THE UNIVERSITY *of* EDINBURGH

Computer Aided Engineering of Biological Systems

Angelo Gaeta

Doctor of Philosophy
University of Edinburgh
2021

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Angelo Gaeta*)

To my family.

Abstract

The rapid growth of Synthetic Biology(SynBio) led to significant breakthroughs. CRISPR/CAS and the synthesis of synthetic chromosomes are just examples of what researchers achieved in the field in the last decade. However, the synthesis of synthetic constructs or organism is still a non-trivial and challenging task. To assure reproducibility and simplify the process, researchers rely on the Design-Build-Test and Learn cycle (DBTL), a workflow deriving from engineering disciplines. Therefore, engineers widely employ DBTL in Electronic Design Automation (EDA) and Computer-Aided Design (CAD), particularly for the design of circuits. As in CAD, researchers can select the components for a specific design and predict the outcome. Nevertheless, the biological nature of DNA components complicates the DBTL. DBTL consists of fours different steps: Design, Build, Test and Learn. The Design step consists of selecting the components of a synthetic construct (e.g. Promoters, Coding Sequences and terminators) and design the theoretical circuit. In the Build step, DNA synthesis and molecular assemblies allow to synthesise the components and assemble constructs. Then, the Verification step verifies that the sequence and the expression of the assembled constructs match with what specified in the Design step. The last step, Learn, evaluates the performances of the constructs and the analysed data informs the next Design step. The purpose of this work is to explore the use of the DBTL cycle in SynBio and introduce two methods focusing on different steps of the cycle: Multi-Objective Optimisation algorithm for the Design and Assembly of DNA constructs (MOODA) and Nanogate. MOODA is the first genetic algorithm implemented for the Design and Assembly of synthetic constructs. One of the main limitations of DBTL is the dependency between steps, particularly between the Design and Build steps. The phosphoramidite synthesis imposes limits on the content and size of the molecule. As a consequence, researchers progressively edit the design to meet DNA synthesis constraints. As a result, the difference between the sequence of assembled and designed construct become significant. MOODA returns the list of optimised trade-offs between the Design and Build constraints. Nanogate instead derives from Nanopore and Golden Gate Assembly and focus on the Test step of the cycle since its purpose is to verify DNA sequences after assembly on an industrial scale. Nanogate takes advantage of Nanopore long reads since the length of a synthetic construct can usually be contained in a single Nanopore read. Nanogate, unlike standard alignment algorithms, employs hash codes to verify the similarity between the reads and the library of parts. The hash codes reduce the complexity of the algorithm and increase the True Positive rate.

Contents

Chapter 1

Introduction

1.1 Synthetic biology

The term synthetic biology appeared for the first in an article by Barbara Hobom, to define bacteria genetically engineered with DNA recombinant technology [10]. She used the expression as a synonymous of what we today refer as bio-engineering because bacteria had been modified by humans (synthetic approach). The term was used again in 2000 by Eric Kool during the American Chemical Society in San Francisco, to describe the artificial synthesis of organic molecules, where the term was used as an extension of biomimetic chemistry, which is the synthesis of artificial molecules having the same function of biological molecules [10].

Currently, there is not a consensus on what is synthetic biology but, in general, we refer to an engineering-related approach to rationally design and construct biological compounds, functions and organisms not found in nature or to redesign existing biological parts and systems to carry out new functions [40]. Synthetic biology embraces different disciplines, including molecular and systems biology, chemistry, biophysics, computer-aided modelling and design, and overlaps with others e.g. genetic and metabolic engineering [40]. The difference between genetic engineering and synthetic biology is subtle; in genetic engineering, modification or transfers of one or few genes are performed in existing living cells, while in synthetic biology multiple genes are involved for the creation of new pathways or *de novo* genomes or chromosomes.

Synthetic biology brings the engineering principles of abstraction and characterisation to biology. Biological units with a specific function, such as Ribosome Binding Site (RBS) or Coding Sequence (CDS), are defined as parts. RBS are mRNA sequences located upstream of a starting codon and allow the ribosome to bind the RNA sequence. CDS are the DNA or RNA sequence coding for a protein. Biological parts can be assembled into genes, and multiple genes can form pathways, finally, more pathways can form a chromosome or a genome [22]. One of the main challenges of synthetic biology is to apply Computer-Aided Design (CAD) principles to biological engineering; while CAD is well established for electronic engineering and has led to the growth of the Electronic Aided Design (EDA) field, CAD application to biology has been limited. In EDA, CAD allows not only to design circuits using standardised parts but even to predict its performances. A similar approach in biology would allow to design of entire chromosomes or genomes and predict their behaviour

1.2 Synthetic biology Applications

Synthetic biology has found application in many scientific and engineering fields, most notably healthcare, environment and energy. In the healthcare field, a synthetic biology approach is used for the production of synthetic viruses able to attack and kill selectively cancer cells. A biological circuit can be designed to produce proteins able to bind exclusively molecules produced by cancer cells[37]. A different application focuses on the production of active compounds. Many drug precursors are extracted from plants, such as artemisinin for treating malaria or taxol as a cancer agent, but their process is usually slow and low throughput; synthetic biology has been

used to maximise their production by transferring and adapting their pathways to Escherichia coli or yeast cells [58, 2].

In the environment field, the creation of biological circuits for detection of pollutants, such as heavy metals or pesticides, is one of the most promising application [40].

A synthetic biology approach in this context allows the computational design of biological circuits, that transferred in bacteria will produce non-natural sensor proteins to detect pollutants, such that the whole cells will be used as a bio-sensor device [4]. Another application consists in the removal of pollutants, even if for their degradation, bacteria or plants are already employed, a synthetic biology approach could allow the construction of chimeric proteins containing at the same time, bacterial and mammalian domains to bind or reduce heavy metal and radioactive ions[68]. An intriguing application is also the use of bacterial cells as a factory to produce chemicals with renewable resources. In this respect, an attempt has been already done, E.Coli cells have been used to produce polyhydroxyalkanoates using 1 gene from a plant species and 8 genes from yeast, all transferred into E.coli cells [44].

In the energy field, one of the most promising application is the production of carbon-based fuel, using renewable resources. A first study focused on using a computer-aided design method to design DNA sequences, that transferred in E.Coli cells would produce butanol and branched-chain higher alcohols, using sugar as a source of energy [67]. A different application instead, focused on obtaining butanol, biodiesel or biocarbon using lignocellulose as source[31]. A different approach involves the use of cyanobacteria or microalgae, to exploit photosynthesis to produce alkanes [36]. An emerging application involves the production of hydrogen to be used as fuel, using light and water as a source. The synthetic biology contribution can be used to optimise the photosynthetic process using heterologous enzymes [42].

Nevertheless, as the number and complexity of synthetic biology application grow, questions about biosafety, biosecurity, environmental, socio-economic risks arise, along with ethical questions about life and its manipulations [40].

1.3 Design Build Test and Learn cycle (DBTL)

The engineering of biological systems relies on the Design, Build, Test and Learn (DBTL) workflow. The design step focuses on obtaining biological parts and modify their DNA sequence to obtain a specific behaviour. The Build step involves preparing the system for manufacturing, which involves adapting the DNA sequence to obey synthesis constraints and choose an assembly plan. The test phase consists of analysing and characterising the engineered system, while the learn phase consists in extracting useful information from the test experiments to improve the design. One of the main challenges of the DBTL cycle is that the Design and Build steps have conflicting objectives; this often leads to designs that cannot be manufactured or DNA constructs that do not meet design requirements. As consequence, engineered systems have often performances that are lower than expected and the DBTL cycle is iterated several times before obtaining the expected results [5]. Moreover, the orthogonality concept adapts perfectly to electronics, where a single circuit or component can be easily isolated. In synthetic biology, instead, the orthogonality concept is purely theoretical. Indeed, interactions within and between cells can not be ignored and may significantly affect the behaviour of a synthetic construct [70].

1.3.1 Design step

The design phase of the Design-Build-Test and Learn cycle, represent the first step to design a biological circuit. It consists in defining the structure or the function of the system, using, for example, mathematical description of system behaviours or rule-based description of system architecture[5]. Then the structure or the function is translated into biological parts (DNA sequences), that can be retrieved from repositories, such as The Registry of Standard Biological Parts or the Inventory of Composable Elements (ICE)[30]. ICE is an open-source registry for biological parts, where the user can download parts in different formats a graphically visualise them. Editing parts underlying DNA sequences allow to adapt them to the host organism and improve their performances in terms of gene expression.

One of the main challenges in sequence design[41],[53] is to find a balance to adjust gene expression [3],[75]. Indeed, circuits performances depend on the genetic context, strains and growth conditions [38], [15]. In this regard, different studies [34][52][45] report how codon usage bias(CUB) is crucial for cell survival. In particular [34],reports how even the position of each synonymous codon affects the gene expression.

In addition, regulators when over-expressed can become toxic for the cell, compromising the circuit performances [7].

Tools and algorithms have been proposed to give support during the design of biological circuits. On the basis of their approach to genetic design, they can be categorised as rational, combinatorial and tools to engineer the codon usage.

Computer Aided Sequence Design

A rational approach to sequence design consists in choosing parts forming biological circuits on the basis of biochemical or biophysical models. The most popular tool belonging to this category is Cello[47].

Cello Cello (cellular logic) accelerates the design of biological circuits, allowing the user to introduce complex gene regulation. Its algorithm is based on cells behaviour, they activate or inactivate a specific function under a specific condition.

Cello algorithm to design the circuit first converts the Verilog file into a first circuit diagram, excluding logic gates that are not included into the UCF file. Second, to each logic gate in the diagram circuit, a set of DNA parts is assigned from the UCF file. For each logical gate, multiple DNA parts set are available, each one generating a different response, then to find the optimal assignment a Monte Carlo algorithm is applied. Third, the diagram circuit with the assigned DNA parts is translated into a linear DNA sequence, each gate is converted into a set of DNA parts and a combinatorial algorithm is used to design the linear DNA sequence respecting constraints. The optimised circuit can be further modified by the user, modifying the Eugene rules in the UCF file, allowing the creation of different variants of a circuits having the same constraints and functions[47]. Nevertheless, Cello algorithms have been tested only on prokaryotic systems, where the gene regulation is considerably less complex than eukaryotic systems.

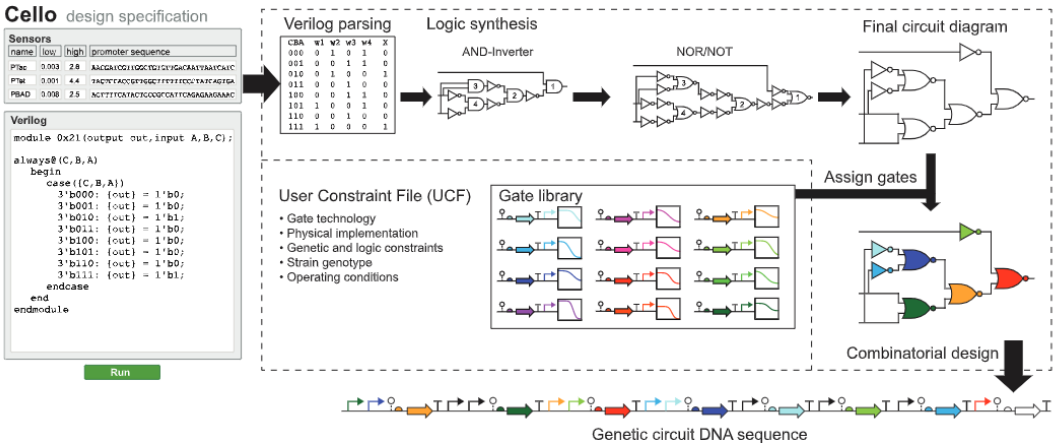

Figure 1.1: **Cello workflow [47]**. Users first write a verilog and upload sensors and a UCF. Cello deisgns a truth table on the basis of the verilog, the truth table is used as input to geberate a circuit diagram. Regulators are assigned from a library to each gate (each color is a different repressor). Combinatorial design is then used to concatenate parts into a linear DNA sequence. For parts representation SBOL[61] language is used, whereas parts colours represent physical gates

.

Combinatorial sequence design

A combinatorial approach to sequence design can be used to test different variants of a biological circuit to optimise its performance, allowing to chose the variant resulting with an higher gene expression for a particular strain. The most popular tool belonging to this category is Double Dutch [59].

Double Dutch Double Dutch uses a mathematical approach called the design of experiments(DOE) and applies formal grammar and heuristic algorithms to build empirical models that test several biosynthetic pathways without testing all the possible variations. The tool allows to create libraries of pathway variation and provides info on statistical analysis based on the pathway, assembly cost and risk of homologous recombination.

The algorithm at first, it takes as input a list of categorised DNA parts or set of parts, where the role, the strength and the sequence of each part are specified. Second, each module is assigned to the factors library or to the levels library. Modules stable throughout the whole pathway are assigned to the factors library, while the rest is assigned to the levels library. Third to each factor a level module is assigned creating a pathway, then the algorithm evaluates for each pathway the number of homologous regions in each DNA part (as the number of homologous regions increase, the probability for the biological circuit to have lower performances or to be not assembled at all increase as well), the cost (the cost is evaluated on the basis of the reusability of each DNA part) and the gene expression, moreover the user can choose the weight of each one of these objectives.

Although, Double Dutch allows the design of combinatorial pathways giving to the user a list of possible solutions for a specific design, libraries are not optimised.As a result, there are usually too many constructs to test, therefore increasing the cost of the entire bio-engineering process.

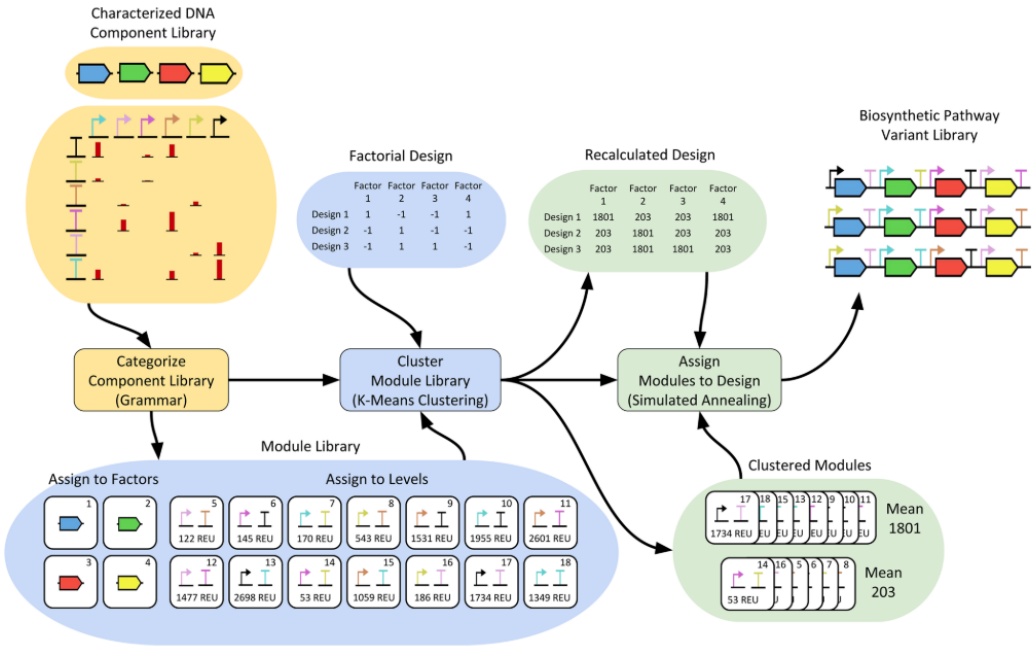

Figure 1.2: **Double Dutch workflow [60]**. The workflow shows how parametrized modules are taken in input, then their mean parameter values are turned into the new low and high levels of the factorial design Then the tool assigns the parametrized modules from their clusters to the corresponding levels of the design. As a result, Double Dutch returns an optimized library of biosynthetic pathway variations based on a factorial design.

.

Codon usage engineering

While tools for combinatorial and rational design are used to design a biological circuit, choosing the biological parts to use and their order without editing the underlying DNA sequences, tools to engineer the codon usage (codons juggling) are used to edit DNA sequences of biological parts.

Due to the genetic redundancy, most of the amino acid in a CDS can be encoded by multiple codons. Moreover, each strain or species has his own preferences about the codon usage, thus the codon usage engineering can be used to enhance circuit performances or adapt it to the host organism. Since both the codons juggling and the build optimisation involve the editing of the underlying DNA sequences, most of the tools used for the codon juggling are used for build optimisation as well(e.g. Gene Designer [69], Build optimisation software tools(BOOST) [49], BioPartsBuilder [73] and Gene Design [55]), then these tools will be described in the sequence building section.

1.3.2 Build Step

The Build phase of DBTL consists in physically build a biological circuit. The assembly of biological parts can be performed in a wet lab *in loco* or by DNA foundries. There are different biological methods to assemble parts, such as Golden Gate [24], Gibson[28] and USER [13]. Each assembly method has its own constraints, thus circuits need to be optimised editing the underlying DNA sequences. Tools are available for this part of the cycle as well, they can be categorised in tools for sequence optimisation and tools for assembly planning. Golden Gate and Gibson are two different assembly methods allowing to join together different DNA fragments, by exploiting DNA overhangs. Golden Gate uses Type IIS restriction enzymes and T4 DNA ligase, to generate overhangs on the DNA fragments and join them in vitro. Gibson assembly method instead can be divided in the following steps and requires the use of exonuclease, DNA polymerase, and DNA ligase :

Via PCR an overlap is added to one of the fragments to join, then the exonuclease digest the $5'$ ends, the fragments with the overlap anneal and ligase then close the gaps.

Sequence optimisation

Sequence optimisation tools are used to optimise DNA sequences underlying a biological circuit to respect assembly constraints. Constraints vary on the basis of the assembly method used by the DNA vendor or wet lab. They may involve adjusting the GC%, modify the sequence length splitting it into blocks, add flanking regions, removal of secondary structures, motives, hairpins or repeated regions. It is important that sequence optimisation tools, do not introduce frameshift mutations during the process. Frameshift mutation are caused by insertion or deletion of one or more bp, when introduced into a CDS it affects all the codons downstream the mutation. For example, if we consider to have the following codons:

$$\text{ATG CCA GAA GGA} \tag{1.1}$$

producing the peptide:

$$\text{M P E G} \tag{1.2}$$

when a frameshift mutation is introduced, for example in the second codon:

$$\text{ATG CTC AGA AGG A} \tag{1.3}$$

All the amino acids downstream to the mutation are affected, indeed after the mutation the sequence of the peptide is:

$$\text{M L R R} \tag{1.4}$$

Frameshift mutations in general severely affects the functions of a protein, and may even introduce premature stop codon.

While some sequence optimisation tools are used to edit DNA sequences in pathways or single genes(Gene Designer[69], Build optimisation software tools[49], BioPartsBuilder[73] and

Gene design [55]) some others are used to edit entire chromosomes or genomes (Genome Calligrapher [17] and Automated Design of Assemblable Modular Synthetic Chromosomes[56])

Gene Designer Gene designer [69] is a software that aims at designing and optimising synthetic DNA segments. Gene design accepts as input sequence objects such as amino acid, DNA or ORF. In Gene designer it is possible to juggle codons on the basis of the CAI score (Codon adaptation index). CAI is defined as the geometric mean of the weight associated with each codon over the length of the gene sequence (measured in codons), the weight is evaluated as follows:

$$w_i = \frac{f_i}{max(f_j)} \quad i, j \in \text{amino acid synonymous codons} \tag{1.5}$$

If CAI is equal to 1 only the most used codon for a specific amino acid is selected, this could represent a drawback, because an overexpressed gene generates an overexpressed mRNA with a high concentration for a specific tRNA, causing imbalances into the tRNA population. Proteins produced from mRNA including exclusively most used codons could be over-expressed at the expense of essential proteins for cellular growth, that could be under expressed. Moreover, with a low codon flexibility, it could be difficult to avoid repetitions and secondary structures.

Gene Designer uses predefined or manually edited codon usage tables, where each codon has a probability score based on his distribution in the host organism. Different sequences are then generated on the basis of a Monte Carlo algorithm, codons distribution are described by the codon usage tables. Then each generated sequence is analysed to search for DNA motives which could result in secondary structures, repeats, restriction site and GC% imbalances. If a secondary structure is found or the GC% is off limits, then codons in that region are replaced by synonymous one, until a feasible sequence is generated.

Gene designer algorithm for juggling codons, help in enhancing gene expression as much as possible into the host organism, without sacrificing his cellular growth. On the other hand, a ranking of the solutions found by the Monte Carlo algorithm is absent, moreover it is applied exclusively for codon juggling and the removal of manufacturing constraints is applied *a posteriori*, modifying the codon usage obtained with the Monte Carlo Algorithm.

Build optimisation software tools BOOST [49]is a suite of build optimization software tools for the automated generation of DNA constructs ready for DNA synthesis. BOOST is a DNA assembly planning software, its workflow can be divided into three microflows: codon adjustment, sequence polishing and sequence partitioning. In the codon adjustment microflow, BOOST algorithm checks if the input sequence is a DNA or a protein. If the sequence is a protein, it is translated into DNA, then depending on the organism, codons within the DNA sequence are replaced. The algorithm for codon juggling allow the user to select between three different strategies: random, mostly used and balanced.

In the random strategy, the algorithm replaces each amino acid codon with a random synonymous one. With the most used strategy, the algorithm replaces each amino acid codon with the organism most used codon. For the balanced strategy the algorithm generates for each amino acid a random number between 0 and 1, at beginning each amino acid is replaced by most used codon, then the codon usage is subtracted from the random number, when it becomes equals to 0, the second most used codon replaces that amino acid and so on.

During the sequence polishing microflow, an algorithm verifies the sequence against constraints. If the sequence violates any of them, the algorithm checks if the violation is inside a coding region, then the sequence containing the violation is translated into a protein, codons are replaced with synonymous one until the violation is solved or a solution with as few violations as possible is returned. If the violation is not within the coding region, the input sequence is returned as it was and the user can plan for a different sequence design.

In the sequence partitioning microflow, BOOST algorithm divides the DNA sequence into blocks trying to minimise their number and to search for overlapping sequences inside the block that match the vendor criteria. If no overlapping sequence is found inside a block, the algorithm increases the number of blocks. To set sequence polishing and partitioning parameter is possible to select a DNA vendor among IDT, Gen9, Thermo Fisher, Twist, DNA2.0 or set personalized constraints.

BioPartsBuilder BioPartsBuilder[73]is a synthetic biology tool for combinatorial assembly of biological parts. BioPartsBuilder can retrieve parts from annotated NCBI sequences, then each part can be optimised for the assembly by juggling codons, removing restriction enzyme constraints, splitting each part into blocks, adding prefix and suffix to each block. Although algorithms for codon optimisation and restriction site removal are the same that are used in Gene Design[58], sequence partitioning algorithms are designed *ad hoc* to make a part feasible by commercial providers. First, the *sequence segmentation* algorithm evaluates the optimal number of blocks to use by dividing the sequence part length by the maximum block length,

$$n = [L/L_max] \tag{1.6}$$

where n is the optimal number of blocks, L is the length of the sequence and L_max is the block maximum size. The *overlap encoding* algorithm introduce overlapping sequences inside blocks(the overlap size L_0 and sequences S_0 are set by the user). The overlap between two block i and $i+1$ is equal to:

$$i = last \quad \frac{L_0}{2} \quad and \quad i+1 = first \quad \frac{L_0}{2} \tag{1.7}$$

If an overlap it is not unique or not included into the user list, the *overlap encoding* algorithm move the split until a feasible overlap is found. The *standardization* algorithm add the prefix Fp and the suffix Fs defined by the user at the beginning end at the end of each block.

Gene Design Gene design [55] is a suite of web-based programs consisting in six modules(reverse translation, codon juggling, silent restriction site insertion, silent restriction site removal, oligonucleotide design module, sequence analysis) that can be used individually or in series to optimise a DNA sequence.

In the first module, "reverse translation", the algorithm converts a protein into a nucleotide sequence, every amino acid is replaced by the most used codon defined by relative RSCU (synonymous codon usage). The genetic code is redundant and universal, therefore multiple codons code for the same amino acid. Nevertheless, redundant codons are in general not used equally, and each species has his on preference on which codon to use for a specific amino acid. This phenomenon is known as codon usage bias and affects both eukaryotes and procaryotes. The RSCU is a method to measure the codon usage bias [63], according to the following formula:

$$RSCU_{i,j} = \frac{n_i x_{i,j}}{\sum_{j=1}^{n_i} x_{i,j}} \tag{1.8}$$

Where i is the amino acid, n_i is the number of codons that code the amino acid i. $x_{i,j}$ instead, represents the number of occurrences of codon j.

The second module "codon juggling", allows the user to juggle codons choosing between four different algorithms, each of those using RSCU data.

The first replace each codon with the one that can maximise the expression. The second replace every codon with the one that can maximise the expression, but that is different from the original one. The third algorithm replaces every codon with a synonymous one that is as different as possible from the original one. The fourth algorithm simply replaces every codon with a random one.

In the third module, "silent restriction site insertion", add a restriction site to a DNA sequence. The algorithm, at first, put the restriction sequence in a frame, translate it into an amino acid sequence, screen the translated input DNA sequence for the same amino acid motif, then replace synonymous codons to add a restriction site. The fourth module, silent restriction site removal is based on the same algorithm of the previous module, but carries out the opposite process, removing restriction site motives.

In the "oligo-nucleotide design" module, the algorithm splits sequences with a length > 800 bp into overlapping chunks with a length ~ 500 bp.

In the last module, "sequence analysis", the algorithm prints information about the analysed sequence: basis%, displays a map of restriction sites contained into the sequence and a list of the absent ones.

Gene design has a large variety of different algorithms that can be used individually or together, furthermore, others can be added to the Gene Design web page to introduce new functions. Moreover, Gene design presents some unique operators that are absent from other tools BOOST, Genome Calligrapher, GeneDesigner, e.g. "silent restriction site insertion" and "silent restriction site removal".

Genome Calligrapher Genome Calligrapher [17] is a computer-aided design tool for genome re-factoring of bacterial chromosomes. It can be used to optimise the GC content and remove features that can interfere with the DNA synthesis.

Genome calligrapher is designed for prokaryotic genomes but can be used on eukaryotic sequences without discontinuous CDS features. The tool accepts as input a DNA GenBank sequence and automatically removes di- and trinucleotide repeats and homopolymeric sequences, instead the motives to be removed are specified by the user.

For the GC% adjusting, the user can set both the upper and lower thresholds for GC content and define the skew factor, a parameter describing the codons frequency to adjust the GC% (the algorithm can even be forced to replace every available codon to adjust GC%).

The codon juggling is performed specifying for a codon table to use related to a specific organism. Genome Calligrapher includes 2776 codon tables representing bacterial organisms in addition to the Saccharomyces Cerevisiae table. The user can even select the recording probability, a parameter defining the frequency of synonymous codon swapping(if the source and the final organism are the same, this parameter should be set to 0) and the offset parameter, that allows the preservation of the first n codons of a CDS to be replaced, due to their importance for translation initialisation.

Moreover, Genome Calligrapher allows specifying for immutable codons or for codons that must be completely erased from the genome. A Monte Carlo algorithm is used for synonymous codons substitution according to the selected codon usage. The percentage of codon to swap is chosen by the user selecting a number between 0 and 1, where 0 means that no codon will be switched except for the one needed to resolve constraints, and 1 means that every codon will be replaced with a synonymous one as described in the codon table.

Genome Calligrapher is designed to edit prokaryotic DNA sequences or entire prokaryotic genomes. Nevertheless, only eukaryotic sequences with no discontinuous coding sequences can be used as input and the codon juggling algorithm could be improved to return a codon usage more similar to the destination organism.

Automated Design of Assemblable Modular Synthetic Chromosomes Richardson et al.[56] developed an optimisation algorithm to engineer entire chromosomes. First, it uses a parallelisable indexing to catalogue edits unable to influence gene functions. Second, it splits the optimisation problem into sub-problems suitable for parallel computing. Third dead-end elimination is applied to exclude dead paths.

The algorithm takes as input only annotated sequences, which can be edited only within coding regions to avoid interference with regulatory or structural features. Within the coding region, restriction sites can be added or deleted, introducing silent mutations that do not affect the protein sequence.

Each chromosome can be split into parts, the number of parts and cut sites depends on the restriction enzyme that is used. The algorithm then uses an objective function to evaluate the cost of chromosome parts depending on the enzyme that is used, then return the plans with the lowest cost.

The algorithm allows to optimise large sequences for restriction enzyme cuts and define different planes allowing the user to choose the less expensive, but the further editing of the DNA sequence is not allowed

Assembly planning

Tools for assembly planning are used to design flanking sequence to use in the assembly and to plan the DNA assembly reactions, maximising parts reuse and minimising the number of reactions to perform. Raven [6] and j5[32] are the most popular in this category.

Raven Raven[6] is a DNA assembly planning software tool suite, that produces optimised assembly plans, by allowing users to modify them interactively. Given an assembly plan, DNA parts can be linked to each other by part junctions(overhangs). Overhangs are single DNA strand, in general generated after the digestion of a restrictions enzyme. Long overhangs can be exploited with a DNA ligase, to bind the DNA fragment with the overhang to a second DNA fragment with a complementary overhang.

To assign overhangs to each part, Raven algorithm first searches for overhangs respecting assembly constraints. Second, maximise the reuse of overhangs trying to minimise the number of assembly reactions.

Third Polymerase Chain Reaction (PCR) oligos are designed on the basis of the chosen overhangs as well as well as the number of cloning and PCR steps.

If a specified step fails, it can be marked and forbidden to reappear in future and new overhangs will be designed, while successful steps can be reused in new assembly plans. Although Raven improves the design process minimising the overhangs number, those assembly failures are not computationally simulated to avoid time loss.

j5 j5[32] is a web-based software for the design automation of DNA assembly plans. The user can manually design the biological circuit with an editor and upload a list of possible DNA sequences for each circuit component, then Eugene rules can be specified to define circuit constraints(frequency of each DNA sequence, a combination of parts to be avoided and the parts that must be linked to each other). Then, the user can choose between assembly methods requiring flanking sequences(Gibson[28]) and assembly methods based on restriction enzymes (Golden Gate[24]).

In the first case, the user specifies flanking region constraints(e.g. minimum homology sequence length) and j5 algorithm design the flanking region for circuit component, if the sequence exceeds the constraints they are sequentially reduced until flanking regions respecting all of the constraints are returned. Then, a different algorithm evaluates if DNA sequences are compatible with each other (flanking regions too similar could generate secondary structures preventing assembly reactions), if not the algorithm design a hierarchical assembly process including in each reaction set of compatible DNA sequences.

In the second case overhangs for each DNA parts should not be self-cohesive or anneal to off-target overhangs, j5 algorithm then searches throughout all possible overhang combination to find: the most compatible to themselves and the other parts, that satisfies the maximum number of overhang base-pair matches(set by the user) and are as neutral as possible (close to parts boundaries).

To perform the cost evaluation, the user can select the cost parameters and j5 algorithm evaluate for each DNA part which synthesis method (direct DNA synthesis, PCR, Oligo embedding) gives as result the minimum cost.

j5 designs combinatorial DNA assembly protocols (allowing custom variants of DNA constructs that combine multiple parts), performs a cost-benefit analysis to identify which portions of an assembly process would be less expensive and designs DNA assembly strategies. Nevertheless, the j5 algorithm can be applied only on a limited number of current assembly strategies, does not scale for large systems, uses suboptimal methods to design DNA sequences (e.g., Primer3 for PCR primers) and reduce assembly constraints instead of optimising DNA sequences.

1.3.3 Test step

The Test step of DBTL aims at verifying the correctness of the Design and Build steps. Therefore the sequence of the assemblies is verified, along with their expression. DNA sequencing is one of the most common methods employed to verify that the assembled sequence matches the original design

Sanger sequencing

Frederick Sanger developed the homonymous Sanger sequencing in 1977 [62]. The method exploits the DNA duplication process with the use of labelled di-deoxynucleotide triphosphates

(ddNTPs). The process consists of four different polymerase reactions, one for each nucleotide (A, C, T, G). Besides ddNTPs and a DNA polymerase, each reaction includes a DNA template, a DNA primer and deoxynucleotide triphosphates (dNTPs). The DNA polymerase uses the DNA primer and the DNA template to generate a DNA molecule complementary to the template. When the DNA polymerase includes a ddNTP, the elongation ends, due to the absence of the -OH group required for the binding of the next nucleotide. As a consequence, each reaction will produce DNA fragments of different sizes ending with a differently labelled ddNTP. Afterwards, gel electrophoresis orders the fragments by size, observing the last nucleotide (the labelled ddNTP) will be possible to derive the sequence of the original DNA molecule. Sanger sequencing is still one of the most employed sequencing technology. Nevertheless, the costs and times required for sequencing exponentially increase with the size of the DNA to sequence. As a consequence, it is mainly used to sequence small DNA sequences or to validate the sequences of regions including variants.

Sequencing by synthesis

The most relevant exponent of sequencing by synthesis(SBS) is Illumina sequencing [11]. Similar to Sanger sequencing, Illumina sequencing employs a DNA template, DNA polymerase and reversible nucleotides Illumina sequencing includes three different workflows, library preparation, sequencing and analysis. During the library preparation, the extracted DNA is divided into fragments of the same length. The length of fragments may vary between 25 and 300 bp. Two different adapters I5 and I7, are added respectively at the beginning and end of each fragment. The sequencing reaction occurs on the flow cell, a glass plate including adapters complementary to the I5 and I7 adapters, thus binding the DNA fragment to sequence. The fragments binding the flow cell form clusters through the bridge amplification process. After cluster generation, complementary oligos bind the fragments, and a DNA polymerase extends the primer sequence with fluorescent nucleotides. Each nucleotide includes a blocker preventing the elongation of the chain. The blocker is removed after the nucleotide binds the sequence and emits fluorescence. Every time there is an emittance of fluorescence, the sequencer detects the frequencies emitted and turn them into data file called .bcl files. During the analysis step, this data is converted into FASTQ files, representing the sequence of the original fragment.

Oxford Nanopore Technology

Oxford Nanopore Technology [35] was patented in 2007. The technology allows DNA sequencing of DNA molecules with a length of up to 2000000 bp. In Nanopore sequencers, the DNA or C-DNA molecule are bound to adapters facilitating the binding with an enzyme and the movement through a pore. The enzyme regulates the speed of the fragments through the pore and assures unidirectionality. The pore is exposed to an ionic current, and every time a new nucleotide of the original template enters the pore, there is a change in the ionic current. The changes in the ionic current are recorded and employed to obtain the DNA sequence of the original template. Compared to the previous methods, Nanopore allows the generation of longer reads facilitating downstream computational analysis. Moreover, reagents and sequencers are, on average, less expensive than Illumina sequencing. Nevertheless, Nanopore sequencing shows a lower accuracy $92 - 97\%$, compared to the 99.9% of Sanger and Illumina sequencing. Given these features, Nanopore is usually preferred for rough analysis, such as the identification of viruses or bacteria in a specific environment. Moreover, due to the small size of the sequencer, Nanopore sequencing is often used for field analysis, in particular in environmental biology.

1.3.4 Learning step

Biological parts have more complex interactions compared to electronic components. The behaviour of a designed construct can often be unpredictable due to: environmental factors, such as pH and temperature, epigenetics factors, chemical interaction between compounds, lack of knowledge of the underlying biological mechanisms, and the general absence of orthogonality. Therefore, a biological construct rarely shows the predicted behaviour at the first DBTL cycle, and different DBTL iterations are necessary to obtain the desired behaviour. The Learn step

uses machine learning and statistical approaches to retrieve data from previous experiments and improve future iterations. For example, Automated Recommendation Tool (ART) [54] evaluates the biological parts of the previous experiments, then applies machine learning and Bayesian statistic to obtain recommendations on future designs.

Chapter 2

Design and assembly of DNA molecules using multi-objective optimisation

2.1 Introduction

I declare that the material and data presented in this chapter are published in the following preprint, [25] accepted for publication in Synthetic Biology (Oxford University Press). I developed the MOODA algorithm, Valentin Zulkower developed the web interface, and Giovanni Stracquadanio supervised the work.

Recent advances in synthetic biology and DNA synthesis technologies are enabling significant scientific and biotechnological breakthroughs, including the engineering of pathways for drug production [51], the construction of minimal bacterial cells [27] and the assembly of synthetic eukaryotic chromosomes [57].

Pivotal to these achievements has been the adoption of an iterative engineering workflow, known as 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, 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 current phosphoramidite synthesis poses limits on molecule length and content. These limits are usually overcome by splitting the designed sequences into shorter fragments, which can be assembled through molecular assembly techniques [24, 28], at the cost of increasing complexity both for the design and manufacturing step. 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 [60], Cello [46], j5 [33], Raven [6], BOOST [50] and BioPartsBuilder [74]. 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 [65, 66] to solve the conundrum of designing DNA for manufacturability. Similar to how electronic circuits design is informed by physical and silicon manufacturing limits, 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 our open-source Python software called Multi-Objective Optimisation algorithm for DNA Design and Assembly (MOODA); executable are available on PyPi and Anaconda, whereas source code through GitHub (http://github.com/stracquadaniolab/mooda) and a ready to use interface at http://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 sequence length.

2.2 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.

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

Let x be a sequence over the DNA alphabet $\Sigma = \{\texttt{A,T,C,G}\}$ 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 optimisation problem (MOP) as follows:

$$\min_{x \in \Sigma} F(x) = (f_1(x), f_2(x), \cdots, f_k(x)) \tag{2.1}$$

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:

$$\exists i \in \{1, \cdots, k\}\ f_i(x_1) < f_i(x_2)\ \wedge\ \forall j \in \{1, \cdots, k\}\ f_j(x_1) \leq f_j(x_2) \tag{2.2}$$

In mathematical terms, the set of trade-off solutions, or Pareto optimal set, is made of all the non-dominated solutions for F, that 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 [72].

A plethora of methods have been proposed in literature to solve multi-objective problems, both deterministic [19, 8] and stochastic [76, 21, 39, 80]. While deterministic methods provide convergence results, 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 strong convergence results.

2.2.2 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 constraints.

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 same size, each one of size $l_{min} \leq l \leq l_{min}$, 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 according 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.

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

```
1:  procedure MOODA(s, n, F, T_max)
2:      P ← Initialise(s, n)
3:      Evaluate(F, P)
4:      t ← 0
5:      while t ≤ T_max do
6:          R ← Clone(P)
7:          R ← Edit(R)
8:          Evaluate(F, R)
9:          P ← Select(P, R)
10:         t ← t + 1
11:     end while
12: end procedure
```

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 input a solution 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 4 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 biological phenotypes; for example, in prokaryotic organisms, the GC content of coding sequences correlates with their optimal growth temperatures [77]. Here we define a GC optimisation operator, which recodes a particular coding sequence $CDS_{editing}$ 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.

Algorithm 2 GC content operator

procedure GC CONTENT(σ_{GC}, T_{GC}, CDS)
  $CDS_{reference} \leftarrow$ RandomSelect(CDS)
  $CDS_{editing} \leftarrow$ Copy($CDS_{reference}$)
  while $|GC(CDS_{editing}) - GC(CDS_{reference})| \leq \sigma_{GC}$ **do**
    $C \leftarrow$ RandomSelect($CDS_{editing}$)
    $A \leftarrow$ Translate(C)
    $CL \leftarrow$ GetCodonsList(A)
    if $T_{GC} \geq$GC($CDS_{editing}$) **then**
      $C \leftarrow$ LowGCSelection(CL) ▷ selection of a codon decreasing GC
    else if $T_{GC} <$ GC($CDS_{editing}$) **then**
      $C \leftarrow$ HighGCSelection(CL) ▷ selection of a codon increasing GC
    end if
  end while
  return $CDS_{editing}$
end procedure

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 maximise the production of a particular protein [78]. However, how to recode the codons of a gene to control its transcription is poorly understood [48]. Our codon optimisation operator probabilistically recodes a fraction σ_c of the codons of a given gene, by silently replacing the current codons, according to the frequency specified in a 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 optimisation to all coding sequences at the same time, but only to one at random (see Alg. 3).

Algorithm 3 Codon usage operator

procedure CODON USAGE(σ_c, T_{CF}, CDS)
  $CDS_{editing} \leftarrow$ RandomSelect(CDS)
  $NCR \leftarrow \sigma_c *$ length($CDS_{editing}$) ▷ NCR= number of codons replacement
  for R in NCR **do**
    $C \leftarrow$ RandomSelect($CDS_{editing}$)
    $A \leftarrow$ Translate (C)
    $C \leftarrow$ SelectCodon(A, T_{CF})
  end for
  return $CDS_{editing}$
end procedure

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. [28]. Indeed, excessive fragmentation can be both expensive and increase the turn-around 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 and 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 2 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 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 [28]; 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 individuals 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 [20].

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

$$w_{f_i}(z_j) = (f_i(z_{j+1}) - f_i(z_{j-1}))/(f_i^{max} - f_i^{min}) \tag{2.3}$$

$$d(z_j) = \sum_{n=1}^{k} w_{f_i}(z_j) \tag{2.4}$$

where $d(z_j)$ is the crowding distance related to the j solution, $w_{f_i}(z_j)$ 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 the similarity between individuals and favours individuals with low similarity to improve the Pareto Front exploration. After the ranking, the top n individuals, 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 a specific 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.

2.2.3 Design and manufacturing objectives

We assessed the performance of our method by studying 4 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:

$$f_1(z) = \sum_{b \in B(z)} |T_{GC} - GC(b)| \tag{2.5}$$

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 [48], ultimately resulting in slower translation. Thus, we considered an objective function that rewards designs using the most frequent codons as follows:

$$f_2(z) = \sum_{c \in C(z)} |Q(aa(c)) - q(c)| \tag{2.6}$$

where z is a candidate solution, Q is the frequency of the most frequent codon for the amino acid $aa(c)$ encoded by c, 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 paper.

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

$$f_3(z) = \frac{1}{|B(z)|} \sum_{b \in B(z)} (l(b) - \bar{l}(b))^2 \tag{2.7}$$

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:

$$f_4(z) = |B(z)| \tag{2.8}$$

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

Achieving an optimal design with respect to all 4 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.

2.3 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 coding sequence (CDS), and a terminator [1]. We then parametrized 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 E. coli genes (288.67 codons in the HUSEC2011 strain) [18], 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 3 datasets consisting of 10 sequences made of $5, 10, 20$ TUs, with a final sequence length ranging from $8,481$ bp to $35,264$ bp.

We then redesigned our 30 sequences with respect to 4 design problems characterized by a varying number objectives, namely P1, P2, P3 and P4 (see Tab. 2.1); ultimately, we tested our method on a benchmark of 120 design problems.

Problem	**Objective functions**	**Parameters**
P1	GC content Block number	$T_{GC} = 50\%$
P2	GC content Block length variance	$T_{GC} = 50\%$
P3	GC content Block length variance Block number	$T_{GC} = 50\%$
P4	GC content Codon usage Block number	$T_{GC} = 50\%$ T_{CF} = E. coli

Table 2.1: Benchmark design problems. For each problem, we report a unique identifier, the objective functions and the corresponding parameters used.

We run the standard MOODA implementation on our benchmark using the parameters reported in Tab. 2.2, and in Tab. 2.3 for the sequence editing operators. Since MOODA is a stochastic algorithm, we performed 5 independent runs for each problem and parameters combination to estimate the expected quality and optimality of the designs.

Pool size (n)	**Number of iterations** (T_{max})
50	100
50	200
100	100
100	200

Table 2.2: Standard parameter settings used to test MOODA.

Operator	**Parameters**
GC content	$T_{GC} = 50\%$, $\sigma_{GC} = 0.05$
Codon usage	T_{CF} =E. coli, $\sigma_c = 0.05$
Block join	$l_{max} = 2000$ bp, $l_{min} = 200$ bp, $\sigma_b = 50$ bp
Block split	$l_{max} = 2000$ bp, $l_{min} = 200$ bp, $\sigma_b = 50$ bp

Table 2.3: Sequence design and manufacturing operators used in MOODA. For each operator we report, its name and the parameters used.

2.3.1 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 [79]. 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 $V_k : \mathbb{R}^k \rightarrow \mathbb{R}$ 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 un-scaled 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 analysed 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 Fig.2.1). Interestingly, we observed negligible differences in design quality between parameters 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. 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 (Fig.2.1C).

Surprisingly, we found worse performances on P4, which includes the codon usage objective function, with $NV_k = 0.5$ on average (see Fig. 2.1D). 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 due to the codon usage procedure changing 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 Supp. Fig. 2). This result suggests that as GC content is taken into account, finding a trade-off with codon usage becomes more difficult.

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 overall robustness of our framework.

2.3.2 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 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 parameters setting θ, normalized according to the

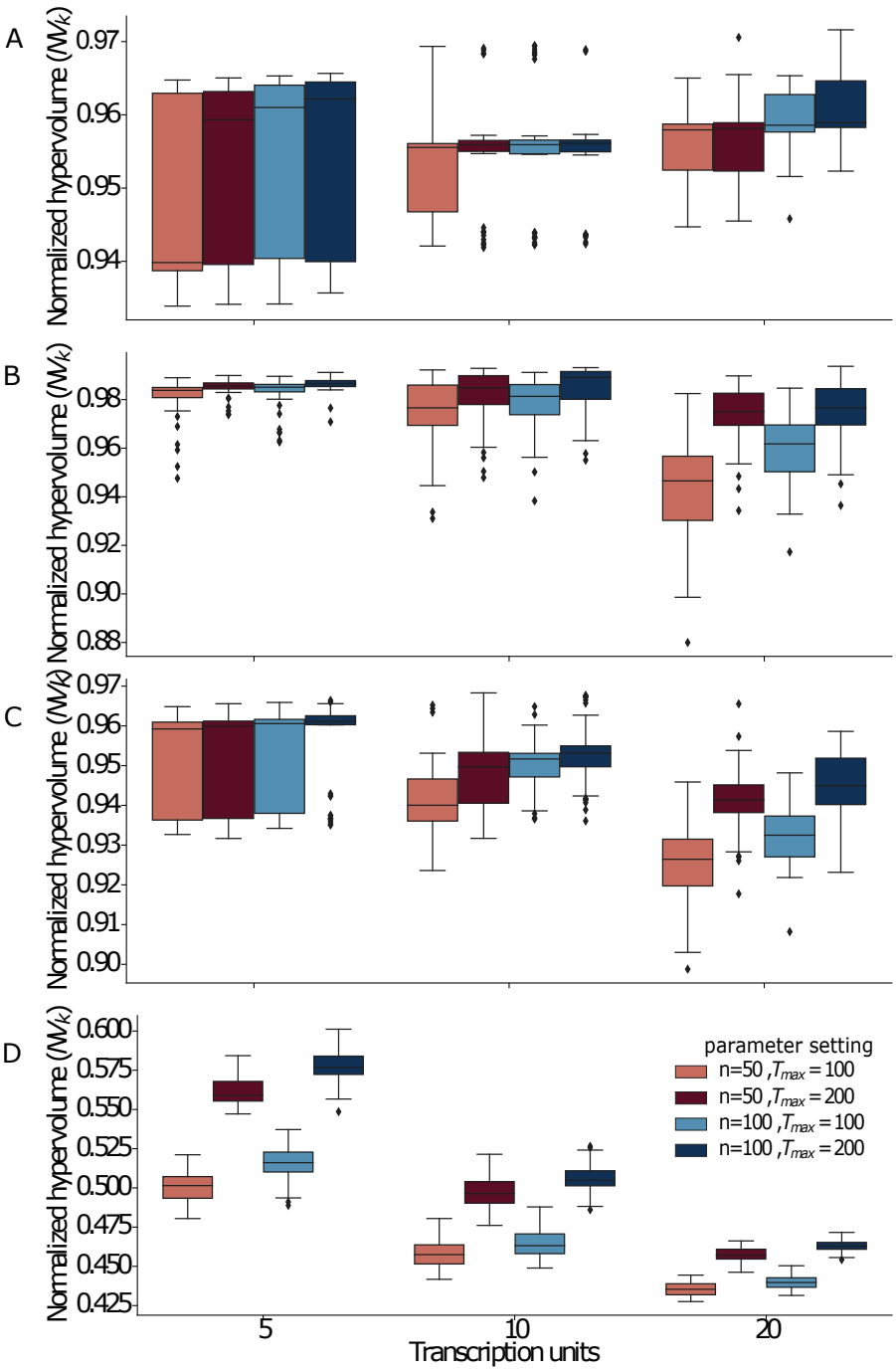

Figure 2.1: **Evaluation of design quality.** We report the normalized hypervolume values, NV_k, for different parameters settings for the design problems a) P1 (GC content, block number), b) P2 (GC content, block variance), c) P3 (GC content, block variance and block number) and d) P4 (GC content, codon 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.

pool size (see Tab. 2.2); intuitively, the best parameters 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 (Fig.2.2). 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.

2.3.3 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 CPUs (16 cores, 2.10Ghz), 128Gb DDR4 RAM and running Scientific Linux 7; we then recorded the user time and averaged across 5 independent runs.

We found the running time to scale linearly with the number of TUs and iterations (see Fig. 2.3), 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.

Pool size(n)	**Number of iterations**(T_{max})	**Archive size** (m)
10	100	100
10	200	100
20	100	100
20	200	100

Table 2.4: Parameter settings used for testing the MOODA archive system.

Since the quality and the number of global Pareto optimal solutions depends more on the number of iterations than the pool 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 Tab. 2.4); 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 parameters settings (see Tab. 2.2), limiting our analysis to experiments with an equal number of iterations.

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 Fig. 2.4), showing negligible differences compared to the design produced by running MOODA with standard parameters settings (see Fig. 2.5), ranging from -0.3 in P3 to 0.8 in P3. Conversely, the difference in running time is extremely significant (see Fig. 2.6), with a drop of 2000 seconds on average; in particular, the archiving system exponentially reduces the running on more complex problems (e.g TUs= 20), leading to MOODA being up to 2.2 h faster for P4.

2.3.4 Testing on biological data

After testing MOODA on ad hoc in-silico testbeds, we decided to test our algorithm on real DNA sequences. Gibson et al. [26] describe the synthesis of the whole DNA molecule of the mouse mitochondria and its integration into the E. Coli genome. At this regard, we reasoned to apply MOODA on the same DNA sequence and set as T_{CF} the mouse nuclear codon usage.

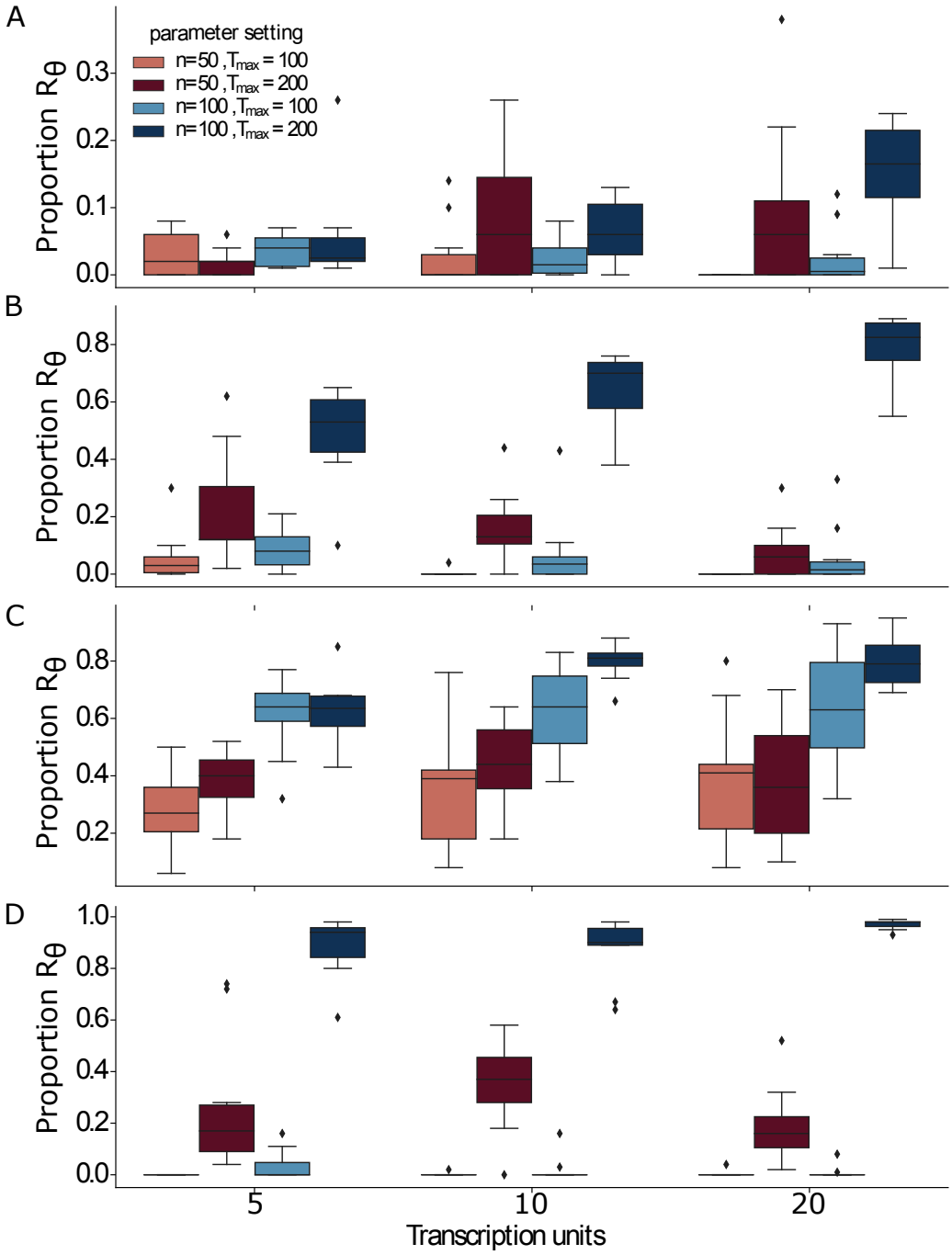

Figure 2.2: **Evaluation of design optimality.** We report the percentage of globally Pareto optimal solutions, R_θ, derived from the global Pareto front, $\hat{P}_f$, for the 4 design problems a) P1 (GC content, blocknumber), b) P2 (GC content, block variance), c) P3 (GC content, block variance and 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.

To achieve this, we edited the input sequence, due to mouse mitochondria unique DNA sequence [12]. Indeed, in mouse mitochondria, the first codon of each CDS codifies for methionine, regardless of what specified by the genetic code [12]. The editing consisted of replacing with ATG the first codon of each CDS not translated into methionine, according to the mouse

mitochondria genetic code. The mitochondria DNA sequence contains two features that are absent in silico testbeds: overlapping CDS, and a source organism genetic code different from the host organism genetic code. MOODA tackles the first by creating for each CDS a list of overlapping codons, which are not edited by the operators working on the DNA sequence. As a result, if the overlap between CDS is not in frame, the amino acid sequence remains identical across the overlapping CDS. To allow MOODA to work with different genetic codes, we implemented an ad hoc initiator. The latter replace in each CDS, each codon which codify for a different amino acid according to the host organisms genetic code. In the eventuality of overlapping CDS with a different genetic code, the initiator removes the overlap by adding between CDS a DNA sequence of 100 bp without starting codons. This procedure is necessary to recode every CDS according to the host organism genetic code without any change to the protein sequence of overlapping CDS. Moreover, the absence of starting codons avoid the generation of small peptides between CDS. To obtain the redesign of the sequence we set as input [9], $T_{max} = 500, 1000n = 10, 20m = 100$. We applied block join, block split, GC content and codon usage as operators, whereas block number, GC content and codon usage as objective functions. Parameters for both operators and objective functions are in table 2.5.

Operator	**Parameters**
GC content	$T_{GC} = 50\%$, $\sigma_{GC} = 0.05$
Codon usage	T_{CF} =Mouse nuclear, $\sigma_c = 0.75$
Block join	$l_{max} = 2000$ bp, $l_{min} = 200$ bp, $\sigma_b = 50$ bp
Block split	$l_{max} = 2000$ bp, $l_{min} = 200$ bp, $\sigma_b = 50$ bp

Table 2.5: Parameter settings used applying MOODA on mouse mitochondria DNA.

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; ultimately, we proved that MOODA can be easily scaled to tackle the design of complex constructs and the quality of solutions found by the algorithm remain unchanged when applied on real problems.

2.4 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 speedup 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 or Anaconda and extended through plugins.

We are also aware of the limitations of our work. In particular, like every optimization methods, 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 optimisation experiments. Nonetheless, as models of biological processes become more accurate, defining

objective functions that can exactly capture biological behaviour will be feasible and our method is ready to take advantage of these advances.

Ultimately, with the advent of large scale synthetic genome projects and after testing the algorithm on mouse mitochondria, we believe that our framework for DNA engineering provides exciting opportunities to do extensive chromosome editing in mammalian and plant systems.

Contributions

AG and GS conceived the algorithm. AG developed MOODA and performed experiments. VZ developed MOODA web application. AG and GS analyzed experimental results. AG and GS wrote the manuscript.

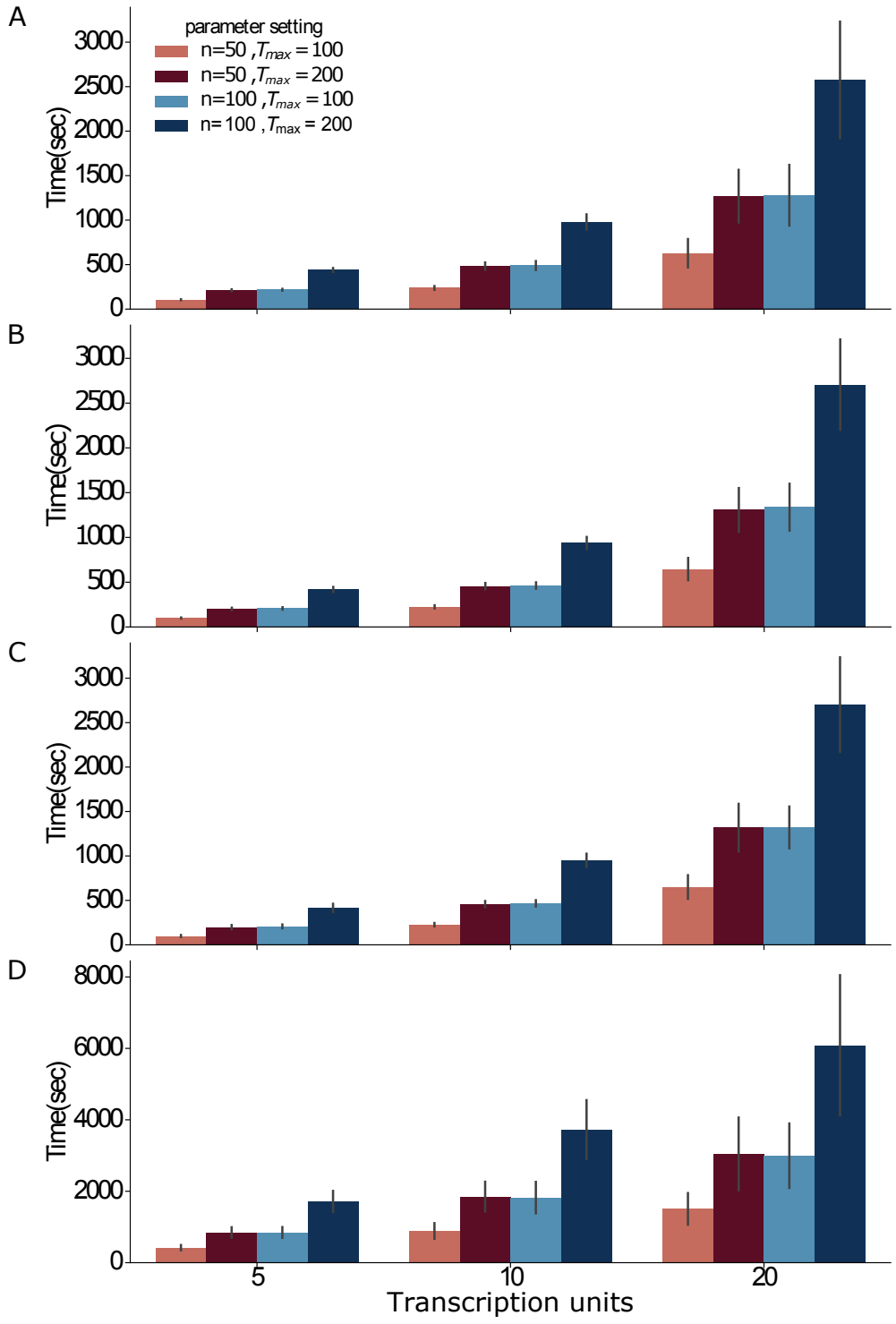

Figure 2.3: **Running time analysis**. We report the average running time, measured in seconds, of each parameter settings for the 4 design problems a) P1 (GC content, block number), b) P2 (GC content, block variance), c) P3 (GC content, block variance and block number) and d) P4 (GC content, codon usage, 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.

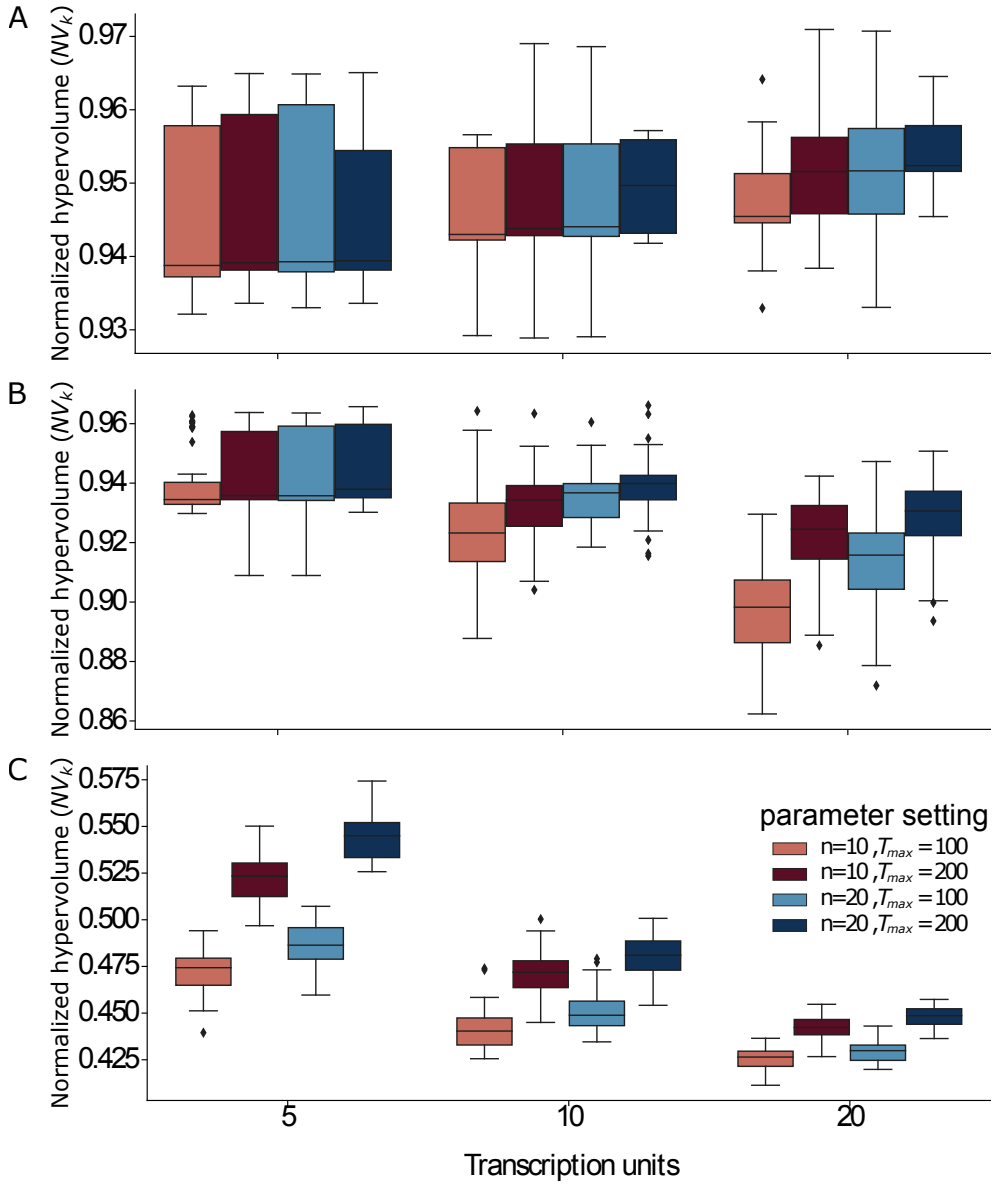

Figure 2.4: **Evaluation of the design quality obtained using MOODA and using the archive system.** We report the normalized hypervolume values, NV_k, for different parameters settings for the design problems a) P1 (GC content, block number), b) P2 (GC content, block variance), c) P3 (GC content, block variance and block number) and d) P4 (GC content, codon usage, block number).

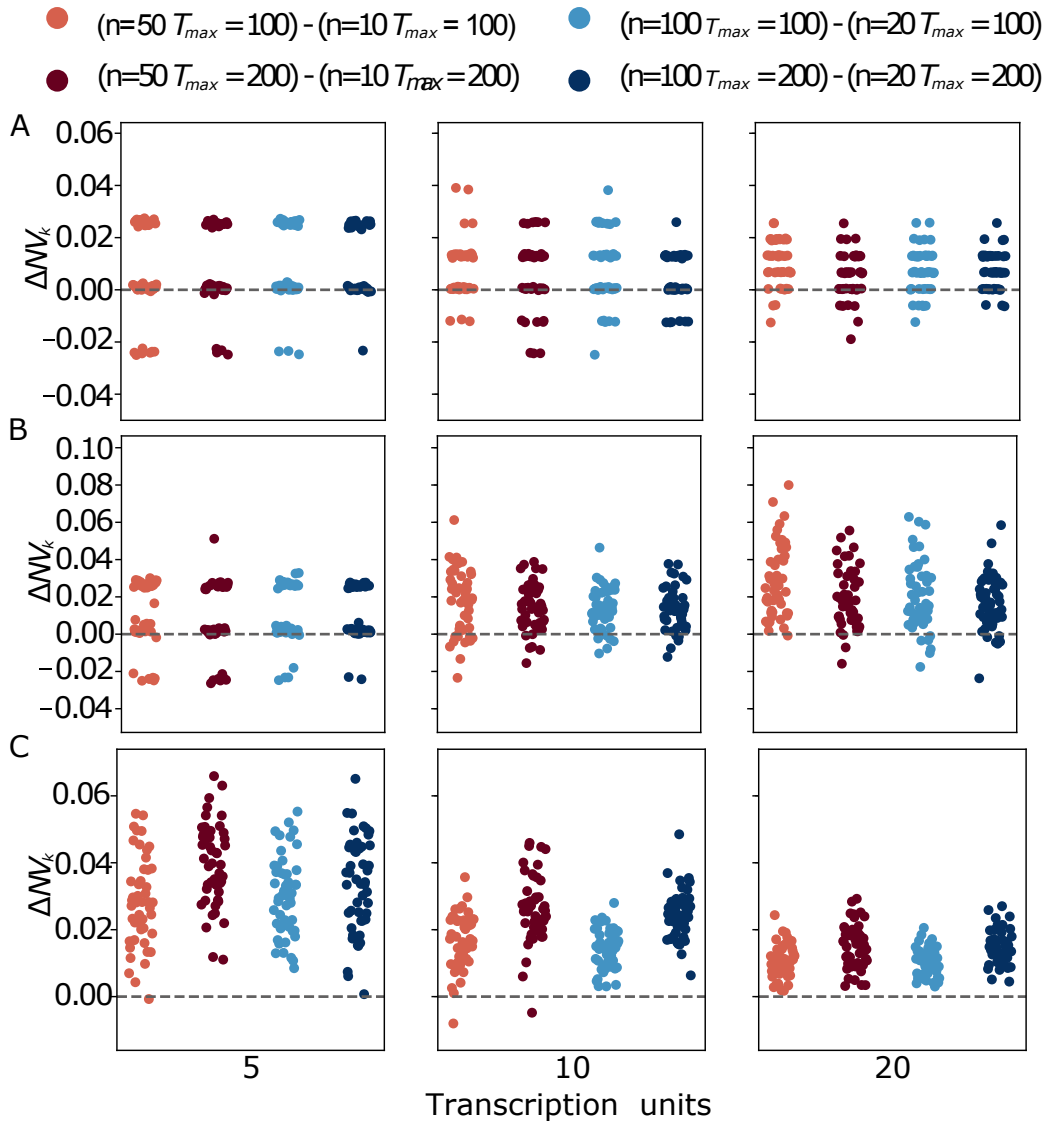

Figure 2.5: **Comparison of design quality between standard MOODA and using the archive system.** We report the difference in normalized hypervolume, ΔNV_k, between standard MOODA and the MOODA with the archive system for the 4 design problems a) P1 (GC content, block number), b) P2 (GC content, block variance), c) P3 (GC content, block variance and block number) and d) P4 (GC content, codon usage, block number). Positive values of ΔNV_k are associated with better quality of the standard MOODA solutions compared to the archive version.

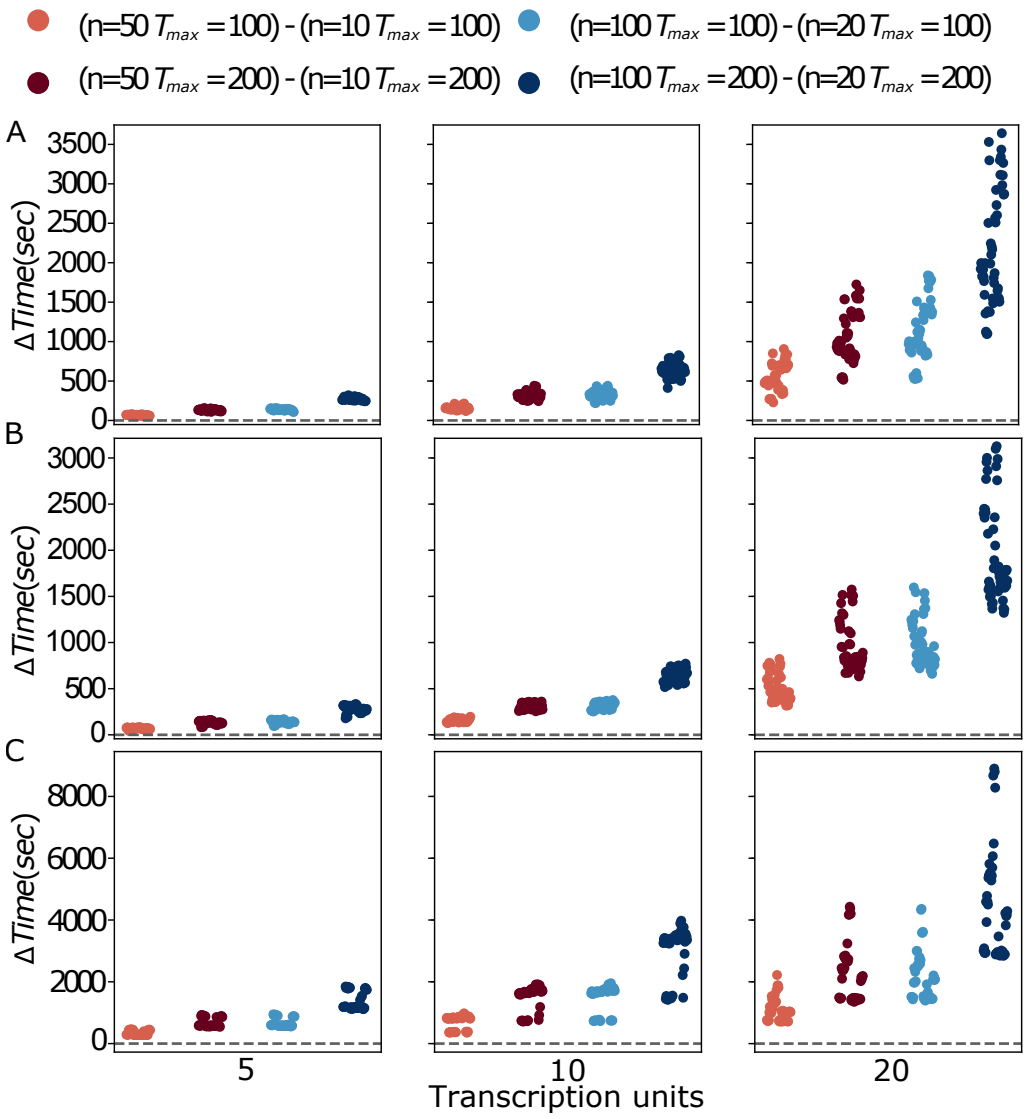

Figure 2.6: **Comparison of the running time between standard MOODA and using the archive system.** We report the difference in running time, measured in seconds, between standard MOODA and MOODA with the archive system for the 4 design problems a) P1 (GC content, block number), b) P3 (GC content, block variance and block number) and c) P4 (GC content, codon usage, block number). Positive values of ΔTime is associated with MOODA archive system being faster than the standard implementation.

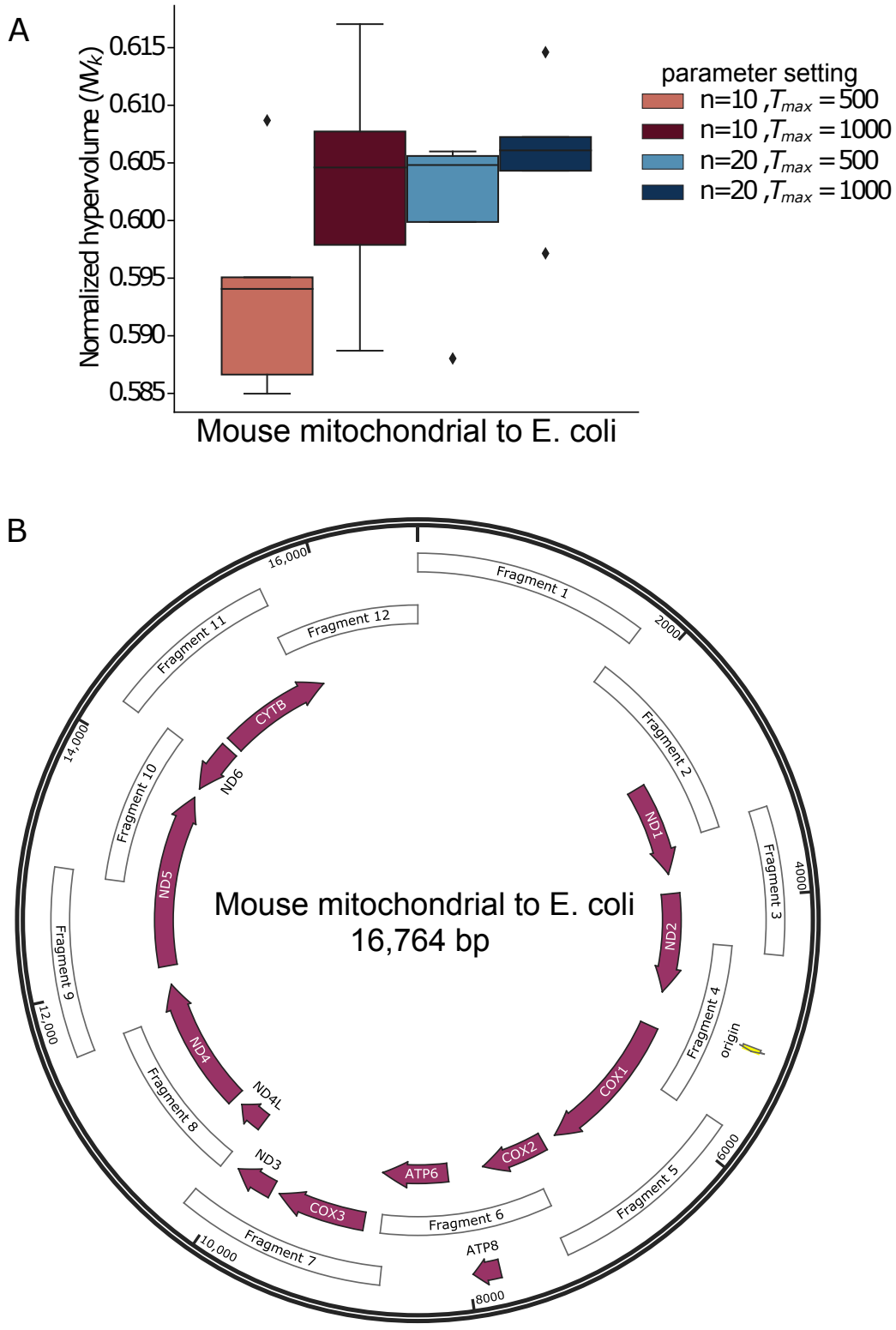

Figure 2.7: **Evaluation of design quality for mouse mitochondrial DNA redesign.** We report the normalized hypervolume after applying MOODA to mouse mitochondrial DNA sequence, setting as T_{cf} the E.coli codon usage.

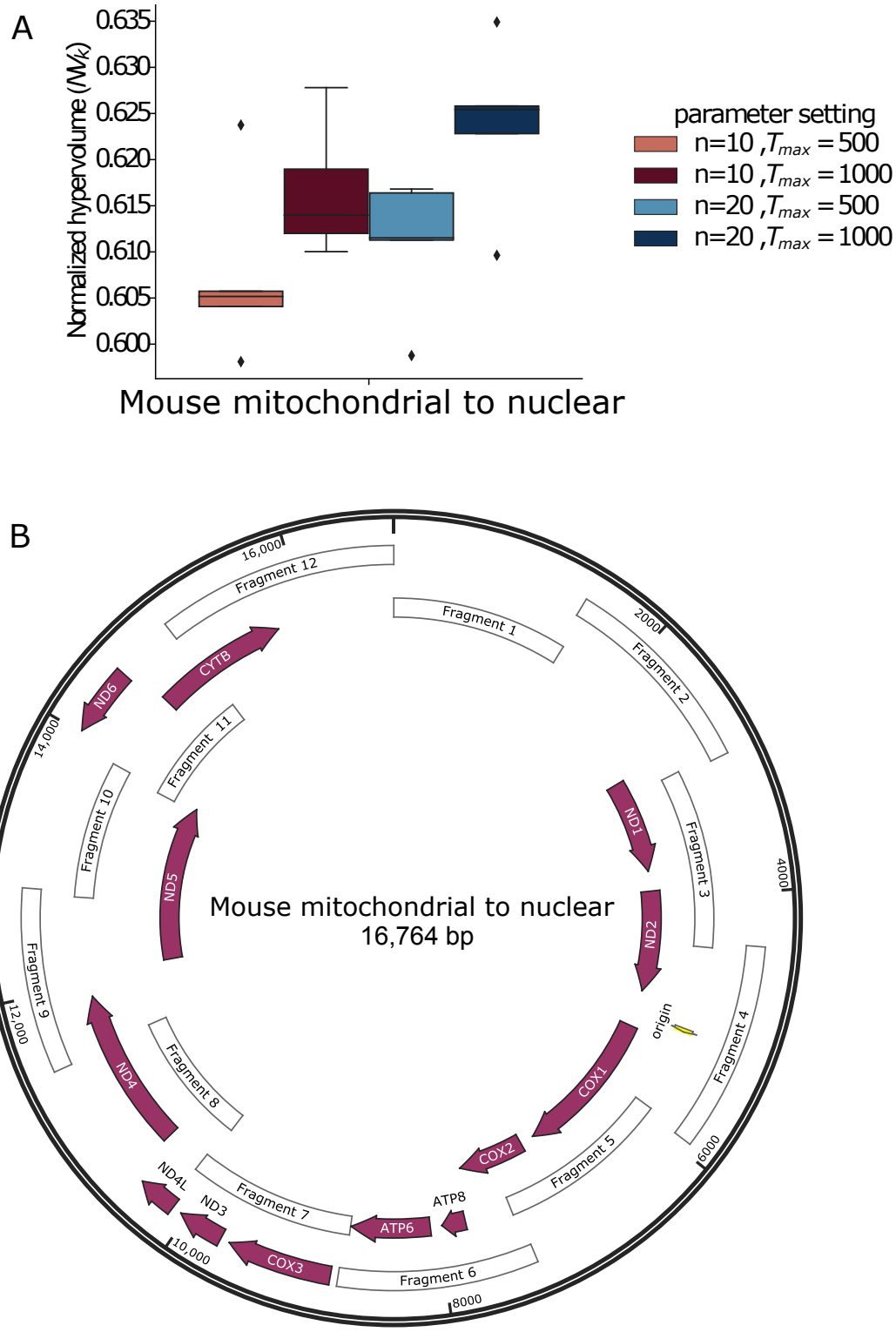

Figure 2.8: **Evaluation of design quality for mouse mitochondrial DNA redesign.** We report the normalized hypervolume after applying MOODA to mouse mitochondrial DNA sequence, setting as T_{cf} the the mouse nuclear codon usage.

Chapter 3

Alignment-free deconvolution of nanopore sequencing data for combinatorial assembly verification

3.1 Introduction

The rapid growth of Synthetic Biology led to significant breakthroughs. CRISPR/CAS and the assembly of synthetic chromosomes are just examples of what researchers achieved in the field in the last decade. However, this rapid growth would not be possible without the sustain of new computational methods. To support CRISPR/CAS experiments, researchers developed more than 60 tools only for the design of gRNAs [64]. The synthesis of synthetic chromosomes required, for instance, the implementation of Biostudio [57] an open-source framework for genome design.

In synthetic biology, assembly methods such as Gibson Assembly [28] and Golden Gate [23] are pivotal for the generation of constructs. They are the glue applied by researchers to put together different functional DNA sequences (DNA parts) such as promoters, coding sequences and terminators, into constructs or biological circuits. Nevertheless, the assembly process is error-prone, and it does not always produce the expected outcome. For instance, when using the Gibson assembly the exonuclease could digest DNA sequences shorter than 200 bp. Therefore, DNA sequence verification of the assembled construct is crucial to identify errors during the process.

3.1.1 Methods for DNA verification

Different methods are available to verify DNA sequences after assembly 1.3.3 relying on either restriction enzymes or sequencing technologies. Nevertheless, when applied on an industrial scale and long constructs, they are impractical, being expensive either time-consuming.

Restriction enzymes

Restriction enzymes methods rely on cutting a construct nearby a restriction site producing fragments, and gel electrophoresis can be applied to measure its size. These methods are more affordable and fast than the others, indeed a digestion reaction takes on average 60 minutes, and the expanses relate only to reagents and consumables. Nevertheless, restriction sites are not always available on a construct and fragments with similar sizes generates bands that are difficult to distinguish on the gel. Moreover, it is not possible to detect error not affecting the restriction site, but different regions of the construct; since the method does not provide information on the entire DNA sequence.

Sequencing methods

Methods for the verification of DNA sequences after assembly also includes sequencing technologies, such as Sanger sequencing [62] or Sequencing-By-Synthesis (SBS) Illumina [14]. Sanger sequencing allowed us to read an unknown DNA sequence for the first time in the year 1977. It does not require expensive instruments but consumables and gel electrophoresis to reveal the DNA sequence. As a consequence, compared to Illumina (SBS) is, on average faster and more affordable. Nevertheless, costs and times of Sanger sequencing increase exponentially with the number of constructs to sequence and their lengths. For example, to sequence 1 Gb with Sanger sequencing would cost $2,400,000$\$. As a consequence, Sanger sequencing is impractical on a large scale. On the other hand, Illumina (SBS) requires both expensive instruments and consumables. Different instruments are available depending on the user purpose. For example, the MiSeq target the sequencing of small genomes such as viruses and bacteria. The NextSeq and the more recent NovaSeq focus on high throughput analysis, such as human exome amplification or whole-genome sequencing. Illumina SBS compared to Sanger sequencing is more precise. Indeed, the guaranteed Q30 is on average above 80% and requires fewer manual labour.

Nevertheless, when applied to the assembly sequence verification on a large scale, it is expensive and time-consuming since each construct in a library has to be sequenced and analysed. The analysis involves alignment algorithms, mapping the reads generated against the expected DNA sequence of the construct. Nevertheless, the similarity between parts and the increasing number of assembled constructs per library complicate the analyses. Therefore, reducing the efficiency and efficacy of standard alignment algorithms.

Nanogate

The DNA parts employed in the assemblies, on an industrial scale, are already sequence-verified. Therefore, I reasoned that sequencing technologies being less accurate, but at the same time, faster and less expensive than Sanger sequencing and SBS would be more practical on a large scale. This concept is the basis of my project for synthetic DNA molecules verification called Nanogate, combining hash functions with Oxford Nanopore technology. Nanogate relies on the Nanopore feature of generating very long reads, up to 2 Mbp. Usually, constructs generated in SynBio do not exceed the 2Mbp, being on average 10 Kbp (check on literature). Therefore, a single read in Nanopore can represent the entire sequence of a construct, simplifying the sequencing analysis. Nanogate does not require any barcoding and takes advantage of k-mers counting to map reads against the assemble parts DNA sequence. I applied Nanogate on different assemblies with increasing parts similarity and compared it to Minimap2, a standard alignment algorithm. The results show that Nanogate provides a significantly higher True Positives Rate and reduced running time along with the possibility of discovering contaminants and junk reads.

3.2 Methods

Nanogate aim is to provide an efficient and effective method to verify DNA sequences after assembly, even for homology libraries. Nanogate is an alignment free-algorithm. Standard alignment algorithms, such as Minimap2 [43], align a query against a reference. If we consider our example of DNA molecules assemblies, the read obtained from Nanopore is the query, and the theoretical DNA sequence of the construct is the reference. As a free alignment algorithm, Nanogate does not need any reference sequence, using in input the single parts employed in the assembly. Nanogate turns both reads and parts sequences into k-mers, then convert them into hash codes and measures if the parts hash codes are a subset of the read hash codes. A parameter called Jaccard containment J_c evaluates to which extent the two sets overlap. To compare hash code rather than strings, reduce the asymptotic complexity making Nanogate faster than standard alignment algorithms such as Minimap2. Another significant advantage is that the hash code comparisons guarantee a significantly higher True Positive rate (TPr). On average, Nanogate TPr is 70 % higher than standard alignment algorithms.

3.2.1 Hash functions

A hash function is a function applied to map data of arbitrary size to a bit string of a fixed size [16]. An ideal hash function has the following properties. It is unidirectional, converting data to bits, but not the opposite. It avoids collisions, meaning that it avoids that different hash codes map to the same data. It has an avalanche effect; the smaller is the difference between data, the higher is the difference between hash codes. The conversion from data to hash codes is easy to compute.

Hash functions can speed up the analysis of sequencing data by reducing the time complexity O. Let s_1 and s_2 be two different DNA sequences of length l. In a character-by-character comparison, the time complexity required to establish the equality criteria is at maximum $O(l)$. The probability of having a mismatch decreases with the similarity between strings. For instance, the time complexity required to compare the strings "DNA" and "CDS" is $O(1)$. To find the difference between the two strings, we stop after comparing the first characters, D and C. In this case, the conversion to hash codes and their comparisons would increase the time complexity. Since, before comparison, we have to apply a function to convert both strings.

On the other hand, if we base our comparison on k-mers, the probability of having strings with high similarity increases since DNA is composed of only 4 different characters(A, G, C, T). Let's compare the k-mers $k_1 = ACTGCTC$ and $k_2 = ACTGCTG$, in this event in the string comparison $O(l)$. If we apply a hash function such as MinHash to both strings, we obtain the hash codes 749946486594621384 for k_1 and 16793412555443311593 for k_2. Since they differ at the first digit, the time complexity required to compare the two hash codes is 1. In this example, applying hash codes, we reduced O from l to 1. When we compare DNA strings since they are composed of only four different characters (C, T, A, G), the probability of having a very similar string compared to the standard alphabet or the decimal alphabet is higher. Moreover, due to the avalanche effect, the difference between hash codes increases with the similarity between the two original string. Therefore, the avalanche effect is crucial for Nanogate since the k-mer lengths I tested vary from 9 to 13, and I run Nanogate even on DNA strings with high similarity. Besides, Nanogate performs the comparisons for all the k-mer of a specific DNA string. Therefore the reduction of O increase even further with the number of k-mer compared.

3.2.2 K-mers in DNA sequence comparison

In the previous section, I discussed how the comparison between k-mers reduces O compared to the string comparisons. Here, I focus on how we can take advantage of k-mers to make comparisons between DNA strings. In a DNA sequence, the k-mers are all the possible substrings of a specific length. The more two DNA sequence are similar, the higher is the number of k-mers they share. To measure the number of k-mers shared by two DNA sequence, we can evaluate two parameters: the Jaccard coefficient J and the Jaccard containment coefficient J_c.

The Jaccard coefficient measures the intersection between two different sets, let h_1 and h_2 be the set of hash codes of the DNA sequences s_1 and s_2. J is defined as:

$$J(h_1, h_2) = \frac{|h_1 \cap h_2|}{|h_1 \cup h_2|} \tag{3.1}$$

If $J = 1$, h_1 and h_2 completely overlap as the sequences s_1 and s_2 they derives from. Vice versa, $J = 0$ indicates no overlap.

On the other hand, the Jaccard containment coefficient (J_c) measures the intersection between two different sets.

$$J_c(h_1, h_2) = \frac{|h_1 \cap h_2|}{|h_1|} \tag{3.2}$$

J_c measures if h_1 is a subset of h_2. $J_c = 1$ if h_1 comprehends every element of h_2, whereas $j = 0$ if h_2 and h_1 do not have any element in common. Nanogate applies the J_c to measure if the set of hash codes of a DNA part is a subset of a DNA read hash code.

3.2.3 Standard alignment algorithms

The first algorithms employed for DNA sequences comparison are the standard alignment algorithms. They input a query and align it against a reference, summarising the similarity between the two sequences in the form of a score. Matching bases increase the score, whereas mismatches and gaps give penalties since they can make every DNA sequence alignable. Let be $s_1 = ATGCT$ and $s_2 = CGAGC$; they have no base in common according to a perfect match algorithm. Nevertheless, if we add gaps, we can align s_1, and s_2 as follow:

Figure 3.1: **Alignment example**

In this regard, standard alignment algorithms add a higher penalty for the initialisation of gaps and a lower to extend them. Although standard alignment algorithms, such as Minimap2 or BWA, are still widely employed, multiple DNA sequences alignments and similarities between queries and between references lower their accuracy. This environment is frequent in synthetic biology, where often large libraries of constructs are assembled to test different promoters or CDS differing even for a single nucleotide. To overcome standard alignment limitations, researchers developed alignment-free algorithms. The latter includes two categories of algorithms: subsequences algorithms and information theory methods. Nanogate belongs to the first category. Indeed subsequences algorithms divide the sequence into substrings of a specific length, the similarity between the DNA sequences compared increase with the number of substrings shared.

3.3 Nanogate algorithm

Nanogate is the algorithm I developed for the verification of DNA sequences after assembly. I conceptualised Nanogate to support the large scale productions of synthetic constructs such as in genome foundries. Here the large size of libraries and the similarity between DNA sequences make standards approaches for the verification of DNA sequences impractical. Indeed SBS technologies are too expensive due to the high number of runs required and the consequent consumables costs, whereas Sanger sequencing is impractical when applied on long DNA constructs. On the other hand, restriction enzymes methods are not consistent since restriction sites are not always present or produce similar size bands. To overcome the above mentioned limitations, Nanogate relies on Oxford Nanopore Technologies. Nanopore generates very long reads up to 2 Mpb, covering potentially a whole construct with a single read. Compared to the sequencing technologies previously mentioned, Nanopore has lower accuracy. However, Nanogate aim is to detect if the assembly reaction performed as predicted and not the sequence of each part since they are verified already. Therefore, the lower accuracy of the Nanopore fit Nanogate purpose.

The inputs of Nanogate are the DNA sequences of the parts employed in an assembly and the reads generated from Nanopore. Nanogate contains three subprocedures Core Nanogate, Filter Nanogate and Parts sorting Nanogate. Core Nanogate, at first, applies a filter to remove short and junk reads. If the user does not provide a specific k-mer length, Nanogate detects the latter from the error rate provided. Then applies MinHash to convert both reads and parts at first into k-mers, then into hash codes. It compares each read against every DNA part in the library and evaluates the J_c for each of them. Filter Nanogate assigns DNA parts to each read based on their J_c value, assigning as first the one with the highest J_c and as last the one with lowest J_c. The last procedure Sorting Nanogate evaluates the order of parts assigned to each read.

Algorithm 4 Nanogate

```
1: procedure NANOGATE(R, P, E, C, J_t, C)
2:     O ← rawnanogate(R, P, E, C, J_t, C)
3:     NO ← filternanogate(O, J_t)
4:     SO ← sortingnanogate(R, P, NO)
5: end procedure
```

3.3.1 Core Nanogate

The first procedure is Core Nanogate. At first, it derives the k-mer length K from the error rate E. E is a known parameter obtained from the Nanopore sequencing and defined in Nanogate as the percentage of non-unique k-mers among all the parts employed in the assembly. The procedure returns K after testing all the integer values between a minimum and a maximum specified by the user until E reaches the target value.

The probability of k-mers to be unique among parts increase with their length. Moreover, long k-mers reduce the amount of data to compute. Nevertheless, the sequencing process introduces errors in the read. Therefore, long k-mers match the reference sequence but not the read sequence. Core Nanogate evaluates the optimal K length balancing the uniqueness between parts and the read errors.

In the next step, Nanogate applies a filter to remove junk and short reads from the analysis. The filter creates a distribution of construct lengths d_c to exclude from the analysis each read not compatible with the distribution obtained. At each cycle of the filter, the procedure samples n parts from the library of DNA parts Pl, where n is the number of DNA parts p in each construct. The procedure adds a new construct length to d_c at each cycle until:

$$\bar{d_c} \text{ cycle } x - \bar{d_c} \text{ cycle } x + 1 < 0.01 \tag{3.3}$$

Or until d_c includes more than 20000 samples. The filter excludes from the analysis every read with a length $< 5^th$ percentile of the last d_c obtained.

The filter excludes short reads but not long reads since molecular assembly methods may generate longer constructs than expected by joining multiple parts together.

The procedure then applies MinHash to covert both reads and reference parts first into k-mers then into hash codes. In the last step, Core Nanogate for each filtered read rf_n evaluates the Jaccard containment coefficient J_c between rf_n and each part in Pl.

$$J_c = \frac{rf_n \cap p_n}{p_n} \tag{3.4}$$

Here Nanogate treats reads and parts as different sets of hash codes and applies the Jc to evaluate to which extent a p_x is a subset of rf_x. The higher is the number of shared hash codes the higher the similarity between the read and part DNA sequences. At the end of the step, each read will include a list of parts ranked based on their J_c.

3.3.2 Filter Nanogate

Filter Nanogate procedure use as input the list of reads returned from Core Nanogate. The user can specify an additional parameter, the Jaccard threshold J_t. Filter Nanogate use the J_t to remove from the list every p respecting the following condition:

$$J_c < J_t \tag{3.5}$$

The list is then reinitialised, and p is added to the list from the one with the highest J_c, until the following condition is true:

$$\sum_{i=1}^{n'} |p_i| >= |r_l| \tag{3.6}$$

Algorithm 5 Core Nanogate

```
1: procedure CORE NANOGATE(R, P, E, C, K)
2:     if E then
3:         K ← evaluateKmerlength(E, V)
4:     else
5:         K ← (K)
6:     end if
7:     PH ← initialize()
8:     for P_i in P do
9:         PH_i ← gethashes(Pi, K)
10:        PH ← add(PH_i)
11:    end for
12:    Rf_i ← lengthdistributionfilter P, R, C
13:    O ← initialize()
14:    for Rf_i in Rf do
15:        RH_i ← gethashes(Rf_i, K)
16:        for P_i in P do
17:            J ← Jaccardcontainment PH, RH_i
18:            O ← add J
19:        end for
20:    end for
21:    return O
22: end procedure
```

where |.| is the length in terms of bp of the sequence.

The procedure aims to assign parts to each read, starting from the highest J_c. The higher is the J_c, the higher is the similarity between the part and the read. The procedure continues until the p assigned cover the entire sequence of the read.

3.3.3 Sorting Nanogate

At the end of the Filter Nanogate procedure, Nanogate returns for each read which are the most similar parts. Nevertheless, we do not know in which order they align against the read. The purpose of Sorting Nanogate is to align the parts assigned against the read. Sorting Nanogate applies Minimap2 for the alignment, which returns the start and ending point of each p aligned. By sorting the starting points, Sorting Nanogate can detect the order of each p for the forward strand. The procedure applies reverse sorting to determine the order on the reverse strand. The procedure evaluates the strand selecting the one with the highest number of parts assigned.

Sorting Nanogate applies Minimap2 to sort parts since the previous procedures Core and Filter Nanogate reduce to 1 the number of queries. Indeed, Minimap2 aligns p parts against each read. In this way, Nanogate reduces the computational complexity for Minimap2, significantly increasing the accuracy, as I analyse in the result section.

3.4 Results

testbeds, avoid bias deriving from DNA synthesis, assembly or sequencing, and speed up the overall analysis.

The framework works as a simulator of assemblies. It samples np CDS from an annotated DNA sequence used as a source. For testing, I employed the DNA sequence of Saccharomyces cerevisiae XIV chromosome [29]. For the assembly of each construct, let p be the number of parts, the framework divides the np CDS into p groups, with each group representing a different position in the construct. In the case of spare parts, the framework assigns them randomly to a group.

In synthetic biology, frequent experiments involve the testing of parts with high similarity. For example, a researcher may test promoters differing for a single base to verify which one

Algorithm 6 Filter Nanogate

```
1: procedure FILTER NANOGATE(O, J_t)
2:     NO ← initialize()
3:     for R_f in O do
4:         R_p ← initialize()
5:         R_l ← length(R_f)
6:         P_l ← 0
7:         while P_l < R_l do
8:             P_j ← getpartsjaccard(O, R_f)
9:             P_il ← getpartlength(PJ)
10:            if J in P_j > j_t then
11:                R_p ← add(P_j)
12:            end if
13:            P_l ← P_l + P_il
14:        end while
15:        NO ← add(RP)
16:    end for
17:    return NO
18: end procedure
```

Algorithm 7 Sorting Nanogate

```
1: procedure SORTING NANOGATE(R, P, NO)
2:     SO ← initialize()
3:     for O in NO do
4:         R_a ← buildread (R)
5:         A ← Minimap2alignment(R_a, P)
6:         S ← evaluatestrand()
7:         if S = + then
8:             A ← forwardsorting()
9:         else
10:            A ← reversesorting()
11:        end if
12:        SO ← add(A)
13:    end for
14:    return SO
15: end procedure
```

returns the highest expression. I adapted the framework to reproduce this condition and test the algorithm limits. The framework includes the similarity between parts and constructs as parameters for the generation of testbeds. In the framework, I define the similarity between parts as the percentage of *bps*, shared between two parts marked as clones *c*. I introduced it by cloning a part *pt* into a clone pt', then replacing $1 - bps\%$ of pt' with a random sequence of the same size. The framework locate the edit in a random position of pt', which satisfy the criteria:

$$S_p + E_l \leq S_e \tag{3.7}$$

Where S_p is the insert position, E_l is the length of the insert, and S_e is the length of the whole part.

On the other hand, the constructs similarity is the percentage of parts marked as a clone in each construct *cp*. To select the parts to clone, the framework has three different modes *begin*, *end* and *random*. In *begin*, the framework clones parts in the first *cp* positions; the lasts in *end*, and they are random in the *random* mode. By assembling at random *p* parts from the parts pool *np*, based on the assigned positions, the framework generates a library of *l* constructs. To generate reads, I run Badreads Tool [71] on the library previously generated and set the parameters in Tab. 3.1 to simulate high-quality(HQ) reads and the parameters in Tab. 3.2 to simulate low-quality reads(LQ).

Paramaters	**Values**
Quantity	5x, 10x, 50x, 100x
Error model	nanopore
Qscore model	nanopore
Glitches	0, 0, 0
Junk reads	0
random reads	0
chimeras	0
identity	95, 100, 4
start adapter sequence	None
end adapter sequence	None

Table 3.1: Parameters set in Bad reads tool for the generation of high quality reads

Paramaters	**Values**
Quantity	5x, 10x, 50x, 100x
Error model	nanopore
Qscore model	nanopore
Glitches	1000, 100, 100
Junk reads	5
random reads	5
chimeras	10
identity	75, 90, 8
start adapter sequence	None
end adapter sequence	None

Table 3.2: Parameters set in Bad reads tool for the generation of low quality reads

Tab. 3.3, describes all the parameters set for the generation of the benchmark.

Paramaters	**Values**
Parts identity(bps)	0, 0.7, 0.8, 0.9
Run(R)	10
Construct library size(l)	10, 50, 100
Parts(p)	5, 10, 20
Coverage(cov)	$5x$, $10x$, $50x$, $100x$
Error rate(E)	0.01, 0.1, 0.3

Table 3.3: Framework Parameters applied for the generation of testbeds

bps is the part similarity parameter aforementioned, r is the number of independents benchmarks we generate to assess Nanogate efficiency. l is the number of constructs in the library, p is the number of parts per construct, *cov* is the coverage set in Badreads Tool, and E is the error rate, the percentage of non-unique k-mers set as a target.

To simplify, I will divide the analysis of Nanogate performances into two different groups: Qualitative and Sorting analysis. The Qualitative analysis evaluates the Core and Filter procedure, estimating the number of Parts that Nanogate assigns to the correct read. Instead, the Sorting analysis analyses the Sorting algorithm, evaluating the number of Parts sorted correctly within the read after the calling. For both Qualitative and Sorting analyses, I run Nanogate over the same testbeds divided into four groups, each defined by different percentages of part similarity: $0\%, 70\%, 80\%, 90\%$. For each group, I employed the framework to generate 10 independent run and applied Nanogate on each of them. For each group, I measured: the precision, the recall, the quality and quantity of the reads, the error and the true positive rate.

3.4.1 Read analysis

One of the main challenges in evaluating Nanogate performances was to discern between Nanogate and sequencing errors. Sequencing errors consists of differences between the read sequence and the original construct. In Nanopore, they are caused mainly by repeats, homopolymers and palindromes. When a single DNA strand containing one of these features pass through the pore, the instrument has a higher probability of calling the wrong nucleotide (check here). The errors might include even reads stopped prematurely, including a number of parts lower than in the original sequence(short reads). In other cases, the read is not long enough to be associated with any reference and is called a junk read.

If Nanogate analyses a short read, it could correctly return the set of parts *sp* matching the sequence, but not the *sp* of the original construct. Therefore, in this event, I would assign to Nanogate an error introduced by the sequencing. To solve the dilemma, I implemented a filter in Core Nanogate to reduce the number of short reads analysed 3.3.1. Nevertheless, in the preliminary analysis, I could see that the number of short reads analysed significantly affected the results. Therefore, to focus only on the Nanogate errors, I decided to flag as "pass" every read to whom Nanogate assigned a number of parts matching those in the original construct. With this edit at the framework, I evaluated Nanogate errors correctly, removing errors due to the sequencing.

The results show that the percentage of reads analysed, passing the Nanogate filter, and flagged as "pass" differs considerably between HQ and LQ settings as expected(Fig 3.2. The reads analysed, for the HQ setting, is on average 70% and decrease slowly with the length of the reads 3.2. For the LQ setting instead, we have an average of 40% of analysed reads for $p = 5$, and $= 10, 20\%$ for $p = 10, 20$ 3.2. In both cases, interestingly, the coverage does not affect the results. Fig. 3.3 shows the amount of reads analysed in relation to the coverage. Although, the percentage does not change acrros the different coverages, the amount of reads analysed increase significantly with the coverage. Therefore, the two figures 3.3 and 3.2 proves that Badreads uniforms the percentage of reads not covering the entire construct across the different coverage. Altogether, these results suggest that to have optimal results with at least 10 reads per construct, Nanogate requires coverage of $50x$ for HQ reads, and $100x$ for the LQ.

3.4.2 Qualitative analysis

The qualitative analysis aims at assessing the Core and Filter procedures. Indeed, here I evaluate their precision, recall and error. I structured the confusion matrix for the qualitative analysis as follow: Given a read r_1 of a construct c_1, a TP is a read to which Nanogate assigns correctly the set of parts ps_1. I define an FP when Nanogate calls a ps_1 matching the composition of a construct c_n in the library, but not the original construct c_1. In a FN Nanogate defines a ps_1 matching neither the composition of c_1, neither those of c_n.

After defining the confusion matrix, I began to measure the recall of the Core and Filter procedures. Here, indeed I evaluate the percentage of reads having a *ps* called correctly. The

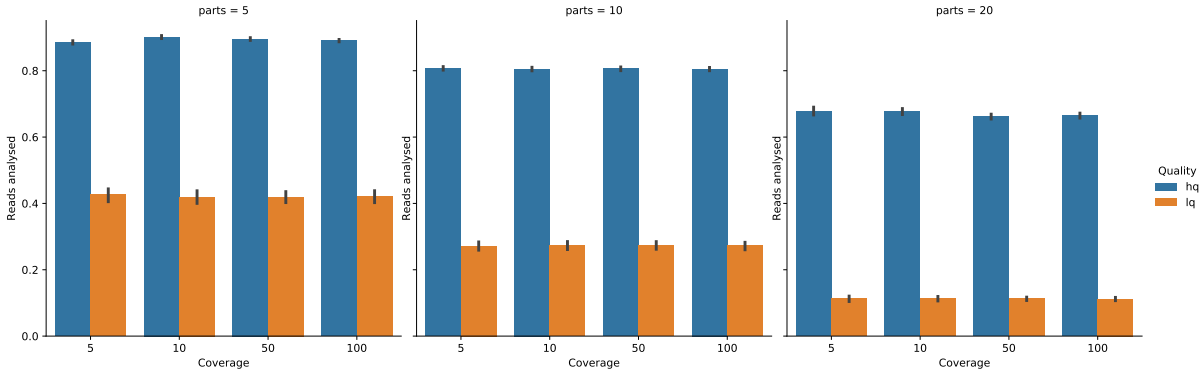

Figure 3.2: **Reads quality** reports the percentage of reads analysed respectively for the HQ and LQ settings both with $s = 0\%$ in relation to the coverage.The percentage counts the number of reads flagged as pass on the number of reads passing only the Nanogate filter.

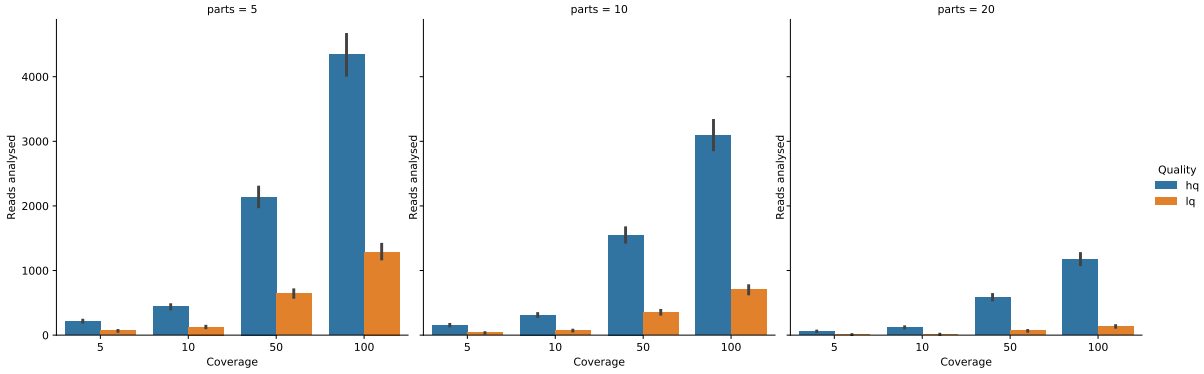

Figure 3.3: **Reads quantity** report the number of reads analysed respectively for the HQ and LQ settings both with $s = 0\%$ in relation to the coverage.The amount takes into account the total number of reads flagged as pass.

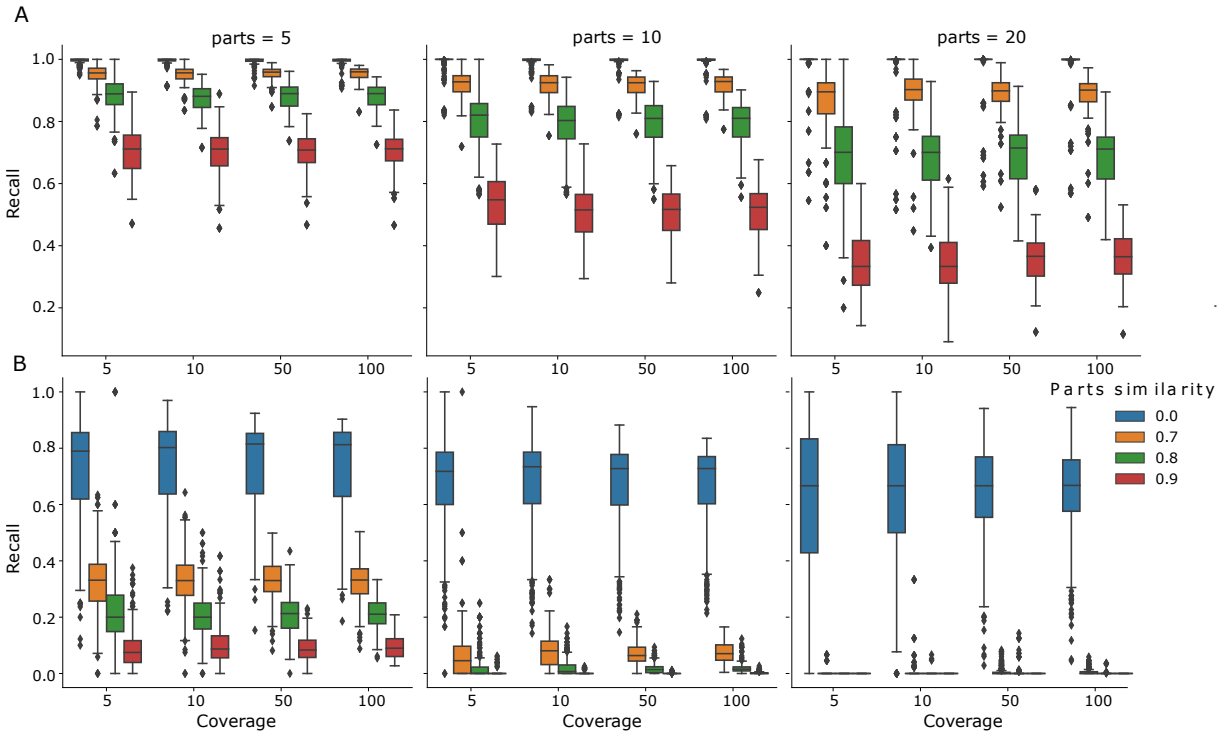

Figure 3.4: **Recall** We report how R, defined as $\frac{TP}{TP+FN}$ varies with the coverage, the number of parts for construct and parts similarity, respectively for the HQ setting A) and LQ setting B)

recall (R) measured as the ratio:

$$R = \frac{TP}{TP + FN} \tag{3.8}$$

It decreases with the similarity between parts and the quality of the reads. In Fig.3.4 , R has a maximum for $s = 0\%$, and a minimum for $s = 90\%$ the variance decrease with the coverage applied during the sequencing. For LQ, R is significantly lower than in HQ 3.4 If we draw a line, considering reliable results with an R higher than 70%, we can assume that Nanogate produces reliable results in the following conditions: HQ, s= $0\%, 70\%, 80\%$ and LQ,$s = 0\%$ for $HQs = 90\%$ results are reliable only for $p = 5$. Interestingly, for HQ $s = 0\%70\%$, R is higher than 90%. These results taken all together show that for low or absent parts similarity, Nanogate detects the correct composition of a read with an error rate lower than 10%, the error decrease with the quality of the reads, and increase with the similarity between parts and the length of the analysed read. Notably, for the LQ settings, the number of reads analysed is sensibly lower than in HQ, particularly for lower coverages $5x$ and $10x$ as described above 3.4.1.

the precision (P) is the second measure I took into consideration, defined as:

$$P = \frac{TP}{TP + FP} \tag{3.9}$$

Here, I measure the probability of calling the ps of a construct, different from the one analysed still present in a library of size l.

As expected, the precision has higher values than the recall. Indeed, FPs are less frequent than FNs. In other terms, the probability of having ps_x not matching the construct x ps, but a ps_y of the construct y is lower than having a ps_n not represented in any construct of the library. As in Fig 3.5, the precision is on average 100%, with outliers increasing the similarity between parts. As a consequence, false positives are rare in qualitative analysis.

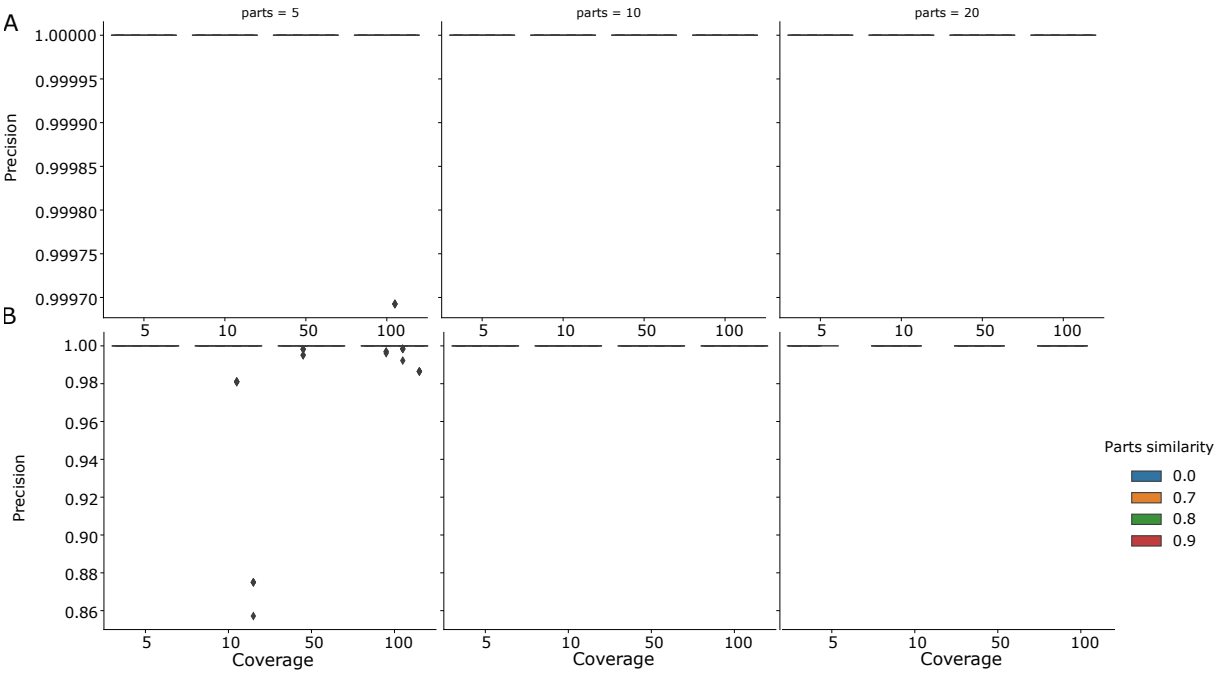

Figure 3.5: **Precision** We report how P defined as $\frac{TP}{TP+FP}$ varies with the coverage, the number of parts for construct and parts similarity, respectively for the HQ setting A) and LQ setting B)

3.4.3 Error characterisation

Here I characterise Nanogate error by measuring e. The algorithm flags as pass every read having the same number of parts of the original construct that have been sequenced. e measures the percentage of parts incorrectly called on every read marked as pass, In the best-case scenario HQ and $s = 0\%$, for each read not matching the original ps, we have less the one parts called incorrectly, Fig 3.6. e decreases with p and slightly increases with the size of the library. P and R for HQ,$s = 0\%$ are higher than 90%, as a consequence e is lower than 2%. As a consequence, outliers errors are present only in large libraries. The J_t as expected affects e as well, and e decreases as J_t increase, indeed a higher J_t reduce the probability of a bad calling in the absence of parts similarity. As expected as s increases e increases as well, Fig. 3.6 nevertheless, the number of bad calls is slightly higher than 1 even in the HQ worst-case scenario $s = 90\%$. Interestingly for $s > 0\%$ the size of the library and the J_t do not affect e. As s increase the number of reads not matching the original construc increase as well, and the effect of the library size is no more evident. The J_t does not affect e for $s > 0\%$, since for $s = 70\%, 80\%$ and 90% the J_c between the similar parts and read is in general higher than the J_t. For the LQ setting, e shows a similar behaviour, although increases overall. Indeed, for $s = 0$ there is on average 1 bad call per construct, and J_t lowers e drastically. e increases with s and for $s = 90\%$ the percentage of bad calls per constructs is on average 50%. As a consequence on average, there are 2, 5 and 10 bad calls respectively for $p = 5, 10$ and 20.

3.4.4 Sorting Analysis

I applied the same principles employed for the qualitative analysis to the sorting analysis. Indeed, to evaluate the performances of the sorting algorithm, I defined a similar confusion matrix similar. Given a read r_1 of a construct c_1, a TP is a read where Nanogate sorted correctly the set of parts ps_1.

In an FP Nanogate sorts a ps_1 matching the parts order of a construct c_n in the library, but not the original construct c_1. In a FN Nanogate defines a ps_1 matching neither the parts order of c_1, neither those of c_n.

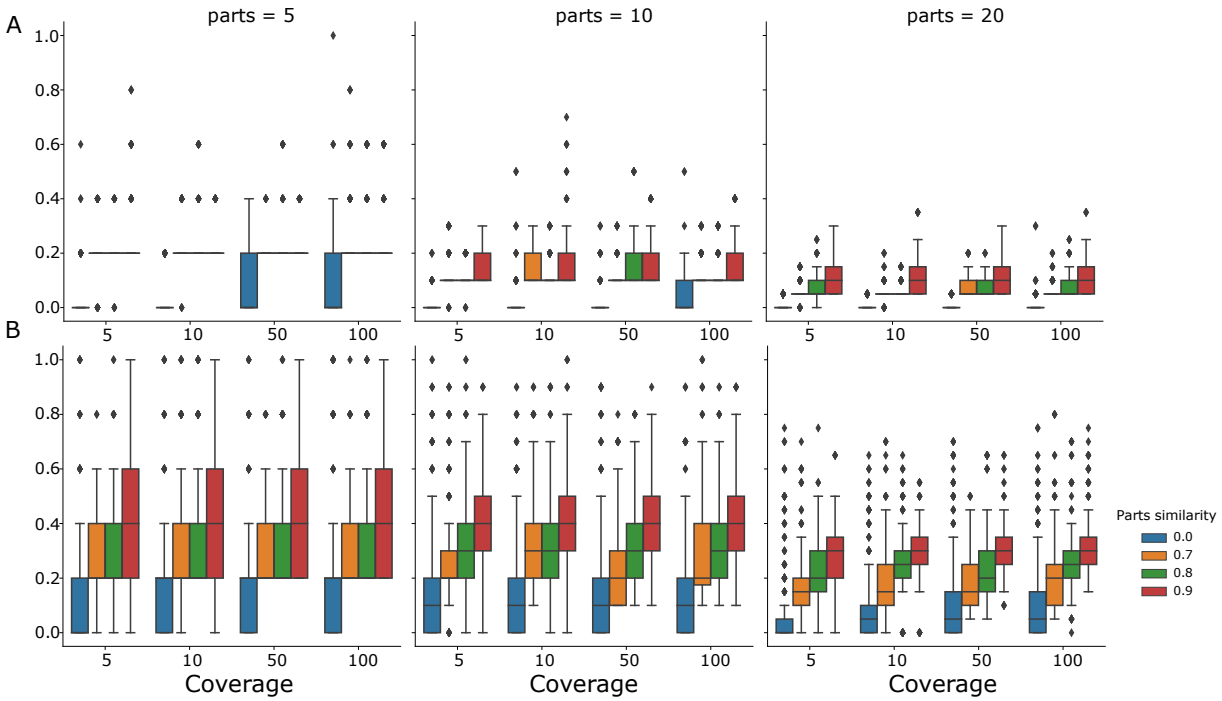

Figure 3.6: **Error rate** We report how e, defined as the percentage of parts incorrectly called on every read marked as pass, which does not match the original ps varies with the coverage, the number of parts for construct and parts similarity, respectively for the HQ setting A) and LQ setting B)

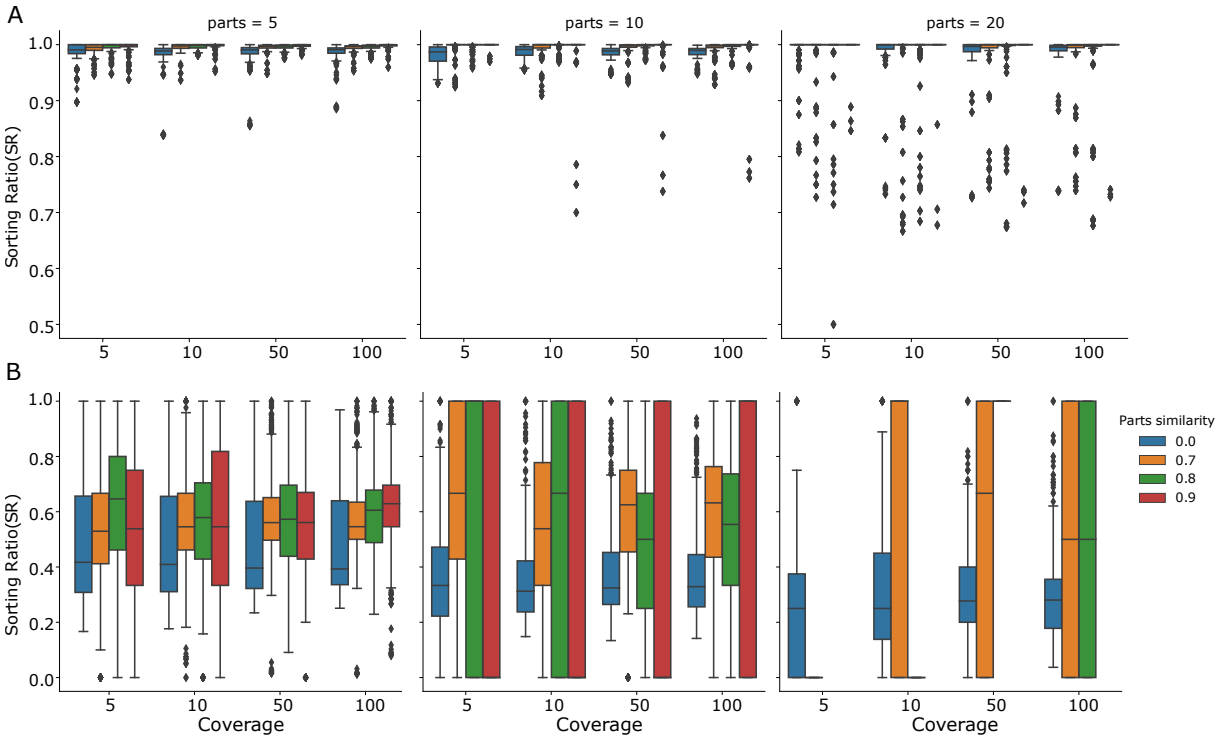

Figure 3.7: **Sorting ratio** We report how SR, defined as the ratio between the true positives found by the qualitative analysis and the true positives found by the sorting algorithm. Here we measures how SR varies with the coverage, the number of parts for construct and parts similarity, respectively for the HQ setting A) and LQ setting B)

Then I measured the precision and recall of the sorting algorithm. In addition to the recall and precision, I introduced a new measure: the Sorting ratio (TPS):

$$SR = \frac{TP_q}{TP_s} \tag{3.10}$$

TP_q and TP_s are both True Positives, but the first refers to the qualitative analysis, the second to the sorting. SR is the ration between the reads which part composition has been detected correctly, and number reads which parts order has been detected correctly. In other terms, it measures how many reads which part composition has been detected correctly, have been even sorted correctly.

The reads quality is the main factor affecting the SR, as shown in Fig. 3.7. Indeed for the HQ setting, SR is on average higher than 90%, with outliers increasing with the similarity between parts. The SR results are consistent with the sorting R. Indeed, for the HQ setting, the qualitative and sorting R show almost no difference. Therefore, for HQ, if Nanogate detects the correct ps, there is a probability = 90% to sort parts without errors. The results are different for LQ. Here the RS has a large variance, in particular for $p >= 10$. This result is a consequence of the low number of reads analysed for the LQ setting. Even in the worst-case scenario, with $s = 90\%$ and LQ, the RS is on average $40 - 50\%$ at least for $p = 5$.

To summarise for high-quality reads, if Nanogate detects the correct ps, the sorting has algorithm a probability higher than 90% to sort the parts correctly, regardless of their similarity.

3.4.5 Contaminants detection

Nanogate can even detect contaminants introduced during the sequencing. Core Nanogate marks a read as a contaminant if none of the library parts can be assigned. Moreover, the user can use genomes databases to search for the sequence in the marked reads to define the nature of the contaminant. As expected, the number of reads with no parts assigned increase with the

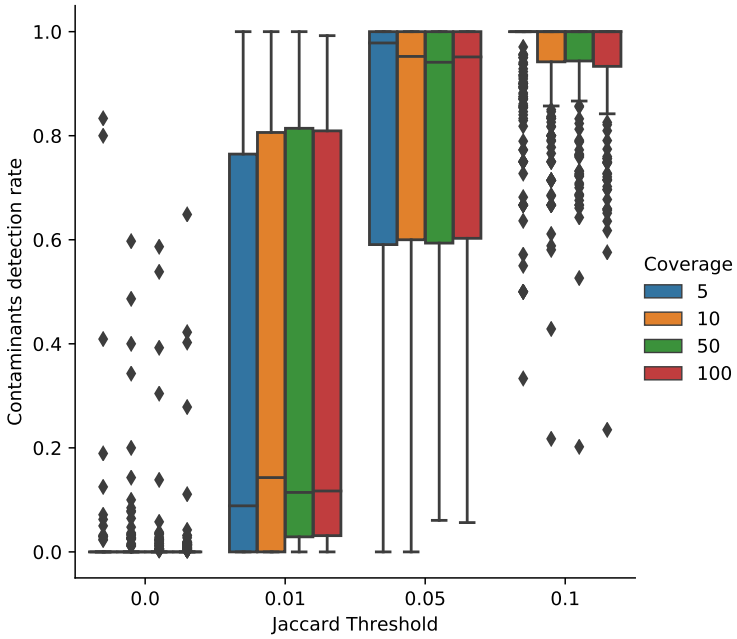

Figure 3.8: **Contaminants discovery rate** We report how the Contaminants discovery rate varies with the Jaccard Threshold (Jt) and Coverage

J_t. Nevertheless, a J_t too high would have, as a consequence, a loss of TP. Indeed, Nanogate would cut off even parts belonging to the correct read if J_t is too stringent. Therefore, we tested low J_t values ranging from 0.05 to 0.1 to minimise the TP loss and maximise the detection. For $J_t = 0.05, 0.1$ the rate of contaminants detected is between $80 - 90\%$ and the loss of TP is negligible, see Fig. 3.8. The detection of contaminants further proves the versatility of our method.

3.4.6 Computational complexity

We analysed the time complexity of our algorithm, and for consistency performed all our experiments on a system with 2 Intel Xeon Gold 6130 CPUs (16 cores, 2.10Ghz), 128Gb DDR4 RAM running Scientific Linux 7. We recorded the user time and averaged on 10 independent run. Fig.3.9 shows that the running time ranges from 5 to 400 seconds. The time complexity increases with p, l and cov. In the worst-case scenario with $l = 100$ and $c = 100x$ and $p = 20$, the whole analysis took 400 seconds.

3.4.7 Standard alignment algorithm

At present, standard alignment algorithms, such as Minimap2, are the most employed to verify DNA sequences. For this reason, I decided to compare Nanogate and Minimap2 performances after applying them on the same testbeds. Nevertheless, the two algorithms take different inputs. Whereas Nanogate takes as input the library of parts and the reads sequences, Minimap2 requires inputting queries and references. In this regards, I decided to divide the comparisons into two different classes. In the first class $Minimap2$, the references for Minimap2 are all the possible constructs generated by the combinatorial assembly of the parts. In the second class of comparisons MMA, I decided to perform an unfair comparison giving a significant advantage to Minimap2. Indeed, I set as references in the alignment algorithm only the constructs generated from the combinatorial assembly. Nevertheless, in combinatorial assemblies, the combinations of parts are not predetermined. Due to the computational amount of time and memory required for the generation of constructs assembled and their alignment with Minimap2, I performed comparisons exclusively for $p = 5$. In Fig. 3.10 I compare, with parts similarity = 70%, the tpr of Minimap2 and Nanogate. Nanogate attains the highest tpr in every scenario, although the

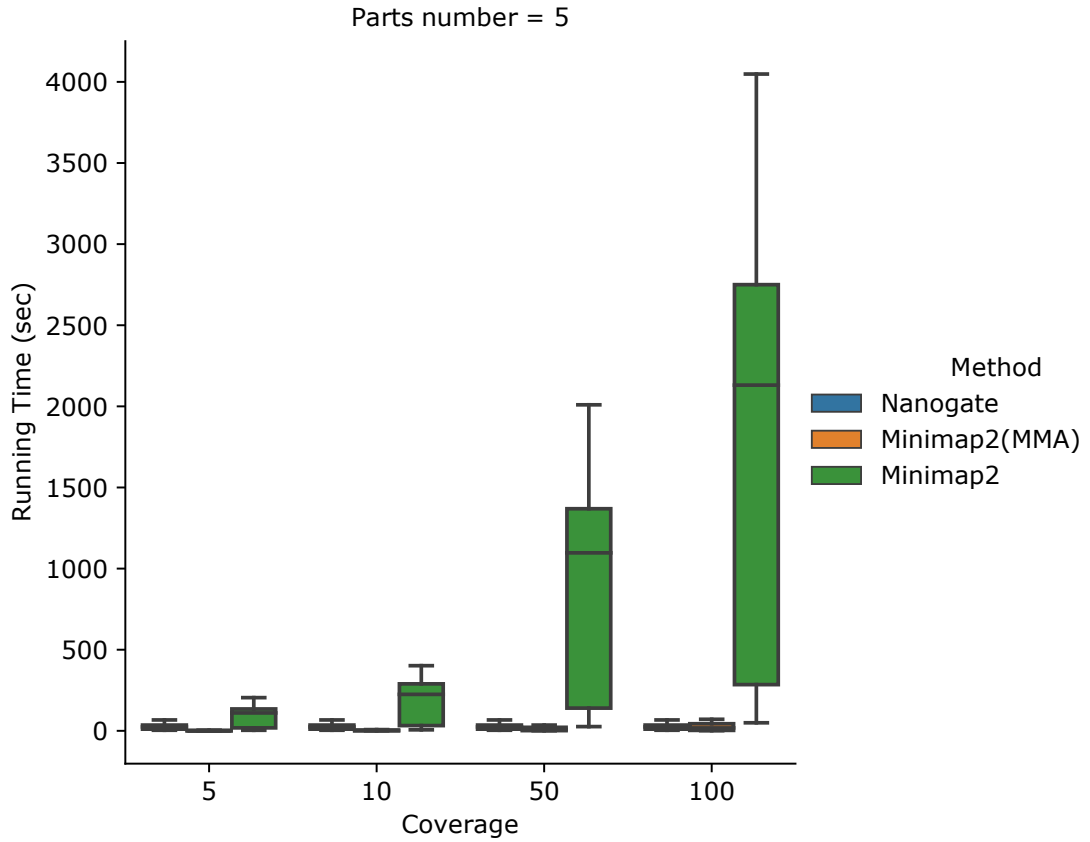

Figure 3.9: **Running time** We compare Nanogate and Minimap2 running times, in Minimap2(MMA).

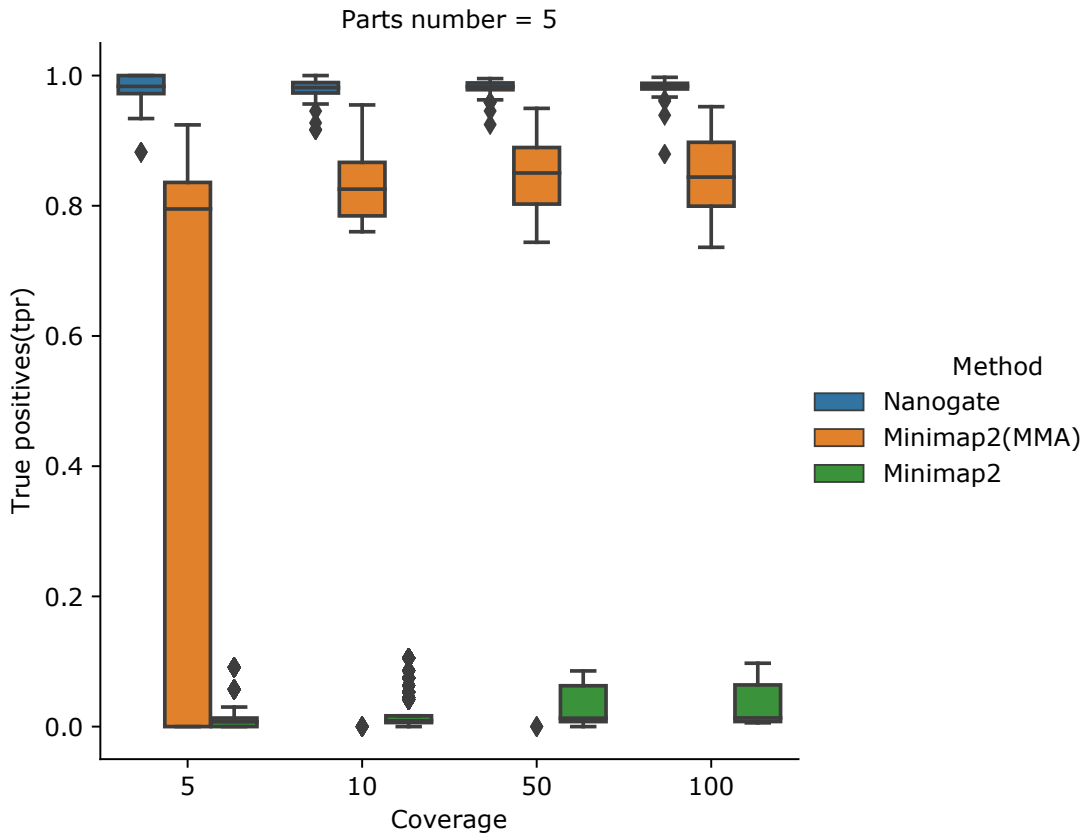

Figure 3.10: **Nanogate comparison similarity 70%** We compare Nanogate and Minimap2 True Positive rate(tpr) with parts similarity= 70%, we analyse how the tpr varies with the coverage. n Minimap2(MMA), I set as references only the constructs that have been generated during the experiment. In Minimap2, I set as references, the constructs generated from all the possible combinations of parts that are possible to obtain. Due to hardware limit, Mimimap2 could be tested only for $p = 5$

advantage decrease with the increase of c. For the $Minimap2$ settings, Minimap2 reaches a tpr below 0.2, The low tpr of Minimap2 is a consequence of the number of references, $7000 - 8000$ for the combinatorial assembly. Fig. 3.11 shows the tpr for $s = 90\%$. Minimap2 tpr, in standard conditions, is still below 0.2%. Nevertheless, for $cov > 5x$ Mma attains a tpr slightly higher. These results suggests that Nanogate considerably overcomes Minimap2 in terms of tpr in standard condition. Moreover, despite giving a significant advantage to Minimap2 with the MMA settings, Nanogate still shows an higher tpr for $s = 70\%$. Interestingly, with the increasing of s, Minimap2(MMA) shows and higher tpr, compared to Nanogate.

3.5 Discussion

The fast pace growth of Synthetic Biology caused an increased demand for DNA parts and their assemblies. The production of synthetic constructs on a large scale requires new support methods for their verification. Approaches relying on restriction enzymes are too error-prone, whereas sequencing methods are unfeasible due to costs and hands-on time. Here we propose a new alignment-free method for the verification of DNA sequences after assembly we called Nanogate. It harnesses Nanopore long reads and uses the MinHash algorithm to reduce the time complexity O. Nanogate recall and precision are nearly 100% on high quality reads. Compared to standards alignment algorithms, we provide a significantly higher True Positives Rate and reduced running time, along with the possibility of discovering contaminants and junk reads.

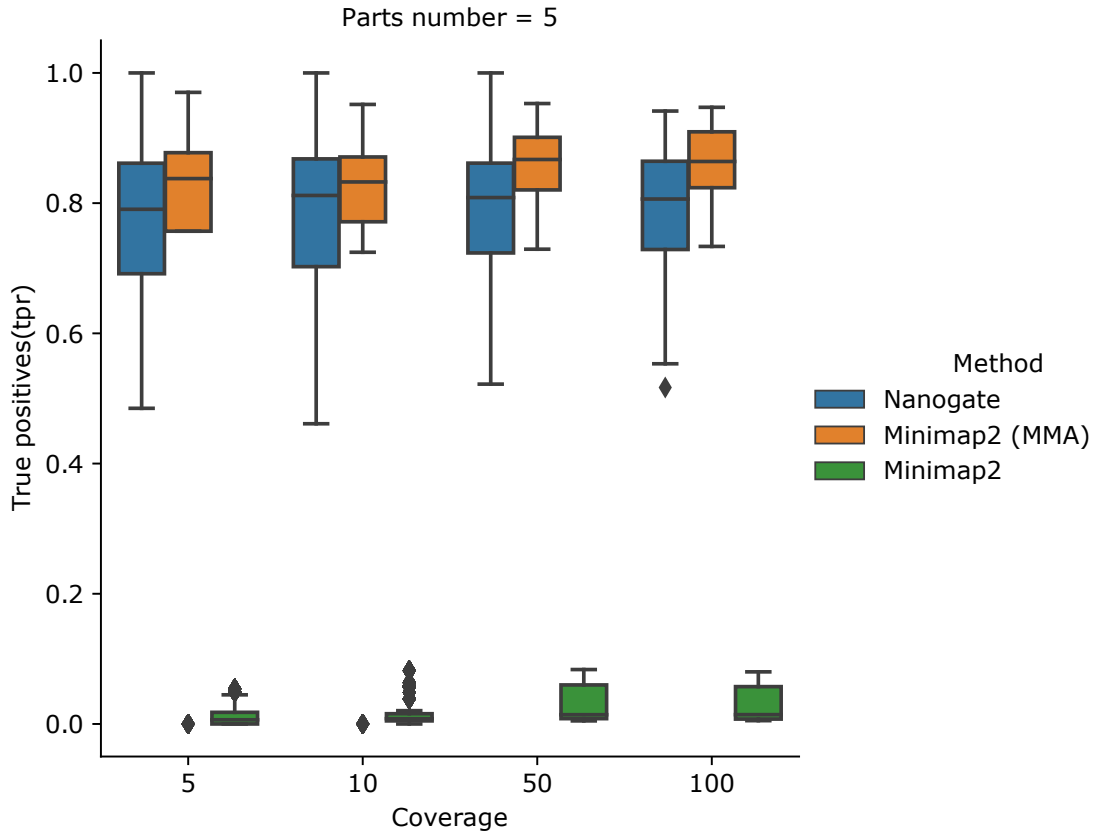

Figure 3.11: **Nanogate comparison similarity 90%** We compare Nanogate and Minimap2 True Positive rate with parts similarity= 90% we analyse how the tpr varies with the coverage. In Minimap2(MMA), the algorithm analyse only the constructs in the library, and not each possible combination of parts

Nevertheless, Nanogate recall and precision strictly depend on the Nanopore reads quality. Moreover, tests of the algorithm on Nanopore reads rather than simulated could give hints on how to improve and adjust the algorithm.

Chapter 4

Conclusions

The Design-Build Test and Learn(DBTL) cycle has been pivotal for the spreading and development of SynBio. DBTL supports the reproducibility and consistency of SynBio experiments. Nevertheless, the engineering of biological construct is still a complex task. The biological nature of cells, increase the complexity of the systems, making more difficult to create ad hoc CAD for SynBio. The lack of orthogonality and the complex chemical interactions within the cell, are examples of what makes challenging to apply the engineering principles to biological systems. In this work here presented, I implemented two different algorithms, MOODA and Nanogate, both have the aim of assisting researchers, during the DBTL cycle. MOODA focuses on the Design and Build steps, whereas Nanogate focuses and the Test step. MOODA is the first multi objective optimisation algorithm implemented for the design and assembly of synthetic constructs. MOODA takes into account the constraints of the Design and Build steps. Indeed, often the designed construct can not be assembled due to the limits imposed by the DNA synthesis and assembly method. Therefore, the construct design is edited to meet the synthesis and assembly constraints, often leading to designs too different from the original one. MOODA simplify this task, returning as output a list of optimised feasible solutions and underlying pros and cons of each of them. Nevertheless, despite the implementation of the crowding, many of the solutions returned by the algorithm are still too similar. Nevertheless, MOODA is just a backbone, researchers can add and remove operators on the basis of their needs. Furthermore, future studies may improve the diversity of the returned solutions by improving the crowding distance system.

Nanogate, on the other hand, is a valid instrument for the verification of DNA sequences after assemblies, in particular on large scales.

The verification of DNA sequences is at present performed mainly via restrictions enzyme methods or SBS sequencing technologies. Restriction enzyme methods are affordable and simple to apply. Nevertheless, assembled constructs do not always have an available restriction site to cut, or it can happen that the fragments generated from the digestion have a similar size, making difficult the visualisation of the two different bands on an electrophoretic gel. On the other hand, to use SBS technology to verify the sequence of the assembled synthetic constructs on a large scale, might be both expensive and time-consuming. Furthemore, the sequence of single parts employed in the assemblies are usually already verified by the providers. Nanogate provides an affordable and practical method for sequence verification of synthetic constructs in bulk. Taking advantage of Nanopore long reds, Nanogate is base on the assumption that each synthetic construct can be contained in a single Nanopore read. Nogate uses a k-mer based approach to provide info on the parts contained in each read, their order, presence of contaminants and short reads. Nevertheless, Nanogate relies on Nanopore Oxford Technologies, and the reads analysed must be returned from a Nanopore sequencer. Future studies may use real Nanopore reads as input to test Nanopore precision and recall, since in this study reads have been generated in silico, using Badreads. Then, Nanogate could be added to a DBTL workflow and employed in a large scale production to verify constructs assembled via Golden Gate or Gibson assembly. Moreover, it would be interesting to test Nanogate on different long-read sequencers, such as PacBio.

Bibliography

[1] Neta Agmon et al. "Yeast Golden Gate (yGG) for the Efficient Assembly of S. cerevisiae Transcription Units". In: *ACS Synthetic Biology* 4.7 (2015), pp. 853–859. ISSN: 21615063. DOI: 10.1021/sb500372z.

[2] Parayil Kumaran Ajikumar et al. "Isoprenoid pathway optimization for Taxol precursor overproduction in Escherichia coli". In: *Science* (2010). ISSN: 10959203. DOI: 10.1126/science.1191652.

[3] J. Christopher Anderson, Christopher A. Voigt, and Adam P. Arkin. "Environmental signal integration by a modular and gate". In: *Molecular Systems Biology* 3 (2007). ISSN: 17444292. DOI: 10.1038/msb4100173.

[4] Rozalyn M. Anderson et al. "Nicatinamide and PNC1 govern lifespan extension by calorie restriction in Saccharomyces cerevisiae". In: *Nature* (2003). ISSN: 00280836. DOI: 10.1038/nature01578.

[5] Evan Appleton et al. "Design Automation in Synthetic Biology". In: (2018).

[6] Evan Appleton et al. "Interactive assembly algorithms for molecular cloning". In: *Nature Methods* 11.6 (2014), pp. 657–662. ISSN: 1548-7091. DOI: 10.1038/nmeth.2939. URL: http://www.nature.com/doifinder/10.1038/nmeth.2939.

[7] Adam P. Arkin and Daniel A. Fletcher. "Fast, cheap and somewhat in control". In: *Genome Biology* 7.8 (Aug. 2006). ISSN: 14747596. DOI: 10.1186/gb-2006-7-8-114.

[8] Charles Audet, John E Dennis Jr, and Sébastien Le Digabel. "Trade-off studies in black-box optimization". In: *Optimization Methods and Software* 27.4-5 (2012), pp. 613–624.

[9] Mariéa Pilar Bayona-Bafaluy et al. "Revisiting the mouse mitochondrial DNA sequence". In: *Nucleic Acids Research* 31.18 (2003), pp. 5349–5355.

[10] Steven A. Benner and A. Michael Sismour. *Synthetic biology.* 2005. DOI: 10.1038/nrg1637.

[11] David R. Bentley et al. "Accurate whole human genome sequencing using reversible terminator chemistry". In: *Nature* 456 (7218 Nov. 2008), pp. 53–59. ISSN: 00280836. DOI: 10.1038/nature07517.

[12] Maureen J Bibb et al. "Sequence and gene organization of mouse mitochondrial DNA". In: *Cell* 26.2 (1981), pp. 167–180.

[13] Jurate Bitinaite et al. "USER™ friendly DNA engineering and cloning method by uracil excision". In: *Nucleic Acids Research* 35.6 (Mar. 2007), pp. 1992–2002. ISSN: 03051048. DOI: 10.1093/nar/gkm041.

[14] Bruno Canarda and Robert S Sarfati. *DNA polymerase fluorescent substrates with reversible 3'-tags.* 1994, p. 6.

[15] Stefano Cardinale, Marcin Pawel Joachimiak, and Adam Paul Arkin. "Effects of genetic variation on the e. coli host-circuit interface". In: *Cell Reports* 4.2 (July 2013), pp. 231–237. ISSN: 22111247. DOI: 10.1016/j.celrep.2013.06.023.

[16] J Lawrence Carter and Mark N Wegman. *Universal Classes of Hash Functions.* 1979, pp. 143–154.

[17] Matthias Christen, Samuel Deutsch, and Beat Christen. "Genome Calligrapher: A Web Tool for Refactoring Bacterial Genome Sequences for de Novo DNA Synthesis". In: *ACS Synthetic Biology* 4.8 (2015), pp. 927–934. ISSN: 21615063. DOI: 10.1021/acssynbio.5b00087.

[18] NCBI Resource Coordinators. "Database resources of the National Center for Biotechnology Information". In: *Nucleic Acids Res* 44 (2016), pp. 853–859. DOI: 0.1093/nar/gkv1290.

[19] Indraneel Das and John E Dennis. "Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems". In: *SIAM journal on optimization* 8.3 (1998), pp. 631–657.

[20] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* (2002). ISSN: 1089778X. DOI: 10.1109/4235.996017.

[21] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 182–197. ISSN: 1089778X. DOI: 10.1109/4235.996017.

[22] Tom Ellis, Tom Adie, and Geoff S. Baldwin. "DNA assembly for synthetic biology: From parts to pathways and beyond". In: *Integrative Biology* (2011). ISSN: 17579694. DOI: 10.1039/c0ib00070a.

[23] Carola Engler et al. "A Golden Gate modular cloning toolbox for plants". In: *ACS Synthetic Biology* 3 (11 Nov. 2014), pp. 839–843. ISSN: 21615063. DOI: 10.1021/sb4001504.

[24] Carola Engler et al. "Golden gate shuffling: A one-pot DNA shuffling method based on type ils restriction enzymes". In: *PLoS ONE* (2009). ISSN: 19326203. DOI: 10.1371/journal.pone.0005553.

[25] Angelo Gaeta, Valentin Zulkower, and Giovanni Stracquadanio. "Design and assembly of DNA molecules using multi-objective optimisation". In: *bioRxiv* (2019). ISSN: 2692-8205. DOI: 10.1101/761320.

[26] Daniel G Gibson et al. "Chemical synthesis of the mouse mitochondrial genome". In: *Nature methods* 7.11 (2010), p. 901.

[27] Daniel G Gibson et al. "Creation of a bacterial cell controlled by a chemically synthesized genome". In: *science* 329.5987 (2010), pp. 52–56.

[28] Daniel G. Gibson et al. "Enzymatic assembly of DNA molecules up to several hundred kilobases". In: *Nature Methods* 6 (5 2009), pp. 343–345. ISSN: 15487091. DOI: 10.1038/nmeth.1318.

[29] A Goffeau et al. "Life with 6000 Genes conveniently among the different interna- Old Questions and New Answers The genome . At the beginning of the se- of its more complex relatives in the eukary- cerevisiae has been completely sequenced Schizosaccharomyces pombe indicate". In: *Science* 274.October (1996), pp. 546–567. ISSN: 0036-8075. DOI: jyu.

[30] Timothy S. Ham et al. "Design, implementation and practice of JBEI-ICE: An open source biological part registry platform and tools". In: *Nucleic Acids Research* (2012). ISSN: 03051048. DOI: 10.1093/nar/gks531.

[31] Wendy Higashide et al. "Metabolic engineering of Clostridium cellulolyticum for production of isobutanol from cellulose". In: *Applied and Environmental Microbiology* (2011). ISSN: 00992240. DOI: 10.1128/AEM.02454-10.

[32] Nathan J Hillson. "DNA Cloning and Assembly Methods". In: 1116 (2014), pp. 245–269. ISSN: 1940-6029. DOI: 10.1007/978-1-62703-764-8. URL: http://link.springer.com/10.1007/978-1-62703-764-8.

[33] Nathan J Hillson. "DNA Cloning and Assembly Methods". In: *Methods in Molecular Biology* 1116 (2014), pp. 245–269. ISSN: 1940-6029. DOI: 10.1007/978-1-62703-764-8. URL: http://link.springer.com/10.1007/978-1-62703-764-8.

[34] Adam J. Hockenberry et al. “Quantifying position-dependent codon usage bias”. In: *Molecular Biology and Evolution* 31 (7 2014), pp. 1880–1893. ISSN: 15371719. DOI: 10.1093/molbev/msu126.

[35] Miten Jain et al. “The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community”. In: *Genome Biology* 17 (1 Dec. 2016). ISSN: 1474-760X. DOI: 10.1186/s13059-016-1103-0.

[36] Reinhard Jetter and Ljerka Kunst. “HARNESSING PLANT BIOMASS FOR BIOFUELS AND BIOMATERIALS Plant surface lipid biosynthetic pathways and their utility for metabolic engineering of waxes and hydrocarbon biofuels”. In: (). DOI: 10.1111/j.1365-313X.2008.03467.x.

[37] Leisa Johnson et al. “Selectively replicating adenoviruses targeting deregulated E2F activity are potent, systemic antitumor agents”. In: *Cancer Cell* (2002). ISSN: 15356108. DOI: 10.1016/S1535-6108(02)00060-0.

[38] M C Kennedy et al. “wt C R”. In: October (2013), pp. 475–480.

[39] Joshua D Knowles and David W Corne. “Approximating the nondominated front using the Pareto archived evolution strategy”. In: *Evolutionary computation* 8.2 (2000), pp. 149–172.

[40] Harald Konig et al. “Synthetic Genomics and Synthetic Biology Applications Between Hopes and Concerns”. In: *Current Genomics* (2013). ISSN: 13892029. DOI: 10.2174/1389202911314010003.

[41] Roberta Kwok. “Five hard truths for synthetic biology”. In: *Nature* 463.7279 (2010), pp. 288–290. ISSN: 00280836. DOI: 10.1038/463288a. URL: http://www.nature.com/doifinder/10.1038/463288a.

[42] James Weifu Lee. “Switchable photosystem-II designer algae for photobiological hydrogen production”. In: (Jan. 2010).

[43] Heng Li. “Minimap2: pairwise alignment for nucleotide sequences”. In: (Aug. 2017). DOI: 10.1093/bioinformatics/bty191. URL: http://arxiv.org/abs/1708.01492%20http://dx.doi.org/10.1093/bioinformatics/bty191.

[44] Vincent J J Martin et al. “Engineering a mevalonate pathway in Escherichia coli for production of terpenoids”. In: *Nature Biotechnology* (2003). ISSN: 10870156. DOI: 10.1038/nbt833.

[45] Oriah Mioduser, Eli Goz, and Tamir Tuller. “Significant differences in terms of codon usage bias between bacteriophage early and late genes: A comparative genomics analysis”. In: *BMC Genomics* 18 (1 Nov. 2017). ISSN: 14712164. DOI: 10.1186/s12864-017-4248-7.

[46] A. A. K. Nielsen et al. “Genetic circuit design automation”. In: *Science* 352.6281 (2016), aac7341–aac7341. ISSN: 0036-8075. DOI: 10.1126/science.aac7341. URL: http://www.sciencemag.org/cgi/doi/10.1126/science.aac7341.

[47] Alec A.K. Nielsen et al. “Genetic circuit design automation”. In: *Science* 352.6281 (Apr. 2016). ISSN: 10959203. DOI: 10.1126/science.aac7341.

[48] Eva Maria Novoa et al. “Elucidation of Codon Usage Signatures across the Domains of Life”. In: *Molecular Biology and Evolution* (May 2019). ISSN: 0737-4038. DOI: 10.1093/molbev/msz124.

[49] Ernst Oberortner et al. “Streamlining the Design-to-Build Transition with Build-Optimization Software Tools”. In: *ACS Synthetic Biology* 6.3 (2017), pp. 485–496. ISSN: 21615063. DOI: 10.1021/acssynbio.6b00200.

[50] Ernst Oberortner et al. “Streamlining the Design-to-Build Transition with Build-Optimization Software Tools”. In: *ACS Synthetic Biology* 6.3 (2017), pp. 485–496. ISSN: 21615063. DOI: 10.1021/acssynbio.6b00200.

[51] C. J. Paddon et al. “High-level semi-synthetic production of the potent antimalarial artemisinin”. In: *Nature* 496.7446 (Apr. 2013), pp. 528–532. ISSN: 00280836. DOI: 10.1038/nature12051.

[52] Sujatha Thankeswaran Parvathy, Varatharajalu Udayasuriyan, and Vijaipal Bhadana. *Codon usage bias*. Jan. 2022. DOI: 10.1007/s11033-021-06749-4.

[53] Priscilla E. M. Purnick and Ron Weiss. "The second wave of synthetic biology: from modules to systems". In: *Nature Reviews Molecular Cell Biology* 10.6 (2009), pp. 410–422. ISSN: 1927629X. DOI: 10.1038/nrm2698. URL: https://www.nature.com/articles/nrm2698.pdf.

[54] Tijana Radivojević et al. "A machine learning Automated Recommendation Tool for synthetic biology". In: *Nature Communications* 11 (1 Dec. 2020). ISSN: 20411723. DOI: 10.1038/s41467-020-18008-4.

[55] Sarah M Richardson et al. "GeneDesign : Rapid , automated design of multikilobase synthetic genes GeneDesign : Rapid , automated design of multikilobase synthetic genes". In: (2006), pp. 550–556. DOI: 10.1101/gr.4431306.

[56] Sarah M. Richardson et al. "Automated design of assemblable, modular, synthetic chromosomes". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6068 LNCS.PART 2 (2010), pp. 280–289. ISSN: 03029743. DOI: 10.1007/978-3-642-14403-5{_}30.

[57] Sarah M. Richardson et al. "Design of a synthetic yeast genome". In: *Science* 355 (6329 2017), pp. 1040–1044. ISSN: 10959203. DOI: 10.1126/science.aaf4557.

[58] Dae Kyun Ro et al. "Production of the antimalarial drug precursor artemisinic acid in engineered yeast". In: *Nature* (2006). ISSN: 14764687. DOI: 10.1038/nature04640.

[59] Nicholas Roehner et al. "Double dutch: A tool for designing combinatorial libraries of biological systems". In: *ACS Synthetic Biology* (2016). ISSN: 21615063. DOI: 10.1021/acssynbio.5b00232.

[60] Nicholas Roehner et al. "Double dutch: A tool for designing combinatorial libraries of biological systems". In: *ACS Synthetic Biology* 5.6 (2016), pp. 507–517. ISSN: 21615063. DOI: 10.1021/acssynbio.5b00232.

[61] Nicholas Roehner et al. "Sharing Structure and Function in Biological Design with SBOL 2.0". In: *ACS Synthetic Biology* 5.6 (2016), pp. 498–506. ISSN: 21615063. DOI: 10.1021/acssynbio.5b00215.

[62] F Sanger, S Nicklen, and A R Coulson. *DNA sequencing with chain-terminating inhibitors (DNA polymerase/nucleotide sequences/bacteriophage 4X174)*. 1977, pp. 5463–5467.

[63] Paul M Sharp, Therese M F Tuohy, and Krzysztof R Mosurskil. "Nucleic Acids Research Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes". In: 14 (1986). DOI: 512543.

[64] Pawel Sledzinski, Mateusz Nowaczyk, and Marta Olejniczak. *Computational Tools and Resources Supporting CRISPR-Cas Experiments*. May 2020. DOI: 10.3390/cells9051288.

[65] Giovanni Stracquadanio, Vittorio Romano, and Giuseppe Nicosia. "Semiconductor device design using the BIMADS algorithm". In: *Journal of Computational Physics* (2013). DOI: 10.1016/j.jcp.2013.01.025.

[66] Giovanni Stracquadanio et al. "Multi-objective optimization of doping profile in semiconductor design". In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM. 2010, pp. 1243–1250.

[67] Donald E Trimbur and Palo Alto. *Application Data Continuation of application No. 12/131,783, filed on*. Tech. rep. 2008.

[68] Marc Valls et al. "Engineering a mouse metallothionein on the cell surface of Ralstonia eutropha CH34 for immobilization of heavy metals in soil". In: *Nature Biotechnology* (2000). ISSN: 10870156. DOI: 10.1038/76516.

[69] Alan Villalobos et al. "Gene Designer: A synthetic biology tool for constructuring artificial DNA segments". In: *BMC Bioinformatics* 7 (2006), pp. 1–8. ISSN: 14712105. DOI: 10.1186/1471-2105-7-285.

[70] Baojun Wang et al. “Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology”. In: *Nature Communications* 2 (1 2011). ISSN: 20411723. DOI: 10.1038/ncomms1516.

[71] Ryan Wick. “Badread: simulation of error-prone long reads”. In: *Journal of Open Source Software* 4 (36 Apr. 2019), p. 1316. ISSN: 2475-9066. DOI: 10.21105/joss.01316.

[72] Ezequiel Di Paolo Xiao-Bing Hu Ming Wang. “Calculating Complete and Exact Pareto Front for Multiobjective Optimization: A New Deterministic Approach for Discrete Problems”. In: *IEEE Transactions on cybernetics* 43.3 (2013), pp. 1088–1101.

[73] Kun Yang et al. “BioPartsBuilder: A synthetic biology tool for combinatorial assembly of biological parts”. In: *Bioinformatics* (2015). ISSN: 14602059. DOI: 10.1093/bioinformatics/btv664.

[74] Kun Yang et al. “BioPartsBuilder: A synthetic biology tool for combinatorial assembly of biological parts”. In: *Bioinformatics* 32.6 (2015), pp. 937–939. ISSN: 14602059. DOI: 10.1093/bioinformatics/btv664.

[75] Yohei Yokobayashi, Ron Weiss, and Frances H Arnold. “Directed Evoln”. In: 99.26 (2002), pp. 1–5. URL: papers2://publication/uuid/FD5CE032-AF3E-45A2-B34C-4FC8D227F8AB.

[76] Qingfu Zhang and Hui Li. “MOEA/D: A multiobjective evolutionary algorithm based on decomposition”. In: *IEEE Transactions on evolutionary computation* 11.6 (2007), pp. 712–731.

[77] Hao Zheng and Hongwei Wu. “Gene-centric association analysis for the correlation between the guanine-cytosine content levels and temperature range conditions of prokaryotic species”. In: *BMC Bioinformatics* 11.SUPPL. 11 (Dec. 2010). ISSN: 14712105. DOI: 10.1186/1471-2105-11-S11-S7.

[78] Zhipeng Zhou et al. “Codon usage is an important determinant of gene expression levels largely through its effects on transcription”. In: *Proceedings of the National Academy of Sciences* 113.41 (2016), E6117–E6125.

[79] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. “Quality assessment of pareto set approximations”. In: *Multiobjective Optimization*. Springer, 2008, pp. 373–404.

[80] Eckart Zitzler and Lothar Thiele. “Multiobjective optimization using evolutionary algorithms — A comparative case study”. In: *Parallel Problem Solving from Nature — PPSN V*. Springer, Berlin, Heidelberg, 1998. ISBN: 3-540-65078-4. DOI: 10.1007/BFb0056872.