Computers and Operations Research 119 (2020) 104932

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Codon optimization by 0-1 linear programming

Claudio Arbib[a,*], Mustafa Ç. Pınar[d], Fabrizio Rossi[b], Alessandra Tessitore[c]

[a] Dipartimento di Scienze/Ingegneria dell'Informazione e Matematica and Center of Excellence DEWS, Università degli Studi dell'Aquila, L'Aquila, Italia
[b] Dipartimento di Scienze/Ingegneria dell'Informazione e Matematica, Università degli Studi dell'Aquila, L'Aquila, Italia
[c] Dipartimento di Scienze Cliniche Applicate e Biotecnologiche, Università degli Studi dell'Aquila, L'Aquila, Italia
[d] Department of Industrial Engineering, Bilkent University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 18 June 2019
Revised 20 February 2020
Accepted 23 February 2020
Available online 29 February 2020

Keywords:
Protein Design
Codon Optimization
Motif Engineering
Integer Linear Programming

ABSTRACT

The problem of choosing an optimal codon sequence arises when synthetic protein-coding genes are added to cloning vectors for expression within a non-native host organism: to maximize yield, the chosen codons should have a high frequency in the host genome, but particular nucleotide bases sequences (called "motifs") should be avoided or, instead, included. Dynamic programming (DP) has successfully been used in previous approaches to this problem. However, DP has a computational limit, especially when long motifs are forbidden, and does not allow control of motif positioning and combination. We reformulate the problem as an integer linear program (IP) and show that, with the same computational resources, one can easily solve problems with much more nucleotide bases and much longer forbidden/desired motifs than with DP. Moreover, IP (*i*) offers more flexibility than DP to treat constraints/objectives of different nature, and (*ii*) can efficiently deal with newly discovered critical motifs by dynamically re-optimizing additional variables and mathematical constraints.

1. Introduction

Optimization models and methods have received increasing attention in genetic engineering in the last decade. Discrete optimization, in particular, proved to be a useful tool in important protein design problems, such as determining the physical deployment of bio-molecular chains in the tri-dimensional space (*inverse protein folding* Pabo (1983); Traoré et al. (2013)) or finding the most productive way of obtaining a protein with desired features (*codon optimization* Kofman et al. (2003); Webster et al. (2017)). In consideration of problem complexity, the large majority of software tools for these applications are heuristics but, in special cases, problems are addressed by Dynamic Programming (DP). In fact, since genomic objects are represented as sequences, DP appears naturally fit for problems of various nature (see e.g. Cohen and Skiena (2003), Condon and Thachuk, Montes et al. (2010); Mueller et al. (2010) and Gould et al. (2014) for more references).

In this paper we investigate the potential of Integer Programming (IP), and precisely of 0-1 Linear Programming, in designing optimization tools for a large number of problems addressed in the relevant literature. Via a specific example, on one hand we numerically demonstrate the potential advantage of IP over DP in terms of efficiency; on the other hand we suggest that this potential has chances to be exploited for more articulated problems, that can easily be modeled resorting to the great expressiveness of IP.

A protein is a (possibly very long) sequence π of elements $\pi_1, \pi_2, \ldots$ called *amino acids*. Each amino acid is in turn generated by a triplet, called *codon*, of *nucleotide bases* taken from the well known DNA alphabet. Viewed in this way, a protein is essentially a combinatorial object: protein design calls for finding one of a huge number of codon sequences (or equivalently, nucleotide sequences) that realizes the desired protein while optimizing factors related to protein yield, folding or other biochemical properties.

The idea of codon optimization in protein design, as described in Kofman et al. (2003), is partly motivated by significant variations of codon usage bias (Holm, 1986; Varenne and Lazdunski, 1986) among different species or even genes within the same organisms. In fact, as we shall detail later in Section 2, the same amino acid can in general be expressed by more than one codon, and the same protein is thus produced by an organism (the *host*) in equivalent forms that differ for the codons used in its synthesis: the form most preferred (or *biased*) by the host is generally expected to give the largest yield. According to this view and in rough terms, then, the best way to construct a protein would be to insert in the host genome the most frequently observed codons that the host uses in nature to express the required amino acids.

* Corresponding author.
E-mail addresses: claudio.arbib@univaq.it (C. Arbib), mustafap@bilkent.edu.tr (M.Ç. Pınar), fabrizio.rossi@univaq.it (F. Rossi), alessandra.tessitore@univaq.it (A. Tessitore).

https://doi.org/10.1016/j.cor.2020.104932
0305-0548/

However, even neglecting studies (Mauro and Chappel, 2014; Webster et al., 2017) that find codon frequency too simplistic — when not contradictory — an indicator, pandering to the host codon-usage bias may not guarantee the best results. In fact, special sub-sequences of nucleotide bases – called *motifs* (Sandhu et al., 2008; Satya et al., 2003) – formed in the synthesis may have positive or negative effects in terms of either protein final use or yield. For instance, in DNA vaccination or gene therapy, some motifs have experimentally been observed to be immuno-stimulatory while certain others are immuno-suppressive. Also, particular sub-structures (*hairpins, pseudoknots*) may reduce/increase protein yield despite host bias/unbias (Charneski and Hurst, 2013), with a behavior influenced by many factors, among which sequence stability. Thus, a finer control of sub-structural motifs is in general regarded as potentially advantageous.

Following the above remarks, along with a DNA or RNA fragment that encodes the desired protein, software packages often provide users with a warning about motifs that may contrast expected features. However, those motifs are not always controlled *ex-ante*: for instance, the *IDT Codon Optimization Tool* checks and notifies hairpins, repeats and extreme genetic code content (IDT Codon Optimization Tool, 2018) just *ex-post*. Furthermore, solution optimality is rarely certified: *Optimumgene*™ OptimumGene, to quote another commercial tool, just uses particle swarm optimization heuristics. For a survey on, and comparison of, commercial tools the interested reader can refer to Gould et al. (2014).

Besides commercial tools, many methods for codon optimization can be found in the scientific literature, and several algorithms can be downloaded from the web and run for experiments (see Condon and Thachuk (2012); Fuglsang (2003); Gao et al. (2004); Sandhu et al. (2008); Satya et al. (2003), just to quote a few). Like commercial tools, those algorithms take as input a target amino acid sequence π (the protein that one wants to produce) and all the codon biases of the host organism, and return a sequence σ of nucleotide bases that corresponds to π and maximizes a quality indicator, such as the *Codon Adaptation Index* (CAI) originally proposed by Sharp and Li (1987). In so doing, however, *motif engineering* methods such as (Condon and Thachuk, 2012; Satya et al., 2003; Skiena, 2001) allow some *ex-ante* motif control by explicitly trying to maximize (minimize) the amount or weight of desired (undesired) motifs in the output encoding σ. Instead of the CAI, other protein synthesis methods try to optimize the stability of *m*RNA secondary structures (Cohen and Skiena, 2003), or the placement of restriction sites Montes et al. (2010). Pareto optimal codon designs obtained using a standard commercial solver, and incorporating (multiple) objectives such as CAI and *m*RNA secondary structures are discussed in Şen et al. (2019).

In this paper we focus on CAI optimizers and use (Condon and Thachuk, 2012), where an exact dynamic programming (DP) algorithm is proposed and tested, as our benchmark reference. This algorithm maximizes the CAI of the output sequence, and implements motif engineering by minimizing (maximizing) the number of undesired (desired) motifs from lists given in advance. The algorithm runs in linear time in the target protein length, but is exponential in the length of the longest (desired or undesired) motif: as a consequence, it works fine for moderate motif lengths but becomes increasingly impractical as the length increases (experiments in Condon and Thachuk (2012) refer to a data set with ten undesired motifs of length ≤ 6 and thirty-three desired ones of length ≤ 8).

Instead of DP, we resort to Integer Programming (IP) to formulate a more general model. We tested our method on a data set that adds new and longer proteins, and much more and longer motifs, to that used in Condon and Thachuk (2012). With our method, we found optimal solutions within 0.36 seconds in the worst case against over 73 seconds of DP; and on average, the CPU time we employ is from 1.4 to over 500 times less than with DP.

Not only is our model much more efficient than Condon and Thachuk (2012) to treat long motifs, but it provides users with more sophisticated control of subsequences. As already remarked, just prohibiting or requiring motifs has been observed to be insufficient in reflecting the intrinsic complexity of DNA/RNA synthesis. Software tools try to address limitations by also considering codon context (e.g., codon pairs), autocorrelation, *t*RNA availability, and RNA secondary structures (e.g. hairpins). Four of the eleven software packages examined in Gould et al. (2014) heuristically treat *m*RNA secondary structures, and two also codon context. All those tools aim at optimizing the CAI, but just three consider motif avoidance. Our contribution is basically computational, so we will not insist on refinements from the viewpoint of biological adequacy. We however remark that our IP approach has a potential to include, quite easily, additional features listed in Gould et al. (2014) due to both its flexibility and efficiency: decision variables can in fact be used to express additional constraints, such as hairpin prevention or codon pair bias, while the strong improvement in the use of computation resources appears promising to implement dynamic structure search and re-optimization, as in a branch-and-cut scheme.

The remainder of the paper is organized in five sections. In Section 2 we survey basic issues of codon optimization with motif engineering and formulate the problem here addressed. In Section 3 we present our 0-1 linear programming model and discuss its main characteristics. In Section 4 we observe some polyhedral properties of the model that partially explain its performance in practice, and describe the potential of our model for coping with more complex sub-structural requirements. A numerical experience including a comparison of our model to the DP algorithm developed in Condon and Thachuk (2012) is provided in Section 5, and conclusions are finally drawn in Section 6.

2. Basics of codon optimization

As it is well known, the DNA alphabet $\mathcal{B}$ consists of four symbols called *bases*, $\mathcal{B} = \{\text{A,C,G,T}\}$. Triplets of elements of $\mathcal{B}$, called *codons*, generate the 20 different *amino acids* reported in Table 1 plus the *stop* signal of protein end: from now on, this set of 21 elements (the protein alphabet) will be denoted by $\mathcal{A}$.

Because there are as many as $4^3 = 64$ possible codons, a single amino acid of $\mathcal{A}$ can be encoded in more than one way: for example, *Phenylalanine* can be either encoded as TTT or as TTC. As a consequence, the same protein $\pi \in \mathcal{A}^n$ can be expressed by a huge number of DNA/RNA sequences $\sigma \in \mathcal{B}^{3n}$ that are equivalent in terms of the amino acids produced, but generally not in terms of bases and therefore secondary biochemical behavior. This behavior has a twofold aspect: one, in terms of protein yield, is related to the host organism; another, in terms of final use, deals with the effects on the protein recipient. The quality indexes normally (and here) adopted to measure the relevant effects are the *Codon Adaptation Index* (CAI) and the number of desired/undesired *motifs* in the output sequence. However, more complex indicators, such as measures of motif absolute or reciprocal positions, can be considered to reveal sub-sequence interaction (Evfratov et al., 2017; Osterman et al., 2013), see also Section 4.2.

2.1. Codon adaptation index

The *Codon Adaptation Index* (CAI) is correlated with protein yield (Quax et al., 2016), and is computed in an elementary way from experimental data that indicate how frequently codons are encoded by the host organism when producing a given amino acid. The frequency with which a codon c appears in nature (meaning,

Table 1
DNA codon table.

I	II base								III
base	T		C		A		G		base
T	TTT	*Phenylalanine*	TCT	*Serine*	TAT	*Tyrosine*	TGT	*Cysteine*	T
T	TTC	"	TCC	"	TAC	"	TGC	"	C
T	TTA	*Leucine*	TCA	"	TAA	*stop*	TGA	*stop*	A
T	TTG	"	TCG	"	TAG	"	TGG	*Tryptophan*	G
C	CTT	"	CCT	*Proline*	CAT	*Histidine*	CGT	*Arginine*	T
C	CTC	"	CCC	"	CAC	"	CGC	"	C
C	CTA	"	CCA	"	CAA	*Glutamine*	CGA	"	A
C	CTG	"	CCG	"	CAG	"	CGG	"	G
A	ATT	*Isoleucine*	ACT	*Threonine*	AAT	*Asparagine*	AGT	*Serine*	T
A	ATC	"	ACC	"	AAC	"	AGC	"	C
A	ATA	"	ACA	"	AAA	*Lysine*	AGA	*Arginine*	A
A	ATG	*Methionine*	ACG	"	AAG	"	AGG	"	G
G	GTT	*Valine*	GCT	*Alanine*	GAT	*Aspartic acid*	GGT	*Glycine*	T
G	GTC	"	GCC	"	GAC	"	GGC	"	C
G	GTA	"	GCA	"	GAA	*Glutamic acid*	GGA	"	A
G	GTG	"	GCG	"	GAG	"	GGG	"	G

in a reference genomic set associated with the host organism) can easily be computed dividing the number of occurrences of c by the total number of occurrences of the codons representing the same amino acid as c.

For example (Table 1), $a =$ *Lysine* has two codon expressions: $c =$ AAA and $c' =$ AAG. In the bacterium Escherichia Coli[1], *Lysine* is found 54,723 times in the form c and 17,729 times in the form c'. Thus, referred to the $54{,}723 + 17{,}729 = 72{,}452$ codons expressing *Lysine*, the relative frequencies are $\rho_c(a) = 0.76$ and $\rho_{c'}(a) = 0.24$. Instead of taking frequencies, the CAI uses codon *fitness numbers* $\tau_c(a)$ normalized to the most frequent codon: hence, in the case of *Lysine*, $\tau_c(s) = 1$ and $\tau_{c'}(a) = \frac{0.24}{0.76} \approx 0.32$.

The CAI of a codon sequence $\sigma = c_1 c_2 \dots c_n$ is obtained by multiplying and normalizing the fitness numbers of the codons σ consists of:

$$CAI(\sigma) = \left(\prod_{k=1}^{n} \tau_{c_k}(\pi_k) \right)^{\frac{1}{n}}$$

Therefore $CAI(\sigma) \leq 1$, where 1 corresponds to an "ideal" codon sequence for the considered host.

2.2. Motif engineering

If one relies on the CAI, getting an ideal sequence of bases σ seems trivial: simply, one has to take the most frequent codon for each amino acid π_k of the target protein. However, as outlined in Section 1, things are complicated by the fact that, in the arrangement provided by σ, the nucleotide bases may form undesired substrings from the viewpoint of the recipient organism. Or, perhaps, by the fact that a σ' with a smaller CAI than σ presents, as a counterpart, some desirable base substrings for the recipient organism. Or, finally, because the presence/absence of a particular motif has an effect on yield that dominates the one measured by the CAI. Known desired or undesired base substrings (the *motifs*) are respectively gathered into sets D and U, that we assume as input data for codon optimization.

It is reasonable to suppose that none of the given motifs is a subsequence of any other motif in D or U. While undesired motifs are simply forbidden in σ, a quite delicate question arises about desired motifs. In fact, a motif μ can appear several times in σ: for example, ATT ATA ATT ATA contains $\mu =$ TTAT twice. The model should then specify whether just one or each occurrence of μ is to be counted to fulfill the condition. The algorithm proposed in Condon and Thachuk (2012) counts every occurrence of a desired motif μ as a new one, and therefore we do the same in our model; however, our model allows to switch to the opposite requirement without any loss of generality.

2.3. Codon optimization with motif engineering

After the discussion in §§2.1-2.2, we can introduce the CODON OPTIMIZATION problem with MOTIF ENGINEERING (COME) that one has to solve in order to obtain an optimal encoding of a target protein $\pi \in \mathcal{A}^n$, where n is the protein length, that is, the number of amino acids it contains.

Problem 2.1.
Given:
A target protein $\pi \in \mathcal{A}^n$, a set $D \subseteq \mathcal{B}^*$ of desired motifs, a set $U \subseteq \mathcal{B}^*$ of undesired ones, and positive integers d, u.
Find:
A sequence $\sigma \in \mathcal{B}^{3n}$ such that:

i) The k-th triplet of σ encodes the k-th amino acid of π, $k = 1, \dots, n$;
ii) σ contains no more than u substrings of U;
iii) σ contains at least d substrings of D;
iv) $CAI(\sigma)$ is maximized.

A COME problem (or a relaxed version in which goals (*ii*) to (*iv*) are hierarchically considered) can be solved by dynamic programming as explained in Condon and Thachuk (2012). In the following section we will describe an alternative approach consisting in formulating the problem in terms of 0-1 Linear Programming.

3. 0-1 linear programming formulation

For the k-th amino acid π_k of the target protein $\pi \in \mathcal{A}^n$, let C_k denote the set of codons that can be used to express it. For instance (Table 1), if $\pi_k =$ *Isoleucine*, then $C_k = \{$ATT, ATC, ATA$\}$. Denote also as $M = D \cup U \subseteq \mathcal{B}^*$ the set of all desired or undesired motifs, and let $|M| = m$.

Let $\mu \in M$ be a motif consisting of $|\mu|$ bases, and suppose that μ is a substring of σ that starts in $i \in [1, 3n - |\mu| + 1]$. Depending on its starting point and according to the amino acids intersected, μ is segmented into base triplets, the first and last of which may not be completely determined.

[1] See http://www.kazusa.or.jp/codon/cgi-bin/showcodon.cgi?species=155864

Example 3.1. *If* $\mu = \text{TTCCATT} \in M$ *and* $i = 1$, *the segmentation is* [TTC][CAT][T · ·]. *If instead* μ *starts from the fifth base of* σ, *i.e.,* $i = 5$, *then its segmentation is* [·TT][CCA][TT·]. □

Let $s = s(i) = \lceil \frac{i}{3} \rceil$, $t = t(i) = \lceil \frac{i+|\mu|-1}{3} \rceil$ be the indexes of the first and last amino acid of π intersected by μ: in Example 3.1 with $i = 5$, $s = \lceil \frac{5}{3} \rceil = 2$ and, as $|\mu| = 7$, $t = \lceil \frac{11}{3} \rceil = 4$: thus μ intersects π_2, π_3, π_4.

For $k = s, \ldots, t$, define then R_k^μ as the set of codons that are compatible with the k-th segment of μ: in Example 3.1, for $i = 5$, the triplet associated with π_2 is [·TT], so $R_2^\mu = \{\text{ATT, TTT, CTT, GTT}\}$.

Definition 3.1. Motif μ is i-compatible with protein π if and only if $C_k^\mu = C_k \cap R_k^\mu \neq \emptyset$ for $k = s(i), \ldots, t(i)$. □

Example 3.2. *Let* $\mu = [\cdot\text{TT}][\text{CCA}][\text{TT}\cdot]$, *segmented as in Example 3.1 for* $i = 5$. *Then*

$R_2^\mu = \{\text{ATT, TTT, CTT, GTT}\}, R_3^\mu = \{\text{CCA}\}, R_4^\mu = \{\text{TTA, TTT, TTC, TTG}\}$

Assume now

$\pi_2 = Isoleucine, \pi_3 = Proline, \pi_4 = Phenylalalanine$

From Table 1, we see

$C_2 = \{\text{ATT, ATC, ATA}\}, C_3 = \{\text{CCT, CCC, CCA, CCG}\}, C_4 = \{\text{TTT, TTC}\}$

Then $C_2^\mu = \{\text{ATT}\}, C_3^\mu = \{\text{CCA}\}, C_4^\mu = \{\text{TTT, TTC}\}$. *All sets being non-empy,* μ *is 5-compatible with* π. □

Note that more motifs can be compatible with π in a given position. For example, also $\nu = \text{TTCCCTT}$, with $R_2^\nu = R_2^\mu, R_4^\nu = R_4^\mu$ and $R_3^\nu = \{\text{CCC}\}$, or $\gamma = \text{TTCCAT}$ with $R_2^\gamma = R_2^\mu, R_3^\gamma = R_3^\mu$ and $R_4^\gamma = \{\text{TTT, TTC}\}$, are 5-compatible with π. Given that, let

M_i be the set of all $\mu \in M$ that are i-compatible with π (M_i can of course be empty, and in particular $M_i = \emptyset$ for $i > 3n - \min\{|\mu| : \mu \in M\}$).
$U_i = M_i \cap U$, $D_i = M_i \cap D$.

Furthermore, denote by $K_i^\mu = \{s(i), s(i)+1, \ldots, t(i)\}$ the index set of the amino acids intersected by μ when starting in i (see e.g. figure 5 in the Appendix, where $K_i^\mu = \{1, 2, 3\}$ for $i = 3, \mu =$ CCCCCTT). We are now ready to formulate Problem 2.1 as 0-1 LP. The formulation uses two types of 0-1 decision variables, namely

Codon variables

x_k^c *set to 1 if and only if* $c \in C_k$ *is used to encode* $\pi_k, k = 1, \ldots, n$

Motif variables

y_i^μ set to 1 if and only if $\mu \in M_i$ is a subsequence of $\sigma \in \mathcal{B}^{3n}$ that starts from position $i, i = 1, \ldots, 3n - |\mu| + 1$

Finally, for $k = 1, \ldots, n$ and all $c \in C_k$ let

$$w_k^c = \log(\tau_c(\pi_k)) \leq 0 \tag{1}$$

Position (1) transforms $CAI(\sigma)$ into a linear function of the codon variables. The problem reads

$$\max \frac{1}{n} \sum_{k=1}^{n} \sum_{c \in C_k} w_k^c x_k^c \tag{2}$$

$$\sum_{c \in C_k} x_k^c = 1 \qquad k = 1, \ldots, n \tag{3}$$

$$\sum_{\mu \in M_i} y_i^\mu \leq 1 \qquad i = 1, \ldots, 3n \tag{4}$$

$$\sum_{i=1}^{3n} \sum_{\mu \in D_i} y_i^\mu \geq d \tag{5}$$

$$-y_i^\mu + \sum_{c \in C_k^\mu} x_k^c \geq 0 \qquad \forall i;\ \mu \in D_i;\ k \in K_i^\mu \tag{6}$$

$$\sum_{i=1}^{3n} \sum_{\mu \in U_i} y_i^\mu \leq u \tag{7}$$

$$-y_i^\mu + x_{s(i)}^a + x_{t(i)}^b + \sum_{k=s(i)+1}^{t(i)-1} x_k^c \leq |K_i^\mu| - 1 \tag{8}$$

$$\begin{aligned} &\forall i;\ \mu \in U_i;\ a \in C_{s(i)}^\mu;\ b \in C_{t(i)}^\mu \\ &x_k^c \in \{0, 1\} \qquad k = 1, \ldots, n;\ c \in C_k \\ &y_i^\mu \in \{0, 1\} \qquad i = 1, \ldots, 3n;\ \mu \in M_i \end{aligned} \tag{9}$$

Objective (2) is equivalent to maximizing the mean of the fitness number logarithms, that in turn corresponds to maximizing the CAI index of σ. Assignment conditions (3),(4) state that exactly one codon out of C_k must be used to encode π_k, and no more than one motif in M_i can start at any given position of i the output sequence σ. Inequalities (5), (7) require a minimum or allow a maximum number of desired and undesired motifs, respectively. Conditions (6) — called *implications* — state that if μ is chosen as a desired subsequence starting from the i-th base of σ, then all the amino acids involved must be encoded accordingly. Conditions (8) — that we here call *cover inequalities* — prohibit codon subsequences that contain μ from position i, unless μ is accepted as an undesired subsequence starting from the i-th base of σ. Inequalities (6) and (8) are written for each forbidden motif μ and position i; moreover, (8) are written for each codon pair with the first element in $C_{s(i)}^\mu$ and the second in $C_{t(i)}^\mu$ (note that for $s(i) < k < t(i)$, C_k^μ reduces to a single codon).

Let us briefly discuss inequality (4). Suppose that, unlike what supposed in this paper, μ is a subsequence of ν and both are (un)desired motifs; then an occurrence of ν may or may not imply an occurrence of μ according to whether (4) is withdrawn or not from the constraint set. This decision clearly implies a different count of motifs. Another case is when one motif is desired and the other undesired. If we accept their co-existence in σ, we can split inequality (4) in two conditions, one with the left-hand side summation extended to D_i, the other to U_i.

Codon variables are as many as

$$\sum_{k=1}^{n} |C_k| \quad \leq \quad 6n$$

because the amino acids with the largest C_k (namely, *Leucine* and *Arginine*) have 6 possible codons. Considering that each inner segment of μ is a blocked triplet (that is, only one combination of three bases is possible, see the intermediate [CAT] in Example 3.1 or [CCA] in Example 3.2), the number of motif variables is

$$\sum_{i=1}^{3n} \sum_{\mu \in M_i} \prod_{k \in K_i^\mu} |C_k^\mu| \leq 3n|M| \max_i \{|C_{s(i)}||C_{t(i)}|\} \leq 108mn$$

The problem has $4n$ assignment conditions, cover inequalities are

$$\sum_{i=1}^{3n} \sum_{\mu \in U_i} |C_{s(i)}^\mu||C_{t(i)}^\mu| \leq 108mn$$

and implications are $\leq mn \max\{|\mu|:\ \mu \in D\}$. Therefore the formulation is compact in m, n and the length of the longest desired motif. In practice, motif variables as well as implications and cover inequalities are much fewer, since their existence is subject to i-compatibility.

4. Nice model features

4.1. Matrix properties

As we will see in Section 5, formulation (2)-(9) is a good basis to develop a solution algorithm which is very efficient in practice. This nice behavior is partially explained by the structure of the inequality set. First, recall that a matrix is said to be *totally unimodular* if the determinant of any of its square sub-matrices is either 0 or ± 1. If $\mathbf{A} \in \{0, \pm 1\}^{m \times n}$ is totally unimodular and $\mathbf{b} \in \mathbb{Z}$, then $\mathbf{Ax} \leq \mathbf{b}$ is an integral polyhedron (for more on total unimodularity the reader is referred to Schrijver (1999)). We then observe the following two properties.

Theorem 4.1. *For any fixed y-vector of motif variables and any given motif μ, the coefficients of the codon variables x_k^c in inequalities (3), (6) form a totally unimodular matrix.*

Proof. Recall that for each k the index set of the variables in (3), C_k, contains that of the variables in (6), C_k^μ. Therefore the matrix has the consecutive 1 property by row. □

Theorem 4.2. *For any fixed y-vector of motif variables and any given motif μ, the coefficients of the codon variables x_k^c in inequalities (8) form a totally unimodular matrix.*

Proof. Let $K_i^\mu = \{s(i), \ldots, t(i)\}$. There exists one cover inequality for each codon pair $a \in C_{s(i)}^\mu$, $b \in C_{t(i)}^\mu$: each intermediate codon variable x_k^c involved in the inequalities, $s(i) < k < t(i)$, is in fact blocked to the single codon $c \in C_k^\mu$ that is compatible with μ in position i. Hence we can associate each inequality (each non-intermediate variable $x_{s(i)}^a, x_{t(i)}^b$) with a vertex (with an edge) of a complete bipartite graph G. The coefficient of the codon variables are then arranged into a matrix where

- the columns corresponding to intermediate variables consist of all 1;
- the remaining columns form the edge-vertex incidence matrix of G.

The matrix is therefore totally unimodular. □

Although Theorem 4.2 admits some generalization, it does not hold in general if assignment constraints (3) are added to the matrix, or if cover inequalities (8) are taken from different motifs.

Proposition 4.3. *The assignment constraints (3) plus the inequalities (8) associated with a single motif can form a matrix which is not totally unimodular for any fixed y-vector of motif variables.*

Example 4.4. *Take π = (Serine, Proline, Leucine) and $\mu =$* CCCCCTT *forbidden in position $i = 2$. This position implies the segmentation*

$\mu =$ [·CC][CCC][TT·]

From the DNA codon table (Table 1), the only compatible encoding of Serine *is* TCC*;* Proline *is blocked to* CCC*; on the contrary, both* TTA *and* TTG *encode* Leucine *while being compatible with μ in position 2. The motif is forbidden by $y_2^\mu = 0$, so the relevant cover inequalities read*

$$x_1^{\text{TCC}} + x_2^{\text{CCC}} + x_3^{\text{TCA}} \leq 2 \qquad x_1^{\text{TCC}} + x_2^{\text{CCC}} + x_3^{\text{TCG}} \leq 2$$

Moreover, by (3),

$$x_3^{\text{TCA}} + x_3^{\text{TCG}} \leq 1$$

Putting those inequalities together, we note that the coefficients of x_1^{TCC} (or x_2^{CCC}), $x_3^{\text{TCA}}, x_3^{\text{TCG}}$ form an odd circulant with determinant ± 2. □

Proposition 4.5. *The inequalities (8) associated with an odd set of motifs can form a matrix which is not totally unimodular for any fixed y-vector of motif variables.*

Example 4.6. *Take π = (Leucine, Glutamic Acid, Aspartic Acid) and*

$\mu =$ TAGA $\quad \nu =$ TTAGAGG $\quad \rho =$ AGAT

respectively forbidden in positions 2, 1, 6*. From Table 1 we see that* Leucine *can be encoded* TTA*,* Glutamic Acid GAA *or* GAG*, and* Aspartic Acid GAT*, therefore π can be encoded either* TTA GAA GAT *or* TTA GAG GAT*. Forbidding motifs in the given positions implies $y_2^\mu = y_1^\nu = y_6^\rho = 0$ and the following segmentations:*

$\mu =$ [·TA][GA·][···] $\quad \nu =$ [TTA][GAG][G··] $\quad \rho =$ [···][··A][GAT]

We then have the cover inequalities

$$x_1^{\text{TTA}} + x_2^{\text{GAA}} \leq 1 \qquad x_1^{\text{TTA}} + x_2^{\text{GAG}} + x_3^{\text{GAT}} \leq 2 \qquad x_2^{\text{GAA}} + x_3^{\text{GAT}} \leq 1$$

and in the corresponding matrix the coefficients of $x_1^{\text{TTA}}, x_2^{\text{GAA}}, x_3^{\text{GAT}}$ form an odd circulant with determinant ± 2. □

Inequalities (8) can actually be enforced by

$$-y_i^\mu + \sum_{k \in K_i^\mu} \sum_{c \in C_k^\mu} x_k^c \leq |K_i^\mu| - 1 \qquad i = 1, \ldots, 3n;\ \mu \in U_i \tag{10}$$

Theorem 4.7. *For any given motif μ, inequality (10) is obtained by any sequential lifting in (8) of variables $x_k^c, k \in \{s(i), t(i)\}, c \in C_k^\mu$.*

Proof. Write (8) for $a \in C_{s(i)}^\mu$, $b \in C_{t(i)}^\mu$, and let $a' \in C_{s(i)}^\mu$, $a' \neq a$ using $r = |K_i^\mu| - 1$ for the right-hand side to simplify notation. Add then variable $x_{s(i)}^{a'}$ multiplied by the lifting coefficient α to the left-hand side:

$$-y_i^\mu + x_{s(i)}^a + \alpha x_{s(i)}^{a'} + \sum_{k=s(i)+1}^{t(i)-1} \sum_{c \in C_k^\mu} x_k^c + x_{t(i)}^b \leq r \tag{11}$$

By assignment condition (3), $x_{s(i)}^{a'} = 1$ implies $x_{s(i)}^q = 0$ for any $q \in C_{s(i)}^\mu$, $d \neq a'$. To enforce (11) we then choose the largest α compatible with the latter position, that is

$$\alpha = \max \left\{ r + y_i^\mu - x_{s(i)}^a - \sum_{k=s(i)+1}^{t(i)-1} \sum_{c \in C_k^\mu} x_k^c - x_{t(i)}^b \,|\, x_{s(i)}^q = 0, q \neq a' \right\} =$$

$$= \max \left\{ r + y_i^\mu - \sum_{k=s(i)+1}^{t(i)-1} \sum_{c \in C_k^\mu} x_k^c - x_{t(i)}^b \right\}$$

But, due to the assignment constraints,

$$- \sum_{k=s(i)+1}^{t(i)-1} \sum_{c \in C_k^\mu} x_k^c \geq 1 - r$$

hence

$$\max \left\{ r + y_i^\mu - \sum_{k=s(i)+1}^{t(i)-1} \sum_{c \in C_k^\mu} x_k^c - x_{t(i)}^b \right\} \geq \max\{y_i^\mu + 1 - x_{t(i)}^b\} = 1$$

Similarly we obtain the lifting coefficients of variables $x_{t(i)}^{b'}, b' \in C_{t(i)}^\mu$, $b' \neq b$. It is easy to see that the lifting procedure returns the same coefficient $\alpha = 1$ whatever is the order in which variables are lifted. □

4.2. Model enrichment

The mathematical form of problem (2)-(9) allows a much finer control on decision making than Condon and Thachuk (2012), and can help avoid possible drawbacks of optimized DNA/RNA sequences, such as those pointed out in (Charneski and Hurst, 2013; Mauro and Chappel, 2014).

For example, instead of maximizing the CAI one can be interested in approaching the wild types distribution as much as possible, see e.g. Consortium (2017). Let f^c be the relative wild type frequency of codon $c \in \mathcal{B}^3$ encoding for amino acid $a \in \mathcal{A}$. The approximation error is defined by slack and surplus variables $u^c, v^c \in \mathbb{R}_+$ associated with c:

$$\frac{1}{n_a}\sum_{k=1}^{n} x_k^c + u^c - v^c = f^c$$

where $n_a = |\{k \in \mathbb{Z} : 1 \le k \le n, \pi_k = a\}|$, and one seeks to minimize $\sum_{c\in\mathcal{B}^3}(u^c + v^c)$.

One can also weigh motifs differently according to some preference: to do so, one has to simply generalize constraints (5), (7) to knapsack inequalities using motif weights $p^\mu \in R_+$

$$\sum_{i=1}^{3n}\sum_{\mu\in D_i} p^\mu y_i^\mu \ge d \qquad \sum_{i=1}^{3n}\sum_{\mu\in U_i} p^\mu y_i^\mu \le u$$

or, in alternative, add/subtract the weighted left hand sides to/from the objective function.

Motifs can also be desired or undesired with different grades, and one can permit no more than — or require at least — a given number of motifs per grade. In this case D and U are partitioned into subsets corresponding to grades, and one can write an inequality of the form (5), (7) for each subset. A combination of weights and grades is also possible.

Hierarchical preferences on desired motifs can be expressed as

$$\sum_{i=1}^{3n-|\mu|+1} y_i^\mu \le \sum_{i=1}^{3n-|\nu|+1} y_i^\nu$$

which implies that motif $\mu \in D$ cannot be found at any point in σ unless some other motif $\nu \in D$ is in turn present somewhere (or perhaps in some specified zone). Let us call COME$_1$ this extended version of Problem 2.1.

One can further enrich the model features by taking advantage of motif position-indexing: in fact, special sub-structures (e.g., hairpins) may not correspond to fully known base sequences, but can contain "don't care" which can be indirectly modeled by controlling absolute or relative positions of motifs. To this purpose, one can add such special constraints as

$$y_i^\mu + y_j^\nu \le 1 \qquad \mu, \nu \in M, \forall i, j : a^{\mu\nu} \le |i - j| \le b^{\mu\nu}$$

that prevent σ from containing motifs μ, ν within a certain range $[a^{\mu\nu}, b^{\mu\nu}]$ of reciprocal positions, whatever are the intermediate bases. Let us refer to this extension as COME$_2$.

Let us explain how COME$_1$ and COME$_2$ intervene in practical genome design.

1. A restriction enzyme is a reagent that cuts DNA at specific patterns. For instance, the enzyme EcoRI cuts at the pattern GAATTC. According to Montes et al. (2010), over 3000 of such enzymes have been studied in detail, and over 600 are commercially available today. A *restriction site* is an occurrence of an enzyme-specific motif, and if unique in σ, it allows to cut the sequence at a certain point and only there. Having many unique restriction sites regularly placed in σ is therefore a feature that helps sequence manipulation, but keeping the sites positional distance under a desired threshold δ was proved to be $\mathcal{NP}$-hard (Montes et al., 2010). This is a COME$_1$-type problem with

$$\sum_{\mu\in D} y_i^\mu \le \sum_{\mu\in D}\sum_{j=i+1}^{i+\delta} y_i^\mu$$

for all positions i, where D is the set of motifs associated with the enzyme considered, with the further condition

$$\sum_{i=1}^{3n-|\mu|+1} y_i^\mu \le 1 \qquad \forall \mu \in D$$

requiring the uniqueness of motif occurrences.

2. *Hairpins* are formed by complementary motifs, namely pairs $\mu, \bar{\mu}$ with $|\mu| = |\bar{\mu}|$, characterized by relatively low levels of free energy that can cause the string to fold. The strongest chemical link is G≡C, followed by the double link A=T. Thus, in a string, G,C and A,T are complementary nucleotide pairs, and string folding can occur when complementary sub-sequences are read (at an appropriate reciprocal distance) in opposite directions: for instance, $\mu =$ TACGGCT, $\bar{\mu} =$ AGCCGTA. To prevent such a folding one can use a special form of COME$_2$:

$$y_i^\mu + y_{i+p}^{\bar{\mu}} \le 1 \qquad p \in [a, b], \mu \in M_i$$

where $[a, b]$ denotes the range of distances for which the folding probability is somehow significant.
For instance, the restriction sequence GCC$\cdot\cdot\cdot\cdot$GGC (BglI) is found with high-frequency in Homo Sapiens and recognized by bacterial endonucleases. Because of the five undetermined bases, forbidding this configuration would require DP to forbid 1024 distinct motifs. With IP this is instead easily obtained by writing the above constraints for $p = 5$ and just for the variables that are i-compatible with π.
In general, such secondary structures can be detected by dynamic programming in the current optimal sequence σ^*, possibly based on an estimation of the total energy of the relevant electrochemical links. Inequalities are then generated in a lazy way and the sequence re-optimized.

Hierarchical preferences or incompatible motif pairs separately make Problem 2.1 $\mathcal{NP}$-hard. For details see the Appendix, Section 7.

5. Numerical experience

In this section we report on a computational experience carried out with model (2)–(9) enforced by (10) on a testbed of Coding DNA Sequences (CDS) taken from two data sets:

- **Dataset I**. A filtered set based on the 3,891 CDS regions of the GENEC-ODE subset of the *Encode* data set (Consortium, 2004) (version hg17 NCBI build 35) and used in the computational test of Condon and Thachuk (2012). This set is further elaborated by framing the sequences and eliminating those with intermediate stop codons. As a result, 3,154 sequences are eventually obtained, with lengths ranging from 72 to 3,106 bases.
- **Dataset II**. CDS list from GRCh38.p10 assembly (Genome Reference Consortium, Consortium (2017)), containing coding sequences for *Ensembl* or "ab initio" predicted genes. The original list contains 52,848 CDS, but with much redundancy (that is, base sequences that encode the same protein). Filtering that redundancy, one obtains 20,376 coding genes. We further reduced this amount by suppressing sequences with ≤ 72 bases or with intermediate stop codons, and a few ones with "don't care" elements. The final set consists of 10,223 sequences with lengths between 72 and 4,696 bases.

From each CDS we obtain a target protein by associating, as per Table 1, amino acids with consecutive base triplets. Problem sizes in the two data sets are distributed as in Figs. 1 and 2 (sequence lengths on the horizontal axis). As in Condon and Thachuk (2012), codon fitness numbers τ_c (and hence the CAI) were computed from the frequencies in http://www.kazusa.or.jp/codon/cgi-bin/showcodon.cgi?species=155864.

Table 2
Dataset I, CPU time of dynamic vs. integer programming.

	Dynamic Programming					Integer Programming				
L_{max}	mean	σ	min	median	max	mean	σ	min	median	max
12	0.013	0.017	$<$ 0.001	0.010	0.230	0.006	0.012	$<$0.001	0.003	0.231
15	0.071	0.093	$<$ 0.001	0.050	1.260	0.006	0.012	0.001	0.003	0.242
18	0.387	0.469	0.080	0.260	6.550	0.006	0.012	0.001	0.003	0.233
21	2.028	2.481	0.480	1.355	34.500	0.006	0.012	0.001	0.003	0.229
24	10.929	13.298	2.830	7.300	181.120	0.006	0.012	0.001	0.003	0.232

Table 3
Dataset II, CPU time of dynamic vs. integer programming.

	Dynamic Programming					Integer Programming				
L_{max}	mean	σ	min	median	max	mean	σ	min	median	max
12	0.033	0.029	$<$ 0.001	0.030	0.470	0.015	0.027	$<$0.001	0.010	1.387
15	0.182	0.156	0.010	0.150	2.610	0.015	0.026	$<$0.001	0.010	1.415
18	0.883	0.760	0.070	0.730	13.760	0.015	0.028	$<$0.001	0.010	1.612
21	4.665	4.035	0.470	3.830	75.040	0.015	0.026	$<$0.001	0.010	1.457
24	25.727	23.421	2.740	21.160	407.740	0.015	0.026	$<$0.001	0.010	1.412

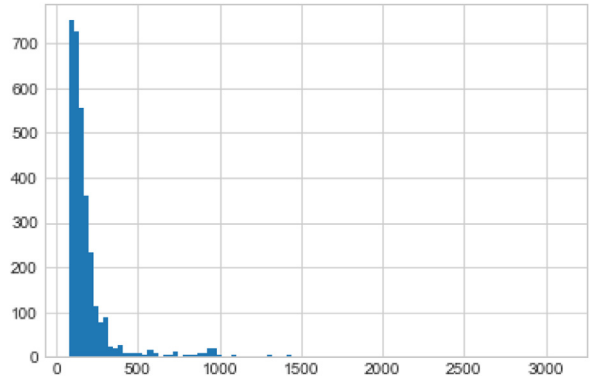

Fig. 1. Dataset I, distribution of sequence length.

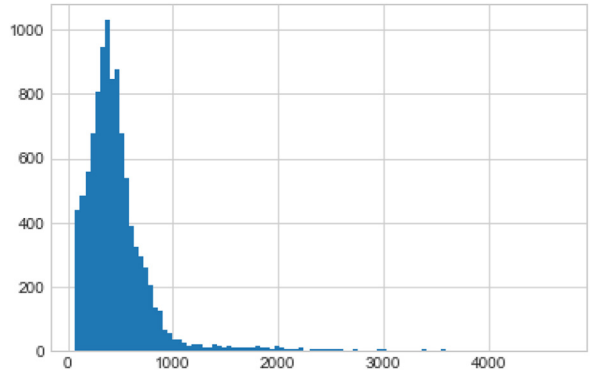

Fig. 2. Dataset II, distribution of sequence lengths.

We carried out experiments with different forbidden and desired motif sets to compare the behavior of our integer programming model (IP) to that of the dynamic programming algorithm (DP) proposed in Condon and Thachuk (2012), downloadable from http://www.cs.ubc.ca/labs/beta/Projects/codon-optimizer. Data sets are available from the website http://optimization.disim.univaq.it/codon-opt/. The code of our algorithm can be freely downloaded from https://github.com/fabrros/codon-opt.

In our experiments we maintained the desired set as in Condon and Thachuk (2012), that is:

$D =$ AACGTT, AACGTTCG, ACGCGT, AGCGCT, GACGTC, GACGTT, AACGAT, AACGCT, AGCGTT, ATCGAT, CACGTG, CACGTT, CTCGAC, CTCGCA, CTCGTA, GACGAT, GACGCT, GACGTA, GACGTG, GGCGTT, GTCGAC, GTCGAT, GCTGCT, GTCGTC, GTCGTT, TACGTA, TACGTT, TGACGTT, TGTCGCT, TGTCGTT, TCAACGTT, TCGCGA.

but we modified the undesired set by increasing to 4 nucleotide bases the length of the shortest motifs and by adding restriction sequences found with high-frequency in Homo Sapiens and recognized by bacterial endonucleases, namely GGGCCC (ApaI), CCTAGG (AvrII), CCTCAGC (BbvCI), TTTAAA (DraI), GCATGC (SphI), AATATT (SspI), TCTAGA (XbaI), TTATTTATT (RNA destabilizing sequence element), GAATTC (EcoRI), AAGCTT (HindIII), GAGCTC (SacI), plus the instance GCCATAGCGGC of GCC$\cdot\cdot\cdot\cdot$GGC (BglII), where $\cdot$ denotes any nucleotide. This leads to the following list of undesired motifs:

$U =$ CCGG, CGGG, CGCGCG, GCGCGC, GGGG, CCCC, TTTT, AAAA, GACTC, GAGTC, GGGCCC, CCTAGG, CCTCAGC, TTTAAA, GCATGC, AATATT, TCTAGA, TTATTTATT, GAATTC, AAGCTT, GAGCTC, GCCATAGCGGC.

The largest length of a motif in the above list is $L_{max} = 11$. To test larger lengths, for each CDS we added a further undesired motif corresponding to a basis sequence of length $L_{max} \in \{12, 15, 18, 21, 24\}$ found in the middle of the CDS (i.e., starting from position $\lfloor \frac{3|\pi|-L_{max}}{2} \rfloor$; choosing an additional motif that exists in the CDS ensures that the corresponding x, y variables exist in our model).

As designed and used in Condon and Thachuk (2012), DP operates a hierarchical optimization by first minimizing undesired motifs, then maximizing desired ones, and finally maximizing the CAI. Thus, for a fair algorithm comparison, we run IP in the same hierarchical way, that is, first we minimize the number of undesired motifs getting u^*, then we maximize the number of desired motifs by fixing the number of undesired motifs to u^* and, at the end, we maximize the CAI with u^* and d^* fixed. It is worth mentioning that this hierarchical multi-objective optimization is directly handled by the ILP solver we use (Gurobi 8.1.1, www.gurobi.com). Tests were conducted on a machine Intel Xeon E5-2698 v4 clocked at 2.20GHz with 256 GB RAM. Gurobi runs with default settings but the number of threads is limited to 4 to simulate a standard desktop computer. We compare the CPU (wall clock) time required for solving each CDS instance disregarding the time to setup both the DP algorithm and the IP model, that can be considered negligible.

The test summary is reported in Tables 2, 3 and Figs. 3, 4. Tables are arranged in rows corresponding to problem groups, where a group contains problems referred to the whole data set (I or II) with L_{max} up to the value in column 1. Columns 2 to 6 report the

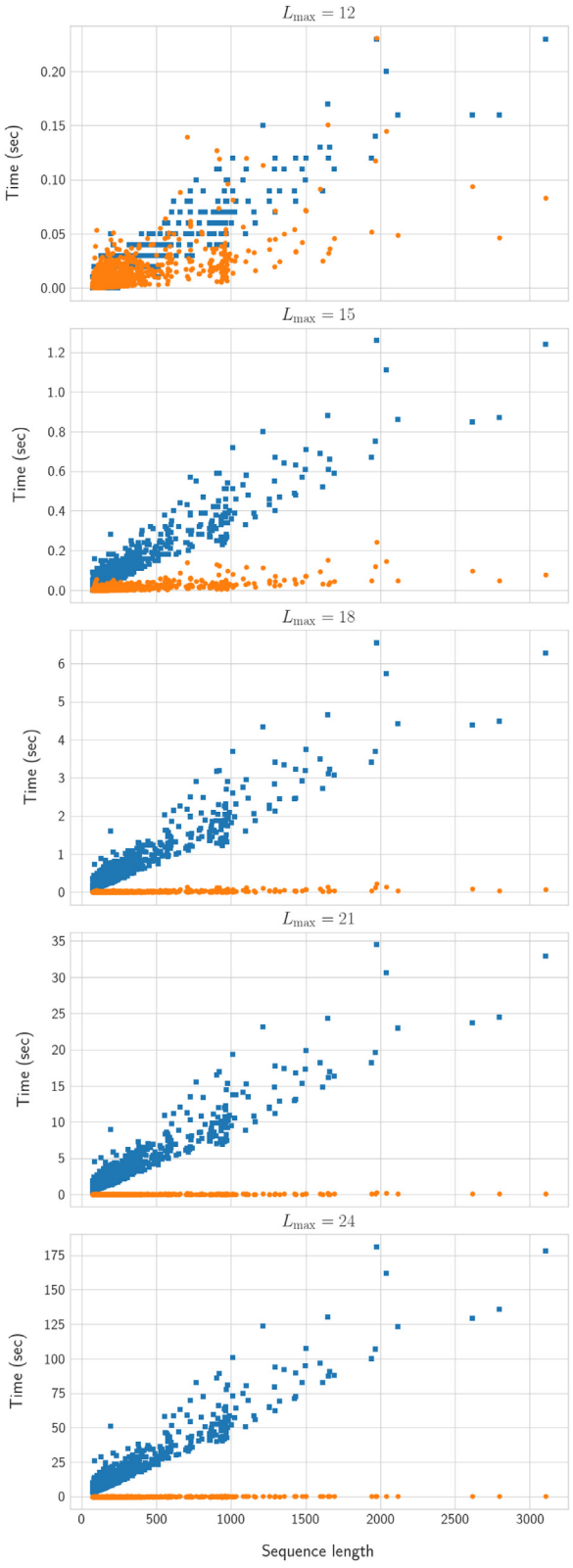

Fig. 3. Dataset I: DP (blue, square) vs. IP (orange, circle).

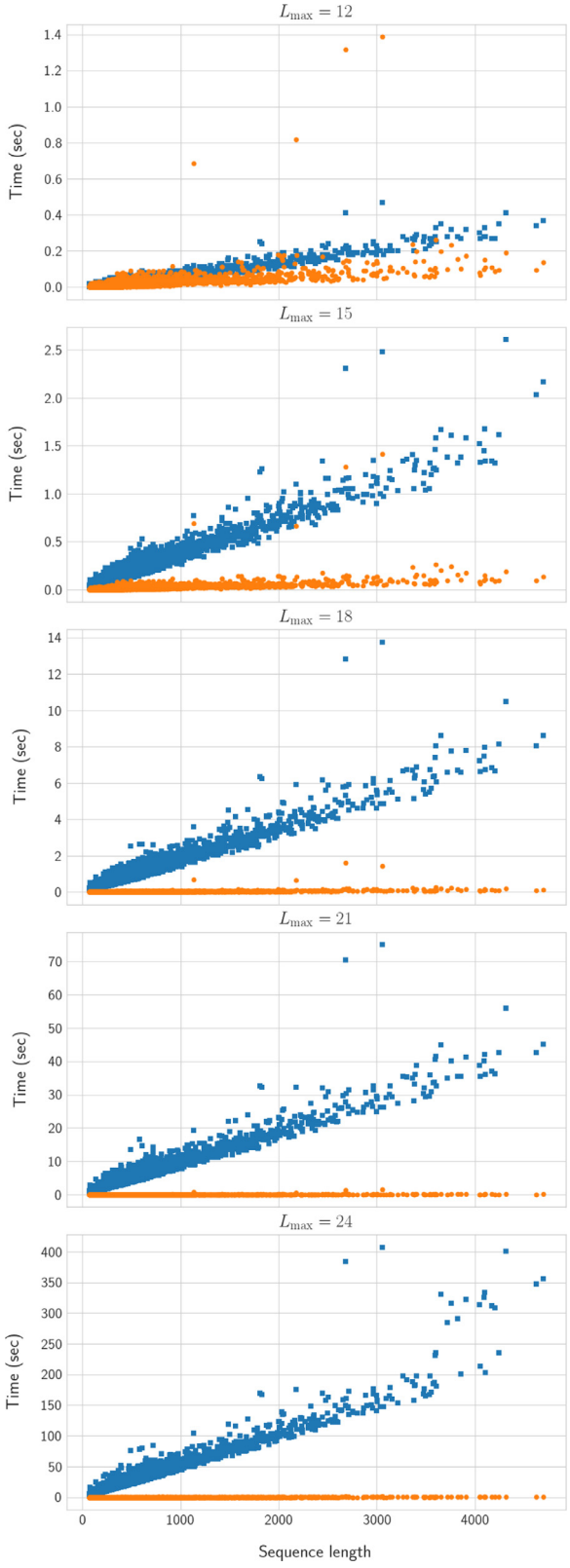

Fig. 4. Dataset II: DP (blue, square) vs. IP (orange, circle).

mean, standard deviation σ, minimum, median and maximum CPU time with DP. Columns 7 to 11 give the values observed with our IP model (2)–(9) enforced by (10).

Distributions of CPU time are shown in Figs. 3 and 4 grouped by $L_{\max}$ (from 12 to 24). Each dot represents a problem solved, the x-axis reports the number of nucleotide basis, while CPU time is indicated in the y-axis. The blue squared dots represent DP while the orange circle dots refer to IP.

Looking at the experiment outcome, we observe that:

- Both methods find an optimal solution in all the problems tested.

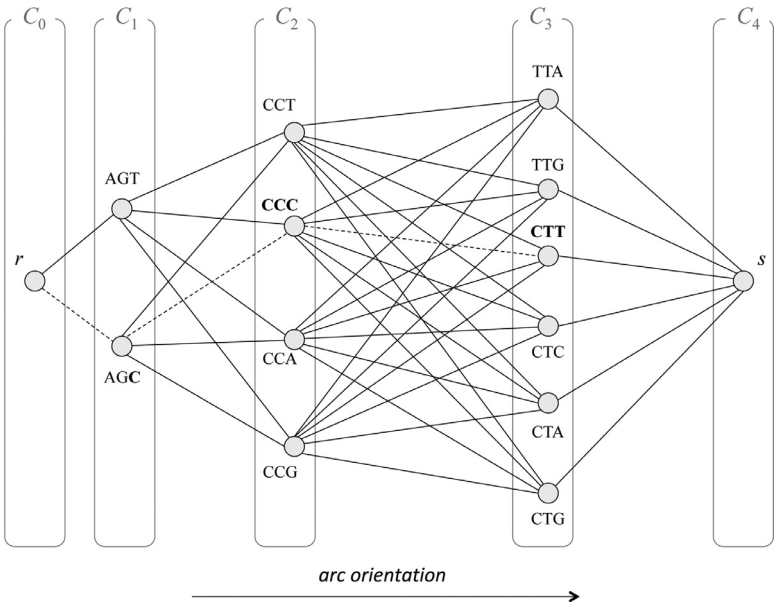

Fig. 5. Digraph G constructed after π = *Serine, Proline, Leucine*; the motif $\mu = \text{CCCCCTT}$ originates the dashed path.

- With IP, CPU time is always very small, never exceeding 0.3 seconds in Dataset I and 1.7 seconds in Dataset II. Problems with this performance are just rare outliers: both average CPU time and standard deviation amount in fact to fractions of seconds.
- With DP, CPU time remains comparable to IP only for motifs with length 12 and, as well as standard deviation, sensibly diverges with $L_{\max} \geq 21$ in both data sets. The worst performance in Dataset I (resp., Dataset II) is with over 3 (resp., 6) minutes CPU time.
- CPU time scales linearly with CDS length for DP, and is apparently insensitive to CDS length for IP. Note however that, for visual comparison of DP and IP, charts for longer $L_{\max}$ use a larger time scale than shorter ones. On average, a very moderate linear increase with sequence length can in fact be observed also for IP. For DP, instead, CPU time increase (roughly) by a factor of 5 when $L_{\max}$ is increased by 3. On the contrary, IP times does not seem to vary with $L_{\max}$.
- DP outperforms IP only in a few cases, concentrated in instances with $L_{\max} = 12$. An explanation for this behavior is that the LP model has a matrix size that cannot be compressed further. However, the very small values involved (0.3 seconds for Dataset I and 1.4 seconds for Dataset II) make this DP dominance not meaningful in practice.

Finally, by running the DP code on instances with $L_{\max} = 24$, we observed a memory peak requirement of about 74 GB (resp., 176 GB) for the largest instances in Dataset I (resp., Dataset II). Such a memory usage may prevent users from running DP in instances with fairly long CDSs and $L_{\max} \geq 24$. On the contrary, the memory peak of IP in the whole testbed never exceeds 120 MB.

As a whole, the experiments took 101h:25':15", of which 101h:11':04" (99.8%) dedicated to DP and only 14':11" to IP. The IP method was in other words over 400 times faster than DP, and required about 1,400 times less memory in the worst cases. All in all, the tests carried out indicate therefore that integer programming is a much faster and more reliable way to solve the codon optimization problems here addressed.

6. Conclusion

The problem of codon optimization with motif engineering originally formulated in Condon and Thachuk (2012), and solved there by dynamic programming was tackled by 0-1 linear programming in this paper. The model we developed has nice properties both in terms of mathematical structure and potential extensions. In a numerical comparison, our solution method behaves very well in absolute terms, and outperforms the competing DP software especially for increasing motif length: in fact, the time complexity of dynamic programming is exponential in the largest motif length, while the CPU time of our method appears very well scalable with both problem size and motifs length. In particular, the speedup and memory requirements of our approach make it possible to extend the hierarchical search of dynamic programming and explore the whole Pareto-efficiency region of the problem with respect to the three goals studied (number of desired/undesired motifs and CAI). These features of the model suggest a good potential for more challenging bio-chemical and genetic applications, such as its use in simulation environments or to establish complex proteomic benchmarks for *in-vitro* experiments.

7. Appendix: Some complexity results

Let us restate problem COME and its variants COME_1 and COME_2 of §4.2 in terms of Constrained Longest Path on a special digraph G constructed from π. The node set is the union of disjoint subsets C_k, $k = 0, 1, \ldots, n, n+1$, with $C_0 = \{r\}, C_{n+1} = \{s\}$, and with each of the remaining C_k corresponding to an amino acid π_k and containing one node per codon. Arcs are directed from all the nodes in C_k to all those in C_{k+1}, $k = 0, \ldots, n$, thus G turns out to be $(n+2)$-layered and acyclic. Fig. 5 shows the graph corresponding to π = (*Serine, Proline, Leucine*).

A subsequence of nucleotides creates in G a set of paths: for instance, motif $\mu = \text{CCCCCTT}$ starting in position 3 (boldface nucleotides in Fig. 5) forms the path P indicated with dashed arcs. Each path can be found in polynomial time in motif and protein length, and can be desired or undesired as the motif and the position from which the motif originates. Weighting by (1) each arc of G except those ending in s, an optimal codon assignment for a COME problem is then a longest (r, s)-path in G that has no more than u undesired sub-paths and at least d desired ones. Similarly, one can formulate problems COME_1 and COME_2 as Constrained Longest Path in G subject to desired/undesired sub-path pairs.

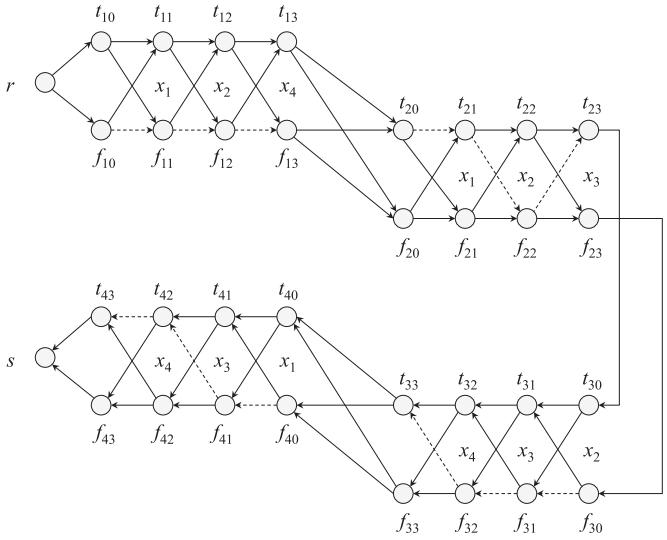

Fig. 6. Graph G for $\varphi = (x_1 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4)$: dashed subpaths are forbidden to satisfy clauses, forbidden arc pairs to enforce truth assignment consistency are not indicated.

Proposition 7.1. *Both COME$_1$ and COME$_2$ polynomially reduce from 3-SATISFIABILITY.*

Proof. Let $\varphi = (\ell_{11} \vee \ell_{12} \vee \ell_{13}) \wedge \ldots \wedge (\ell_{m1} \vee \ell_{m2} \vee \ell_{m3})$ be a Boolean formula in conjunctive normal form. Each literal ℓ_{ij} is a directed or negated logical variable x_k or $\neg x_k$ for some $k = 1, \ldots, n$, in which case we say that x_k occurs in ℓ_{ij}. For each clause $\varphi_i = \ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$ define a graph G_i with four node pairs t_{ij}, f_{ij}, $j = 0, \ldots, 3$. Four directed arcs connect t_{ij}, f_{ij} to $t_{i,j+1}, f_{i,j+1}$ in all possible ways for $j = 0, 1, 2$. Using the above components G_i construct then a new digraph G by adding two nodes r, s, four arcs from r to t_{11}, f_{11} and from t_{3m}, f_{3m} to s, and four arcs from t_{i3}, f_{i3} to $t_{i+1,1}, f_{i+1,1}$, $i = 1, \ldots, m-1$ (see Fig. 6).

Let us stipulate that, for $j = 1, 2, 3$, an arc entering node t_{ij} (node f_{ij}) corresponds to assigning value TRUE (value FALSE) to the variable occurring in ℓ_{ij}. An (r, s)-path P of G corresponds to a consistent truth assignment if and only if for any two literals ℓ_{ij}, ℓ_{hk} associated with the same variable x_k (with complementary variables x_k, $\neg x_k$), P does not contain an arc entering t_{ij} if it also contains one entering f_{hk} (entering t_{hk}), or an arc entering f_{ij} if it also contains one entering t_{hk} (entering f_{hk}). Moreover, clause φ_i is satisfied by the assignment corresponding to P if and only if P does not contain the subpath associated with the only assignment that falsifies φ_i (dashed arcs in Fig. 6).

In conclusion, ϕ is satisfiable if and only if G has an (r, s)-path that does not include special sub-paths or sub-path pairs. The introductory reformulation of COME$_1$ and COME$_2$ is a generalization of this problem, so the proof is completed. □

Of course we do not expect to construct the reduction of Proposition 7.1 via a *real* codon optimization problem. Moreover, whilst COME$_1$ and COME$_2$ are $\mathcal{NP}$-hard, to the best of our knowledge the computational complexity of COME as stated in Problem 2.1 is open — although its mathematical structure appears indeed non-trivial. In fact, shortest path problems with forbidden transitions have been proved $\mathcal{NP}$-hard for several classes of graphs Kanté et al. (2016); Szeider (2003). However, we observe that COME can be efficiently solved in special cases:

Proposition 7.2. *COME can be solved in polynomial time when just constrained to desired motifs that can repeatedly occur in σ.*

Proof. Let d be the minimum number of desired motifs in a solution σ. Construct graph $G = (V, E)$ as in Fig. 5, together with the set $\mathcal{D}$ of its paths corresponding to motifs in D that start in any position. Then elaborate a new path set $\widetilde{\mathcal{D}} \supseteq \mathcal{D}$ by recursively concatenating the paths of $\mathcal{D}$, so that if $P', P'' \in \widetilde{\mathcal{D}}$ share an arc, $P' \neq P''$, and $P = P' \cup P''$ is a path, then $P \in \widetilde{\mathcal{D}}$. Clearly, the construction of $\widetilde{\mathcal{D}}$ can be carried out in time polynomial in $|\pi|$, $|D|$ and in the largest motif length.

Next, expand G by creating $d+1$ copies $v_0, v_1, \ldots, v_d$ of each node $v \in V$ except r, s. The expanded graph $G_{exp} = (V_{exp}, E_{exp})$ will contain arc $u_i v_i$ for any $uv \in E$ not originating in r or ending in s, plus all arcs of the form rv_0, $v_d s$. Moreover, if P is a (u, v)-path of $\widetilde{\mathcal{D}}$ associated with h desired motif, then $u_i v_{\min\{i+h,d\}} \in E_{exp}$ for all i. Finally, we weight arcs $u_i v_i$ as in G and give the remaining arcs the weight of the associated path in G.

By construction, G_{exp} is acyclic and can be constructed from π, D in time polynomial in the protein and the largest motif length. Let Q be an (r, s)-path of G_{exp}, if any. If Q reaches node v_i, we infer that the protein encoding up to that point includes i desired motifs: in fact, if Q contains a desired (u_i, v_i)-path, we can replace it by the single arc $u_{i-1} v_i$ at the same cost. Because $v_i s \notin E_{exp}$ for $i < d$, Q necessarily contains an arc of the form $v_d s$ for some $v \in V$. Therefore, Q corresponds to a feasible encoding of π, and to an optimal one if of maximum weight. As the longest (r, s)-path of G_{exp} can be computed in polynomial time, the result follows. □

Throughout the paper we assumed that no desired motif contains any other motif of D, but the proof of Proposition 7.2 still holds under the opposite assumption. The hypothesis that a desired motif can occur with no limits in the encoding sequence σ is instead crucial for the proof given.

CRediT authorship contribution statement

Claudio Arbib: Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing. **Mustafa Ç. Pınar:** Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing. **Fabrizio Rossi:** Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing. **Alessandra Tessitore:** Conceptualization, Funding acquisition, Formal analysis, Writing - original draft, Writing - review & editing.

Acknowledgement

Claudio Arbib and Fabrizio Rossi are partially supported by the Italian Ministry of Education, National Research Program (PRIN) 2015, contract n. 20153TX-RX9. The authors are grateful to the Editor and two anonymous Reviewers for helping them to improve the value of the initial submission.

References

Charneski, C.A., Hurst, L.D., 2013. Positively charged residues are the major determinants of ribosomal velocity. PLoS Biol. 11 (3), e1001509.
Cohen, B., Skiena, S., 2003. Natural selection and algorithmic design of mRNA. J. Comp. Biol. 10 (3-4). 419-432CS03
Condon, A., Thachuk, C.,. Codon optimizer. http://www.cs.ubc.ca/labs/beta/Projects/codon-optimizer/.
Condon, A., Thachuk, C., 2012. Efficient codon optimization with motif engineering. J. Discrete Algor. 16, 104–112.
Consortium, E., 2004. The *encode* project (*enc*yclopedia of *d*NA *e*lements). Science 306 (5696), 636–640.
Consortium, G. R., 2017. http://dec2017.archive.ensembl.org/Homo_sapiens/Info/Index.
Evfratov, S.A., Osterman, I.A., Komarova, E.S., Pogorelskaya, A.M., Rubtsova, M.P., Zatsepin, T.S., Semashko, T.A., Kostryukova, E.S., Mironov, A.A., Burnaev, E., Krymova, E., Gelfand, M.S., Govorun, V.M., Bogdanov, A.A., Sergiev, P.V., Dontsova, O.A., 2017. Application of sorting and next generation sequencing to study 5-UTR influence on translation efficiency in escherichia coli. Nucleic Acids Res. 45 (6), 3487–3502.

Fuglsang, A., 2003. Codon optimizer: a freeware tool for codon optimization. Protein Express. Purif. 31 (2), 247–249.
Gao, W., Rzewski, A., Sun, H., Robbins, P.D., Gambotto, A., 2004. Upgene: application of a web-based DNA codon optimization algorithm. Biotechnol. Progress 20 (2), 443–448.
Gould, N., Hendy, O., Papamichail, D., 2014. Computational tools and algorithms for designing customized synthetic genes. Front. Bioeng. Biotech. 2 (41), 1–14.
Holm, L., 1986. Codon usage and gene expression. Nucleic Acids Res. 14 (7), 3075–3087.
Kanté, M.M., Moataz, F.Z., Momége, B., Nisse, N., 2016. Finding paths in grids with forbidden transitions. In: Mayr, E.W. (Ed.), WG 2015, Lecture Notes in Computer Science 9224, pp. 154–168.
Kofman, A., Graf, M., Bojak, A., Deml, L., Bieler, K., Kharazova, A., Wolf, H., Wagner, R., 2003. HIV-1 gag expression is quantitatively dependent on the ratio of native and optimized codons. Tsitologiia 45 (1), 86–93.
Mauro, V.G., Chappel, S.A., 2014. A critical analysis of codon optimization in human therapeutics. Trend Mol. Med. 20 (11), 604–613.
Montes, P., Memelli, H., Ward, C., Kim, J., Mitchell, J.S.B., Skiena, S., 2010. Optimizing restriction site placement for synthetic genomes. Comb. Pattern Match. Proceed. 6129, 323–337. doi:10.1007/978-3-642-13509-5_29.
Mueller, S., Coleman, J.R., Papamichail, D., Ward, C.B., Nimnual, A., Futcher, B., et al., 2010. Live attenuated influenza virus vaccines by computer-aided rational design. Nat. Biotechnol. 28, 723–726. doi:10.1038/nbt.1636.
OptimumGene,. ™ - codon optimization, US patent 8,326,547: Document identifier u. s. 20110081708 a1.
Osterman, I.A., Evfratov, S.A., Sergiev, P.V., Dontsova, O.A., 2013. Comparison of mRNA features affecting translation initiation and reinitiation. Nucleic Acids Res. 41 (1), 474–486.
Pabo, C., 1983. Molecular technology: designing proteins and peptides. Nature 301 (200).
Quax, T.E.F., Claassens, N.J., Söll, D., van der, O.J., 2016. Codon bias as a means to fine-tune gene expression. Mol. Cell. 59 (2), 149–161. doi:10.1016/j.molcel.2015.05.035.
Sandhu, K.S., Pandey, S., Maiti, S., Pillai, B., 2008. GASCO: genetic algorithm simulation for codon optimization. Silico Biol. 8 (2), 187–192.
Satya, R.V., Mukherjee, A., Ranga, U., 2003. A pattern matching algorithm for codon optimization and cpg motif-engineering in DNA expression vectors. Proc. IEEE Comp. Soc. Bioinform. Conf. 2, 294–305.
Schrijver, A., 1999. Theory of linear and integer programming. John Wiley & Sons Ltd., New York. (reprint) 266-278
Şen, A., Kargar, K., Akgün, E., Pınar, M.Ç., 2019. Codon optimization: a mathematical programming approach. Bilkent University Technical Report, under revision for Bioinformatics.
Sharp, P.M., Li, W.H., 1987. The codon adaptation index – a measure of directional synonymous codon usage bias, and its potential applications. Nucleic Acids Res. 15 (3), 1281–1295.
Skiena, S.S., 2001. Designing better phages. Bioinformatics 17, 253–261.
Szeider, S., 2003. Finding paths in graphs avoiding forbidden transitions. Discrete Appl. Math. 126, 261–273.
IDT Codon Optimization Tool, 2018. https://eu.idtdna.com/pages/education/decoded/article/using-a-codon-optimization-tool-how-it-works-and-advantages-it-provides.
Traoré, S., Allouche, D., André, I., de Givry, S., Katsirelos, G., Schiex, T., Barbe, S., 2013. A new framework for computational protein design through cost function network optimization. Bioinformatics 29 (17), 2129–2136.
Varenne, S., Lazdunski, C., 1986. Effect of distribution of unfavourable codons on the maximum rate of gene expression by an heterologous organism. J. Theor. Biol. 120 (1), 99–110.
Webster, G.R., Teh, A.Y.-H., Ma, J.K.C., 2017. Synthetic gene design – the rationale for codon optimization and implications for molecular pharming in plants. Biotechnol. Bioeng. 114 (3), 492–592. doi:10.1002/bit.26183.