Streamlining the Design-to-Build Transition with Build-Optimization Software Tools

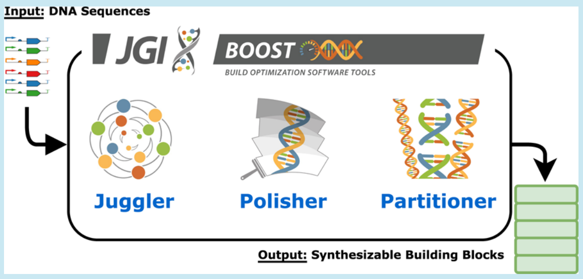
Ernst Oberortner,[†,‡] Jan-Fang Cheng,[†,‡] Nathan J. Hillson,[†,§,∥] and Samuel Deutsch*[,†,‡,∥]

[†]DOE Joint Genome Institute, 2800 Mitchell Drive, Walnut Creek, California 94598, United States

[‡]Environmental Genomics and Systems Biology Division, and [∥]Biological Systems and Engineering Division, Lawrence Berkeley National Laboratory, 1 Cycloton Road, Berkeley, California 94720, United States

[§]Fuels Synthesis and Technology Divisions, DOE Joint BioEnergy Institute, 5885 Hollis Street, Emeryville, California 94608, United States

Supporting Information


Input: DNA Sequences
JGI
BOOST
BUILD OPTIMIZATION SOFTWARE TOOLS
Juggler
Polisher
Partitioner
Output: Synthesizable Building Blocks

ABSTRACT: Scaling-up capabilities for the design, build, and test of synthetic biology constructs holds great promise for the development of new applications in fuels, chemical production, or cellular-behavior engineering. Construct design is an essential component in this process; however, not every designed DNA sequence can be readily manufactured, even using state-of-the-art DNA synthesis methods. Current biological computer-aided design and manufacture tools (bioCAD/CAM) do not adequately consider the limitations of DNA synthesis technologies when generating their outputs. Designed sequences that violate DNA synthesis constraints may require substantial sequence redesign or lead to price-premiums and temporal delays, which adversely impact the efficiency of the DNA manufacturing process. We have developed a suite of build-optimization software tools (BOOST) to streamline the design-build transition in synthetic biology engineering workflows. BOOST incorporates knowledge of DNA synthesis success determinants into the design process to output ready-to-build sequences, preempting the need for sequence redesign. The BOOST web application is available at https://boost.jgi.doe.gov and its Application Program Interfaces (API) enable integration into automated, customized DNA design processes. The herein presented results highlight the effectiveness of BOOST in reducing DNA synthesis costs and timelines.

KEYWORDS: *DNA synthesis, synthetic biology, design−build−test, bioCAD/CAM*

Ongoing sequencing efforts across a diversity of species and biomes continue to reveal large numbers of novel genes and pathways, expanding the range of hypothetical biochemical activities performed within living cells.[1−4] Synthetic biology tools enable access to these new biochemical reactions, for applications in fuel, chemical production, or cellular-behavior engineering.[5−7]

The synthetic biology engineering process involves (i) identifying relevant genetic components (e.g., coding and regulatory elements), (ii) composing selected components into genetic designs, (iii) optimizing the designed DNA sequences for function in the selected host, (iv) building the optimized synthetic DNA, and (v) transforming/transfecting the synthetic constructs into the host. Significant efforts have been devoted to automate this process through the development of biological computer-aided design and manufacture tools (bioCAD/CAM) that address different portions of this synthetic biology workflow. Sequence and design repositories (e.g., ICE,[8] IMG,[9] KEGG,[10] NCBI,[11] SBOLStack[12]), enable querying and retrieving genetic components for reuse in new designs. Software tools, such as Cello,[13] DeviceEditor,[14] GenoCAD,[15] Eugene,[16] or DoubleDutch[17] automate the composition of genetic components. Depending on the type of component, tools can also be used to (i) reverse-translate or codon optimize protein coding sequences (e.g., GeneDesign[18]), or (ii) design and analyze the sequences based on biophysical models (e.g., RBS and Operon Calculators[19−22]). The design phase of the synthetic biology process outputs DNA sequences to be manufactured, which will include the assembly of synthetic fragments typically ordered from commercial DNA synthesis providers. Tools such as j5[23] and Raven[24] can design and optimize the DNA assembly process. Software tools such as PR−PR[25] automate the execution of DNA assembly protocols by generating instructions for liquid-handling robotics and microfluidic devices. Data-exchange standards, such as the Synthetic Biology Open Language (SBOL)[26,27] have emerged for software tools to exchange design-specific information. However, multiple remaining gaps preclude the full automation of scalable synthetic biology processes.

Over the past decade, through the synthesis of many millions of DNA base pairs, and the generation of constructs ranging in complexity from single genes to entire chromosomes,[28−30]

Received: July 17, 2016
Published: December 6, 2016

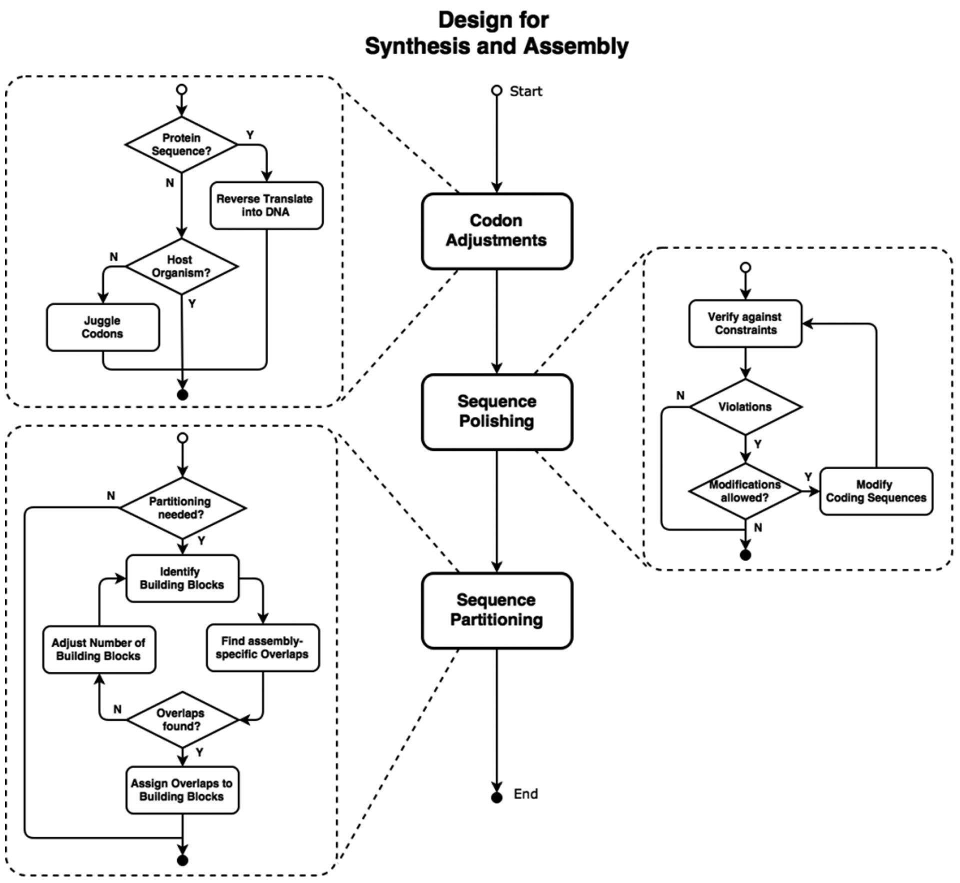


Figure 1. A flowchart illustrating the macro−microflow of the "Design for Synthesis and Assembly" workflow. Empty circles denote the start, and filled circles the end, of macro- and microflows. Rectangles with rounded corners represent tasks, and diamonds represent conditions, the evaluation of which determines the next step in the workflow.

there has been a collective realization that not all DNA sequences are equally synthesizable. Empirical success/failure data analyzed through regression and machine learning methods have enabled the identification of sequence features that significantly adversely impact the likelihood of correctly synthesizing and assembling a DNA construct, including repeats, secondary structure, and sequence stretches with extreme %GC content. Current synthesis technologies accumulate errors through oligo synthesis, amplification, and assembly,[31] limiting the number of synthesized bases before each requisite sequence verification step.

Commercial DNA synthesis vendors streamline their DNA manufacturing process by screening DNA sequences for features that are predictive of synthesis failure, which we term "DNA synthesis constraints". Different vendors have different sets of DNA synthesis constraints depending on their machine learning and data analytics as well as internal manufacturing and QA/QC processes. Sequences that violate the vendor's DNA synthesis constraints are flagged, and most vendors provide expressive violation reports through their web-based UIs for ordering synthetic DNA. Constraint violations can lead to workflow inefficiencies, as sequences may need to be substantially redesigned and/or noncompliant sequence synthesis may incur price-premiums and temporal delays.

Currently, DNA synthesis constraints are not considered as part of an automated design process through bioCAD/CAM software tools. Software tools, which provide the functionality of codon optimization, can be utilized to redesign protein coding sequences in case of violations. After every codon optimization step, the redesigned sequence must be verified against the DNA synthesis constraints. Such an approach, however, constitutes an inefficient trial-and-error process that could also unnecessarily introduce new violations or regress to previously resolved violations. Most DNA synthesis vendors offer web-based software tools that facilitate the resolution of DNA constraint violations, but this is done one sequence at a time, and assumes that sequences exclusively comprise coding DNA, which is often not the case. The integration of functionalities provided by unrelated web-based software tools into an automated design process is challenging to implement and maintain in the absence of provided APIs. In addition, ad-hoc sequence redesign poses a number of problems for tracking sequence changes that impact downstream processes including assembly, cloning, and sequence verification.

Automating the detection and resolution of synthesis constraints removes process inefficiencies and reduces the cost and turnaround times of DNA synthesis and assembly. Therefore, we have developed a suite of build optimization software tools (BOOST) for the automated generation of DNA constructs ready for DNA synthesis through commercial DNA vendors. BOOST includes tools for reverse-translation/codon juggling of sequences, detection, and resolution of DNA synthesis constraint violations in an annotation-dependent manner, and partitioning of long DNA sequences into shorter fragments optimized for assembly. The modular architecture of BOOST follows state-of-the-art software engineering principles (i) by providing a Web UI for human interactions, (ii) by providing APIs that can be invoked by other software tools, and

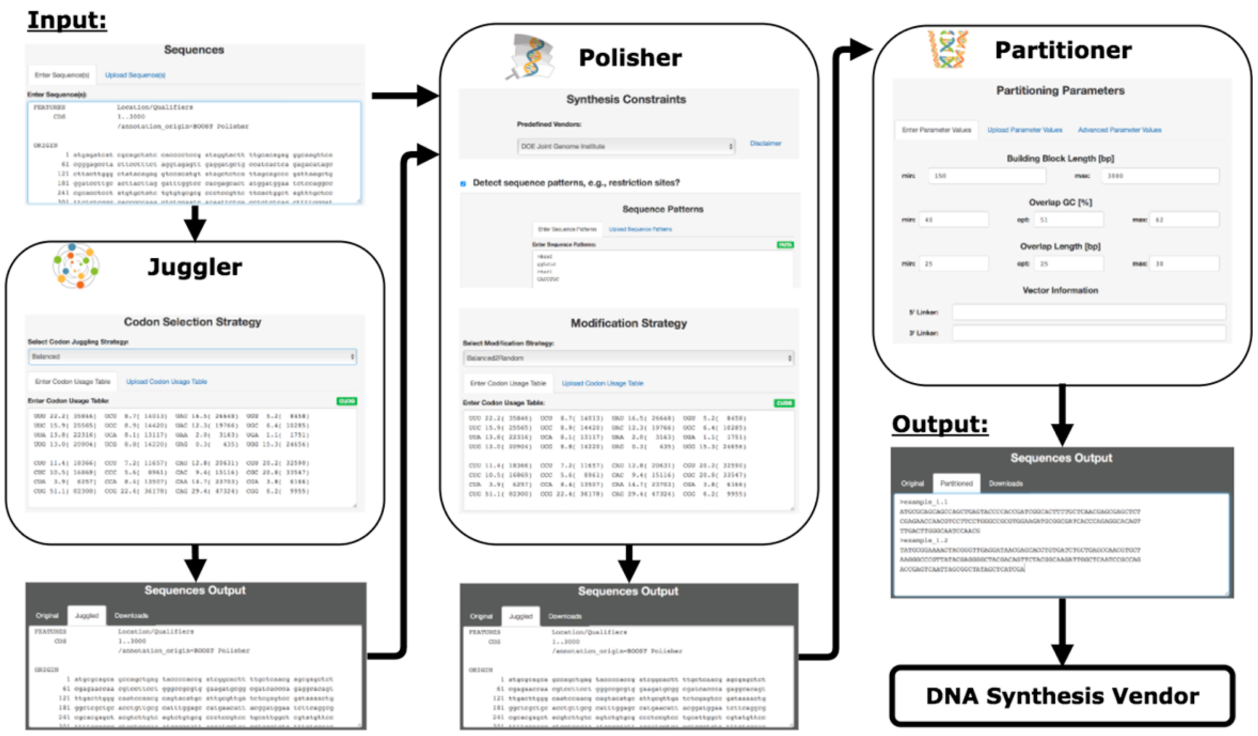


Figure 2. A walkthrough of the BOOST web application to automate the design for synthesis and assembly process.

(iii) through compliance with existing and emerging standards. The BOOST web application (available at https://boost.jgi.doe.gov) provides easy and interactive access to all BOOST functionalities. The BOOST web application builds upon a representational state transfer (REST) application program interface (API) that enables the integration of BOOST functionalities into bioCAD/CAM tools for the automated design of ready-to-order DNA constructs at minimal synthesis costs and turn-around time. Lastly, BOOST supports community standard data-exchange formats including FASTA, GenBank,[32] and SBOL.[26,27]

RESULTS AND DISCUSSION

A High-Level Design Workflow for DNA Synthesis and Assembly. The goal of BOOST is to streamline, in a scalable fashion, the process of designing readily synthesizable DNA fragments. In Figure 1, we present a flowchart illustrating the "Design for Synthesis and Assembly" workflow. Our workflow design follows the macro−microflow pattern,[33] which separates a workflow into a flow of short-running activities ("microflow") and long-running activities ("macroflow"). The macroflow of our DNA synthesis and assembly design workflow includes three activities: "Codon Adjustments", "Sequence Polishing", and "Sequence Partitioning" each of which has its own microflow process.

The type of input sequence determines the first step of the "Codon Adjustment" microflow. If the input comprises protein sequences, each protein sequence is "Reverse Translated" into a DNA according to a user-determined strategy and desired codon usage table (e.g., that of the target host organism). For each input DNA sequence not originating from the target host organism, the "Juggle Codons" step adjusts the codon usage within protein coding (sub)sequences according to a user-determined strategy and desired codon usage table. The result of the "Codon Adjustment" microflow is a set of DNA sequences (with desired codon usage).

The "Sequence Polishing" microflow starts with the 'Verify against Constraints' step, which analyzes the input set of DNA sequences for violations of DNA synthesis constraints. To resolve violations (if any) detected in modifiable protein coding regions, the "Modify Coding Sequence" step replaces codons within violation regions according to a user-determined strategy and codon usage table. The verification and modification tasks are performed iteratively until either all violations have been resolved or violations persist only in noncoding regions (Figure 1). For example, if the remaining violations occur in intergenic, intronic, or regulatory regions such as promoters, ribosomal-binding-sites, or terminators, no further sequence modifications are performed. The outcome of the "Sequence Polishing" microflow is a set of DNA sequences with a minimal number of DNA synthesis constraint violations.

Lastly, the "Sequence Partitioning" microflow checks if sequences are within a specified DNA length range. Sequences within the allowed range will not be further modified. If a sequence is too short, it will simply be flagged. For sequences exceeding the maximum length, partitioning will be performed according to a set of parameters which maximize the likelihood of successful assembly. The resulting sequences of the "Sequence Partitioning" can then be synthesized and/or ordered from a commercial DNA synthesis vendor.

Our design of the "Design for Synthesis and Assembly" workflow enables its customization depending on the type of input sequences. For example, the tasks of reverse translation and codon juggling do not have to be performed for DNA sequences that have been codon optimized for the desired target host organism. Also, not every sequence needs to be partitioned into synthesizable building blocks. Our workflow ensures, however, that every sequence is verified against DNA synthesis constraints.

BOOST Automates Design for Synthesis and Assembly. The design of DNA sequences that violate synthesis constraints impact the success rate of synthesis and can lead to workflow inefficiencies. We have therefore developed and implemented BOOST, each tool automating one step of the macroflow (Figure 1). Each tool in BOOST can be integrated with upstream bioCAD/CAM tools (via a RESTful API) in order to fully automate the process of converting the designs of DNA constructs into physical DNA molecules.

Figure 2 shows screenshots of the BOOST Web UI, and the organization of the screenshots provides a process-oriented walkthrough of the Juggler, Polisher and Partitioner tools of BOOST that respectively automate the tasks of "Codon Adjustment", "Sequence Polishing", and "Sequence Partitioning" of the design for DNA synthesis and assembly workflow (see Figure 1). As shown in Figure 2 the output sequences of one tool serve as the input of the next downstream tool.

In all three tools of BOOST, the user can enter either single sequences or a batch of sequences in CSV, Fasta, Genbank, or SBOL file format. Every tool automatically detects the file format and the sequence type. Since any DNA sequence can theoretically be a protein sequence, the user can overrule the detected sequence type. Both, the Juggler and Polisher work better with annotated input sequences, such as provided by the Genbank and SBOL sequence file formats. However, both tools allow the user to specify that all sequences are 5′-3′ in-frame, protein coding sequences exclusively if the input DNA sequences are encoded in a sequence format that does not support the annotation of sequence features, such as CSV or FASTA. Currently, only the Juggler supports the input of protein sequences for the task of reverse translation. In the Juggler and Polisher, the user must select a codon selection strategy and, depending on the strategy, needs to provide a codon usage table, for example, of the target host organism. A list of commonly used codon usage tables is provided on BOOST's Web UI of the Juggler and Polisher.

The Juggler tool in BOOST automates the "Reverse-Translate" and "Juggle Codons" tasks of the "Codon Adjustments" microflow. The Juggler outputs the reverse-translated or codon juggled input sequences in a sequence format selected by the user.

The Polisher tool in BOOST automates the "Verify Against Constraints" and "Modify Coding Sequences" tasks of the "Sequence Polishing" microflow. The user can select between two options: "Detect" and "Polish". When selecting "Detect", then the Polisher verifies the input DNA sequences against the DNA synthesis constraints of a selected DNA synthesis vendor. IDT, GeneArt-Thermo Fisher, and SGI DNA kindly provided their set of DNA synthesis constraints. The synthesis constraints of Gen9 are publicly available at http://help.gen9bio.com/docs/gene-synthesis-guidelines. Besides selecting a DNA synthesis vendor, the Polisher tool also supports a language that enables the end-user to specify customized DNA synthesis constraints. For the time being, the language supports the specification of %GC content, repeats, and length constraints. In addition, the user can enter sequence patterns, such as restriction sites, instructing the Polisher to treat them as additional constraints. When the user selects the "Polish" task, the Polisher verifies and additionally modifies the protein coding regions within the input DNA sequences. For both "Detect" and "Polish" tasks, the Polisher outputs a tree-view of all input sequences and flags those that violate the DNA synthesis constraints. Expanding the node of a violating sequence provides detailed information about the violations, such as the constraint type and the violation location. In the case of the "Polish" task, the Polisher outputs the modified sequences, a log of the performed modifications, and a tree-view to compare the constraint violations before and after the modifications.

The Partitioner tool in BOOST checks if DNA sequences are within a defined length range. The user can configure the minimum and maximum length. Sequences that are above the maximum length are decomposed into building blocks. The partitioning algorithm searches for sequence overlaps among the building blocks that are optimized for downstream DNA assembly. The user can configure the %GC content and the length of the overlap sequences to be compatible with downstream DNA assembly methods, but other features such as overlap uniqueness, and excluding the presence of hairpins and homopolymers, are hard coded into the algorithm. If the partitioning function finds overlaps that comply with the parameters for each pair of building blocks, then an output will be generated. Otherwise the user needs to adjust the partitioning parameters.

The input sequences for any BOOST functionality are sequences that are either under design or the output of upstream bioCAD tools, allowing BOOST to be integrated into pre-existing design pipelines. The modular structure of BOOST enables to customize the execution of an automated design for synthesis and assembly process depending on design-specific requirements. The automation of internal and cross-organizational workflows requires functionalities to be invoked in a programmatic manner. As a result, every BOOST functionality is accessible via a RESTful API. Further information about the RESTful API is provided on the BOOST web site (https://boost.jgi.doe.gov).

More detailed information and examples about the supported sequence formats, the codon selection strategies and algorithms for reverse-translation, codon juggling, violation resolution, and partitioning are provided in the Methods section and the Supporting Information.

BOOST Results in Efficiency Gains. BOOST has been used internally as the final design step for over 6 Mbp of synthetic DNA over the last 18 months. To evaluate potential efficiency gains derived from the implementation of BOOST, we focused on a set of 1950 sequences (1424 DNA and 526 Proteins) that were ordered from the same commercial synthesis provider over a 1-year period, for a total of 2 918 254bp. For this set, the Polisher tool in BOOST identified 546 sequences (28%, 1 194 138bp) that violate at least one synthesis constraint with the remaining 1404 sequences (1 724 116bp) being compliant.

In total, 53 543 individual violations were detected that, when merged, resulted in 2608 sequence regions that violated DNA synthesis constraints (see Methods). Out of the 546 complex sequences, the Polisher was able to resolve all violations in 447 instances (747 756bp) using the balanced-to-random modification strategy (see Methods). We analyzed the 99 DNA sequences that still contained violations after the polishing procedure. Out of these, two sequences did not contain any annotated sequence features and the BOOST polisher was not able to annotate the sequences automatically, resulting in entirely immutable sequences. Eleven sequences code proteins with long stretches of repeating amino acids, making it hard for the BOOST heuristics to find a DNA sequence that complies with the DNA synthesis constraints. The remaining 86

Table 1. Comparing BOOST's Functionalities with the Functionalities Provided at (i) the Web Portals of Commercial DNA Synthesis Providers and (ii) DNA Design Software Tools with Respect to Processing a Batch of Sequences

	DNA Synthesis Provider					DNA Design Software		
	IDT	Gen9	Thermo Fisher	Twist	DNA2.0 Gene Designer	Gene Design	Genome Calli-grapher	BOOST
Reverse Translation	Y	Y	Y	Y	Y	Y	N	Y
Codon Juggling Batch of annotated Sequences	N	N	N	N	N	N	Y	Y
Detect Constraint Violations	Y	Y	Y	Y	N	N	Y	Y
Polish Individual Sequences	Y	Y	Y	N	N	N	Y	Y
Polish Batch of annotated Sequences	N	N	N	N	N	N	Y	Y
Sequence Partitioning	N	N	N	N	N	N	N	Y
APIs	N	N	N	N	N	N	N	Y

sequences had feature annotations including both modifiable coding sequences and immutable regulatory elements. A detailed analysis of the remaining 86 sequences with constraint violations revealed that all 299 remaining violations (of the original 2608) occurred in immutable features, demonstrating that the BOOST polisher addressed the violations in the modifiable coding regions and left the immutable features unchanged. Next, we analyzed the immutable features and discovered that 69 constraint violations occurred in sequence regions annotated as promoters (promoter), 16 in origins of replication (rep_origin), 73 in terminators (terminator), 2 in miscellaneous signals (misc_signal), 12 in regions annotated as source, 8 in regions annotated as loci, 2 in recombination features (misc_recomb), and 117 in miscellaneous noncoding features (misc_feature). Then, we analyzed the 117 miscellaneous noncoding features and identified 12 promoter (incl. RBS sequences) and 23 protein coding sequences. The remaining features were homology arms required for genome editing/integration. Our closer analysis of the miscellaneous noncoding features clearly highlights the need for users to provide more precise sequence feature annotations, either manually or with the support of bioCAD/CAM tools.

For the set of 1950 sequences described above the total cost of synthesis would be \$291,825 assuming an industry average of \$0.10/bp. However, sequences that violate synthesis constraints are typically synthesized at a premium price ranging from \$0.20-\$0.50/bp depending on the sequence complexity. If we conservatively assume a \$0.20/bp cost for all fragments that violate constraints, then the total synthesis cost would increase to \$411,239.20. However, the actual synthesis cost when using the BOOST toolkit would be \$336,463.60 assuming the cost model above, which constitutes a 18% savings. In addition to cost benefits, using BOOST results in much improved cycle times. First, it eliminated any time spent in sequence redesign. Second, fragments without sequence complexities are usually delivered in about 25 business days, whereas those with complexities usually require about 40 business days and have higher failure rates. For DNA foundry operations that closely monitor cost and cycle time metrics, using BOOST can result in very significant efficiency gains.

BOOST-Related Software Tools. BOOST was developed with the main goal of providing DNA synthesis design functionalities (described above) that can operate on batches of annotated sequences, either for researchers/scientists through a Web UI and for bioCAD/CAM tools through APIs. Some individual BOOST features can be performed through existing software, however the integrated functionalities to automatically codon optimize, polish, and partition complex sequence batches in an annotation-dependent manner to produce ready-to-synthesize sequences, fill an existing gap in the design-to-build transition.

In Table 1 we compare the BOOST functionalities against the functionalities provided on the web portals of commercial DNA synthesis providers (IDT, Gen9, ThermoFisher, Twist, and DNA2.0) and DNA design tools (DNA2.0s GeneDesigner,[34] GeneDesign,[18] GenomeCalligrapher[35]). Recently, some vendors augmented their web portal with web-based DNA design software tools, such as Gen9's collaboration with Benchling (https://benchling.com/), and Twist's GenomeCompiler (http://www.genomecompiler.com/).

All web portals from DNA synthesis vendors support the identification of DNA synthesis constraints (Table 1). However, in order to resolve the violations, current tools operate on individual sequences and require (i) manual annotation of coding sequences, (ii) additional upload of the protein sequences, or (iii) that all DNA sequences exclusively comprise in-frame coding sequences. All of these options are time-consuming, and and DNA sequence modification at this stage needs to be captured and tracked appropriately which is not an easy task. BOOST takes advantage of standardized sequence formats for the specification and exchange of sequence features and annotations such as GenBank or SBOL. As such, BOOST can automatically polish within coding regions using vendor specific constraints and track all the changes as part of the design process.

To the best of our knowledge, BOOST is the first published software tool to decompose (i.e., partition) large sequences into synthesizable building blocks with sequence overlaps that can be configured to be compatible with downstream DNA assembly methods. Partitioning large sequences into synthesizable building blocks is a design-specific functionality that needs careful tracking during (automated) downstream assembly and cloning steps.

One technical requirement of BOOST was to provide APIs, making it possible to integrate its functionalities into bioCAD/CAM tools. While invoking the functionalities of a web-based portal such as those from commercial DNA synthesis providers is theoretically possible (e.g., via HTTP), the associated complications (e.g., building and sending the request, parsing and interpreting the response, secure communication, authentication mechanisms) make it hard to implement in practice.

Future Plans. During our extensive use of the current BOOST release we identified several areas that should be considered for future improvements.

The major challenge to keep BOOST's DNA synthesis constraints up-to-date with rapidly evolving DNA synthesis methods and technologies could, for example, be addressed through a language-based approach. DNA synthesis constraints can then be specified in such a language and can be exchanged in a standardized manner. In BOOST, we now provide a prototype for such a language, but this would have to be adopted by the vendors for it to constitute an effective solution. A more realistic alternative is for DNA synthesis providers to provide APIs in addition to their existing Web UIs. This would allow for, bioCAD/CAM tools, such as BOOST, to programmatically invoke these APIs to verify DNA sequences against the vendor's current synthesis constraints. At a minimum, these APIs should report the sequence region (i.e., start- and end-positions) and constraint violation type. Recently IDT has provided such a functionality, which will be invoked by BOOST in future releases. Our approach, going forward, will be to use a combination of these two alternatives to keep BOOST's DNA synthesis constraints up-to-date and consistent with DNA synthesis vendors.

Additional areas of future development are (i) development of a workflow manager that combines the BOOST functionalities for the automation of customized workflows depending on the input sequences and values, (ii) deployment of a queuing/management system enabling the use of the BOOST functionalities in an asynchronous manner, and (iii) availability of BOOST as a cloud-based infrastructure.

■ METHODS

Sequence Exchange Formats. One challenge in the exchange of sequences is to support the different types of input sequences (i.e., DNA, RNA, or protein) represented in commonly used and standardized exchange formats. Every tool in BOOST supports the input and output of sequences in FASTA, GenBank,[32] Synthetic Biology Open Language (SBOL),[26,27] and a simple Comma Separated Values (CSV). The import and export of sequences represented in FASTA and Genbank is implemented using the BioJava library.[36] BOOST integrates libSBOLj[37] to input and output SBOL files.

BOOST prefers Genbank and SBOL sequence exchange formats, which support sequence feature annotation, because coding sequence (e.g., CDS) features cue BOOST as to which DNA sequence regions may be altered without changing their corresponding translated protein products (see below; the user is responsible for ensuring that coding sequence regions have been properly annotated). While BOOST also supports sequence specification via CSV or FASTA, these formats do not support feature annotations and the user must enable the 'Auto-Annotate' feature that annotates each sequence as a coding sequence ("CDS") if its length is a multiple of 3bp in order to take advantage of BOOST's codon juggling and sequence polishing features (see below).

Reverse-Translation and Codon Juggling. Reverse-translation (also known as back-translation) is the process of mapping a given protein sequence back into one of several possible RNA or DNA sequences that could encode it. That is, reverse-translation is the inverse process of translating an RNA or DNA sequence into its corresponding protein sequence. Following the standard genetic code, each 3-nucleotide RNA or DNA sequence ('codon') is translated into a specific amino acid (or possibly a translation stop signal). Since there are 64 codons (4^3) but only 20 standard natural amino acids, multiple codons may encode the same amino acid. For example, there are two codons (TTT and TTC) for phenylalanine and four codons (TCT, TCC, TCA, and TCG) for serine. Consequently, there are multiple valid reverse-translations of the same protein sequence.

The process of "codon juggling" is useful when transferring coding genes between organisms. Two organisms may have different preferences for using particular codons to encode the same amino acid. Consequently, DNA sequences that encode the same protein will likely differ across organisms with distinct codon usage preferences.

Genome sequencing reveals to what extent an organism preferentially uses certain codons over others to encode a given amino acid. Codon usage tables suggest to what extent a given organism may prefer one putative valid DNA sequence to another to encode the same protein. To the best of our knowledge, there is no standardized format for the specification and automated exchange of codon usage tables. Therefore, BOOST supports the following formats:

1. The Relative Synonymous Codon Usage (RSCU) format,[38] as used and generated by the GeneDesign toolkit (http://genedesign.jbei.org),[18]
2. a two-column comma-separated values (CSV) format, in which the first column contains the codon and the second contains its usage value
3. both formats provided by the Codon Usage Database (http://www.kazusa.or.jp/codon/)
    a. [triplet] [frequency: per thousand] ([number])
    b. [triplet] [amino acid] [fraction] [frequency: per thousand] ([number])

Regardless of the format, the codon usages are normalized so that the sum of codon usages is 1.00 for every amino acid.

BOOST uses these codon usage tables to guide the codon selection process. BOOST reverse-translates protein sequences one amino acid at a time. BOOST provides three codon selection strategy options:

- **Random:** For each amino acid, select a codon randomly.

- **Mostly Used:** For each amino acid, select the codon with the highest codon usage.
- **Balanced:** For each amino acid, distribute usage across codons so that the resulting DNA sequence statistically resembles the codon usage table. This codon selection algorithm works as follows: For every amino acid, generate a random number between zero (0) and one (1). Pick the mostly used codon and subtract its usage from the random number. If the difference is below or equal to zero, then select the mostly used codon. Otherwise, select the second mostly used codon, subtract its usage from the remaining difference, and check again if the difference is below or equal to zero. If so, then pick the second mostly used codon. This loop of iterative selection and subtraction repeats until the remaining difference is below or equal to zero. In BOOST, the algorithm first normalizes the codon usages so that their sum equals to one (1) and sorts the codons by their descending usage. Hence, the random number is generated between zero (0) and one (1). If the codon usages are not normalized, then the random number should be generated between zero (0) and the sum of the codon usages of each amino acid's codons.

For codon juggling, BOOST provides an additional codon selection strategy option:

- **Least Different:** Replace each codon with a codon that has the most-similar usage among the codons that encode the same amino acid. If two or more codons are equally similar, then select randomly among them.

All codon selection strategy options have been inspired by Richardson et al.[18] All codon selection strategies iterate only once over the amino acids in a protein sequence or codons in a coding DNA sequence. Furthermore, the codon replacement strategies do not select codons with zero usage (although these codons could be accessible via a random strategy with no usage table provided). Currently, BOOST does not support the specification of a minimum codon usage threshold (greater than zero).

Verification of a DNA Sequence against Common DNA Synthesis Constraints. BOOST supports the verification of DNA sequences against a range of sequence features that define the success/failure rate of synthesis. In collaboration with commercial DNA synthesis providers, we have identified the following common sequence features:

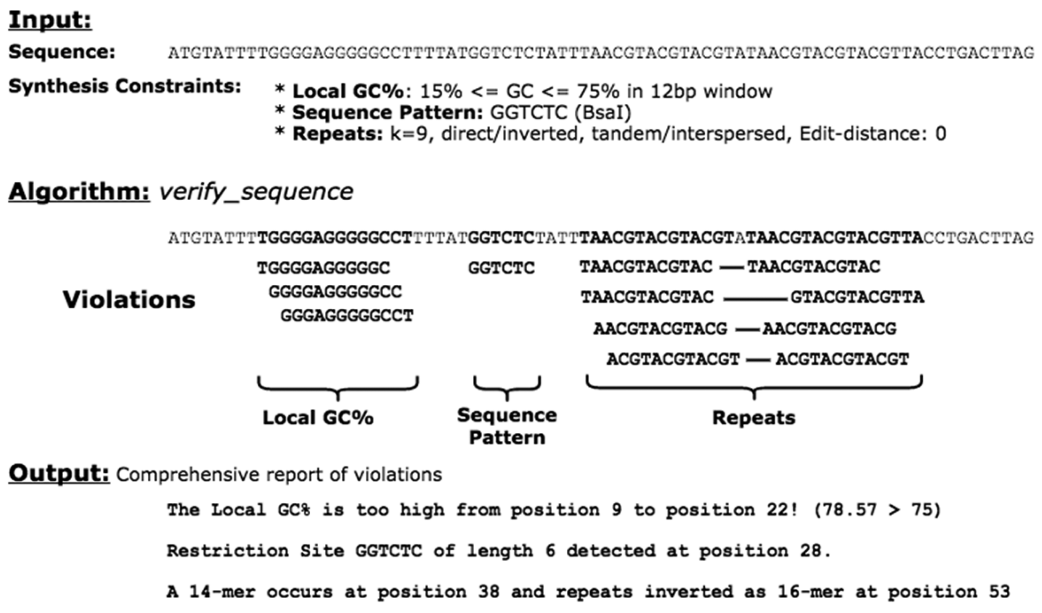
%GC Content: The percentage of Gs and Cs in a sequence must be balanced in a specific range of a minimum value and a maximum value. The %GC content must be in the range either in the entire sequence (global) and/or within sequence windows of a specific length (local). For global %GC content constraints, BOOST calculates the %GC content of the entire sequence. For local %GC content constraints, BOOST calculates the %GC content for each sequence window of specified length. In either case, if the calculated %GC content is lower/higher than the minimum/maximum %GC content (respectively) then BOOST reports a violation, outputting the %GC content and whether it is too low/high. For local %GC content violations, BOOST also reports the sequence window location.

Repeats: Readily synthesizable sequences are often constrained to be free of repeating sequences greater than a minimum length *k*. Repeats can occur (i) within the entire sequence (global) or within certain regions of the sequence (local), (ii) on the same (direct) or opposite (inverted) strand, (iii) immediately after each other (tandem) or with a number of base pairs between them (interspersed), and (iv) identical (exact) or differing by a number of base pair substitutions/insertions/deletions (mutated). To detect direct or inverted repeats of a minimum length *k* within a maximum edit-distance *d* and interspersed by a maximum number of base pairs *l*, BOOST creates a key/value-list lookup-table for the strands and location(s) of each *k*-mer (and the sequences within an edit distance *d* thereof). BOOST calculates the keys of the lookup-table by mapping each *k*-mer on both top and reverse complement strands into a pair of numeric values. For example, the 4-mer ATCG is transformed into the binary code 00110110, which is 54 in decimal. Its reverse complement CGAT (binary: 01100011) results in 99. For each pair of mapped numeric values (top strand and reverse complement), BOOST appends to the key/value-list lookup-table a triplet that represents the k-mer's location, strand, and edit distance. For example, if the ATCG 4-mer occurs on the top strand at position 10 without any substitutions and insertions/deletions, then BOOST appends the ⟨10,+1,0⟩ triplet to the value-list of key 54. BOOST integrates BBTools to create the lookup-tables efficiently (https://sourceforge.net/projects/bbmap/). Depending on the types of repeats to be detected, for each key in the lookup-table, BOOST identifies any sets of repeated locations/orientations that meet the detection criteria. For example, if we are detecting both direct and inverted repeats of 4-mers with a maximum edit distance of 0 base pairs, which are interspersed by less than 10 base pairs, and the sequence contains ATCG at location 10 and CGAT at location 20, then the value-list for key 54 would be [⟨10,+1,0⟩, ⟨20,−1,0⟩]. BOOST would report this as an inverted repeat of ACTG interspersed by six base pairs.

Sequence Patterns: Restriction enzymes cut DNA sequences at specific motifs, so-called restriction sites. If a sequence contains restriction sites, then restriction enzymes can impact not only the success rate of gene synthesis, but also the function of the designed sequence. The plethora of restriction enzymes and motifs requires a flexible approach for specification and verification. In BOOST, the user can input a CSV- or FASTA-format file containing unwanted sequence patterns, such as restriction sites, that should not appear in a sequence. BOOST compiles the sequence patterns into a single regular expression in order to detect the occurrence each sequence pattern on both DNA strands. BOOST reports the location, strand, and sequence of each identified sequence pattern.

Sequence Length: DNA synthesis vendors provide synthetic DNA blocks of certain size ranges as part of their product portfolio. If a designed sequence construct is longer than the vendor's specified ranges, then the sequence needs to be decomposed into synthesizable fragments (building blocks) that need then be assembled at a later point. BOOST calculates the length of the sequence and compares it against the minimum/maximum allowable gene synthesis length.

Additional DNA synthesis constraints exist, such as repeat coverage, the percentage of unique *k*-mers, or the maximum number of occurrences of sequence patterns. Such constraints can be specified as a function of the four enumerated constraints combined with conditions. For example, if the repeat coverage is greater than a specific threshold, then the sequence cannot be synthesized at minimal cost. That is, the repeat coverage is a function of the number of *k*-mers and the sequence length and the threshold defines the condition.


Input:
Sequence: ATGTATTTTGGGGAGGGGGCCTTTTATGGTCTCTATTTAACGTACGTACGTATAACGTACGTACGTTACCTGACTTAG
Synthesis Constraints:
* Local GC%: 15% <= GC <= 75% in 12bp window
* Sequence Pattern: GGTCTC (BsaI)
* Repeats: k=9, direct/inverted, tandem/interspersed, Edit-distance: 0
Algorithm: verify_sequence
ATGTATTTTGGGGAGGGGGCCTTTTATGGTCTCTATTTAACGTACGTACGTATAACGTACGTACGTTACCTGACTTAG
Violations
TGGGGAGGGGGC
GGGGAGGGGGCC
GGGAGGGGGCCT
GGTCTC
TAACGTACGTAC — TAACGTACGTAC
TAACGTACGTAC — GTACGTACGTTA
AACGTACGTACG — AACGTACGTACG
ACGTACGTACGT — ACGTACGTACGT
Local GC%
Sequence Pattern
Repeats
Output: Comprehensive report of violations

```
The Local GC% is too high from position 9 to position 22! (78.57 > 75)

Restriction Site GGTCTC of length 6 detected at position 28.

A 14-mer occurs at position 38 and repeats inverted as 16-mer at position 53
```

Figure 3. An example of verifying a DNA sequence against gene synthesis constraints and reporting violations in a comprehensible manner.

In Figure 3, we visualize an example of verifying a DNA sequence against synthesis constraints. The *verify_sequence* algorithm evaluates the input sequence against all prescribed synthesis constraints. In this example, we first evaluate each 12 bp sequence window to ensure that the %GC content is greater than 15% and less than 75%. Then, we check if the sequence pattern for the BsaI restriction site occurs on either DNA strand. The last constraint searches both DNA strands of the input sequence for perfect (zero edit distance) repeats of at least 9 bp, which may be direct or inverted, tandem or interspersed (separated by 1 or more bps).

The *verify_sequence* algorithm finds three overlapping sequence windows that violate the local %GC constraint. The sequence pattern for the BsaI restriction site occurs once on the top strand. The algorithm finds various repeats that are close to the 3′-end of the sequence.

Reporting Sequence Regions that Violate Gene Synthesis Constraints. As illustrated in Figure 3 and described immediately above, the *verify_sequence* algorithm may detect various constraint violations including extreme %GC, sequence patterns, and repeats. To generate a comprehensive violation report for the user, BOOST performs a *merge_violations* step that takes as input all detected violations, merges overlapping sequence regions that violate the same constraint, and outputs the consolidated sequence regions with their corresponding constraint violation information.

Modifying Coding Sequences to Comply with Gene Synthesis Constraints. The user can enable BOOST to attempt to automatically modify a DNA sequence to resolve identified constraint violations. If so, then BOOST determines the modifiable sequence regions, which are features annotated as coding sequence (CDS). If a violation occurs entirely in a modifiable region, then BOOST attempts to modify sequence region to resolve the violation. For example, if the %GC content is too low or too high in a coding sequence region, then BOOST attempts to make silent mutations (following one of several strategies) to increase or decrease the %GC. If a violation spans modifiable and immutable regions, then BOOST attempts to modify only the modifiable region to resolve the violation. For example, if a sequence were repeated in a promoter sequence region (immutable) and in a coding sequence region (modifiable), BOOST would only attempt to make silent mutations in the coding sequence region to resolve the repeat. If BOOST detects a violation that occurs entirely in an immutable region, then BOOST only reports the violation and does not perform any modifications (the user must manually resolve the constraint violation by, for example, redesigning the sequence and/or synthesizing the sequence at higher cost).

Figure 4 (a continuation of Figure 3) exemplifies BOOST's *modify_sequence* functionality, which takes as input the sequence, its constraint violations, and a codon usage table. First, *modify_sequence* brings each modifiable violation region into the appropriate reading-frame. Next, the violated region is translated into its protein sequence. Then, each amino acid is reverse-translated into a nucleotide codon. BOOST reverse-translates using the same codon selection strategies as listed in the Reverse-Translation and Codon Juggling section. That is, every violated sequence region is codon juggled with the goal being to resolve violations without introducing new violations. Therefore, BOOST must verify the entire sequence against the DNA synthesis constraints after modifying a violated region. If the modifications introduced new violations or the violations have not been resolved, then BOOST tries to iterate over the violated sequence regions again in order to modify them. BOOST performs these steps until all violations are resolved or the maximum number of iterations has been reached.

For the *modify_sequence* functionality, we have developed two additional strategies. Both strategies adjust codon usage values when some violations remain unresolved after a prescribed maximum number of iterations.

The **Relaxed Weight** strategy progressively reduces the codon usage values of the most frequently used codons and redistributes the usage equally among the other codons. That is, the most frequently used codons are progressively selected with lower frequency, whereas the other codons are progressively selected with higher frequency.

The **Balanced-to-Random** strategy modifies the codon usage values according to the formula $(i/N)\cdot(1/M) + (N - i)/N\cdot$ codon_usage(c), where i counts the number of codon usage

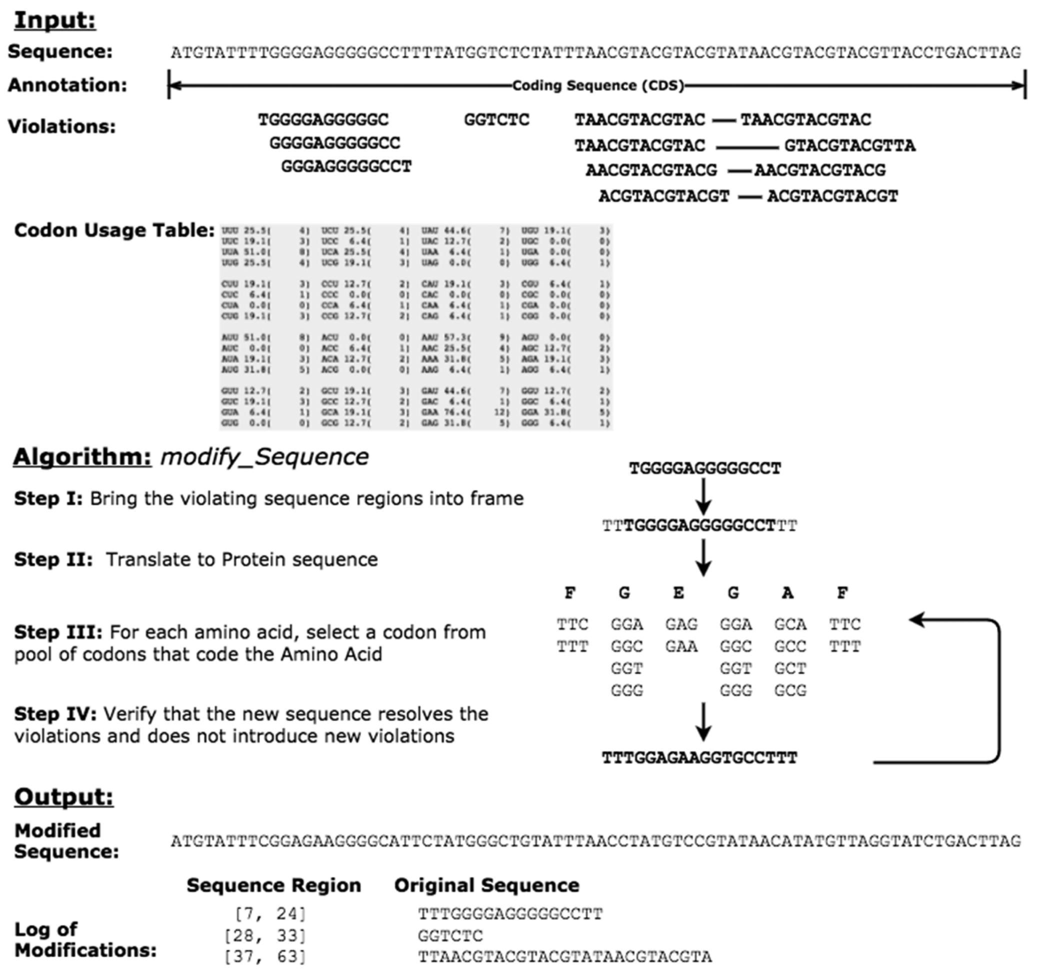

Input:
Sequence: ATGTATTTTGGGGAGGGGGCCTTTTATGGTCTCTATTTAACGTACGTACGTATAACGTACGTACGTTACCTGACTTAG
Annotation:
Coding Sequence (CDS)
Violations:
TGGGGAGGGGGC
GGGGAGGGGGCC
GGGAGGGGGCCT
GGTCTC
TAACGTACGTAC — TAACGTACGTAC
TAACGTACGTAC — GTACGTACGTTA
AACGTACGTACG — AACGTACGTACG
ACGTACGTACGT — ACGTACGTACGT
Codon Usage Table:
Algorithm: modify_Sequence
Step I: Bring the violating sequence regions into frame
Step II: Translate to Protein sequence
Step III: For each amino acid, select a codon from pool of codons that code the Amino Acid
Step IV: Verify that the new sequence resolves the violations and does not introduce new violations
TGGGGAGGGGGCCT
TTTGGGGAGGGGGCCTTT
F G E G A F
TTC GGA GAG GGA GCA TTC
TTT GGC GAA GGC GCC TTT
GGT GGT GCT
GGG GGG GCG
TTTGGAGAAGGTGCCTTT
Output:
Modified Sequence: ATGTATTTCGGAGAAGGGGCATTCTATGGGCTGTATTTAACCTATGTCCGTATAACATATGTTAGGTATCTGACTTAG
Log of Modifications:
Sequence Region
Original Sequence
[7, 24] TTTGGGGAGGGGGCCTT
[28, 33] GGTCTC
[37, 63] TTAACGTACGTACGTATAACGTACGTA

Figure 4. Modifying a protein coding sequence to resolve constraint violations using codon replacement strategies according to a genetic code and a codon usage table.

adjustment iterations, N denotes the maximum number of codon usage adjustments and M is the number of codons of an amino acid. The function codon_usage(c) refers to the codon usage of a codon. At early iterations, this codon selection strategy resembles the balanced strategy and at later iterations it resembles the random codon selection strategy.

BOOST outputs the modified sequence regardless if all violations have been resolved or not. In addition, BOOST outputs a log of the codon replacement modifications as illustrated in Figure 4. BOOST merges neighboring codon replacements into consolidated sets of contiguous replaced codons, in a fashion analogous to the violation reporting process (see above). The log of modifications contains the start- and end-position of each modified sequence region along with the original sequence. BOOST generates a Genbank format file that annotates the modifications as misc_difference sequence features.

Partitioning DNA Sequences into Synthesizable Building Blocks. Here, we describe and exemplify the *partition_sequence* algorithm that partitions a sequence that exceeds the maximum length of commercial gene synthesis. In Figure 5, we illustrate graphically a step-by-step example in order to demonstrate the concepts of the algorithm. The pseudocode of *partition_sequence* algorithm is provided in the Supporting Information.

Step I: The algorithm initializes the working number of building blocks to the absolute minimum number of building blocks required to partition the sequence. For example, if the maximum gene synthesis length is 3 kb (3000 base pairs), then the algorithm would determine that at least four building blocks are required to partition a 10 kb (10 000 base pairs) sequence.

Step II: The algorithm divides the sequence without gaps into the working number of equally sized nonoverlapping blocks. If the nonoverlapping block length is less than the minimum gene synthesis length, then the algorithm fails to partition the sequence, and terminates accordingly.

Step III: As illustrated in Figure 5, the ***partition_sequence*** algorithm determines putative overlap sequences based on a sequence region (neighborhood) defined by the block-boundary (neutral position) and the specified minimum, optimum, and maximum overlap length (see Supporting Information). The lengths of putative overlap sequences must be greater than or equal to the minimum overlap length, and less than or equal to the maximum overlap length. The total search space for overlap sequences is the neutral position ± the maximum overlap sequences.

Putative overlap sequences are verified against additional DNA assembly constraints, to eliminate overlap sequences anticipated to perform poorly. For example, the overlap

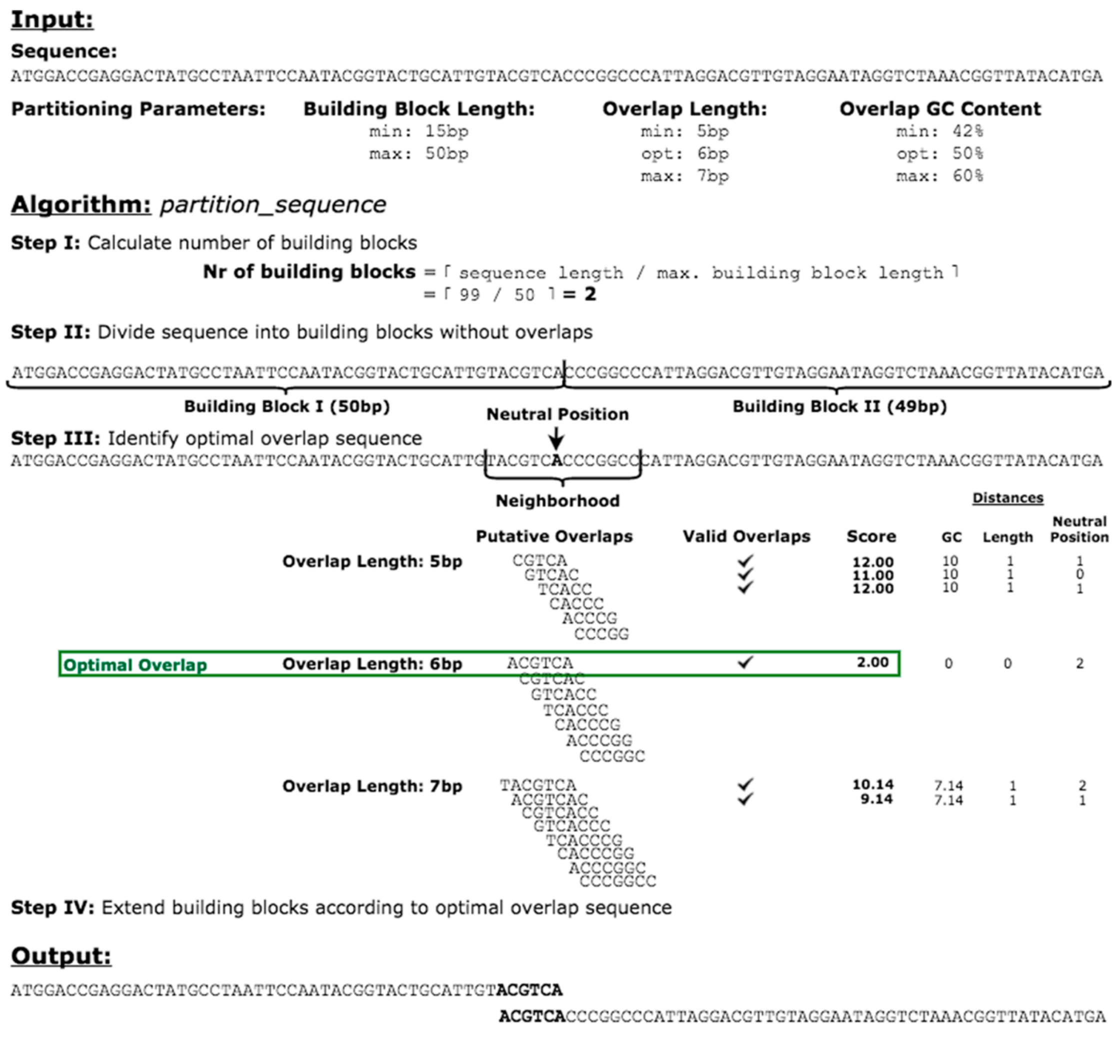

Input:
Sequence:
ATGGACCGAGGACTATGCCTAATTCCAATACGGTACTGCATTGTACGTCACCCGGCCCATTAGGACGTTGTAGGAATAGGTCTAAACGGTTATACATGA
Partitioning Parameters:
Building Block Length: min: 15bp max: 50bp
Overlap Length: min: 5bp opt: 6bp max: 7bp
Overlap GC Content min: 42% opt: 50% max: 60%
Algorithm: partition_sequence
Step I: Calculate number of building blocks
Nr of building blocks = ⌈ sequence length / max. building block length ⌉ = ⌈ 99 / 50 ⌉ = 2
Step II: Divide sequence into building blocks without overlaps
Building Block I (50bp)
Building Block II (49bp)
Neutral Position
Step III: Identify optimal overlap sequence
Neighborhood
Putative Overlaps
Valid Overlaps
Score
Distances
GC
Length
Neutral Position
Overlap Length: 5bp
CGTCA GTCAC TCACC CACCC ACCCG CCCGG
12.00 11.00 12.00
Optimal Overlap
Overlap Length: 6bp
ACGTCA CGTCAC GTCACC TCACCC CACCCG ACCCGG CCCGGC
2.00
Overlap Length: 7bp
TACGTCA ACGTCAC CGTCACC GTCACCC TCACCCG CACCCGG ACCCGGC CCCGGCC
10.14 9.14
Step IV: Extend building blocks according to optimal overlap sequence
Output:
ATGGACCGAGGACTATGCCTAATTCCAATACGGTACTGCATTGTACGTCA
ACGTCACCCGGCCCATTAGGACGTTGTAGGAATAGGTCTAAACGGTTATACATGA

Figure 5. A step-by-step example of partitioning a DNA sequence into synthesizable building blocks with assembly specific overlap sequences. We have chosen a short randomly generated sequence and small parameter values in order to facilitate explaining the concepts of the *partition_sequence* algorithm.

sequence constraints tailored for chew-back assembly methods, such as Gibson,[39] include the following:

- the %GC content of the overlap must be ≥40% and ≤62%
- no exact 6-mer repeats, direct or inverted, tandem or interspersed
- no homopolymer stretches >5bp
- no tandem repeats of trimers that cover the overlap by >50%
- the overlap sequence cannot occur more than once in the entire sequence with a maximum edit-distance (number of substitutions, insertions/deletions) of 5bp

Putative overlap sequences that comply with these constraints are then scored. If no putative overlap sequence complies with the constraints, then the algorithm increments the working number of building blocks and then continues at **Step II** (with building blocks of shorter length).

Otherwise, the algorithm selects the best (lowest) scoring overlap sequences (in case of a tie, the algorithm selects among the winners randomly). The scoring function to determine an optimal overlap sequence places equally weighted penalties on the distance from the block-boundary to the center of the overlap sequence, deviation from optimal overlap sequence length, and the deviation from the optimal overlap sequence %GC content.

Step IV: Overlap sequences have now been successfully identified for all pairs of neighboring blocks, and the nonoverlapping blocks are extended to fully contain the overlap sequence identified at each block boundary, yielding overlapping blocks not exceeding the maximum gene synthesis length (the algorithm successfully terminates).

Implementation Details. The core functionalities of BOOST are purely implemented in Java 1.7, and the BOOST web application is deployed on a Apache Tomcat web server (http://tomcat.apache.org) on a cluster of the DOE National Energy Research Scientific Computing Center (NERSC) (http://www.nersc.gov). The BOOST web application's UI (front-end) is implemented in jQuery (https://jquery.com) and its styling is based on Twitter Bootstrap (http://getbootstrap.com). The RESTful API (back-end) of the BOOST web application is developed in Java 1.7 and based on the Jersey framework (https://jersey.java.net).

Availability and System Requirements. BOOST is available in three different formats: as an executable Java ARchive (JAR), as a RESTful API, and as a web application.

The executable JAR file can either be executed on the command line or integrated into larger, sophisticated Java-based software tools. The RESTful API version of BOOST is deployable on a web server and can be invoked programmatically from other software tools, which are not necessarily implemented in Java. Lastly, the BOOST web application is build atop the RESTful API, can be deployed on a web server, and it provides a UI, enabling humans to utilize the BOOST functionalities using a web browser. All three versions of BOOST are available based on the Docker deployment framework (http://www.docker.com) and further information is provided at the BOOST Web site.

Executing the BOOST JAR file requires the installation of a Java Runtime Environment (JRE), which is available for common platforms. If the execution of BOOST runs out of memory, then additional memory can be requested using the -Xms and -Xmx command line input arguments. The deployment of the BOOST RESTful API as well as the BOOST web application require a web server that supports Java, such as Apache Tomcat. At the DOE JGI, we have deployed the RESTful API and web application of BOOST on an Apache Tomcat 8 Web server.

Licensing Information. BOOST is available at no cost to noncommercial (e.g., academic, nonprofit, or government) users through the public BOOST web server (https://boost.jgi.doe.gov), under a Lawrence Berkeley National Lab end-user license agreement (https://boost.jgi.doe.gov/License). Manuals describing how to use the BOOST Web UI, and technical documentation for programmatically invoking the RESTful API, are provided on the public BOOST web server. Commercial use is available through the Innovations and Partnerships Office of Lawrence Berkeley National Laboratory (ipo@lbl.gov).

■ ASSOCIATED CONTENT

Ⓢ Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acssynbio.6b00200.

Formal description of the *partitioningSequence* algorithm (PDF)

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: eoberortner@lbl.gov.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work has been supported by the DOE Joint Genome Institute (http://jgi.doe.gov) and the DOE Joint BioEnergy Institute (http://www.jbei.org) by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research, through Contract DE-AC02-05CH11231 between Lawrence Berkeley National Laboratory and the U.S. Department of Energy. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan). The authors thank Adam Clore (IDT), Thomas Hofmeister and Axel Trefzer (Thermo Fisher Scientific - Life Technologies), Gisela Canales, and Rahul Gautam (SGI-DNA) for sharing, explaining, and discussing DNA synthesis constraints. The DNA synthesis guidelines of Gen9 are freely available at http://help.gen9bio.com/docs/gene-synthesis-guidelines. We also thank Elizabeth Nickerson, Andrew Bond, and Ann Longueville (Gen9) as well as Amanda Skeen and Rahul Gautam (SGI-DNA) for useful discussions. The development of BOOST is a collaborative effort, managed between end-users and software developers. We thank Cameron Coates and Robert Evans for testing the software tools and for providing constructive feedback on each individual tool's usability and applicability. Also we thank Jacob Coble, Mark Forrer, Hector Garcia Martin, Sarah LaFrance, Xianwei "John" Meng, William Morrell, Oge Nnadi, Hector Plahar, and Lisa Simirenko for reviewing the BOOST source code and for their helpful suggestions regarding the BOOST Web UI. We thank Sarah Richardson for her insights into the development of the algorithms. Lastly, we thank the DOE National Energy Research Scientific Computing Center (NERSC) for providing the hard- and software facilities for executing, deploying and hosting both, the BOOST command-line tool and the BOOST web application.

■ REFERENCES

(1) Koppel, N., and Balskus, E. P. (2016) Exploring and Understanding the Biochemical Diversity of the Human Microbiota. *Cell Chem. Biol. 23*, 18−30.

(2) Walker, M. C., and Chang, M. C. (2014) Natural and engineered biosynthesis of fluorinated natural products. *Chem. Soc. Rev. 43*, 6527−6536.

(3) Rinke, C., Schwientek, P., Sczyrba, A., Ivanova, N. N., Anderson, I. J., Cheng, J. F., Darling, A., Malfatti, S., Swan, B. K., Gies, E. A., Dodsworth, J. A., Hedlund, B. P., Tsiamis, G., Sievert, S. M., Liu, W. T., Eisen, J. A., Hallam, S. J., Kyrpides, N. C., Stepanauskas, R., Rubin, E. M., Hugenholtz, P., and Woyke, T. (2013) Insights into the phylogeny and coding potential of microbial dark matter. *Nature 499*, 431−437.

(4) Long, P. E., Williams, K. H., Hubbard, S. S., and Banfield, J. F. (2016) Microbial Metagenomics Reveals Climate-Relevant Subsurface Biogeochemical Processes. *Trends Microbiol. 24* (8), 600−610.

(5) Paddon, C. J., Westfall, P. J., Pitera, D. J., Benjamin, K., Fisher, K., McPhee, D., Leavell, M. D., Tai, A., Main, A., Eng, D., Polichuk, D. R., Teoh, K. H., Reed, D. W., Treynor, T., Lenihan, J., Fleck, M., Bajad, S., Dang, G., Dengrove, D., Diola, D., Dorin, G., Ellens, K. W., Fickes, S., Galazzo, J., Gaucher, S. P., Geistlinger, T., Henry, R., Hepp, M., Horning, T., Iqbal, T., Jiang, H., Kizer, L., Lieu, B., Melis, D., Moss, N., Regentin, R., Secrest, S., Tsuruta, H., Vazquez, R., Westblade, L. F., Xu, L., Yu, M., Zhang, Y., Zhao, L., Lievense, J., Covello, P. S., Keasling, J. D., Reiling, K. K., Renninger, N. S., and Newman, J. D. (2013) High-level semi-synthetic production of the potent antimalarial artemisinin. *Nature 496*, 528−532.

(6) Burgard, A., Burk, M. J., Osterhout, R., Van Dien, S., and Yim, H. (2016) Development of a commercial scale process for production of 1,4-butanediol from sugar. *Curr. Opin. Biotechnol. 42*, 118−125.

(7) Cheong, S., Clomburg, J. M., and Gonzalez, R. (2016) Energy- and carbon-efficient synthesis of functionalized small molecules in bacteria using non-decarboxylative Claisen condensation reactions. *Nat. Biotechnol. 34*, 556−561.

(8) Ham, T. S., Dmytriv, Z., Plahar, H., Chen, J., Hillson, N. J., and Keasling, J. D. (2012) Design, implementation and practice of JBEI-ICE: an open source biological part registry platform and tools. *Nucleic Acids Res. 40*, e141.

(9) Markowitz, V. M., Chen, I. M., Palaniappan, K., Chu, K., Szeto, E., Grechkin, Y., Ratner, A., Jacob, B., Huang, J., Williams, P., Huntemann, M., Anderson, I., Mavromatis, K., Ivanova, N. N., and Kyrpides, N. C. (2012) IMG: the Integrated Microbial Genomes database and comparative analysis system. *Nucleic Acids Res. 40*, D115−122.

(10) Kanehisa, M., and Goto, S. (2000) KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res. 28*, 27−30.

(11) Geer, L. Y., Marchler-Bauer, A., Geer, R. C., Han, L., He, J., He, S., Liu, C., Shi, W., and Bryant, S. H. (2010) The NCBI BioSystems database. *Nucleic Acids Res. 38*, D492−496.

(12) Madsen, C., McLaughlin, J. A., Misirli, G., Pocock, M., Flanagan, K., Hallinan, J., and Wipat, A. (2016) The SBOL Stack: A Platform for Storing, Publishing, and Sharing Synthetic Biology Designs. *ACS Synth. Biol. 5*, 487−497.

(13) Nielsen, A. A., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016) Genetic circuit design automation. *Science 352*, aac7341.

(14) Chen, J., Densmore, D., Ham, T. S., Keasling, J. D., and Hillson, N. J. (2012) DeviceEditor visual biological CAD canvas. *J. Biol. Eng. 6*, 1.

(15) Czar, M. J., Cai, Y., and Peccoud, J. (2009) Writing DNA with GenoCAD. *Nucleic Acids Res. 37*, W40−47.

(16) Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011) Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS One 6*, e18882.

(17) Roehner, N., Young, E. M., Voigt, C. A., Gordon, D. B., and Densmore, D. (2016) Double Dutch: A Tool for Designing Combinatorial Libraries of Biological Systems. *ACS Synth. Biol. 5*, 507−517.

(18) Richardson, S. M., Wheelan, S. J., Yarrington, R. M., and Boeke, J. D. (2006) GeneDesign: rapid, automated design of multikilobase synthetic genes. *Genome Res. 16*, 550−556.

(19) Espah Borujeni, A., Channarasappa, A. S., and Salis, H. M. (2014) Translation rate is controlled by coupled trade-offs between site accessibility, selective RNA unfolding and sliding at upstream standby sites. *Nucleic Acids Res. 42*, 2646−2659.

(20) Na, D., and Lee, D. (2010) RBSDesigner: software for designing synthetic ribosome binding sites that yields a desired level of protein expression. *Bioinformatics 26*, 2633−2634.

(21) Reeve, B., Hargest, T., Gilbert, C., and Ellis, T. (2014) Predicting translation initiation rates for designing synthetic biology. *Front. Bioeng. Biotechnol. 2*, 1.

(22) Salis, H. M., Mirsky, E. A., and Voigt, C. A. (2009) Automated design of synthetic ribosome binding sites to control protein expression. *Nat. Biotechnol. 27*, 946−950.

(23) Hillson, N. J., Rosengarten, R. D., and Keasling, J. D. (2012) j5 DNA assembly design automation software. *ACS Synth. Biol. 1*, 14−21.

(24) Appleton, E., Tao, J., Haddock, T., and Densmore, D. (2014) Interactive assembly algorithms for molecular cloning. *Nat. Methods 11*, 657−662.

(25) Linshiz, G., Stawski, N., Goyal, G., Bi, C., Poust, S., Sharma, M., Mutalik, V., Keasling, J. D., and Hillson, N. J. (2014) PR-PR: cross-platform laboratory automation system. *ACS Synth. Biol. 3*, 515−524.

(26) Galdzicki, M., Clancy, K. P., Oberortner, E., Pocock, M., Quinn, J. Y., Rodriguez, C. A., Roehner, N., Wilson, M. L., Adam, L., Anderson, J. C., Bartley, B. A., Beal, J., Chandran, D., Chen, J., Densmore, D., Endy, D., Grunberg, R., Hallinan, J., Hillson, N. J., Johnson, J. D., Kuchinsky, A., Lux, M., Misirli, G., Peccoud, J., Plahar, H. A., Sirin, E., Stan, G. B., Villalobos, A., Wipat, A., Gennari, J. H., Myers, C. J., and Sauro, H. M. (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol. 32*, 545−550.

(27) Roehner, N., Beal, J., Clancy, K., Bartley, B., Misirli, G., Grunberg, R., Oberortner, E., Pocock, M., Bissell, M., Madsen, C., Nguyen, T., Zhang, M., Zhang, Z., Zundel, Z., Densmore, D., Gennari, J. H., Wipat, A., Sauro, H. M., and Myers, C. J. (2016) Sharing Structure and Function in Biological Design with SBOL 2.0. *ACS Synth. Biol. 5*, 498−506.

(28) Gibson, D. G., Glass, J. I., Lartigue, C., Noskov, V. N., Chuang, R. Y., Algire, M. A., Benders, G. A., Montague, M. G., Ma, L., Moodie, M. M., Merryman, C., Vashee, S., Krishnakumar, R., Assad-Garcia, N., Andrews-Pfannkoch, C., Denisova, E. A., Young, L., Qi, Z. Q., Segall-Shapiro, T. H., Calvey, C. H., Parmar, P. P., Hutchison, C. A., 3rd, Smith, H. O., and Venter, J. C. (2010) Creation of a bacterial cell controlled by a chemically synthesized genome. *Science 329*, 52−56.

(29) Annaluru, N., Muller, H., Mitchell, L. A., Ramalingam, S., Stracquadanio, G., Richardson, S. M., Dymond, J. S., Kuang, Z., Scheifele, L. Z., Cooper, E. M., Cai, Y., Zeller, K., Agmon, N., Han, J. S., Hadjithomas, M., Tullman, J., Caravelli, K., Cirelli, K., Guo, Z., London, V., Yeluru, A., Murugan, S., Kandavelou, K., Agier, N., Fischer, G., Yang, K., Martin, J. A., Bilgel, M., Bohutski, P., Boulier, K. M., Capaldo, B. J., Chang, J., Charoen, K., Choi, W. J., Deng, P., DiCarlo, J. E., Doong, J., Dunn, J., Feinberg, J. I., Fernandez, C., Floria, C. E., Gladowski, D., Hadidi, P., Ishizuka, I., Jabbari, J., Lau, C. Y., Lee, P. A., Li, S., Lin, D., Linder, M. E., Ling, J., Liu, J., Liu, J., London, M., Ma, H., Mao, J., McDade, J. E., McMillan, A., Moore, A. M., Oh, W. C., Ouyang, Y., Patel, R., Paul, M., Paulsen, L. C., Qiu, J., Rhee, A., Rubashkin, M. G., Soh, I. Y., Sotuyo, N. E., Srinivas, V., Suarez, A., Wong, A., Wong, R., Xie, W. R., Xu, Y., Yu, A. T., Koszul, R., Bader, J. S., Boeke, J. D., and Chandrasegaran, S. (2014) Total synthesis of a functional designer eukaryotic chromosome. *Science 344*, 55−58.

(30) Hutchison, C. A., 3rd, Chuang, R. Y., Noskov, V. N., Assad-Garcia, N., Deerinck, T. J., Ellisman, M. H., Gill, J., Kannan, K., Karas, B. J., Ma, L., Pelletier, J. F., Qi, Z. Q., Richter, R. A., Strychalski, E. A., Sun, L., Suzuki, Y., Tsvetanova, B., Wise, K. S., Smith, H. O., Glass, J. I., Merryman, C., Gibson, D. G., and Venter, J. C. (2016) Design and synthesis of a minimal bacterial genome. *Science 351*, aad6253.

(31) Kosuri, S., and Church, G. M. (2014) Large-scale de novo DNA synthesis: technologies and applications. *Nat. Methods 11*, 499−507.

(32) Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., and Sayers, E. W. (2009) GenBank. *Nucleic Acids Res. 37*, D26−31.

(33) Hentrich, C., and Zdun, U. (2006) Patterns for business object model integration in process-driven and service-oriented architectures, In *Proceedings of the 2006 conference on Pattern languages of programs*, pp 1−14, ACM, Portland, Oregon, USA.

(34) Villalobos, A., Ness, J. E., Gustafsson, C., Minshull, J., and Govindarajan, S. (2006) Gene Designer: a synthetic biology tool for constructing artificial DNA segments. *BMC Bioinf. 7*, 285.

(35) Christen, M., Deutsch, S., and Christen, B. (2015) Genome Calligrapher: A Web Tool for Refactoring Bacterial Genome Sequences for de Novo DNA Synthesis. *ACS Synth. Biol. 4*, 927−934.

(36) Holland, R. C., Down, T. A., Pocock, M., Prlic, A., Huen, D., James, K., Foisy, S., Drager, A., Yates, A., Heuer, M., and Schreiber, M. J. (2008) BioJava: an open-source framework for bioinformatics. *Bioinformatics 24*, 2096−2097.

(37) Zhang, Z., Nguyen, T., Roehner, N., Misirli, G., Pocock, M., Oberortner, E., Samineni, M., Zundel, Z., Beal, J., Clancy, K., Wipat, A., and Myers, C. J. (2015) libSBOLj 2.0: A Java Library to Support SBOL 2.0. *IEEE Life Sciences Letters 1*, 34−37.

(38) Sharp, P. M., Tuohy, T. M., and Mosurski, K. R. (1986) Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes. *Nucleic Acids Res. 14*, 5125−5143.

(39) Gibson, D. G. (2011) Enzymatic assembly of overlapping DNA fragments. *Methods Enzymol. 498*, 349−361.