Synthetic Biology, 2021, **6(1)**, 1–9
DOI: https://doi.org/10.1093/synbio/ysab026
Advance access publication date: 11 October 2021
Research Article

Design and assembly of DNA molecules using multi-objective optimization

Angelo Gaeta[1], Valentin Zulkower[2], and Giovanni Stracquadanio[1,*]

[1]School of Biological Sciences, The University of Edinburgh, Edinburgh EH9 3BF, UK
[2]Edinburgh Genome Foundry, School of Biological Sciences, The University of Edinburgh, Edinburgh EH9 3BF, UK
*Corresponding author: E-mail: giovanni.stracquadanio@ed.ac.uk

Abstract

Rapid engineering of biological systems is currently hindered by limited integration of manufacturing constraints into the design process, ultimately reducing the yield of many synthetic biology workflows. Here we tackle DNA engineering as a multi-objective optimization problem aiming at finding the best tradeoff between design requirements and manufacturing constraints. We developed a new open-source algorithm for DNA engineering, called Multi-Objective Optimisation algorithm for DNA Design and Assembly, available as a Python and Anaconda package, as well as a Docker image. Experimental results show that our method provides near-optimal constructs and scales linearly with design complexity, effectively paving the way to rational engineering of DNA molecules from genes to genomes.

Key words: biodesign; optimization; combinatorial assembly; computer-aided design; multi-objective optimization

1. Introduction

Recent advances in synthetic biology and DNA synthesis technologies are enabling significant scientific and biotechnological breakthroughs, including the engineering of pathways for drug production [1], the construction of minimal bacterial cells [2] and the assembly of synthetic eukaryotic chromosomes [3].

Pivotal to these achievements has been the adoption of an iterative engineering workflow, known as the Design-Built-Test-Learn cycle (DBTL). The DBTL workflow starts with a design step where biological components, such as genes or promoters, are selected to be assembled into a construct to obtain a specific phenotype; usually, the output of this process is a sequence of DNA to be synthesized. The designed sequence is then built and cloned into a host organism, and then tested to assess whether the design requirements are met, e.g. gene expression levels and protein abundance. The testing phase then informs the learning step, which in turn aims at improving the design of the initial construct using the phenotypic information acquired.

Interestingly, the inherent waterfall structure of the DBTL workflow introduces dependencies between steps, making engineering biological systems still a complex task. This is especially true for the design and build steps; in particular, the design space is strongly constrained by the DNA synthesis process, since phosphoramidite synthesis poses limits on molecule length and content. These limitations are usually overcome by splitting the designed sequences into shorter fragments, which can be then put together through molecular assembly techniques [4, 5], at the cost of increasing complexity both for the design and manufacturing steps. Ultimately, recoding the design to meet manufacturing constraints often leads to molecules with substantially different content and properties, effectively breaking the DBTL workflow.

Software have been developed to assist biological engineers in implementing the DBTL cycle, in particular for the design step, with tools such as Double Dutch [6], Cello [7], j5 [8], Raven [9], BOOST [10] and BioPartsBuilder [11]. Nevertheless, current software simply automates the process of designing and adapting the sequence for synthesis, often leading to suboptimal designs and lacking quantitative measures to evaluate design quality.

Here we build on our experience in mathematical programming methods for electronic design automation [12, 13] to solve the conundrum of designing DNA for manufacturability. Similar to how electronic circuits design is informed by physical and silicon manufacturing requirements, we formulated the design and build steps as a multi-objective optimization problem, aiming at finding the best trade-off between design and manufacturing requirements. Thus, rather than a single construct, biological engineers will be presented with a set of manufacturable designs to choose from for downstream work. We also introduce analytical measures to assess design quality and algorithmic performances, which are sorely lacking in the biological design automation field.

We developed a new optimization algorithm to solve this complex multi-objective problem, which is implemented as part of an open-source Python software called Multi-Objective Optimisation algorithm for DNA Design and Assembly (MOODA). The software

Submitted: 15 February 2021; **Received (in revised form):** 7 July 2021; **Accepted:** 24 August 2021

is released on PyPi, Anaconda, and as a Docker image, whereas the source code, documentation and examples are available on GitHub (https://github.com/stracquadaniolab/mooda).

We tested MOODA on an extensive synthetic DNA constructs dataset to assess the quality of the proposed designs and its computational efficiency. Experimental results show that MOODA can effectively identify near-optimal designs regardless of sequence complexity and its running time scales linearly with the number of objectives and the sequence length.

2. Materials and methods

Here we introduce a multi-objective formulation of the DNA design and assembly problem. We assume that the input is a DNA sequence, where coding regions have been annotated. We then propose an optimization algorithm to identify trade-off solutions for an arbitrary number of design and manufacturing requirements.

2.1 A multi-objective formulation of the DNA design and manufacturing problem

Let x be a sequence over the DNA alphabet $\Sigma = \{A, T, C, G\}$ and $F = (f_1, f_2, \cdots, f_k)$, with $f_k : \Sigma \to \mathbb{R}$ and k being the number of design and manufacturing requirements, which we also call objectives. We assume that requirements can be evaluated computationally by a function, which returns a fitness measure for the sequence. Without loss of generality and to avoid ambiguities, we assume that all objectives must be minimized.

We can define a multi-objective optimization problem as follows:

$$\min_{x \in \Sigma} F(x) = (f_1(x), f_2(x), \cdots, f_k(x)) \tag{1}$$

where for $k = 1$ the problem reduces to a standard single-objective optimization problem; however, for $k > 1$, it is usually not possible to find x such that all objectives are simultaneously minimized and, instead, we look for trade-off solutions. Let x_1, x_2 be two sequences over the DNA alphabet, x_1 dominates x_2, denoted as $x_1 \prec x_2$, if:

$$\exists i \in \{1, \cdots, k\}\ f_i(x_1) < f_i(x_2) \ \wedge\ \forall j \in \{1, \cdots, k\}\ f_j(x_1) \leq f_j(x_2) \tag{2}$$

In mathematical terms, the set of trade-off solutions, or Pareto optimal set, is made of all the non-dominated solutions for F, which is the set of sequences that cannot improve an objective without worsening at least another one. The image of the non-dominated solutions with respect to the mapping F is called Pareto front; geometrically, the Pareto front is bounded by an ideal point, which is the vector defined by all the minima, and the nadir point, which is the vector defined by all the maxima, thus representing the theoretical worst possible solution. In general, we cannot find the true Pareto optimal set unless boundary conditions are met, but approximations are usually sufficient in practice [14].

A plethora of methods have been proposed in literature to solve multi-objective problems, both deterministic [15, 16] and stochastic [17–20]. While deterministic methods provide strong proofs of convergence and optimality, they are usually difficult to apply to non-numerical problems. Conversely, stochastic methods, such as genetic algorithms or evolutionary strategies, are domain agnostic and work well in practice, although lacking theoretical guarantees on convergence and optimality.

2.2 A Multi-Objective Optimisation algorithm for DNA Design and Assembly

Here we describe a new stochastic optimization algorithm, called Multi-Objective Optimisation algorithm for DNA Design and Assembly (MOODA). The basic unit of operation is the solution data structure $z = (s, b)$, where s is a DNA sequence and b is the list of DNA fragments (or blocks) required to assemble arbitrary long sequences. Blocks are represented as sequence intervals to take advantage of interval algebra for downstream operations. Hereby we refer to z as the solution for a problem F involving k design and manufacturing objectives.

The algorithm takes as input a DNA sequence s, which is cloned n times to build an initial pool P of n solutions; the initial sequence is randomly split into fragments of approximately the same size, each one of size $l_{min} \leq l \leq l_{max}$, with l_{min} and l_{max} being the minimum and maximum DNA fragment that can be synthesized. Then, at each iteration t, each solution in P is cloned, randomly edited and evaluated with respect to the objective functions F. From the resulting pool of $2n$ solutions, n are selected for the next iteration. The algorithm stops when the maximum number of iterations T_{max} is reached. An overview of the algorithm is presented in Alg. 1.

Hereby we describe the edit and selection procedures, which are the key components of our method.

Sequence editing and assembly operators. The edit operators are local search procedures, which take in a solution as input and return a new, possibly better design. We defined procedures to edit both DNA sequences and blocks; sequence edits are limited to coding regions because we can safely introduce silent mutations to match requirements, whereas block edits are limited by the minimum and maximum DNA fragment size that is possible to synthesize. We defined four edit procedures that cover most common scenarios; however, MOODA can be easily extended with custom functions to introduce different types of changes.

GC optimization operator. The GC content of a DNA fragment is often a major hurdle to its synthesis; usually, synthesis providers have stringent admissible ranges on GC content and sequences have to be recoded to meet this requirement. Nonetheless, the GC content is often associated with specific phenotypes [21], thus any GC optimization procedure should edit a DNA sequence in such a way that the encoded biological function is not affected. This requirement restricts the application of GC content optimization only to coding regions, since GC can be optimized by using synonymous codons, which guarantee that the final protein product is not altered.

Algorithm 1 Multi-Objective Optimisation algorithm for DNA Design and Assembly

```
1.  procedure MOODA($s, n, F, T_{max}$)
2.      $P \leftarrow$ Initialise($s, n$)
3.      Evaluate($F, P$)
4.      $t \leftarrow 0$
5.      while $t \leq T_{max}$ do
6.          $R \leftarrow$ Clone($P$)
7.          $R \leftarrow$ Edit($R$)
8.          Evaluate($F, R$)
9.          $P \leftarrow$ Select($P, R$)
10.         $t \leftarrow t + 1$
11.     end while
12. end procedure
```

Algorithm 2 GC content operator

```
1.  procedure GC CONTENT($\sigma_{GC}$, $T_{GC}$, $CDS$)
2.      $CDS_r \leftarrow$ RandomSelect($CDS$)
3.      $CDS_e \leftarrow$ Copy($CDS_r$)
4.      while $|GC(CDS_e) - GC(CDS_r)| \leq \sigma_{GC}$ do
5.          $C \leftarrow$ RandomSelectCodons($CDS_e$)
6.          $A \leftarrow$ Translate($C$)
7.          $L \leftarrow$ GetCodonsForAminoacid($A$)
8.          if $T_{GC} \geq$ GC($CDS_e$) then
9.              $CDS_e \leftarrow$ UseLowGCCodon($L$)
10.         else if $T_{GC} <$ GC($CDS_e$) then
11.             $CDS_e \leftarrow$ UseHighGCCodon($L$)
12.         end if
13.     end while
14.     return $CDS_e$
15. end procedure
```

Here we define a GC optimization operator, which recodes a random coding sequence (CDS) by probabilistically using codons that bring its GC content closer to a user-defined target T_{GC} (see Alg. 2). The GC procedure acts only on one coding region at the time and allows improvement of at most σ_{GC} percent respect to the original sequence; here, we adopted this strategy to increase design diversity and avoid biases and divergent sequences.

Codon optimization operator. Transplanting genes and pathways between organisms often require changing their primary sequence at the codon level to ensure expression. Moreover, coding regions are often recoded to increase gene expression, as a way to maximize the production of a particular protein [22]. However, the relationship between codon usage and gene transcription is poorly understood [23]. Our codon optimization operator probabilistically recodes a fraction σ_c of the codons of a given gene, by silently replacing the current codons with the most frequent one, in accordance with an input codon usage table T_{CF}. As for the GC optimization operator, to increase the diversity of the pool of designs generated by our method, we do not apply codon optimization to all CDSs at the same time, but only to one at random.

Block split operator. Current technologies do not allow synthesis of arbitrary long DNA molecules, thus requiring a construct to be split into shorter fragments and then reassembled using DNA assembly techniques. [24]. However, excessive fragmentation can be both expensive and increase the turnaround of the assembly process. The block split operator divides a DNA sequence into shorter fragments, whose length is between l_{min} and l_{max} nucleotides; by design, the operator enforces homogeneity in block length by splitting sequences into blocks of discrete length, which is controlled by a parameter σ_b.

Block join operator. The block join operator reduces the number of blocks by joining two consecutive blocks, thus decreasing the number of parts to assemble. The procedure selects two blocks at random and join them into a new longer block; if the new block exceeds the block maximum size, it is divided again into two new blocks with a size that is a multiple of the step size parameter σ_b and within the maximum and minimum block length, respectively, l_{max} and l_{min}. As for all our operators, we enforce diversity in our pool of designs by applying the join procedure only to a pair of blocks at the time.

All our operators are designed to generate overlaps between adjacent blocks compatible with Gibson assembly [24]; however, new assembly methods can be easily defined in Python and integrated with our package.

Selection of trade-off solutions. A crucial step of our method is the selection procedure, where non-dominated solutions are picked for the next iteration. To do that, all solutions are compared to each other and assigned a rank based on the number of solutions they are dominated by; in this case, non-dominated solutions are those with the lowest rank. The domination criteria give the same weight to every objective function, improving the probability to find balanced trade-offs [25].

Once all solutions are ranked, they are ordered first based on their rank and then based on a distance metric, called crowding distance, defined as follows:

$$w_{f_i}(z_j) = (f_i(z_{j+1}) - f_i(z_{j-1}))/(f_i^{max} - f_i^{min}) \tag{3}$$

$$d(z_j) = \sum_{n=1}^{k} w_{f_i}(z_j) \tag{4}$$

where $d(z_j)$ is the crowding distance related to the j solution and $w_{f_i}(z_j)$ is the crowding distance with respect to the objective function f_i, whereas $f_i(z_{j+1})$ and $f_i(z_{j-1})$ are the closest solutions to z_j with respect to f_i. We also denote with f_i^{max} and f_i^{min} the maximum and the minimum value found by the algorithm for the objective function f_i, respectively. The crowding distance is a measure of how close two solutions are on the objective functions space and enable the algorithm to select solutions in unexplored regions of the Pareto Front. After the ranking, the top n solutions are selected for the next iteration.

The selection step is the most critical step for two reasons; first, since the non-dominated sorting procedure has complexity $O(kn^2)$ and it is executed at each iteration, using large pool sizes will dramatically increase the running time of the algorithm. Second, since at most n solutions are selected at each iteration, other non-dominated solutions can be discarded because of poor crowding distance score, effectively causing loss of information.

Here we address these problems by storing all solutions in an *ad hoc* data structure, called archive, whose size $m >> n$ is set by the user. When the archive is full, m non-dominated solutions are retained, eventually discarding the others based on their crowding distance value. By setting the pool size smaller than the archive size, we are decreasing the running time of the sorting procedure with only a negligible cost in terms of memory consumption; moreover, by storing $m >> n$ non-dominated solutions found during the optimization process, we are effectively returning more solutions at a fraction of the running time required for optimizing a pool of size m.

Software implementation and availability. We implemented MOODA in Python following object oriented programming best practice. The software requires as input an annotated GenBank file and a YAML configuration file, which specifies objective functions, and design operators and the assembly method to use to join DNA blocks. Our software can be easily customized by implementing new design operators and objective functions. New design operators should extend the base OPERATOR class and override the APPLY method, which implements a user-defined procedure to manipulate a solution. Analogously, a new objective function should extend the base OBJECTIVEFUNCTION class and override the EVAL method, which in turn will return a real value associated with the fitness of the input solution. The source code, documentation and a fully working example are available at https://github.com/stracquadaniolab/mooda, while packages are released on Anaconda and PyPi repositories and as a Docker image.

2.3 Design and manufacturing objectives

We assessed the performance of our method by studying four competing design and manufacturing requirements; these are common to most DNA engineering tasks and have an easily interpretable form useful to assess the performance of our method.

GC content objective function. The GC content of each designed DNA fragment must be within the limits specified by a DNA synthesis provider. Here we assume that an ideal GC value, T_{GC}, is provided in input. Thus, we can mathematically define the GC content objective as follows:

$$f_1(z) = \sum_{b \in B(z)} |T_{GC} - GC(b)| \tag{5}$$

where z is a solution, and $B(z)$ are the set of blocks defined in z. The optimal value for f_1 is 0, which is obtained when $GC(b) = T_{GC}$. To obtain an upper bound we used a heuristic procedure, where we replaced the codon of each coding region in the input sequence with the one maximizing the difference with respect to T_{GC}; successively, we divided the sequence into the maximum admissible number of blocks and evaluated the objective function.

Codon usage objective function. One of the most common operations in synthetic biology is the transfer of genes or pathways from one organism to another. Nevertheless, each organism has its codon usage, since for each amino acid some codons are less common than others, and so are the related tRNAs [23], ultimately resulting in slower translation. Thus, we considered an objective function that rewards designs using the most frequent codons as follows:

$$f_2(z) = \sum_{c \in C(z)} |Q(aa(c)) - q(c)| \tag{6}$$

where z is a candidate solution, Q is the frequency of the most frequent codon for the amino acid $aa(c)$ encoded by the codon c in a given species, and q is the frequency of codon c used in z. The lower bound for the codon usage objectives function is 0, which is obtained when each amino acid is encoded by the most frequent codon in the target species; conversely, the upper bound is obtained when all rare codons are used. Although our objective function is not accurate, introducing a new accurate model for evaluating translation efficiency is outside the scope of this research work.

Block length variance objective function. DNA assembly methods work best when the fragments of DNA have approximately the same size. Thus, we reward designs with blocks of homogeneous size by defining the following objective function:

$$f_3(z) = \frac{1}{|B(z)|} \sum_{b \in B(z)} (l(b) - \hat{l}(b))^2 \tag{7}$$

where b belongs to the set of blocks $B(z)$ of the solution z, l is the length of the block and $\hat{l}$ is the average block length in the design z. The block variance minimum is 0 when each block has the same length, whereas its maximum is $(l_{max} - l_{min})^2/4$, with l_{min} and l_{max} being the minimum and maximum admissible fragment length, respectively.

Block number objective function. A small number of blocks usually simplifies and speeds up the assembly process. Thus, we evaluated each solution considering the number of blocks required for the assembly as follows:

$$f_4(z) = |B(z)| \tag{8}$$

where $B(z)$ is the set of blocks defined by z. Obviously, the minimum value is $l(s)/l_{max}$, whereas the maximum number of blocks is simply $l(s)/l_{min}$.

Achieving an optimal design with respect to all four requirements is not trivial, as they have conflicting objectives. For example, optimizing codon usage can introduce AT/GC rich regions in a construct; similarly, while splitting the construct in fragments could overcome GC restrictions, it increases manufacturing complexity. It is clear that as the complexity of the constructs and the number of requirements increase, finding an optimal trade-off is challenging.

Finally, it is important to note that the minimum and maximum of an objective function might be difficult or impossible to compute; however, using heuristic estimates, such as the value of the best known solution, usually works well in practice.

3. Results

We tested MOODA on an extensive dataset of DNA constructs to assess the quality of solutions and its computational efficiency.

Currently, no benchmark is available to evaluate DNA design methods, effectively hindering a fair assessment of the methods available in literature. Therefore, as part of our work, we developed a testbed to generate DNA sequences with tunable features.

Here we assume that our input sequences represent modular designs consisting of a set of transcription units (TUs) made of a promoter, a CDS and a terminator [26]. We then parameterized our dataset considering the number of TUs encoded, the length of the constructs, their GC content and codon usage. The length of the CDS of each TU was set by sampling the number of codons from a Poisson distribution with $\lambda = 250$, which is approximately equal to the average number of codons in *Escherichia coli* genes (288.67 codons in the HUSEC2011 strain) [27], whereas the amino acid sequence and the frequency of each codon were generated at random. We then set the length of promoters and terminators by sampling from a Poisson distribution with $\lambda = 500$ bp. For each TU component, the GC content of the sequence was set at random by sampling from a Beta distribution with $\alpha = k \times t$ and $\beta = k \times (1 - t)$ with $k = 150$ and $t = 0.55$; this leads to TUs with a GC content of $\sim 55\%$ on average. Finally, we generated three datasets consisting of 10 sequences made of 5, 10, 20 TUs, with a final sequence length ranging from 8481 bp to 35 264 bp.

We then redesigned our 30 sequences with respect to four design problems characterized by a varying number objectives, namely P1, P2, P3 and P4 (see Supplementary Table S1); ultimately, we tested our method on a benchmark of 120 design problems.

We run the standard MOODA implementation on our benchmark using the parameters reported in Supplementary Table S2, and in Supplementary Table S3 for the sequence editing operators. Since MOODA is a stochastic algorithm, we performed five independent runs for each problem and parameters combination to estimate the expected quality and optimality of the designs.

3.1 Evaluation of design quality

Evaluating the quality of solutions returned by multi-objective optimization algorithms is not trivial, since standard metrics, such as the root mean square error (RMSE), are poor performance indicators. Instead, we used the hypervolume indicator, which is a robust metric used for assessing the quality of a set of Pareto optimal solutions [28]. Let $y \in \mathbb{R}^k$ be a vector of size k, where y_i is the value of the i-th objective function. The hypervolume indicator is a function $V_k : \mathbb{R}^k \to \mathbb{R}$ returning the volume enclosed by the union of the polytopes $p_1, \cdots, p_i, \cdots, p_k$, where p_i is the intersection of the hyperplanes arising from y_i and the axes. In practice, V_k provides an approximation of how many solutions are dominated by a set of Pareto optimal solutions, where the higher the values of V_k, the better is the quality of the non-dominated set. Computing the hypervolume requires the definition of a reference point, estimated either analytically or numerically; in our case, we used the minimum value of each objective function as reference point. It is important to note that the hypervolume value is an unscaled metric; thus, its interpretation is not straightforward. To overcome this problem, we first evaluate the hypervolume of the search space, V_Ω, by computing the hypervolume for the polytope bounded by the reference point and the nadir point; here we defined the nadir point as the vector of the maxima of each objective function. Then, we computed the normalized hypervolume, NV_k, as V_k/V_Ω; intuitively, NV_k values close to 1 are associated with optimal trade-off solutions.

We analyzed the quality of solutions for Problems P1 and P2 and observed that MOODA achieves near-optimal results regardless the number of TUs in each construct, with an average normalized hypervolume of 0.95, ranging from 0.93 to 0.97 (see Figure 1). Interestingly, we observed negligible differences in design quality between parameter settings, although better solutions are usually found with a higher number of iterations rather than large design pools.

We then analyzed solutions for the 3-objective Problems P3 and P4. Consistent with our previous findings, we obtained excellent results for P3 regardless of the number of TUs, with an average $NV_k = 0.94$, ranging from 0.90 to 0.97; as expected, we see a linear decrease in quality with the increasing number of TUs, albeit always approximately > 0.93 on average. As already observed, better solutions are usually obtained by increasing the number of iterations rather than the size of the pool; this difference becomes evident when designing constructs with 20 TUs (see Figure 1C and Supplementary Figure S1).

Surprisingly, we found worse performances on P4, which includes the codon usage objective function, with $NV_k = 0.5$ on average (see Figure 1D). Upon inspection of the non-dominated sets, we found that the codon usage objective function was consistently far from the optimal value. We then reasoned that this could be because the codon usage procedure changes very few codons, resulting in extremely suboptimal designs. Thus, we run MOODA on P4 by allowing the codon usage operator to alter more codons, by setting $\sigma_c = 0.75$; as expected, we observed an increase in quality, albeit limited to 0.64 on average (see Supplementary Figure S2). This result suggests that as GC content is taken into account, finding a trade-off with codon usage becomes significantly more difficult.

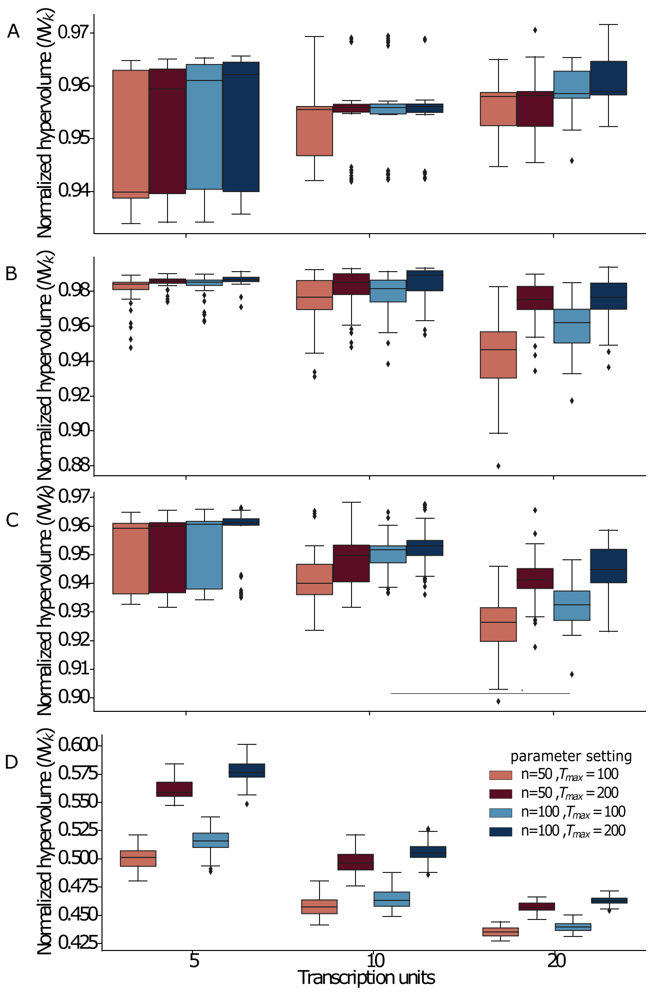

Figure 1. Evaluation of design quality. We report the normalized hypervolume values, NV_k, for different parameter settings for the Design Problems A) P1 (GC content, block number), B) P2 (GC content, block variance), C) P3 (GC content, block variance and block number) and D) P4 (GC content, codon usage, block number). The normalized hypervolume, NV_k, is the ratio between the hypervolume, V_k, of the trade-off solutions generated by MOODA and the hypervolume of the design space, V_Ω. We report normalized hypervolume values for each design problem at increasing number of transcription units; here n and T_{max} represent the pool size and the number of iterations, respectively.

Taken together, we showed that MOODA provides near-optimal designs for the vast majority of test cases. We found that the algorithm performs remarkably well despite no tuning of the editing operators, suggesting the overall robustness of our framework.

3.2 Evaluation of design optimality

The normalized hypervolume indicator provides a quantitative measure of solutions quality, but it does not inform on whether the solutions found by the algorithm are the best trade-offs possible. Here we studied which parameter settings are likely to provide optimal trade-off solutions, that is solutions that are globally Pareto optimal. In general, rigorous proof of global optimality is non-deterministic polynomial acceptable problems (NP)-hard; thus, we relaxed our requirements and reverted to an approximate measure.

We defined the approximately global Pareto optimal set as the union of all non-dominated solutions identified by u independent algorithms for a given set of objective functions. In our experiments, for each design problem, we obtained an approximate global Pareto optimal set, $\hat{P}_f$, by combining non-dominated solutions obtained by running MOODA with different parameters settings. Then, we computed R_θ, that is the proportion of global Pareto optimal solutions found by running MOODA with parameter setting θ, normalized according to the pool size (see Supplementary Table S2); intuitively, the best parameter setting will have values of R_θ close to 1.

We found that MOODA consistently finds the vast majority of global Pareto optimal solutions when setting the pool size to $n = 100$ and the maximum number of iterations to $T_{max} = 200$, with R_θ values ranging from 0.3 for Problem P1 to 1 for Problem P4 (see Figure 2). Consistent with our design quality analysis, we observed a linear dependency between the number of iterations and higher R_θ values (0.5 on average), with significant differences depending on the number of TUs in the construct, ranging from 0.05 for P1 to 1 P4. Conversely, we observed that the algorithm requires large pools when increasing the number of objectives in P3 and P4, suggesting that as the design space becomes bigger, more solutions need to be sampled.

Here we showed that the probability of finding globally optimal trade-off depends on the number of iterations the algorithm is allowed to perform. This result suggests that promising solutions are likely to be found as a result of iterative improvements, rather than by simple stochastic sampling.

3.3 Computational complexity analysis

We then analyzed the running time of our algorithm on all instances of our benchmark. For consistency, we performed all our experiments on a system with 2 Intel Xeon Gold 6130 central processing units (16 cores, 2.10 Ghz), 128 Gb DDR4 random access memory and running Scientific Linux 7; we then recorded the user time and averaged across five independent runs.

We found the running time to scale linearly with the number of TUs and iterations (see Figure 3), with a running time ranging from 100 to 8000 seconds. Moreover, while the time remains comparable across P1, P2 and P3, we found MOODA to be substantially slower on P4; this can be explained by the use of the codon usage operator, which is computationally taxing.

Since the quality and the number of global Pareto optimal solutions depend more on the number of iterations than the pool

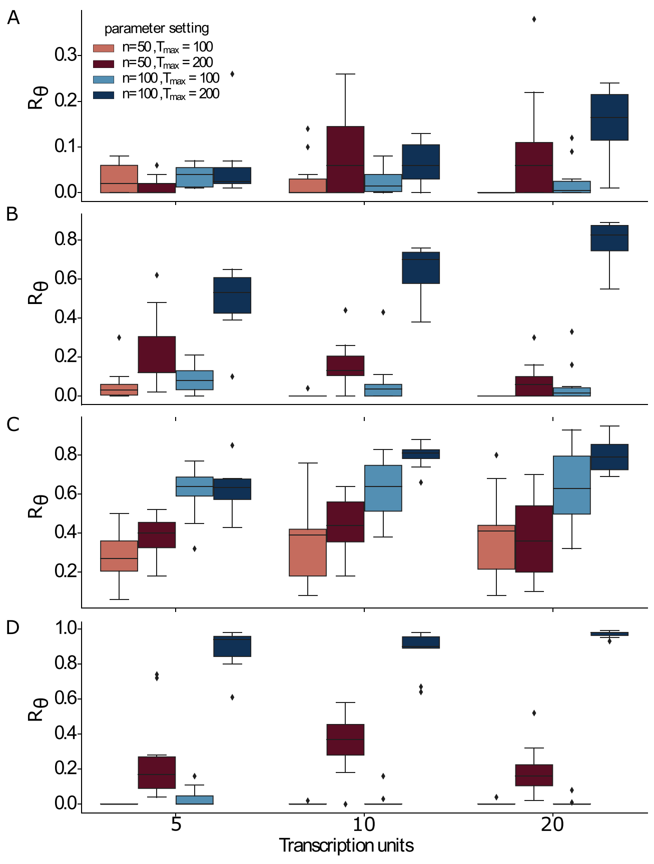

Figure 2. Evaluation of design optimality. We report the percentage of globally Pareto optimal solutions, R_θ, derived from the global Pareto front, $\hat{P}_f$, for the four Design Problems A) P1 (GC content, block number), B) P2 (GC content, block variance), C) P3 (GC content, block variance and block number) and D) P4 (GC content, codon usage, block number). We report R_θ values for each design problem at increasing number of transcription units; here n and T_{max} represent the pool size and the number of iterations, respectively.

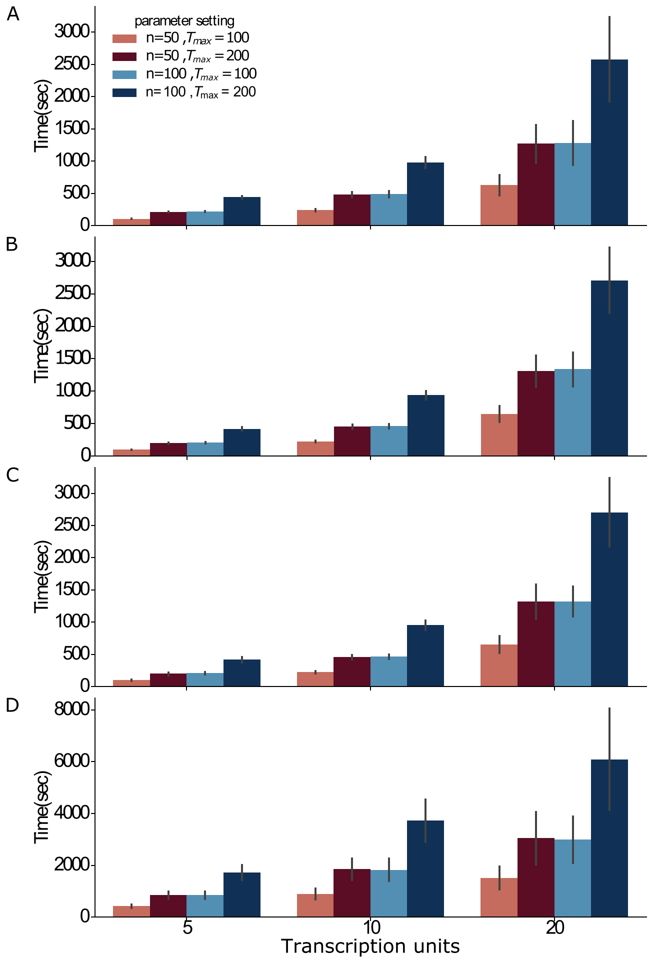

Figure 3. Running time analysis. We report the average running time, measured in seconds, of each parameter settings for the four Design Problems A) P1 (GC content, block number), B) P2 (GC content, block variance), C) P3 (GC content, block variance and block number) and D) P4 (GC content, codon usage, block number). We report the average running time at increasing number of transcription units; here n and T_{max} represent the pool size and the number of iterations, respectively.

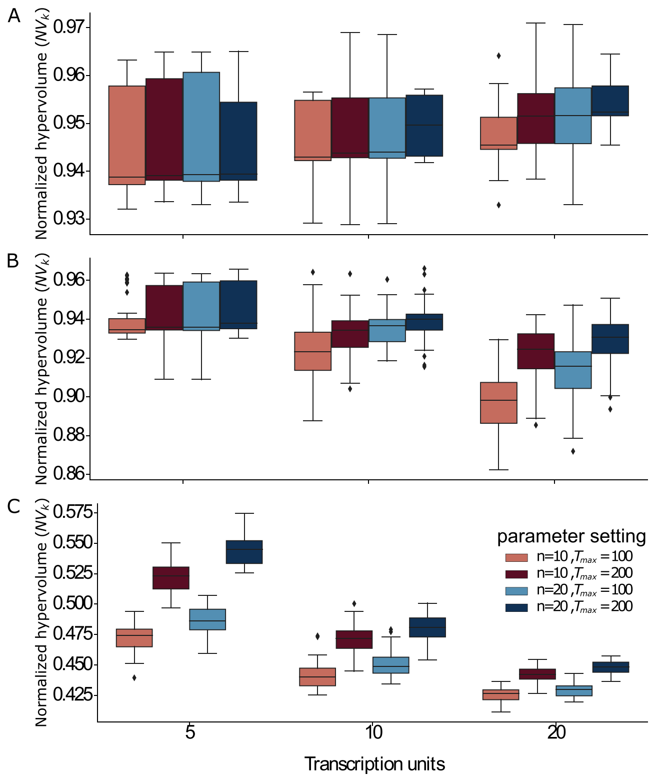

Figure 4. Evaluation of the design quality obtained using MOODA and using MOODA using the archive system. We report the normalized hypervolume values, NV_k, for different parameter settings for the Design Problems A) P1 (GC content, block number), B) P3 (GC content, block variance and block number) and C) P4 (GC content, codon usage, block number).

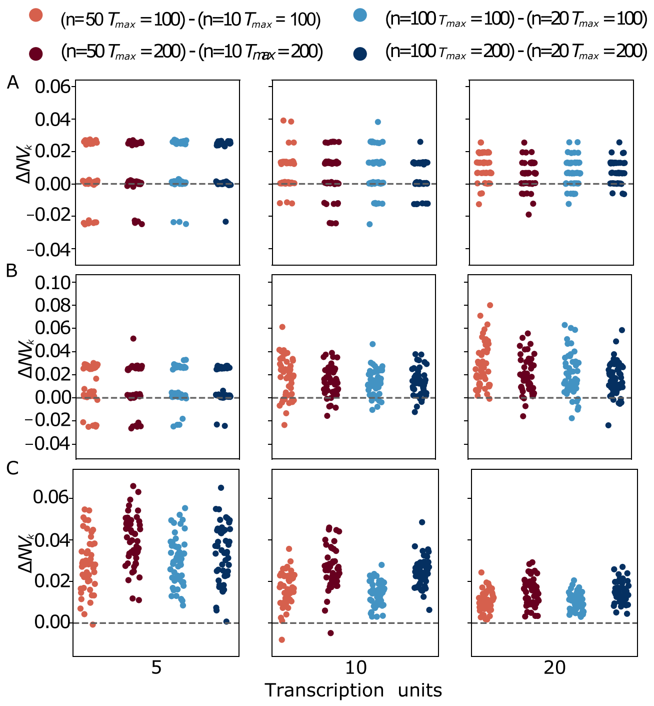

Figure 5. Comparison of design quality between standard MOODA and MOODA using the archive system. We report the difference in normalized hypervolume, ΔNV_k, between standard MOODA and the MOODA with the archive system for the Design Problems A) P1 (GC content, block number), B) P3 (GC content, block variance and block number) and C) P4 (GC content, codon usage, block number). Positive values of ΔNV_k are associated with better quality of the standard MOODA solutions compared to the archive version.

size, we decided to test whether we could obtain the same performances at a lower computational cost, by using the same number of iterations but reducing the pool size by 5-fold. To mitigate the risk of finding fewer Pareto optimal solutions, we used the archive system implemented in MOODA, by setting its size $m = 100$ for all experiments (see Supplementary Table S4); with these settings, the maximum number of non-dominated solutions remains approximately comparable between different experiments. We then evaluated the quality of the designs obtained in terms of normalized hypervolume and compared these values to the normalized hypervolume values obtained with standard parameter settings (see Supplementary Table S2), limiting our analysis to experiments with an equal number of iterations for Problems P1, P3 and P4.

Interestingly, we found that using the MOODA archive system effectively compensates for the smaller pool size; specifically, the algorithm was able to produce near-optimal results (see Figure 4), showing negligible differences compared to the design produced by running MOODA with standard parameter settings (see Figure 5). Conversely, the difference in running time is extremely significant (see Figure 6), with a drop of 2000 seconds on average; in particular, the archiving system exponentially reduces the running time on more complex problems (e.g. TUs = 20), leading to MOODA being up to 2.2 h faster for P4.

Taken together, we showed that MOODA has a running time growing linearly with sequence complexity. The use of an archiving system to keep track of non-dominated solutions effectively reduces the computational burden of our method.

4. Discussion

Advances in chemical synthesis and molecular assembly techniques have enabled a plethora of synthetic biology applications of increasing complexity. Nonetheless, designing a DNA construct that can be easily manufactured remains a complex and time-consuming task.

Here we developed a new mathematical framework and a companion algorithm to tackle the design and assembly of a biological construct as a multi-objective optimization problem, aiming at finding the best trade-offs between conflicting design and manufacturing requirements. To the best of our knowledge, this is the first time that the concept of Pareto optimality has been proposed to simultaneously design and plan the assembly of DNA molecules. Moreover, we introduced quantitative measures of design quality, which provide useful information to speed up the design-build-learn-test cycle.

We performed extensive experiments and showed that our approach can find near-optimal manufacturable designs for arbitrary long and complex DNA molecules. We found that the probability of finding optimal trade-off solutions scales linearly with the number of iterations allowed to our method, and it is only marginally affected by the size of the pool of solutions. We further refined our algorithm by adding an archiving system to keep track of non-dominated solutions found throughout the optimization process, which dramatically reduces the running time of our method and ultimately allows end users to run complex analyses on standard desktop machines. We released our software as an open-source Python package, which can be easily installed from PyPi, Anaconda or Docker and extended with plug-ins.

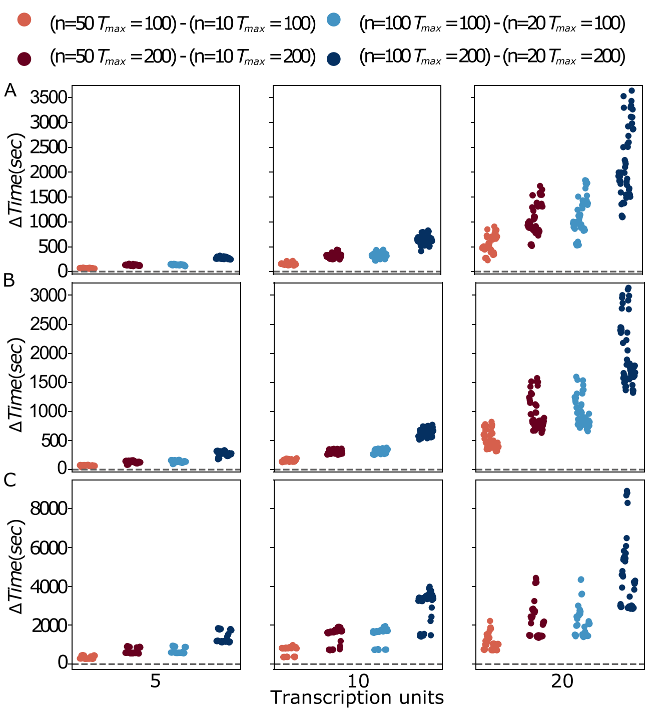

Figure 6. Comparison of the running time between standard MOODA and MOODA using the archive system. We report the difference in running time, measured in seconds, between standard MOODA and MOODA with the archive system for the Design Problems A) P1 (GC content, block number), B) P3 (GC content, block variance and block number) and C) P4 (GC content, codon usage, block number). Positive values of ΔTime are associated with MOODA archive system being faster than the standard implementation.

We are also aware of the limitations of our work. In particular, like every optimization method, the quality of solutions depends on the effectiveness of the search procedures and the accuracy of the objective functions to capture specific requirements; in biology, this has often proved to be a complex problem itself, as we experienced in our codon usage optimization experiments. Nonetheless, as models of biological processes become more accurate, defining objective functions that can exactly capture biological behavior will be feasible, and our method is ready to take advantage of these advances.

Ultimately, with the advent of large-scale synthetic genome projects, our framework provides exciting opportunities to do extensive chromosome editing in mammalian and plant systems.

Supplementary data

Supplementary data are available at SYNBIO Online.

Data availability

Source code and Docker images are available at: https://github.com/stracquadaniolab/mooda.

The Python package is available at: https://pypi.org/project/mooda-dna.

The Anaconda package is available at: https://anaconda.org/stracquadaniolab/mooda.

Author contributions

A.G. and G.S. conceived the algorithm. A.G. developed MOODA and performed experiments. V.Z. developed supporting software. A.G. and G.S. analyzed experimental results. A.G. and G.S. wrote the manuscript.

Conflict of interest statement. The authors declare that there is no conflict of interest.

References

1. Paddon,C.J., Westfall,P.J., Pitera,D.J., Benjamin,K., Fisher,K., McPhee,D., Leavell,M.D., Tai,A., Main,A., Eng,D. *et al.* (2013) High-level semi-synthetic production of the potent antimalarial artemisinin. *Nature*, **496**, 528–532.
2. Hutchison,C.A., Chuang,R.-Y., Noskov,V.N., Assad-Garcia,N., Deerinck,T.J., Ellisman,M.H., Gill,J., Kannan,K., Karas,B.J., Ma,L. *et al.* (2016) Design and synthesis of a minimal bacterial genome. *Science*, **351**, aad6253–aad6253.
3. Richardson,S.M., Mitchell,L.A., Stracquadanio,G., Yang,K., Dymond,J.S., DiCarlo,J.E., Lee,D., Huang,C.L.V., Chandrasegaran,S., Cai,Y., Boeke,J.D. and Bader,J.S. (2017) Design of a synthetic yeast genome. *Science*, **355**, 1040–1044.
4. Engler,C., Gruetzner,R., Kandzia,R. and Marillonnet,S. (2009) Golden gate shuffling: a one-pot DNA shuffling method based on type IIs restriction enzymes. *PLoS One*, **4**, e5553.
5. Gibson,D.G., Young,L., Chuang,R.-Y., Craig Venter,J., Hutchison,C.A. and Smith,H.O. (2009) Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat. Methods*, **6**, 343–345.
6. 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.
7. Nielsen,A.A.K., 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–aac7341.
8. Hillson,N.J. (2014) DNA cloning and assembly methods. *Methods Mol. Biol.*, **1116**, 245–269.
9. Appleton,E., Tao,J., Haddock,T. and Densmore,D. (2014) Interactive assembly algorithms for molecular cloning. *Nat. Methods*, **11**, 657–662.
10. Oberortner,E., Cheng,J.-F., Hillson,N.J. and Deutsch,S. (2017) Streamlining the design-to-build transition with build-optimization software tools. *ACS Synth. Biol.*, **6**, 485–496.
11. Yang,K., Stracquadanio,G., Luo,J., Boeke,J.D. and Bader,J.S. (2016) BioPartsBuilder: a synthetic biology tool for combinatorial assembly of biological parts. *Bioinformatics*, **32**, 937–939.
12. Stracquadanio,G., Romano,V. and Nicosia,G. (2013) Semiconductor device design using the BIMADS algorithm. *J. Comput. Phys.*, **242**, 304–320.
13. Stracquadanio,G., Drago,C., Romano,V. and Nicosia,G. (2010) Multi-objective optimization of doping profile in semiconductor design. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. ACM, Portland, OR, USA, pp. 1243–1250.
14. Hu,X.-B., Wang,M. and Di Paolo,E. (2013) Calculating complete and exact pareto front for multiobjective optimization: a new deterministic approach for discrete problems. *IEEE Trans. Cybern.*, **43**, 1088–1101.
15. Das,I. and Dennis,J.E. (1998) Normal-boundary intersection: a new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM J. Optim.*, **8**, 631–657.
16. Audet,C., Dennis Jr,J.E. and Le Digabel,S. (2012) Trade-off studies in blackbox optimization. *Optim. Methods Software*, **27**, 613–624.

17. Zhang,Q. and Li,H. (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.*, **11**, 712–731.
18. Deb,K., Pratap,A., Agarwal,S. and Meyarivan,T. (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, **6**, 182–197.
19. Knowles,J.D. and Corne,D.W. (2000) Approximating the nondominated front using the pareto archived evolution strategy. *Evol. Comput.*, **8**, 149–172.
20. Zitzler,E. and Thiele,L. (1998) Multiobjective optimization using evolutionary algorithms — A comparative case study. *Parallel Problem Solving From Nature — PPSN V*, Springer, Berlin, Heidelberg.
21. Zheng,H. and Wu,H. (2010) Gene-centric association analysis for the correlation between the guanine-cytosine content levels and temperature range conditions of prokaryotic species. *BMC Bioinformatics*, **11**, S7.
22. Zhou,Z., Dang,Y., Zhou,M., Li,L., Yu,C.-H., Fu,J., Chen,S. and Liu,Y. (2016) Codon usage is an important determinant of gene expression levels largely through its effects on transcription. *PNAS*, **113**, E6117–E6125.
23. Novoa,E.M., Jungreis,I., Jaillon,O., Kellis,M. and Leitner,T. (2019) Elucidation of codon usage signatures across the domains of life. *Mol. Biol. Evol.*, **36**, 2328–2339.
24. Gibson,D.G., Young,L., Chuang,R.-Y., Venter,J.C., Hutchison,C.A. and Smith,H.O. (2009) Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat. Methods*, **6**, 343–345.
25. Deb,K., Pratap,A., Agarwal,S. and Meyarivan,T. (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, **6**, 182–197.
26. Agmon,N., Mitchell,L.A., Cai,Y., Ikushima,S., Chuang,J., Zheng,A., Choi,W.-J., Martin,J.A., Caravelli,K., Stracquadanio,G. and Boeke,J.D. (2015) Yeast golden gate (yGG) for the efficient assembly of S. cerevisiae transcription units. *ACS Synth. Biol.*, **4**, 853–859.
27. NCBI Resource Coordinators (2016) Database resources of the national center for biotechnology information. *Nucleic Acids Res.*, **44**, 853–859.
28. Zitzler,E., Knowles,J. and Thiele,L. (2008) Quality assessment of Pareto set approximations. In: Branke J, Deb K, Miettinen K, Słowiński R (eds). *Multiobjective Optimization*. Springer, Berlin, Heidelberg, pp. 373–404.