Optimal Codon Pair Bias Design (Extended Abstract)

Nolan Donoghue Justin Gardin Bruce Futcher Steven Skiena
Stony Brook University
Stony Brook NY 11794-2424, USA

***Abstract*—Codon pair bias is the species-specific phenomenon that pairs of adjacent codons appear in genomes with frequencies different than would be predicted under an independence assumption, and thus is indicative of evolutionary selection. The synthetic attenuated virus engineering (SAVE) paradigm to design vaccines creates weak viruses by designing coding sequences that favor underrepresented codon pairs.**

Designing genes which achieve the absolute minimum codon pair bias with an arbitrary codon distribution is computationally difficult. In this paper, we develop new algorithms for constructing provably optimal codon pair designs under coding constraints of up to 186 codons in under one minute. We explore a variety of search mechanisms, lower bounds, and pruning strategies to optimize sequences. Our results make it possible for the first time to truly evaluate the performance of commonly used design methods, and quantify the potential improvement possible through better algorithms.

I. Introduction

Live vaccines, such as those widely-used for flu and polio, are particularly effective because such vaccines activate responses at the cellular and anti-body level. They must balance two competing objectives: remaining similar enough to the disease-causing agent to create an effective immune response, while being sufficiently attenuated (weakened) so as to not cause the disease itself.

The synthetic attenuated virus engineering (SAVE) approach to vaccine design exploits the redundancy of the genetic code to design genes which preserve the exact protein sequences of the wildtype pathogen (thus creating an effective immune response) while incorporating translational defects that result in attenuated strains. The SAVE approach has been used to create vaccine candidates for a number of pathogens, including polio [1], influenza [8], HIV [7], and Dengue [10].

Codon bias is the well-known observation that the genome of each species favors certain synonymous codons over others. Efforts to match the host cell's codon bias have had dramatic effects when designing synthetic genes that maximize cross-species expression [3]. Conversely, gene designs which feature underrepresented codons have been shown to result in attenuated viruses [9].

However, the phenomenon of *codon pair bias* has proven to be the most popular source of translational defects for SAVE [1], [8]. Codon pair bias is the species-specific observation that certain pairs of adjacent codons appear in genomes with different frequencies than would be predicted under an independence assumption, and are thus indicative of evolutionary selection. Let $f(x)$ $[f(xy)]$ denote the frequency of a specific codon/amino acid [pair] in a genome, and c_i a specific codon coding for amino acid a_i. Codon pair bias scores are then computed as

$$p(c_i, c_j) = \ln \frac{\text{observed } c_i c_j}{\text{expected } c_i c_j} = \ln \frac{f(c_i c_j)}{f(a_i a_j) \frac{f(c_i)}{f(a_i)} \frac{f(c_j)}{f(a_j)}}$$

Codon pair bias gene designs carefully position codons to maximize the frequency and potency of under-utilized codon pairs to reduce translation efficiency, that is designing the gene g for a given target protein such that

$$g = \arg\min \sum_{i=0}^{n-1} p(g_i, g_{i+1})$$

where g_i is the i-th codon of g.

The problem of optimal codon pair bias gene design for an n-residue protein under codon constraints occupies a curious place algorithmically. Without codon frequency constraints, the problem can be solved in linear time using the Viterbi algorithm, however such designs often dramatically change the codon-usage frequency from the host. In a hypothetical world of n mutually-synonymous codons with an arbitrary pair-bias matrix, design reduces to the traveling salesman problem, making it NP-complete. However, in the real world of the triplet code, dynamic programming yields an $O(n^{64})$-time algorithm. This is indeed polynomial time, but renders it impossible to design optimal genes of even modest length.

For this reason, simulated annealing is widely employed to produce good but presumably non-optimal gene designs. However, we seek designs that achieve maximum viral attenuation under the SAVE technique, and the margins between success and failure can be quite small. For example, the design of a dengue vaccine [10] needs codon pair optimization that grows well in insect cells yet is simultaneously deoptimized for mammalian cells.

In this paper, we explore the problem of better algorithms for codon-constrained codon pair gene design. The main contributions of our work are:

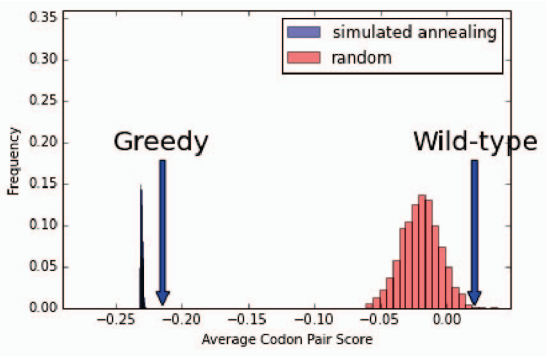
- *Novel Upper/Lower Bounds on Codon-Constrained Gene Design* – We develop interesting new pruning strategies and upper/lower bounds on constrained codon pair costs that greatly increase the search range of traditional backtracking and A* approaches.
- *Provably Optimal Codon Pair Designs* – We demonstrate that our sequence design algorithm is capable of designing provably optimal genes of up to 186 codons

978-1-5090-3050-7/17/$31.00 ©2017 IEEE

in under one minute, on a modest computer. Further, we develop a faster heuristic based on local optimization which outperforms simulated annealing on larger genes. Our implementations are available at https://bitbucket.org/ndonoghue/codonpairbias.

- *Validation of Simulated Annealing Design Heuristics* – The primary open question about codon pair design heuristics was how good were they. Did significantly better designs exist which were not being found by existing tools? Our optimal designs enable us for the first time to judge how well simulated annealing actually works. Generally speaking, not bad: achieving better than 98.6% of optimality over 100 runs on each of seven different genes.

Fig. 1. Distribution of average codon pair scores for heuristics applied to his3



II. Optimization via Heuristics

Since the problem of finding an optimal solution can quickly become intractable, previous work has turned to heuristics for SAVE designs. In our discussion we will refer to the function $s(g)$ where g is an n-codon long sequence and s computes the sum of the codon pair score for all adjacent codons in g. That is,

$$s(g) = \sum_{i=0}^{n-1} p(g_i, g_{i+1})$$

where g_i is the i-th codon of g and $p(x, y)$ is the codon pair score of x followed by y. We define $\text{swap}(g, i, j)$ to be the operation which produces a sequence h where $h_x = g_x$ for $x \notin \{i, j\}$, $h_i = g_j$, and $h_j = g_i$. We use the notation $\alpha \approx \beta$ to denote that codons α and β are *synonymous*, coding for the same amino acid.

Our notation is as follows. Let c be the cardinality of the largest set of synonymous codons, $c \leq 6$. Let $\text{len}(x)$ denotes the number of codons in a codon sequence x, and g denote the wild-type codon sequence.

Swapping two distinct, synonymous codons ensures that coding sequence continues to encodes the same protein, while maintaining the codon distribution. The greedy heuristic repeatedly swaps pairs of such codons, accepting the swap if it improves the codon pair score. In each iteration, we seek:

$$\underset{(i,j)\in\{(x,y)|0\leq x,y<\text{len}(g)\wedge g_x\approx g_y\wedge g_x\neq g_y\}}{\arg\min} s(\text{swap}(g,i,j))$$

Simulated annealing is a heuristic that draws inspiration from the thermodynamics involved in cooling a material slowly, or annealing [6]. We use the transition probability:

$$\mathbb{P}(a, \text{swap}(a,i,j), T) = \begin{cases} 0 & a_i \not\approx a_j \\ 1 & s(a) \geq s(\text{swap}(a,i,j)) \\ e^{(s(a)-s(b))/(k_B \cdot T)} & \text{else} \end{cases}$$

where $\mathbb{P}(a, \text{swap}(a, i, j), T)$ is the probability of moving from state a to state $\text{swap}(a, i, j)$ at temperature T. The value of simulated annealing is that it favours transitions which improve score, but it allows itself the freedom to make transitions in the opposite direction to avoid becoming trapped in local minima.

Our implementation of simulated annealing performs $10,000$ iterations of a geometric cooling process with initial $T := 1.0$, cooling factor 0.99, constant $k_B := 1,000$, and at least $1,000$ swaps are made at each temperature. After all iterations are complete, we perform the simple greedy heuristic.

Figure 1 demonstrates the performance of gene design heuristics on his3, a gene in *S. cerevisiae*. For this gene, most random permutations have a lower average codon pair score $(s(g)/n)$ than wild-type. Performing simulated annealing on each of these starting sequences yields a much tighter distribution, all substantially better than the greedy swap heuristic, but of unknown quality relative to the optimal design.

III. Data Set

In this paper we will perform computational experiments on several representative genes. We summarize information about the genes in Table I. Each of our seven genes are drawn from either human or yeast (*Saccharomyces cerevisiae*). The codon adaptation index (CAI) is a standard measure of the degree to which a particular coding sequence favors the most frequently used codons for each amino acid. More highly expressed genes typically have higher CAI values, but for our purposes higher CAI scores imply tighter constraints on codon distribution and hence easier design optimization. Sequence lengths are measured in codons. The *best ACPS* column records the best average codon pair score (ACPS) from 20 runs of simulated annealing on the whole gene, while *optimal ACPS* indicates the optimal score for the prefix of each gene whose length is given in the previous column. Prefix genes are defined by the initial codons of each sequence, to create a test case which can be solved to optimality in one minute for experimentation.

IV. Lower Bounds on Codon Pair Designs

For any fixed prefix x of a given gene, for an arbitrary *admissible* completion z,

$$s(xz) = s(x) + s'(x, z) \geq s(x) + \ell(x),$$

where $s'(x, z)$ is the score obtained from the last codon of x as well as all of the codons of z and $\ell(x)$ is a lower bound on the

TABLE I
GENE INFORMATION SUMMARY

Gene	Accessor/ID	CAI	Gene length	Best ACPS	Prefix length	Optimal ACPS
hbb	3043	0.751	535	-0.4302	81	-0.31110
hba1	3039	0.769	288	-0.4369	69	-0.38656
his3	YOR202W	0.689	221	-0.2322	70	-0.15615
his4	YCL030C	0.813	800	-0.2247	70	-0.17642
pgk1	YCR012W	0.826	417	-0.1103	172	-0.09130
tef1	YPR080W	0.845	459	-0.0988	186	-0.10465
adh1	YOL086C	0.819	349	-0.1001	152	-0.11358

remaining score. It follows that for any allowable sequence, v, if $s(v) < s(x) + \ell(x)$, no sequences with x as a prefix can be the solution. Pruning this subspace reduces the search time.

We evaluate four distinct lower bounds:

- *Naive bounds* – Better than the trivial lower bound of $-\infty$ is our baseline (κ indicating the set of all codons appearing in x), assuming the smallest possible pair-value in each position of the sequence:

$$\ell(x) = \Big(n - \text{len}(x)\Big) \cdot \Big(\min_{\alpha,\beta\in\{\kappa\}} p(\alpha,\beta)\Big)$$

- *The Viterbi Bound* – At each position i, we record the minimum partial score achieved by the suffix starting in this position and using codon j at position i. This value $A(i,j)$ can be computed:

$$A(i,j) = \min_{k\approx g_{i+1}, j\approx g_i} \big[A(i+1,k) + p(j,k)\big]$$

with the base case $A(n-1,j) = 0$ when $j \approx g_{n-1}$ and ∞ otherwise. This yields a bound, $\ell(x) = A(\text{len}(x) - 1, x_{\text{len}(x)-1})$ that can be computed in $O(c^2 n)$ time. Further, it can be pre-computed and stored using $O(cn)$ space permitting $O(1)$ access during search. The Viterbi algorithm was originally developed to find the most likely sequence of states in a hidden Markov model [2].

- *Dynamically Computed Viterbi Bound* – We can tighten the previous bounds by ignoring exhausted codons. Let $\eta(y,a)$ be a function which counts the number of occurrences of a codon a in a codon sequence y. Then:

$$A(i,j) = \min_{k\approx g_{i+1}, j\approx g_i, \eta(x,j)<\eta(g,j)} \big[A(i+1,k) + p(j,k)\big]$$

with the base case $A(n-1,j) = 0$ when $j \approx g_{n-1} \wedge \eta(x,j) < \eta(g,j)$ and ∞ otherwise. To gain this tighter bound, we sacrifice our ability to pre-compute. This loss can be mitigated by caching bound results.

- *Greedy Assignment* – We can also restrict codon assignment greedily within each synonymous codon class independently. Then we can compute a score using this assignment. For each codon class, we form a list of all possible codon assignments (considering all possible $O(c^2)$ neighbor assignments) for all positions corresponding to the codon class. We pre-compute and sort this list, storing it for constant-time retrieval in $O(c^3 n)$ space. Thus in $O(c^3 n)$ time, for any fixed prefix, we can greedily assign within each class as shown below.

This gives the bound:

$$\ell(x) = \sum_{d\in\{\text{codon classes}\}} \frac{1}{2} \sum_{m=0}^{|P(x,d)|} r(x,d)$$

where

$$r(x,d) = \min_{\substack{i\in P(x,d), j\in J(x,d),\\ a\in N(x,i-1), b\in N(x,i+1)}} \big[p(a,j) + p(j,b)\big],$$

$$P(x,d) = \{i | \text{len}(x) \le i < n \wedge g_i \in d\},$$

$$J(x,d) = \{j | j \in d \wedge \eta'(x,j) < \eta(g,j)\},$$

$$N(x,p) = \{a | a \approx g_{i-1} \wedge \eta(x,a) < \eta(g,a)\},$$

and $\eta'(x,j)$ is shorthand for the number of times j has been used in x and the partial completion corresponding to its codon class.

Experimentally, this lower bound frequently proves tighter than dynamic Viterbi. One can obtain a bound no less tight than either by taking the maximum of the greedy bound and the dynamic Viterbi bound. However, both greedy-based methods incur a computational cost at each search node.

Table II considers prefixes of several genes and then examines the effects of our bounds on computation time. For each bound-gene pair, we report the maximum-sized prefix computed in under a minute. These results use the backtracking search mechanism discussed in Section V for consistency and brevity but the same trends hold with the other mechanisms. These experiments and others discussed were performed on a Linux machine with 64 cores (although only 4 were used at a time) and 378 GiB of RAM.

TABLE II
MAXIMUM CODON PREFIX PROCESSED IN ≤ 1 MINUTE FOR EACH BOUND

Lower bound	hbb	hba1	his3	his4	pgk1	tef1	adh1
Naïve	43	46	47	43	85	74	75
Viterbi	56	58	59	52	106	125	91
Dynamic Viterbi	**60**	**68**	**64**	**53**	**113**	**151**	**96**
Greedy	51	49	54	51	104	92	76
Dynamic Viterbi-Greedy	**60**	62	63	**53**	107	137	**96**

Table II shows that dynamic Viterbi performs consistently better than other bounds. Thus we use the dynamic Viterbi bound in all subsequent experiments.

V. SEARCH PROCEDURES

There are two major concerns in determining a search mechanism: the order of enumeration and the implicit structure of the space. Here we first consider three variations on the former and a crucial variation on the latter:

- *Backtracking* – Here we use depth-first enumeration with bound-based pruning mechanisms and constraints to backtrack when a node was deemed undesirable [4]. If the lower bound for solutions descending from this node is worse than the optimal solution yet found, then we can safely backtrack. Let $C(x)$ be the set of candidate new

partial solutions that are children of a partial solution x. Then:

$$C(x) = \{xj | j \in \{\text{codons}\} \wedge j \approx g_{\text{len}(x)} \wedge \eta(x,j) < \eta(g,j) \wedge s(x) + \ell(xj) \leq s(u)\}$$

- $A*$ – The A* algorithm considers nodes using a best-first strategy [5].
  Each iteration of A* considers the node

$$\nu = \arg\min_{\mu} f(\mu) = \arg\min_{\mu} s(i(\mu)) + \ell(i(\mu))$$

  where $i(\mu)$ is the codon sequence corresponding to node μ. Once we have examined a node whose value equals its lower bound, we are guaranteed to have the optimal solution and can stop without considering the rest of the queue.

Our discussion until now implicitly assumed that state was encoded by a prefix of codons. But the completion cost of a prefix is completely determined by the counts of codons available for future assignment and the last codon assigned, the only codon exposed to future assigned codons. Therefore, we can consider a smaller space in which we ignore the prefix order, effectively collapsing multiple search tree nodes into one.

With the reduced search space size we can perform a breadth-first enumeration using a dynamic programming strategy to generate the set of candidate nodes at the next level of depth. Let $L(i)$ be the set of nodes to be explored at depth i. For $0 < i < n$, $L(i)$ can be generated from $L(i-1)$ as follows:

$$L(i) = \bigcup_{\nu \in L(i-1)} \{(\delta(\nu, j) | j \approx g_i \wedge k(\nu, j) < \eta(g, j) \wedge \sigma(\nu) + \lambda(\nu) \leq s(u)\}$$

Here $k(\nu, j)$ is the count of the codon j assigned in the node ν, $\delta(\nu, j)$ outputs the node formed by assigning codon j to the next position using node ν as the parent, $\sigma(\nu)$ is the score of the lowest scoring prefix associated with this node, and $\lambda(\nu)$ is a lower bound on the score to be gained by any completion of ν. The dynamic programming begins by initializing the set

$$L(0) = \{\delta(\epsilon, j) | j \approx g_0 \wedge \eta(g, j) > 0 \wedge \lambda(\epsilon) \leq s(u)\}$$

where ϵ is the empty start node. The solution to the dynamic programming is simply:

$$\arg\min_{\nu \in L(\text{len}(g)-1))} \sigma(\nu)$$

Table III shows the effects our search mechanisms have on computation time for our test genes. We see that A* and dynamic programming consistently outperform backtracking. Further, dynamic programming generally outperforms A*, and by larger margins on contested examples. Note that the maximum codon prefix varies dramatically across genes, because genes have different codon distributions. Strong codon biases result in fewer choices within each codon class, resulting in smaller search spaces.

TABLE III
MAXIMUM CODON PREFIX PROCESSED IN UNDER ONE MINUTE FOR EACH SEARCH MECHANISM USING DYNAMIC VITERBI LOWER BOUND.

Search algorithm	hbb	hba1	his3	his4	pgk1	tef1	adh1
Backtracking	60	68	64	53	113	151	96
A*	**79**	68	**70**	**72**	162	174	**156**
Dynamic programming	**79**	**69**	**70**	70	**172**	**186**	152

VI. Simulated Annealing and Optimality

Here we exploit our optimal gene design algorithm to analyze the performance of simulated annealing on codon-constrained codon pair gene design. We used simulated annealing to redesign the appropriate prefix of each of our seven genes, and compared this to the optimal design.

Details appear in the full paper, but over 100 runs for six of seven genes (all except his4), simulated annealing always achieved the optimum design. Even the worst simulated annealing design for his4 achieved 98.68% of optimality. However, it should be noted that simulated annealing can have more variance on larger, more diverse genes.

VII. Conclusion

We have explored the feasibility of optimal codon pair bias deoptimization, showing experimentally that the dynamic Viterbi algorithm offers the right balance between tightness of bound and speed of computation, outperforming its competitors by substantial margins. Our algorithms allow for perfect deoptimization of codon sequences of length 186 in under a minute, significantly larger than previous methods.

Future work could explore the effects of increased computation on our algorithms, including parallelization.

References

[1] J. R. Coleman, D. Papamichail, S. Skiena, B. Futcher, E. Wimmer, and S. Mueller. Virus Attenuation by Genome-Scale Changes in Codon Pair Bias. *Science*, 320(5884):1784–1787, 2008.

[2] Jr. Forney, G.D. The viterbi algorithm. *Proc. IEEE*, 61(3):268–278, 1973.

[3] C Gustafsson, S Govindarajan, and J Minshull. Codon bias and heterologous protein expression. *Trends in Biotechnology*, 22(7):346–53, 2004.

[4] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems, 1980.

[5] PE Hart, NJ Nilsson, and B Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[6] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):pp. 671–680, may 1983.

[7] G. Martrus, M. Nevot, C. Andres, B. Clotet, and M. Martinez. Changes in codon-pair bias of human immunodeficiency virus type 1 have profound effects on virus replication in cell culture. *Retrovirology*, 10:78–, 2013.

[8] Steffen Mueller, J Robert Coleman, Dimitris Papamichail, Charles B Ward, Anjaruwee Nimnual, Bruce Futcher, Steven Skiena, and Eckard Wimmer. Live attenuated influenza virus vaccines by computer-aided rational design. *Nature biotechnology*, 28(7):723–6, jul 2010.

[9] Steffen Mueller, Dimitris Papamichail, J Robert Coleman, Steven Skiena, and Eckard Wimmer. Reduction of the rate of poliovirus protein synthesis through large-scale codon deoptimization causes attenuation of viral virulence by lowering specific infectivity. *Journal of virology*, 80(19):9687–96, oct 2006.

[10] Sam H Shen, Charles B Stauft, Oleksandr Gorbatsevych, Yutong Song, Charles B Ward, Alisa Yurovsky, Steffen Mueller, Bruce Futcher, and Eckard Wimmer. Large-scale recoding of an arbovirus genome to rebalance its insect versus mammalian preference. *Proceedings of the National Academy of Sciences*, 112(15):4749–4754, 2015.