allocation in option 1 spans multiple oligos, one should be able to split the set at an appropriate point to yield an option 2 type allocation. In addition, option 2 has the added benefit of lower average coverage. In option 1, all allocated oligos have 6x coverage, whereas in option 2 the oligos have either 2x or 3x coverage.

dx.doi.org/10.1021/sb300086d | ACS Synth. Biol. 2013, 2, 237−244

This observation is generalized and formalized at the end of the section with a theorem.

Comparing options 2 and 3 above, we see that the difference is characterized by whether the oligo interval with higher Δ_i content gets assigned the larger (option 3) or smaller (option 2) factor of n_i. We would expect the region with larger Δ_i content to require an allocation of more oligos, so it would be better if this region had smaller coverage. Unfortunately there are examples where this is not true, so one has to consider all possible assignments of Δ_i's to regions corresponding to the distinct prime factorizations of n_i in order to identify the optimal design.

Suppose $n_i = p_1 \cdot, \ldots, \cdot p_{q_i}$ with p_j prime for $1 \leq j \leq q_i$ (but not necessarily distinct). We now describe a way to formulate the design with q_i components corresponding to this factorization. Let the component corresponding to p_1 be $\langle 0, 1, \ldots, p_1 - 1\rangle$, the second component be $\langle 0, p_1, \ldots, p_1(p_2 - 1)\rangle$, the third $\langle 0, p_1p_2, \ldots, p_1p_2(p_3 - 1)\rangle$, continuing up to the q_ith component $\langle 0, p_1 \cdot \ldots \cdot p_{q_i-1}, \ldots, p_1 \cdot \ldots \cdot p_{q_i-1}(p_{q_i} - 1)\rangle$. This construct is similar to generating natural numbers using a hybrid system of representation, where each digit uses a prime number of values. The difference of the number of Δ_i's between options in a component we will call the *order* o_i of the component, with $o_1 = 1$ and $o_i = p_1p_2 \ldots p_{i-1}$. As an example, the order of design $\langle 0, 9, 18\rangle$ is 9.

To illustrate with an example, consider $n_i = 12$. There are three distinct prime factorizations, $2 \cdot 2 \cdot 3$, $2 \cdot 3 \cdot 2$, and $3 \cdot 2 \cdot 2$, that yield three different fully factored designs: $\{\langle 0,1\rangle, \langle 0,2\rangle, \langle 0,4,8\rangle\}$, $\{\langle 0,1\rangle, \langle 0,2,4\rangle, \langle 0,6\rangle\}$ and $\{\langle 0,1,2\rangle, \langle 0,3\rangle, \langle 0,6\rangle\}$ respectively. Any of these might potentially produce an optimal total design for a particular amino acid sequence.

In order to synthesize the full design space, we must choose a oligo-design assignment for each amino acid of interest, a_i. Each oligo-design assignment will specify a coverage for each oligo of the amino acid chain, and the total coverage needed for each oligo will be the product of the coverages from the oligo-design assignments. The objective of the assignment algorithm will be to minimize the total number of oligos needed to synthesize the design space, i.e. the sum of the total coverage of the individual oligos.

With the framework set forth above our algorithm is described as follows:

1. Divide the amino acid chain into oligos of size l specified by the user, which we will label $s_1, s_2, \ldots, s_N$.

2. For each interior and overlap region of the oligos count the occurrences of a_i and compute the corresponding number of Δ_i's (this value may be a noninteger) for each amino acid.

3. For each amino acid a_i and each fully factored design for n_i, compute all minimal oligo intervals containing at least the maximum number of Δ_i's needed for each component of the design. An oligo interval is considered minimal if it cannot be contracted without dropping below the required number of Δ_i's. For example, for the design $\{\langle 0, 1, 2\rangle, \langle 0, 3\rangle\}$ we need to compute all minimal oligo intervals that contain at least $3\Delta_i$'s worth of a_i to create the $\langle 0, 3\rangle$ component and we also need to compute all minimal oligo intervals that contain at least $2\Delta_i$'s to create the $\langle 0, 1, 2\rangle$ component. Note that an a_i is only considered to be part of the oligo interval if it is in or between the interior of oligo i and the interior of oligo j and not in the outer overlap regions.

4. For each design type for a_i, compute all oligo-design assignments by taking all disjoint combinations of the minimal oligo intervals found in step 3, one from each component type. The notation for an oligo-design assignment will take the form $[(i_1, j_1), (i_2, j_2)] \rightarrow \{\langle comp_1\rangle, \langle comp_2\rangle\}$ to indicate an oligo-design assignment that assigns the oligo interval (i_1, j_1) to the component $\langle comp_1\rangle$ and the oligo interval (i_2, j_2) to the component $\langle comp_2\rangle$.

5. Now that all oligo-design assignments are identified for each amino acid of interest, we only need to find which yields the smallest number of overall oligos to synthesize. This reduces to scoring every possible combination of oligo-design assignments, one taken from each amino acid's list of oligo-design assignments. The scoring can be computed by taking a component-wise product of the coverage of each oligo for each amino acid's oligo-design assignment and summing the resulting N-vector. As a simple example suppose we have $N = 5$ and only two amino acids of interest, a_1 and a_2. Suppose the oligo-design assignment for a_1 is $[(1, 2), (4, 4)] \rightarrow \{\langle 0, 1, 2\rangle, \langle 0, 3\rangle\}$ and the oligo-design assignment for a_2 is $[(2, 3)] \rightarrow \{\langle 0, 1, 2\rangle\}$. Then the total number of oligos required to use these two oligo-design assignments would be $3 + (3 \cdot 3) + 3 + 2 + 1 = 18$ because s_1 requires 3 variations from a_1, s_2 requires 3 variations from a_1 and 3 variations from a_2 hence 9 variations in total, s_3 requires 3 variations from a_2, s_4 requires 2 variations from a_1, and s_5 is not part of either oligo-design assignment, so only one copy of it needs to be synthesized. After scoring all possible combinations of oligo-design assignments in this manner, the combination with the lowest score will be used to generate the optimal library.

6. Using the optimal combination of oligo-design assignments, one from each amino acid, assign codons c_i appropriately, starting with designating a $\min_i$ proportion of the a_i's not in the oligo-design assignment, and then creating the varied oligos based on the components of the optimal total design. Some rounding might be required as the integer multiples of Δ_i may not correspond to integer multiples of a_i's depending on total number of occurrences of a_i in the sequence. Assign the remainder of the codons for a_i according to a user specified distribution (which typically follows either original gene distribution or target host codon distribution, the latter when optimizing heterologous gene expression).

7. Validate the uniqueness of all overlaps, swapping synonymous codons between overlaps and regions outside of the allocated subset for the corresponding amino acids, when nonunique overlaps are found.

We conclude this subsection with the proof of the aforementioned theorem:

Theorem 1. *The number of oligos necessary to assemble a library of single codon frequency gene variants is minimized when the number of varied contiguous regions of oligos is equal to the number of prime factors of n_i and the number of options per component is equal to the corresponding prime factor.*

Proof. If a design has a component of order o that has a number of options $t \in \mathbb{N}$ that is not prime, then $t = p_1 \times p_2$, where $p_1, p_2 \in \mathbb{N}$. This component can be decomposed into two components by splitting the corresponding oligo interval into two disjoint oligo intervals, each containing an appropriate number of Δ_i's that, based on the assignment procedure described above, can be assembled to generate all codon frequencies in the original component. To find the division point for the two oligo intervals, we start at the left of the original interval and proceed until we reach the codon that completes the necessary Δ of the decomposition. If the last codon of the first part of the split and the first codon of the

dx.doi.org/10.1021/sb300086d | ACS Synth. Biol. 2013, 2, 237−244

second half of the split are not in the same overlap, then we need p_1 coverage in the first section (for $\langle 0, 1{\cdot}o, ..., p_1{\cdot}o\rangle$) and p_2 coverage in the second half (for $\langle 0, p_1{\cdot}o, 2{\cdot}p_1{\cdot}o, ..., (p_2 - 1){\cdot}p_1{\cdot}o\rangle$). If there exists a shared overlap, then the oligos immediately on either side of the overlap need $p_1{\cdot}p_2$ coverage, where the rest of the sections are structured as described above. Since $\max(p_1, p_2) \leq p_1{\cdot}p_2$, no oligo after the split will require higher coverage than before the split.

Runtime Analysis. We will consider the size of the input to be the length of the amino acid chain m, which is a multiple of the total number of oligos N. Since the oligo size l and overlap size v are bounded by constants (determined by the method used), the only variable in our input is m, and consequently N. The final loop of combining individual codon designs in our algorithm is by far the part of largest computational complexity, where the score is computed for every combination of oligo-design assignments. Although the asymptotic complexity increases exponentially for each new codon that is processed, the total number of codons that can be varied is bounded by a small constant. The number of oligo-design assignment combinations is equal to

$$\prod_{i=1}^{k} (\text{no. of oligo-design assignments for } a_i)$$

Suppose $n_i = p_1{\cdot}...{\cdot}p_{q_i}$ where the p's are prime numbers. There are $q_i!$ ways to order the primes, resulting in up to $q_i!$ fully factored designs for a_i. Each design has q_i components and each component could have up to $O(N)$ different minimal oligo intervals. The algorithm proceeds by processing all disjoint combinations of these minimal oligo intervals, taking one from each component type. In the worst case, this step requires $O(q_i!N^{q_i})$ time. If each amino acid has $O(q_i!N^{q_i})$ oligo-design assignments, then the number of oligo-design assignment combinations is $O(N^{q_1+\cdots+q_k}(\prod_{i=1}^{k} q_i!))$. The final $O(N)$ step of computing the score for every oligo-design assignment combination yields an upper bound of $O(N^{1+q_1+\cdots+q_k}(\prod_{i=1}^{k} q_i!))$, which can be a large polynomial but only involves very small constants and limited size input.

In practice, a python implementation of the algorithm, processing inputs of medium size proteins and generating designs varying 8 codons at 4 frequency levels each, results in computations taking in the order of a few minutes to complete in moderate hardware (laptop computer with Intel i3 processor). The space complexity of the algorithm is linear as a function of the input size.

AUTHOR INFORMATION

Corresponding Author

* E-mail: dimitris@cs.miami.edu.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their comments and suggestions, which were helpful in improving the manuscript.

REFERENCES

(1) Bugl, H., Danner, J. P., Molinari, R. J., Mulligan, J. T., Park, H. O., Reichert, B., Roth, D. A., Wagner, R., Budowle, B., Scripp, R. M., Smith, J. A., Steele, S. J., Church, G., and Endy, D. (2007) *Nat. Biotechnol. 25*, 627−9.

(2) Czar, M. J., Anderson, J. C., Bader, J. S., and Peccoud, J. (2009) *Trends Biotechno.l 27*, 63−72.

(3) Tian, J., Gong, H., Sheng, N., Zhou, X., Gulari, E., Gao, X., and Church, G. (2004) *Nature 432*, 1050−4.

(4) Welch, M., Govindarajan, S., Ness, J. E., Villalobos, A., Gurney, A., Minshull, J., and Gustafsson, C. (2009) *PLoS One 4*, No. e7002.

(5) Kudla, G., Murray, A. W., Tollervey, D., and Plotkin, J. B. (2009) *Science 324*, 255−8.

(6) Supek, F., and Muc, T. (2010) *Genetics 185*, 1129−34.

(7) Plotkin, J. B., and Kudla, G. (2011) *Nat. Rev. Genet. 12*, 32−42.

(8) Stemmer, W. P., Crameri, A., Ha, K. D., Brennan, T. M., and Heyneker, H. L. (1995) *Gene 164*, 49−53.

(9) Cello, J., Paul, A. V., and Wimmer, E. (2002) *Science 297*, 1016−8.

(10) Smith, H. O., Hutchison, r., C. A., Pfannkoch, C., and Venter, J. C. (2003) *Proc. Natl. Acad. Sci. U.S.A. 100*, 15440−5.

(11) Gao, X., Yo, P., Keith, A., Ragan, T. J., and Harris, T. K. (2003) *Nucleic Acids Res. 31*, e143.

(12) Gibson, D. G., Young, L., Chuang, R. Y., Venter, J. C., Hutchison, r., C. A., and Smith, H. O. (2009) *Nat. Methods 6*, 343−5.

(13) Gibson, D. G., Smith, H. O., Hutchison, r., C. A., Venter, J. C., and Merryman, C. (2010) *Nat. Methods 7*, 901−3.

(14) Gibson, D. G., et al. (2008) *Science 319*, 1215−20.

(15) Ramon, A., and Smith, H. O. (2011) *Biotechnol. Lett. 33*, 549−55.

(16) Beck, M., and Robins, S. (2006) *Computing the Continuous Discretely*, pp 3−23, Springer, New York.

(17) Mueller, S., Papamichail, D., Coleman, J. R., Skiena, S., and Wimmer, E. (2006) *J. Virol. 80*, 9687−96.

(18) Coleman, J. R., Papamichail, D., Skiena, S., Futcher, B., Wimmer, E., and Mueller, S. (2008) *Science 320*, 1784−7.

(19) Li, G. W., Oh, E., and Weissman, J. S. (2012) *Nature 484*, 538−41.