Journal of Discrete Algorithms 16 (2012) 104–112

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Efficient codon optimization with motif engineering ☆

Anne Condon *, Chris Thachuk

Department of Computer Science, University of British Columbia, Vancouver, BC, V6T 1Z4, Canada

ARTICLE INFO

Article history:
Available online 3 May 2012

Keywords:
Codon optimization
Dynamic programming
Sequence design
Motif engineering

ABSTRACT

It is now common to add synthetic protein coding genes into cloning vectors for expression within non-native host organisms. Codon optimization is the task of choosing a sequence of codons that specify a protein so that the chosen codons are those used with the highest possible frequency in the host genome, subject to certain constraints, such as ensuring that occurrences of pre-specified "forbidden" motifs are minimized. Codon optimization supports translational efficiency of the desired protein product, by exchanging codons which are rarely found in the host organism with more frequently observed codons. Motif engineering, such as removal of restriction enzyme recognition sites or addition of immuno-stimulatory elements, is also often necessary. We present an algorithm for optimizing codon bias of a gene with respect to a well motivated measure of bias, while simultaneously performing motif engineering. The measure is the previously studied codon adaptation index, which favors the use, in the gene to be optimized, of the most abundant codons found in the host genome. We demonstrate the efficiency and effectiveness of our algorithm on the GENCODE dataset and provide a guarantee that the solution found is always optimal. The implementation and source code of our algorithm are freely accessible at http://www.cs.ubc.ca/labs/beta/Projects/codon-optimizer.

1. Introduction

Gene synthesis is now an economical and technically viable option for the construction of non-natural genes. Synthetic genes can be novel or derivatives of those found in nature. In either case, the expression levels of these genes, when inserted into the genome of a host organism, depend on many factors. One important factor is the bias of codon usage, relative to the host organism [8,11,18]. Note that for each amino acid in a protein, there may be many (up to six) valid codons, as given by the genetic code. Loosely speaking, the codon bias of a gene for the protein measures how well—or poorly—codons used in the gene match codon usage in the genome of a host organism (we describe specific measures later in this paper). In a study by Lithwick and Margalit [12] on the effects of sequence-dependent features associated with prokaryotic translation, the authors concluded that codon bias is the feature most highly associated with the level of protein expression. It was demonstrated by Kane [11] that usage of rare codons, especially when clustered, is detrimental to protein expression levels [11]. Gao et al. [4] noted that a key obstacle to DNA-based vaccines for the human immuno-deficiency virus (HIV) is low expression levels of HIV genes in mammalian cells, which they attribute to rare codon usage and AU-rich elements. These studies indicate that it is desirable to choose codons that have high usage in a host's genome, in order to ensure that designed genes are maximally expressed within the host.

☆ A preliminary version of this work appeared in the Proceedings of the 2011 International Workshop on Combinatorial Algorithms (IWOCA 2011) (Condon and Thachuk, 2011 [2]).

* Corresponding author.
E-mail addresses: condon@cs.ubc.ca (A. Condon), cthachuk@cs.ubc.ca (C. Thachuk).

1570-8667/$ – see front matter
http://dx.doi.org/10.1016/j.jda.2012.04.017

In addition to optimizing the codon bias of a gene, relative to the genome of a host, it is often desirable or necessary to add or remove certain motifs, i.e., subsequences, via silent mutation: altering the DNA sequence, provided the amino acid sequence it encodes remains unchanged. This is common practice and important for the elimination of restriction enzyme recognition sites of a host organism [7,16]—or inclusion of these elements for diagnostic purposes—and removal of polyhomomeric repeat regions [19]. Exclusion of these elements can be seen as a hard constraint. Yet, in other instances, removal or addition of motifs can be treated more naturally as optimization criteria to be minimized or maximized. This is the case, for instance, with immuno-regulatory CpG motifs in mammalian expression vectors [14], where it is desirable to minimize immuno-inhibitory elements and maximize immuno-stimulatory motifs. In the remainder of this work, we will refer to inclusion or exclusion of motifs, via silent mutation, as motif engineering.

A number of published software tools are capable of codon optimization, i.e., of choosing a sequence of codons that specify a protein so that the product of the frequencies of the chosen codons in the host genome is as high as possible, subject to certain constraints such as exclusion or inclusion of certain subsequences. These tools include DNA Works [9], Codon Optimizer [3], GeMS [10], Gene Designer [19], JCat [5], OPTIMIZER [13], the Synthetic Gene Designer [20], UpGene [4] and a method by Satya et al. [14]. Some of these methods also consider the other problem considered here, motif engineering. Of these, only the method of Satya et al. provides a mathematical guarantee of finding an optimal solution to codon optimization when one exists. However, their method—based on the graph theoretic approach of finding a critical path—runs in $O(n^2)$ time and space, where n is the length of the DNA sequence being optimized. Skiena [16] presents an efficient algorithm for minimizing forbidden motifs when choosing a sequence of codons that specify a protein, assuming that the length of forbidden sequences is bounded by a constant. Skiena also shows that a decision version of the problem is NP-complete when there is no bound on the length of forbidden sequences. However, Skiena's work does not address codon optimization. In this work, we propose the first linear time and space codon optimization algorithm that is guaranteed to find an optimal solution and that also satisfies motif engineering constraints. We have focused our attention on optimizing codon usage according to the Codon Adaptation Index (CAI). The index, originally proposed by Sharp and Li [15], is based on the premise of each amino acid having a 'best' codon for a particular organism. This perspective evolved from the observation that protein expression is higher in genes using codons of high fitness and lower in genes using rare codons [7]. It is believed that this is due to the relative availability of tRNAs within a cell.

We also provide an experimental study of the performance of our algorithm on a biological data set comprising 3157 coding sequence regions of the GENECODE subset of the ENCODE dataset [17]. The implementation and source code of our algorithm are freely accessible at http://www.cs.ubc.ca/labs/beta/Projects/codon-optimizer.

The remainder of this paper is structured as follows. In the Preliminaries section, we formally define the problem of codon optimization. We detail the general objectives of the problem, and formalize the goals of motif engineering. We then present our algorithm, providing a proof of correctness and time and space analysis. In the Empirical Results section, we describe the performance of our algorithm, both in terms of run-time efficiency and also in terms of the quality of optimization achieved. Finally, we conclude with a summary of our major findings and directions for future work.

2. Preliminaries

A DNA strand is a string over the alphabet of DNA. A *codon* is a triple over the DNA alphabet and therefore there are at most $4^3 = 64$ distinct codons. An *amino acid sequence* is a string over the alphabet of amino acids, $\Sigma_{AA} = \{Ala, Arg, Asn, Asp, Cys, Glu, Gln, Gly, His, Ile, Leu, Lys, Met, Phe, Pro, Ser, Thr, Trp, Tyr, Val, stop\}$ with each symbol representing an amino acid and the special symbol '*stop*' denoting a string terminal. We assume there is a predetermined ordering of amino acids, for example, lexicographic.

Therefore, we can represent an amino acid sequence A as a sequence of integers, with $A = \alpha_1, \alpha_2, \ldots, \alpha_{|A|}$, where $1 \leqslant \alpha_k \leqslant 21$, for $1 \leqslant k \leqslant |A|$. For example, the amino acid sequence of the problem instance in Fig. 1 can be represented as $A = 19, 11, 1, 19$. The *genetic code* is a mapping between amino acids and codons. However, as there are 64 possible codons and only 20 amino acids (plus one stop symbol), the code is degenerate, resulting in a one-to-many mapping from each amino acid to a set of corresponding codons.

The process of gene translation can be thought of more naturally as a mapping of codons to amino acids, however, we define our mapping as the inverse for convenience. We denote the set of codons for the ith amino acid by $\lambda(i)$. Therefore, for the first amino acid, namely `Ala`, $\lambda(1) = \{$`GCA`, `GCC`, `GCG`, `GCU`$\}$. Similarly, the second amino acid is `Arg` and $\lambda(2) = \{$`AGA`, `AGG`, `CGA`, `CGC`, `CGG`, `CGT`$\}$, and so on. We let $\lambda_j(i)$ be the jth codon in the set $\lambda(i)$, $1 \leqslant j \leqslant |\lambda(i)|$, where again we use lexicographic ordering. Therefore, we can define a DNA encoding for an amino acid sequence as a sequence of codon indices. Again consider the problem instance in Fig. 1. For the Leucine amino acid (`Leu`) which is the 11th amino acid in lexicographic order, $|\lambda(11)| = 6$ and $\lambda_3(11)$ is the codon `CUG`. The DNA sequence `UAC CUC GCC UAC` can be represented by the codon index sequence $S = 1, 2, 2, 1$; that is, $\lambda_1(19)\lambda_2(11)\lambda_2(1)\lambda_1(19) =$ `UAC CUC GCC UAC`. We refer to S as the *codon design*.

A *codon's frequency* is the number of times that it appears in nature, divided by the total number of times that all codons corresponding to the same amino acid appear in nature. By "in nature", we mean codon frequencies present in some reference sequence or set of sequences such as a genome or set of genomes. As an example, if for some amino acid index i, $|\lambda(i)| = 2$, and the codon $\lambda_1(i)$ is observed 37 times in nature, while $\lambda_2(i)$ is observed 63 times, we can define the frequency of $\lambda_1(i)$ to be $\frac{37}{37+63} = 0.37$. Let $\rho_j(i)$ denote the frequency of the jth codon of the ith amino acid, $1 \leqslant j \leqslant |\lambda(i)|$.

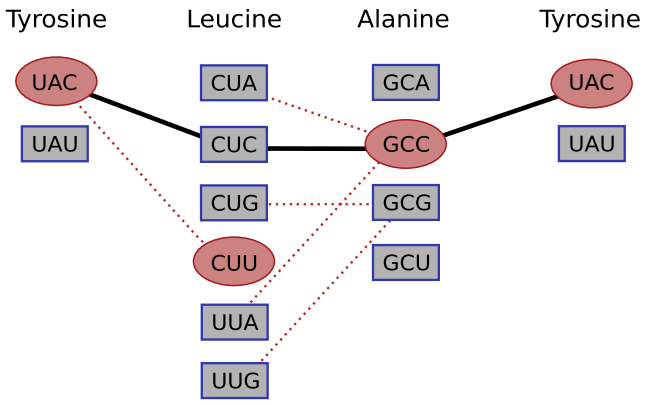

Fig. 1. An instance consisting of four amino acids and a forbidden set $\mathcal{F} = \{\text{CCUU}, \text{AGC}, \text{UGGC}\}$. Best codons are indicated as red ovals. For this example, a forbidden motif can span at most two codons. A valid codon assignment for this instance must contain no occurrence of a forbidden motif. Leucine has six corresponding codons. However, if UAC is the codon chosen for the first Tyrosine, then the codon CUU for Leucine, a desired motif, cannot be chosen because the concatenation of UAC and CUU includes the forbidden motif CCUU. This and three additional concatenations of codons that are not valid are indicated by dotted red lines. The path connected by bold black lines denotes an assignment containing the best codons UAC, GCC and UAC. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Note that $\sum_{j=1}^{|\lambda(i)|} \rho_j(i) = 1.0$, for any i, assuming that the ith codon is in the reference set. In the example above, we say that $\lambda_2(i)$ is the *most frequent codon*. Note that it is possible for more than one codon to have this property.

A *codon's fitness* is the number of times that it appears in nature, divided by the number of occurrences of the corresponding most frequent codon for the same amino acid (originally referred to as the relative adaptiveness of a codon [15]). Let $\tau_j(i)$ denote the fitness value of the jth codon of the ith amino acid. Returning to our previous example, if the ith amino acid has two codons with frequencies $\rho_1(i) = 0.37$ and $\rho_2(i) = 0.63$, then their fitness values, denoted by $\tau_1(i)$ and $\tau_2(i)$ respectively, are $0.37/0.63 \approx 0.59$ and $0.63/0.63 = 1.0$. Note that a most frequent codon will always have a fitness value of 1.0.

2.1. Motif engineering

We focus our attention on designing codon sequences which minimize occurrences of forbidden motifs from a predetermined set, $\mathcal{F}$, while maximizing occurrences of desired motifs from a predetermined set, $\mathcal{D}$. A codon design—a sequence of codon assignments—is said to be *valid* with respect to an amino acid sequence it codes for if it satisfies the following constraints, in order: the DNA sequence corresponding to the codon design (1) contains the minimum possible number of forbidden motifs, and (2) contains the maximum possible number of desired motifs, given that (1) is satisfied. Here, (1) is minimized with respect to all occurrences of forbidden motifs, whether or not they span multiple codons, and similarly (2) is maximized with respect to desired motifs regardless of whether or not they span multiple codons. It is important to recognize that a valid design does not necessarily guarantee that the number of occurrences of desired motifs is the maximum number possible, of all possible codon designs.

Again, consider the problem instance of Fig. 1. The bold path represents the valid design containing no forbidden motifs and no desired motifs that has the highest codon adaptation index score (discussed in the next section). Best codons for each amino acid are shaded in red. In this instance, selecting all four best codons would result in an invalid design that contains a forbidden motif.

We now develop some notation for motif engineering. For a sequence of amino acid indices $A = \alpha_1, \alpha_2, \ldots, \alpha_{|A|}$, a corresponding codon design $S = c_1, c_2, \ldots, c_{|A|}$, a set of forbidden motifs $\mathcal{F}$ and a set of desired motifs $\mathcal{D}$, let $M_{\mathcal{F}}(\lambda_{c_i}(\alpha_i)\ldots\lambda_{c_j}(\alpha_j))$ and $M_{\mathcal{D}}(\lambda_{c_i}(\alpha_i)\ldots\lambda_{c_j}(\alpha_j))$ be the number of occurrences of forbidden motifs and desired motifs, respectively, in the DNA sequence $\lambda_{c_i}(\alpha_i)\ldots\lambda_{c_j}(\alpha_j)$, where $j \geqslant i$. For convenience in our algorithms, we also introduce $M'_{\mathcal{F}}(\lambda_{c_i}(\alpha_i)\ldots\lambda_{c_j}(\alpha_j))$ and $M'_{\mathcal{D}}(\lambda_{c_i}(\alpha_i)\ldots\lambda_{c_j}(\alpha_j))$ which respectively determine the number of forbidden and desired motifs in $\lambda_{c_i}(\alpha_i)\ldots\lambda_{c_j}(\alpha_j)$ that end within the last codon position (the last 3 bases), here indexed by j. For instance consider the codon design $S = 1, 4, 2, 1$ of the problem instance in Fig. 1. $M_{\mathcal{F}}(\text{UACCUUGCCUAC}) = 1$ as it contains the forbidden motif CCUU. However, $M'_{\mathcal{F}}(\text{UACCUUGCCUAC}) = 0$ as no forbidden motif ends within the last codon.

In practice, forbidden and desired motifs are short and we assume their length is bounded by a constant, g [14].

Observation 1. If the largest forbidden or desired motif is of length g, then any forbidden or desired motif can span at most $k + 1$ consecutive codons, where $k = \lceil g/3 \rceil$.

2.2. Codon optimization

We now review a measure commonly employed for codon optimization and the one we optimize in our algorithm. The codon adaptation index (CAI), originally proposed by Sharp and Li [15], is based on the premise of each amino acid having a 'best' codon for a particular organism. This perspective evolved from the observation that protein expression is higher in genes using codons of high fitness and lower in genes using rare codons [7].

The 'best' codon, referred to above, is the most frequent codon and therefore, by definition, has a fitness value of 1.0. For some codon design $S = c_1, c_2, \ldots, c_{|A|}$, which correctly codes for a desired amino acid sequence $A = \alpha_1, \alpha_2, \ldots, \alpha_{|A|}$, the CAI value for S with respect to A, $\mathrm{CAI}(S, A)$, can be calculated as in Eq. (1). Based on this definition, if S consists only of most frequent codons, it would have a CAI value of 1.0. Intuitively, the higher the CAI value, the better.

$$\mathrm{CAI}(S, A) = \left(\prod_{i=1}^{|A|} \tau_{c_i}(\alpha_i) \right)^{\frac{1}{|A|}} \tag{1}$$

With the previously defined definitions, notation, and optimization criteria, we now formally define the problem of codon optimization with motif engineering.

2.3. The CAI codon optimization problem with motif engineering

Instance: Amino acid sequence represented by the sequence of indices $A = \alpha_1, \alpha_2, \ldots, \alpha_{|A|}$, a set of forbidden motifs $\mathcal{F}$, a set of desired motifs $\mathcal{D}$, and a fitness value in the range $[0, 1]$ for each codon.

Problem: Find a codon design S^*, with $|S^*| = |A|$, corresponding to A such that S^* is *valid* with respect to $\mathcal{F}$ and $\mathcal{D}$, and $\mathrm{CAI}(S^*, A) = \max\{\mathrm{CAI}(S, A) \mid S \in \mathbf{S}(A)\}$, where $\mathbf{S}(A)$ is the set of all valid codon designs corresponding to A. S^* is an *optimal codon design with respect to the CAI measure*.

3. A DP algorithm for CAI optimization

We now propose a linear time and space dynamic programming algorithm guaranteed to maximize the CAI measure, such that the codon design is *valid*. In terms of efficiency, this is a direct improvement in both run-time and space over the current state-of-the-art, previously proposed by Satya et al. [14]. Although we have chosen to first ensure forbidden motifs are minimized, then desired motifs maximized and finally the CAI value maximized, it should be clear that the algorithm we present can be adapted to optimize these criteria in any order.

One necessary feature of a codon optimization algorithm is an efficient means to detect if a forbidden motif from $\mathcal{F}$, or a desired motif from $\mathcal{D}$, is present in a potential design. For both algorithms proposed in this work, we utilize an Aho–Corasick search for this purpose. Briefly, the Aho–Corasick algorithm builds a keyword tree (trie) for $\mathcal{F}$ and transforms the structure into an automaton with the addition of failure links. Space and time complexity for building the initial structure is $O(h)$, where h is the sum of the lengths of the motifs in $\mathcal{F}$. Queries to determine if a sequence b contains any forbidden motif take $O(|b|)$ time [1]. Likewise, a second tree is constructed for the desired motifs in $\mathcal{D}$. For a detailed description of the algorithm and existing applications of its use in computational biology, see Gusfield [6]. Satya et al. [14] use the same approach for motif detection in their $\theta(n^2)$ algorithm. We note that any dictionary matching algorithm can be employed for the same task; however, Aho–Corasick automata were chosen due to their simpler implementation.

We first define three quantities that will be important in describing our algorithm. The first quantity, $F^i_{c_{i-k+1},\ldots,c_{i-1},c_i}$, denotes the minimum possible number of forbidden motifs in a DNA sequence which codes for an amino acid sequence $A = \alpha_1, \alpha_2, \ldots, \alpha_i$, given that the last k codons (of i total codons) have indices denoted as $c_{i-k+1}, \ldots, c_{i-1}, c_i$. Similarly, the second quantity, $D^i_{c_{i-k+1},\ldots,c_{i-1},c_i}$, denotes the maximum possible number of desired motifs, among those sequences which contain a minimum number of forbidden motifs. $P^i_{c_{i-k+1},\ldots,c_{i-1},c_i}$ denotes the maximum possible CAI score among all valid sequences.

Our algorithm stores a k-dimensional entry for each position i, $k \leqslant i \leqslant |A|$, of the input amino acid sequence, where $k = \lceil g/3 \rceil$ and g is the longest forbidden or desired motif. The base case occurs when $i = k$ and is computed as follows. Every combination of codons for the first k amino acids is evaluated to determine, independently, the number of forbidden and desired motifs fully contained within the k consecutive codons (Eq. (2) and Eq. (3), respectively) and the CAI value (Eq. (4)).

$$F^k_{c_1,\ldots,c_k} = M_{\mathcal{F}}\big(\lambda_{c_1}(\alpha_1) \ldots \lambda_{c_k}(\alpha_k)\big) \tag{2}$$

$$D^k_{c_1,\ldots,c_k} = M_{\mathcal{D}}\big(\lambda_{c_1}(\alpha_1) \ldots \lambda_{c_k}(\alpha_k)\big) \tag{3}$$

$$P^k_{c_1,\ldots,c_k} = \prod_{i=1}^{k} \big(\tau_{c_i}(\alpha_i)\big) \tag{4}$$

The recursive case occurs for $i > k$. By Observation 1, a forbidden motif could span $k+1$ codons. Therefore, it is necessary to evaluate the last $k+1$ codons of a potential design to ensure codons are selected which 1) minimize forbidden motifs, then 2) maximize desired motifs, then 3) maximize the CAI score.

For any arbitrary assignment of the last k codons, we select the codon preceding them, denoted by the index c_{i-k}, such that the sum of forbidden motifs ending at position $i-1$, $F^{i-1}_{c_{i-k},\ldots,c_{i-2},c_{i-1}}$, and the count of new forbidden motifs which end in the new codon c_i, determined by the function $M'_{\mathcal{F}}$, is minimized. The number of forbidden motifs is recorded. This recurrence is similar to those of Skiena [16], used for a related problem of phage design.

$$F^{i}_{c_{i-k+1},\ldots,c_i} = \min_{1 \leqslant c_{i-k} \leqslant |\lambda(\alpha_{i-k})|} \left\{ F^{i-1}_{c_{i-k},\ldots,c_{i-1}} + M'_{\mathcal{F}}\left(\lambda_{c_{i-k}}(\alpha_{i-k}) \ldots \lambda_{c_i}(\alpha_i)\right) \right\} \tag{5}$$

Similarly, D is calculated in the same manner, after ensuring that the minimal number of forbidden motifs criteria is first satisfied.

$$D^{i}_{c_{i-k+1},\ldots,c_i} = \max_{1 \leqslant c_{i-k} \leqslant |\lambda(\alpha_{i-k})|} \left\{ \begin{array}{l} -\infty, \\ \quad \text{if } F^{i-1}_{c_{i-k},\ldots,c_{i-1}} + M'_{\mathcal{F}}(\lambda_{c_{i-k}}(\alpha_{i-k}) \ldots \lambda_{c_i}(\alpha_i)) \neq F^{i}_{c_{i-k+1},\ldots,c_i} \\ D^{i-1}_{c_{i-k},\ldots,c_{i-1}} + M'_{\mathcal{D}}(\lambda_{c_{i-k}}(\alpha_{i-k}) \ldots \lambda_{c_i}(\alpha_i)), \\ \quad \text{otherwise} \end{array} \right\} \tag{6}$$

Likewise, P is calculated to first ensure forbidden motifs are minimized, followed by desired motifs being maximized. Of these possible codon assignments, the one with the highest CAI value is selected and the score recorded.

$$P^{i}_{c_{i-k+1},\ldots,c_i} = \max_{1 \leqslant c_{i-k} \leqslant |\lambda(\alpha_{i-k})|} \left\{ \begin{array}{l} -\infty, \\ \quad \text{if } (F^{i-1}_{c_{i-k},\ldots,c_{i-1}} + M'_{\mathcal{F}}(\lambda_{c_{i-k}}(\alpha_{i-k}) \ldots \lambda_{c_i}(\alpha_i)) \neq F^{i}_{c_{i-k+1},\ldots,c_i}) \\ \quad \vee (D^{i-1}_{c_{i-k},\ldots,c_{i-1}} + M'_{\mathcal{D}}(\lambda_{c_{i-k}}(\alpha_{i-k}) \ldots \lambda_{c_i}(\alpha_i)) \neq D^{i}_{c_{i-k+1},\ldots,c_i}) \\ \tau_{c_i}(\alpha_i) \times P^{i-1}_{c_{i-k},\ldots,c_{i-1}}, \\ \quad \text{otherwise} \end{array} \right\} \tag{7}$$

Eq. (10) determines the optimal CAI score up to position i of the input amino acid sequence. Therefore, the optimal CAI value of some input sequence A of length $|A|$ is given by $\widetilde{P^{|A|}_k}$, where

$$\widetilde{F^{i}_k} = \min_{\substack{1 \leqslant c_i \leqslant |\lambda(\alpha_i)| \\ 1 \leqslant c_{i-1} \leqslant |\lambda(\alpha_{i-1})| \\ \vdots \\ 1 \leqslant c_{i-k+1} \leqslant |\lambda(\alpha_{i-k+1})|}} \left\{ F^{i}_{c_{i-k+1},\ldots,c_i} \right\} \tag{8}$$

$$\widetilde{D^{i}_k} = \max_{\substack{1 \leqslant c_i \leqslant |\lambda(\alpha_i)| \\ 1 \leqslant c_{i-1} \leqslant |\lambda(\alpha_{i-1})| \\ \vdots \\ 1 \leqslant c_{i-k+1} \leqslant |\lambda(\alpha_{i-k+1})|}} \left\{ \begin{array}{ll} D^{i}_{c_{i-k+1},\ldots,c_i}, & \text{if } F^{i}_{c_{i-k+1},\ldots,c_i} = \widetilde{F^{i}_k} \\ -\infty, & \text{otherwise} \end{array} \right\} \tag{9}$$

$$\widetilde{P^{i}_k} = \max_{\substack{1 \leqslant c_i \leqslant |\lambda(\alpha_i)| \\ 1 \leqslant c_{i-1} \leqslant |\lambda(\alpha_{i-1})| \\ \vdots \\ 1 \leqslant c_{i-k+1} \leqslant |\lambda(\alpha_{i-k+1})|}} \left\{ \begin{array}{ll} P^{i}_{c_{i-k+1},\ldots,c_i}, & \text{if } (F^{i}_{c_{i-k+1},\ldots,c_i} = \widetilde{F^{i}_k}) \wedge (D^{i}_{c_{i-k+1},\ldots,c_i} = \widetilde{D^{i}_k}) \\ -\infty, & \text{otherwise} \end{array} \right\} \tag{10}$$

3.1. Algorithm correctness

The correctness of the algorithm can be shown by induction on the position in the amino acid sequence. Lemma 1 shows that Eq. (7) gives an optimal score under the assumption that the previous k codons are fixed.

Lemma 1. *$P^{i}_{c_{i-k+1},\ldots,c_{i-1},c_i}$ of Eq. (7) correctly determines the score of the optimal valid codon design up to the ith codon position, having the codon assignment $c_{i-k+1}, \ldots, c_{i-1}, c_i$ for the last k codons, given that the maximum length of any motif is $3k$.*

Proof. We will argue by induction. The base case ($i = k$) is trivially *valid* as Eq. (4) correctly determines the CAI score of the first k codons, by definition.

Assume $P^{i-1}_{c'_{i-k},\ldots,c'_{i-2},c'_{i-1}}$ correctly determines the score of an optimal *valid* codon assignment, up to position $i-1$, having the codon assignment $c'_{i-k}, \ldots, c'_{i-2}, c'_{i-1}$ for the last k codons. Similarly, assume F^{i-1} and D^{i-1} are also correct for

the corresponding codon assignment. When moving one position ahead, from $i-1$ to i, we must consider the case of any new motifs we may introduce. By Observation 1, any new motif which ends within codon c_i could not extend past codon c_{i-k}. There are at most 6 possible codon assignments to position c_{i-k} that can directly precede a specific codon assignment $c_{i-k+1},\ldots,c_{i-1},c_i$ ending at position i as there are at most 6 codons for any amino acid. Therefore, the optimal assignment(s) to c_{i-k} must be a subset of these possibilities. $M'_{\mathcal{F}}(\lambda_{c_{i-k}}(\alpha_{i-k})\ldots\lambda_{c_{i-1}}(\alpha_{i-1})\lambda_{c_i}(\alpha_i))$ calculates the number of new forbidden motifs introduced in the codon assignment $c_{i-k},\ldots,c_{i-1},c_i$ which end in codon c_i. By our assumption, $F^{i-1}_{c_{i-k},\ldots,c_{i-2},c_{i-1}}$ correctly determines the minimum number of forbidden motifs having codon assignment $c_{i-k},\ldots,c_{i-2},c_{i-1}$, ending at position $i-1$. Therefore, the sum of these two quantities correctly determines the minimum number of forbidden motifs. As codons $c_{i-k+1},\ldots,c_{i-1},c_i$ are fixed, and Eq. (5) evaluates every possible assignment to c_{i-k} to determine a minimum, then it must be the case that $F^{i}_{c_{i-k+1},\ldots,c_{i-1},c_i}$ is the minimum number of forbidden motifs up to position i, having the codon assignment $c_{i-k+1},\ldots,c_{i-1},c_i$ for the last k codons. We argue similarly for $D^{i}_{c_{i-k+1},\ldots,c_{i-1},c_i}$ in Eq. (6) with the addition that any assignment of c_{i-k} also be forbidden motif minimal ensured by line 1 of the equation.

Finally, consider $P^{i}_{c_{i-k+1},\ldots,c_{i-1},c_i}$. Line 1 of Eq. (7) assigns the value $-\infty$ if a codon assignment ending in $c_{i-k},\ldots,c_{i-1},c_i$ is not *valid*. For all assignments which are *valid*, the equation (line 2) determines the CAI score by multiplying the fitness of the codon represented by c_i for amino acid α_i with the optimal CAI score up to position $i-1$ (guaranteed optimal by our assumption), for each possible choice of c_{i-k} such that there is a *valid* codon assignment ending in $c_{i-k},\ldots,c_{i-1},c_i$. Since every assignment to codon c_{i-k} is evaluated and the maximum is determined, then it must be the case that $P^{i}_{c_{i-k+1},\ldots,c_{i-1},c_i}$ correctly determines the score of the optimal *valid* codon design up to the ith codon position, having the codon assignment $c_{i-k+1},\ldots,c_{i-1},c_i$ for the last k codons, given that the maximum length of any motif is $3k$. □

Since Eq. (10) evaluates all combinations of the previous k codons, Theorem 1 states that an optimal design must be found, if one exists.

Theorem 1. *$\widetilde{P}^{i}_{k}$ of Eq. (10) correctly determines the score of an optimal* valid *codon design, with respect to CAI value, up to the ith codon, given that the maximum length of any motif is $3k$.*

Proof. Lemma 1 guarantees that $P^{i}_{c_{i-k+1},\ldots,c_{i-1},c_i}$ correctly determines the score of the optimal *valid* codon design up to the ith codon, having the codon assignment $c_{i-k+1},\ldots,c_{i-1},c_i$ for the last k codons, given that the maximum length of any motif is $3k$. Therefore, if every possible assignment of the last k codons is evaluated, a maximum of 6^k possibilities, the score of an optimal *valid* codon design ending at position i can easily be determined.

First, consider that $\widetilde{F}^{i}_{k}$ correctly determines the minimum number of forbidden motifs possible, up to position i, by evaluating all possible assignments of that last k codons. Similarly, $\widetilde{D}^{i}_{k}$ evaluates the maximum number of desired motifs possible, by first ensuring that the minimum number of forbidden motifs criteria is satisfied. Finally, by evaluating all possible codon assignments of the last k codons, and determining the maximum score of all those which are *valid*, $\widetilde{P}^{i}_{k}$ must determine the optimal *valid* CAI score, up to position i. □

3.2. Algorithm complexity

Under the assumption that the maximum length of any motif is constant, Theorem 2 proves that the overall time and space complexity is linear.

Theorem 2. *The dynamic programming algorithm for CAI optimization finds a valid nucleic acid sequence design for an amino acid sequence A in $O(|A|+h)$ time and $O(|A|+h)$ space, where h is the total length of forbidden and desired motifs and all motifs are of constant length.*

Proof. Let A be the input amino acid sequence of length $|A|$, and let $\mathcal{F}$ and $\mathcal{D}$ be the sets containing forbidden motifs and desired motifs, respectively. Let g be the length of the longest motif. Set $k=\lceil g/3\rceil$. We assume g, and therefore k, is constant. An Aho–Corasick automaton containing all forbidden motifs, and a separate one for desired motifs, is built only once, in $O(h)$ time. As the maximum number of codons for any amino acid is 6, the base case must evaluate 6^k possible codon designs in the worst case. Determining if a design of length k contains a forbidden motif can be accomplished in $O(3k)$ time and we assume the product of k numbers can be computed in $O(k)$ time. Therefore, $O(6^k\cdot 3k)$ time in total is required to compute the base case. For every position i, $k<i\leqslant|A|$, $O(6^k)$ possible designs must be evaluated. Each of these sub designs could potentially be prefixed by 1 of 6 different codons (at position $i-k$). Evaluating each of these possibilities requires checking for forbidden and desired motifs, in $O(3(k+1))$ time, and performing one multiplication. Determining the best of the previous codons therefore takes $O(6\cdot 3(k+1))$ time. All $O(6^k)$ possible designs at some position i can therefore be calculated in $O(6^{k+1}\cdot 3(k+1))$ time. This effort must be repeated $|A|-k$ times. The optimal score can be determined by finding the maximum of the previously calculated scores at position $|A|$ in $O(6^k)$ time. Therefore, the total time complexity is $O(h+6^k\cdot 3k+(|A|-k)\cdot 6^{k+1}\cdot 3(k+1))=O(h+|A|)$, as k is constant.

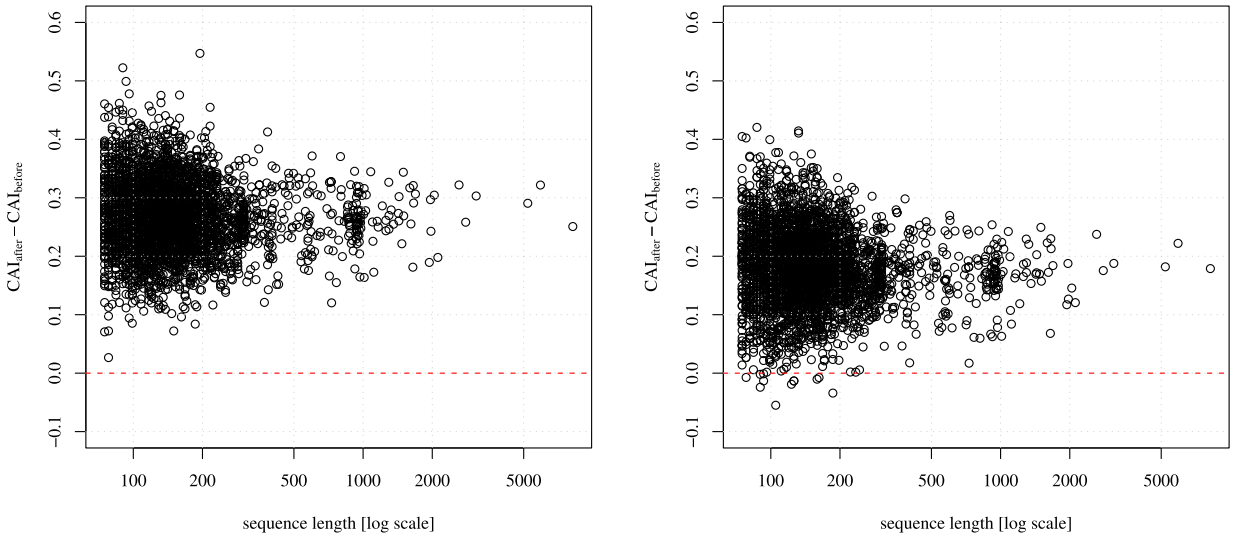

Fig. 2. Results are shown for the difference between the optimal CAI value and the original CAI value, plotted against sequence length, for each of the 3157 sequences. On the left, results are shown when only the forbidden motif set is considered. The right side shows the results when both the forbidden and desired motif sets are considered.

Storing an Aho–Corasick automaton takes $O(h)$ space. Furthermore, $O(6^k)$ entries must be maintained in the dynamic programming table for each of $|A| - k$ codon positions. Therefore, $O(h + 6^k \cdot (|A| - k)) = O(h + |A|)$ space is needed. If only a score is required, the space can be reduced to $O(h)$. □

4. Empirical results

4.1. Experimental setup

4.1.1. Data set

We use a filtered set of the 3891 CDS (coding DNA sequence) regions of the GENECODE subset of the ENCODE dataset [17] (version hg17 NCBI build 35). This curated dataset comprises approximately 1% of the human genome and is representative of several its characteristics such as distribution of gene lengths and GC composition (54.31%). After filtering any sequences less than 75 bases in length, the remaining 3157 CDS regions range in length from 75 to 8186 bases, averaging 173 bases with 267 bases standard deviation.

4.1.2. Codon frequencies

In all cases, we use the codon frequencies of *Escherichia coli* as reported by the Codon Usage Database [http://www.kazusa.or.jp/codon].

4.1.3. Implementation and hardware

All algorithms were implemented in C++ and compiled with g++ (GCC 4.1.0). Experiments were run on our reference Pentium IV 2.4 GHz processor machines, with 1 GB main memory and 256 Kb of CPU cache, running SUSE Linux version 10.1.

4.2. Results

To evaluate the effectiveness and efficiency of our algorithm, a forbidden and desired motif set were constructed which could be considered typical in practice. It is common for a gene synthesis experiment to use a single restriction enzyme. Furthermore, for reasons affecting gene expression, a common task is the removal of polyhomomeric regions (consecutive repeat region of identical nucleotides). Therefore, we have created a forbidden motif set containing ten elements including GAGTC, GACTC, AAAA, TTTT, GGGG, and CCCC where GAGTC is the motif for the *MlyI* restriction enzyme, GACTC is its reverse complement and the other motifs ensure no polyhomomeric regions greater than length three are permitted. The other four elements of the forbidden motif set (not shown) are immuno-inhibitory motifs originally used in the work of Satya et al. [14]. That work also used a desired motif set consisting entirely of thirty-three immuno-stimulatory motifs. We use this same desired motif set in our study.

4.3. Performance of the CAI optimization algorithm

Results are shown for all 3157 sequences in Fig. 2. On the left side of the figure, the difference in optimal CAI value and the original CAI value of each sequence, when forbidden motifs are minimized, is plotted against sequence length.

Table 1
The mean values and standard deviations (averaged over 3157 sequences) of CAI score, number of forbidden motifs, and number of desired motifs are shown for the original sequences (wild-types), the optimized sequences with forbidden motifs minimized and the optimized sequences with forbidden motifs minimized and then desired motifs maximized.

Motif sets	CAI value (std. dev.)	# forbidden (std. dev.)	# desired (std. dev.)
none (wild-types)	0.6477 (0.06)	9.2372 (16.24)	0.4869 (1.06)
forbidden	0.9161 (0.04)	0.1384 (0.45)	0 (0.00)
forbidden and desired	0.8280 (0.05)	0.1384 (0.45)	10.1324 (14.84)

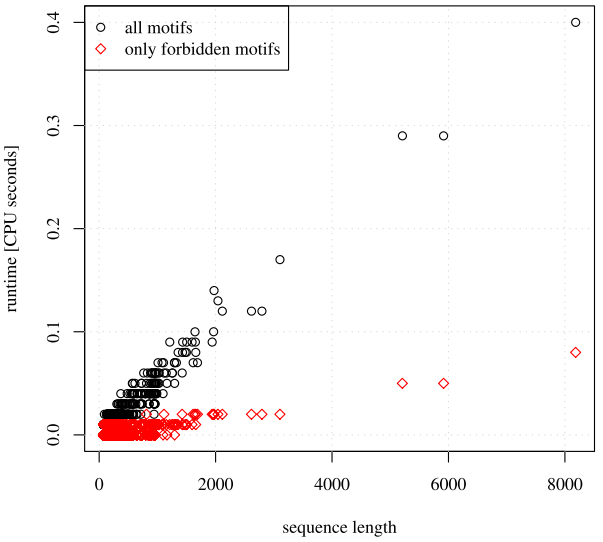

Fig. 3. CPU run-time performance of the CAI dynamic programming algorithm is plotted for each of the 3157 sequences, against their length. Results are plotted for two cases: when only forbidden motifs are considered, and when both forbidden and desired motifs are considered.

Desired motifs were not considered. For all sequences, the CAI value is improved compared with the original, with an average improvement of approximately 0.27. Shown on the right is the difference in CAI value for each sequence when the forbidden motifs are minimized and then the desired motifs are maximized. For this case, the average improvement of CAI value drops to 0.18, with only 12 sequences (∼0.4%) reporting a worse CAI value than the original. A summary of CAI statistics is presented in Table 1. In virtually all cases, forbidden motifs were eliminated entirely. Less than 2% of all sequences contained more than one forbidden motif after optimization, with only 0.6% containing more than two. On average 10 motifs were added to optimized sequences, when desired motifs were considered. These results demonstrate that it is possible to engineer motifs while still optimizing codon usage considerably.

Fig. 3 shows the run-time performance of the CAI dynamic programming algorithm for both the case of considering only forbidden motifs and when considering both forbidden and desired motifs. Run-times are plotted for all 3157 sequences versus their sequence length. The algorithm clearly scales linearly with respect to sequence length. Considering desired motifs, in addition to forbidden motifs, increases run-time by a small constant factor, on average. In the worst case, the algorithm terminates in 0.43 CPU seconds for the longest sequence (8141 bases).

5. Conclusions

In this work we have presented the first linear time and space algorithm for the problem of optimizing the codon adaptation index (CAI) value of a gene. The algorithm guarantees that codon designs for a protein will be found that have a minimum number of forbidden motifs from some user-defined set. The algorithm can also ensure the inclusion of desired motifs, when applicable. We provided a formal proof of correctness and time and space analysis. The algorithm runs in time and space that is linear in the input size, assuming that g, the maximum length of a forbidden or desirable motif, is constant.

The constant hidden in the linear worst-case bound on the running time and space does, however, include the term $6^{\lceil g/3 \rceil + 1}$, where g is the length of the longest forbidden or desired motif. This is the same complexity, including the exponential term, for the pre-processing step that checks for motifs in the algorithm of Satya et al. [14]. As indicated by our experimental data (Section 4), typical gene coding regions in the human genome are often hundreds and sometimes thousands of bases long. In our experimental data, forbidden or desired motifs were up to eight nucleotides long, in which case the term $6^{\lceil g/3 \rceil + 1}$ is at most 1296. An extensive empirical analysis of the algorithm has shown it to be efficient in

practice for these typical maximum lengths of motifs and gene coding regions. Additional experiments (data not shown) demonstrate that the algorithm still completes in less than one second for typical gene lengths when the maximum motif length is doubled to sixteen nucleotides. Aside from improving the run-time performance and memory over the method of Satya et al., our dynamic programming algorithm is also amenable to a further reduction in space consumption by standard techniques [6]. This may prove useful when considering even longer motifs.

An efficient algorithm is a crucial first step towards designing genes while considering other important sequence features. For instance, designing genes with a guarantee that the resulting nucleic acid sequence does not form stable nucleic acid secondary structure is an interesting future direction, and one that may greatly effect translational efficiency.

References

[1] A.V. Aho, Algorithms for Finding Patterns in Strings, MIT Press, Cambridge, MA, USA, 1990, pp. 255–300.
[2] A. Condon, C. Thachuk, Efficient codon optimization with motif engineering, in: Proceedings of the 2011 International Workshop on Combinatorial Algorithms, 2011, pp. 337–348.
[3] A. Fuglsang, Codon optimizer: a freeware tool for codon optimization, Protein Expression and Purification 31 (2) (2003) 247–249.
[4] W. Gao, A. Rzewski, H. Sun, P.D. Robbins, A. Gambotto, UpGene: application of a web-based DNA codon optimization algorithm, Biotechnology Progress 20 (2) (2004) 443–448.
[5] A. Grote, K. Hiller, M. Scheer, R. Münch, B. Nörtemann, D.C. Hempel, D. Jahn, JCat: a novel tool to adapt codon usage of a target gene to its potential expression host, Nucleic Acids Research 33 (Web Server issue) (2005) 526–531.
[6] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge Press, New York, NY, USA, 1997.
[7] C. Gustafsson, S. Govindarajan, J. Minshull, Codon bias and heterologous protein expression, Trends in Biotechnology 22 (7) (2004) 346–353.
[8] L. Holm, Codon usage and gene expression, Nucleic Acids Research 14 (7) (1986) 3075–3087.
[9] D.M. Hoover, J. Lubkowski, DNAWorks: an automated method for designing oligonucleotides for PCR-based gene synthesis, Nucleic Acids Research 30 (10) (2002) e43.
[10] S. Jayaraj, R. Reid, D.V. Santi, GeMS: an advanced software package for designing synthetic genes, Nucleic Acids Research 33 (9) (2005) 3011–3016.
[11] J.F. Kane, Effects of rare codon clusters on high-level expression of heterologous proteins in Escherichia coli, Current Opinion in Biotechnology 6 (5) (1995) 494–500.
[12] G. Lithwick, H. Margalit, Hierarchy of sequence-dependent features associated with prokaryotic translation, Genome Research 13 (12) (2003) 2665–2673.
[13] P. Puigbo, E. Guzman, A. Romeu, S. Garcia-Vallve, OPTIMIZER: a web server for optimizing the codon usage of DNA sequences, Nucleic Acids Research 35 (Suppl. 2) (2007) W126–131.
[14] R.V. Satya, A. Mukherjee, U. Ranga, A pattern matching algorithm for codon optimization and CpG motif-engineering in DNA expression vectors, in: CSB'03: Proceedings of the IEEE Computer Society Conference on Bioinformatics, IEEE Computer Society, Washington, DC, USA, 2003, pp. 294–305.
[15] P.M. Sharp, W.H. Li, The codon adaptation index—a measure of directional synonymous codon usage bias, and its potential applications, Nucleic Acids Research 15 (3) (1987) 1281–1295.
[16] S. Skiena, Designing better phages, Bioinformatics (2001) 253–261.
[17] The ENCODE Consortium, The ENCODE (ENCyclopedia of DNA elements) project, Science 306 (5696) (2004) 636–640.
[18] S. Varenne, C. Lazdunski, Effect of distribution of unfavourable codons on the maximum rate of gene expression by an heterologous organism, Journal of Theoretical Biology 120 (1) (1986) 99–110.
[19] A. Villalobos, J.E. Ness, C. Gustafsson, J. Minshull, S. Govindarajan, Gene Designer: a synthetic biology tool for constructing artificial DNA segments, BMC Bioinformatics 7 (2006) 285.
[20] G. Wu, N. Bashir-Bello, S.J. Freeland, The synthetic gene designer: a flexible web platform to explore sequence manipulation for heterologous expression, Protein Expression and Purification 47 (2) (2006) 441–445.