Automatic restriction strategy finder for Synthetic Biology Constructs

Jorge Gomes[1,3], Bjorn Johansson[2,3]

[1] School of Engineering, University of Minho
Jorge.gomes12@gmail.com
[2] CBMA, Campus of Gualtar, University of Minho
bjorn_johansson@bio.uminho.pt
[3]Campus of Gualtar, 4710-057, Braga, Portugal

Abstract. Synthetic Biology has established itself in the recent years, aided by the convergence of molecular biology and the engineering principles, underpinned by the massive advances in biological technologies, that have allowed the creation of full length genes, operons and even genomes *de novo.* Restriction enzymes used for analysis of genetic constructs are normally chosen ad-hoc and this choice becomes both more difficult and critical with the increasing number of available enzymes, and the increasing length and complexity of the constructs. The main goal of this project is to design an algorithm for the automatic selection of the most effective restriction enzymes for DNA analysis based on user defined criteria, whilst minimizing the number of enzymes necessary, thus reducing the costs of the operation and the time spent on it. Using Python coding language and pydna package, an algorithm has been designed to effectively search for the enzyme or enzyme set that could produce distinguishable agarose gel results, for each inputted sequence. Using two different sequence sets with very distinct characteristics, the algorithm shown to be efficient and fully functional, providing one enzyme for each sequence set, that was able to digest all of the sequences and provide a method of distinction for the sequences.

Keywords: Synthetic Biology, DNA, Agarose gel, Restriction enzymes, Python, pydna.

1 Introduction

The presented introduction will focus on reviewing all of the important intelligence on synthetic biology and cloning methodologies, emphasizing its development and current state. The objectives of this project will also be presented, along with the problem definition that is the rationale for this project.

1.1 State of the art on Synthetic Biology and Cloning

DNA, as any other form of data, can be both read and written. For DNA, the process of reading is something known as DNA sequencing, whereas writing is achieved by

gene synthesis. Over the last decade the main concern of molecular biology has been set on reading and analyzing naturally occurring DNA sequences, as revealed by massive sequencing efforts worldwide, on a very wide range of organisms. In contrast, the emerging field of Synthetic Biology aims to write new genetic informatic, thereby creating designed, non-natural genes, proteins, biological processes, and even organisms [17].

Gene synthesis was conceived as a means of gene acquisition in the 1970s and early 1980s [1,11], meant to be applied on living systems, as recombinant DNA technologies allowed biologist to deliberately change the molecular structure of the organisms, and the chemical synthesis of DNA became widely available [15]. However, this approach would be soon overtaken by cloning libraries and PCR in the following years. More recently, protein and DNA sequences have become much easier to obtain electronically from databases than physically from library clones. In the meantime, gene synthesis technology, has matured, allowing direct gene synthesis to become the most efficient way of producing functional genetic constructs, enabling a variety of applications from codon optimization to protein engineering [17].

Synthetic Biology has established itself in the recent years, aided by the convergence of molecular biology and the engineering principles, underpinned by the massive advances in biological technologies [7], that have allowed the creation of full length genes, operons and even genomes *de novo* [17]. In the last 40 years, the length of synthetic genes that can be obtained in a laboratory has increased by four orders of magnitude, ranging from 10^2 to 10^6 base pairs [16]. In 2010 it was estimated that at this rate, the synthesis of a genome as complex as the human (10^9 base pairs) could be feasible in 10 years' time [7]. In the same year Gibson and his coworkers managed to create a fully functional bacterial cell controlled only by its 1.08 mega base pair chemically synthetized genome [5]. Further developments among bacterial genomes have been made, however eukaryotic genomes present additional challenges, as the genome is much larger and much more complex. Another 4 years later another landmark would be achieved, the first eukaryotic chromossome was synthetized for *Saccharomyces cerevisiae*, a fully functional copy of its 316,617-base pair chromossome III [2]. In fact, the synthesis of the Human Genome might not be that far away, since scientists have already announced the next take on the Human Genome Project - HGP-Write – a 10-year project which aims to completely synthetize the human genome [3]. One of the facts that supports this project is the continuous downfall of the price for synthetizing genes, with prices coming as low as 0.01 $ per base pair [7].

Synthetic biology holds now a tremendous potential as both an investigative and therapeutic modality, since it can represent a major breakthrough in some of humanity biggest concerns', namely, the production of environmentally friendly biofuels, or inexpensive pharmaceutical drugs [7, 18].

For those goals, Synthetic Biology makes the best use for a variety of microorganisms and plants, which have evolved to produce a myriad array of complex molecules known as natural products or secondary metabolites that are of biomedical and biotechnological importance [8]. The ability to synthetize said molecules requires prior knowledge on a species' genome and metagenome, which represents a rich source for discovery of

novel pathways involved in natural product biosynthesis [13]. This is commonly referred as pathway or metabolic engineering, a process where complete biosynthetic pathways are often transferred from native hosts to heterologous organisms in order to obtain products as referred before, but with higher yields than in a non-engineered organism. Consequently, gene expression needs to be balanced, promoter strength needs to be tuned and the endogenous regulatory network needs to be modified [14]. One of two approaches can be chosen here, the combinatorial one, where genes are rearranged in a construct to generate small molecules, or the synthetic way, whereas complex combination of genetic elements are used to create new circuits, with designed properties. In any of these approaches, the conventional multi-step, sequential-cloning method including primer design, PCR amplification, restriction digestion, *in vitro* assembly and transformation, is typically involved and multiple plasmids are often required. This method however can be time consuming and relies on unique restriction sites that become limited for large recombinant DNA molecules [14]. The lack of facile, highly efficient manipulation techniques for libraries of interchangeable genetic elements has stood as a significant hurdle for biosynthetic constructs. As this necessity arose, efforts have been made to develop revolutionary techniques, to transform arduous constructions into routine tasks [4].

Modern DNA assembly techniques can be classified into two groups: those based on homology, and those based on ligation. Homology based methods require neighboring DNA fragments to share identical sequences, such that splicing can occur either by annealing by either annealing followed by extension of the homologous ends *in vitro* or by homologous recombination *in vivo,* a method known as overlap-directed assembly. The most distinguishable *in vitro* technique is probably the one proposed by Gibson and his coworkers, colloquially know as Gibson Assembly [6], which, as stated above, made possible the assembly of a fully functional bacterial cell. Worth mentioning is also the Shao 'DNA assembler' method which enables design and rapid construction of large biochemical pathways in one-step fashion by exploitation of *in vivo* homologous recombination mechanism in *S. cerevisiae*. Due to its high efficiency and ease to work with, *in vivo* homologous recombination in yeast has been widely used for gene cloning, plasmid construction and library creation in the past, yet yeast *in vivo* homologous recombination was only successfully used to assemble multiple-gene biochemical pathways in a plasmid, by Shao and his co-workers in 2009 [13]. A more recent example of the same effort is the work published by Mitchell and his colleagues who managed to develop a method for assembling genetic expression pathways for expression in *Saccharomyces cerevisiae*¸ which takes advantage of the organism capacity to perform homologous recombination, efficiently joining sequences with terminal homologies. The versatile genetic assembly system (VEGAS) uses a modified version of the Golden Gate cloning method, in which each Transcription Unit is assigned a pair of VEGAS adapters that assemble up and down stream, providing terminal homology for overlap-directed assembly by homologous recombination *in vivo.* Through this approach, it was possible to assemble the β-carotene and the violacein biosynthetic pathways [10].

While in the past, small DNA constructs incorporating few parts were common, the complexity of new constructs has grown with advancing technology. Techniques like

the ones mentioned earlier are commonly referred to as “next generation cloning”, since these protocols describe the assembly of ten to twenty PCR fragments into a complex DNA construct. The complexity of these strategies results in high risks of cloning errors and omissions, and without proper documenting and *in silico* simulation these strategies are not fully reproducible by other laboratories [12]. A possible solution to these problems is a strategy description that is both readable by humans and executable by a computer to simulate the individual steps of the protocol as well as the result. To assess this necessity several efforts were driven by the bioinformatic community [12]. Here I will be discussing Pydna, as it is the platform where my project will be developed upon.

Pydna is software tool that provides high level computer simulation of DNA manipulation procedures and aid the design of complex constructs. It allows automated primer design for homologous recombination cloning or Gibson assembly, as well as a simulator of DNA assembly. This software package was implement exclusively in Python and makes uses of the Biopython and NetworkX packages. Most of the Pydna functionalities are implemented as methods for the Dseqrecord class, designed to hold all the sequence information necessary for describing a double-stranded DNA molecule. Dseqrecord objects reflect much of the functionality of SeqRecord objects from Biopython. Even though Pydna works through a command line interface, the code semantically resembles the molecular biology unit operations, making it easy to read even for non-programmers. An example using this package is presented in Figure 1, where Pydna executes the assembly of a circular molecule from 3 PCR products also obtained from Pydna *in silico* simulation. Pydna also supports integration with IPython notebook, a format where figures, code and text can be combined, allowing for the creation of effective workflow description for complex constructs [12].

```
In [19]: asm = pydna.Assembly((yep_bgl,cycl_prd, gfp_prd))
In [20]: cnt = asm.circular_products[0]
In [21]: cnt
Out[21]: Contig(o10681)
In [22]: cnt.figure()
Out[23]:
```

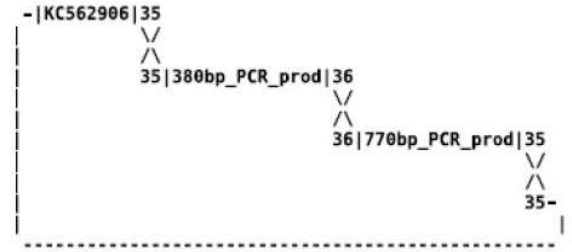

Fig. 1: The above code executes the assembly of a circular molecule of DNA based on previously obtained PCR products (yep_bgl, cycl_prd, gfp_prd), and prints in a text based figure the way the sequences were assembled. The input in [19] assembles the fragments, in several different recombined constructs, both linear and circular, whereas the code input in [20] accesses only the first circular product of said assembly. Assemblies are stored as Contig objects which can be represented with the **figure** method.

1.2 Problem definition and objectives

As we've seen there are many new cloning strategies by which very large and complex genetic constructs can be synthesized. The size and complexity means that verification of the construct becomes a challenge. One of the fastest and most robust ways to analyze DNA molecule structure continues to be restriction analysis with a set of carefully chosen restriction enzymes. Restriction enzymes are incontestably the most fundamental tool used in molecular biology, and over the past years, more than 3500 restriction have been isolated [9]. However, because of the large number of restriction enzymes now available commercially, it is becoming truly difficult to navigate through an ever-increasing number of choices, to find the enzyme or enzymes that we actually need. Restriction enzymes used for analysis of genetic constructs are normally chosen ad-hoc and this choice becomes both more difficult and critical with the increasing number of available enzymes, and the increasing length and complexity of the constructs. This method can be both time and resource consuming, since it's commonly based on a trial and error approach, even with the available bioinformatic tools.

As we discussed previously, most constructs are based on homologous overlapping sequences, so one way of analyzing those constructs is to assess where those homologies are located, and for that we need to digest them. That requires a restriction enzyme set that can cut through all the sequences of the construct, and differentiate them based on the digestion products. Many of these DNA molecules are plasmids, whose nucleotides are arranged in a circular-fashion. Therefore, the search for restriction sites within a plasmid constitutes a circular string pattern matching problem, which requires a string to be rotated several times in order to find a match. However, that problem is already addressed by Pydna Dseqrecord objects' which support circular sequences, and by Biopython restriction algorithms'.

The main goal of this project is to design an algorithm for the automatic selection of the most effective restriction enzymes for DNA analysis based on user defined criteria, whilst minimizing the number of enzymes necessary, thus reducing the costs of the operation and the time spent on it. The final use of the implemented algorithm should be to differentiate between several plasmids or constructs that share similarities, but feature different genes in its multiple cloning site, as the 5 different constructs produced by Mitchell and its associates [10], or any results from cloning, to assess if the cloning experiment was successful, i.e., the insert was introduced. The algorithm will then be implemented in Python and embedded within Pydna software package, or make use of some of its capabilities.

2 Methodology

In this section, several points will be discussed, with the main focus being set on the algorithm design and explanation. Both graphical and textual explanations are provided, to facilitate the understanding of the algorithm. Testing methodologies will also be presented.

2.1 Algorithm design and implementation

In order to differentiate between any number of sequences using restriction enzymes, the selected restriction enzyme(s) must produce a different pattern when observed in an agarose gel [13]. Since we will be working with plasmids and long sequences (over 2kb), and possibly many sequences, this could be hard to achieve given the fact that every sequence has a different insert. Because one of the main goals of this project is to reduce costs of a diagnostic digest, using one restriction enzyme or more per sequence is not an option, and should only be made if it's the only possible way, what is highly unlikely. So, our algorithm must search for the least number of enzymes that are able to cut through several sequences, and produce a unique pattern for each one of them, since that will be the only way to differentiate them. Also, enzymes can only produce fragments bigger than a user defined threshold, to produce viewable results in an agarose gel. The algorithm design followed essentially the steps described in the preliminary report, with some improvements.

The chosen pipeline, explicitly shown in Figure 2, was to search for the biggest contiguous sequence between the given sequences, and search for restriction enzymes that would only produce two fragments in this area, as a way of trying to generate similar number of bands at least on the contiguous area, and after that, search for the enzymes that could cut through the sequences non-contiguous area more than once, to produce distinguishable results. After that, it's given preference to enzymes that feature in both searches, since it allows for cost reduction, but in cases where it is not possible, it tries every single combination of enzymes featuring the first and the second set.

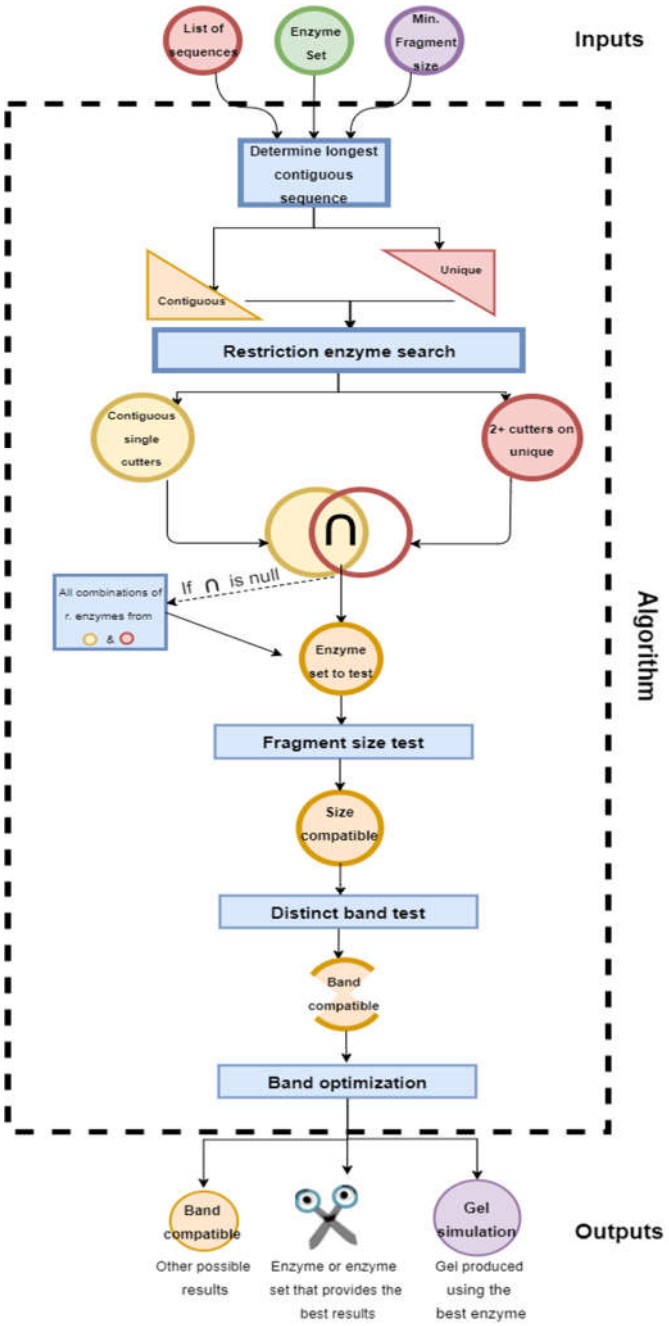

Fig. 2: The above pipeline resembles the algorithm steps performed with the objective of finding the best enzyme or enzyme set, that could produce the best results, based on the inputted sequences and other user-defined parameters.

A possible case-scenario for this problem is illustrated in Figure 3, where we want to differentiate between 4 hypothetical plasmids, and the given enzymes are known to cut through each plasmid insert zone. The best possible solution would be to use EcoRI since it cuts through every single sequence. However, we do not know if EcoRI will produce unique patterns for each one of them, and if the distinguishing bands fulfill the minimum size criteria, thus falling in the non-viewable range of the gel. In some cases, it might not even exist an enzyme that is able digest all of them, so we should, for example instead use a combination of EcoRV and XbaI, or PstI and NotI, or PvuII and SmaI, but again we don't know if any of those sets will produce distinguishable fragments. So, the algorithm starts in the center of the presented Venn diagram, and tests the restriction enzymes

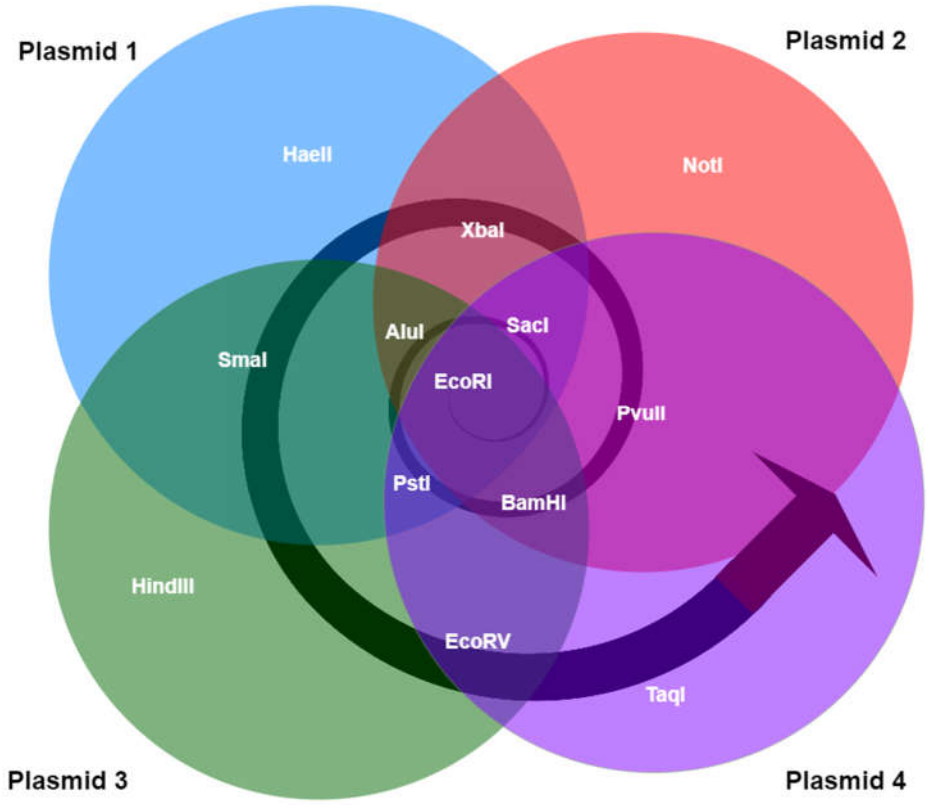

Fig. 3: Venn diagram explaining the enzyme selection step of the algorithm. Each circle represents one different hypothetical plasmid, with enzymes the cut throughs it's insert region inside, and the spiral arrow resembles the algorithm progression, which starts with the most versatile restriction enzymes (centre) and when there's no common enzyme it starts out a spiralling motion trying to find combinations of enzymes that produce the same results as a versatile one would produce.

for fragment distinction and size, but if it founds no results it starts to seek for combinations of enzymes that, together cut through each plasmid insert. It keeps its spiral motion until it eventually reaches the outer limit of the diagram, and the worst-case scenario which requires one different restriction enzyme for each plasmid. This design was chosen with the objective of minimizing the costs of the diagnostic digestion.

After the enzyme selection step, the algorithm narrows down the possible results, by searching for the restriction enzyme(s) that produces distinguishing bands in a gel, i.e., each sequence digestion must produce at least one fragment that is not present in any other sequence. Finally, the chosen enzyme was the one in which the minimal difference between any two fragments digestion is the biggest. This last step, assures better gel readability and thus better chances of differentiating between the enzymes. However, in some cases this step might take some more seconds, so it can be skipped or not,

by user choice. Before this step every enzyme still being considered produces distinguishing results so skipping this step will still produce results, and since it assumes a randomized fashion, it can even give the best result possible. Making use of pydna capabilities, the program can even simulate the agarose gel produced by using the best restriction enzyme or set. After results are prompted we can even check if the best result is an isoschizomer, meaning that, if True, we can indeed use other enzymes, which should also be present in the results.

This algorithm was designed to issue one specific problem, and thus it works only for the presented cases, in which we try to differentiate several sequences which share similarity but have different regions, commonly called as inserts. As stated previously it's usage addresses for cloning experiments, and synthetic biology constructs, trying to provide a solution for plasmid differentiation. Providing real-world examples: let's imagine someone exchanges the labels for two plasmid Eppendorfs, or wants to see if their cloning experiment was successful, using this program solves both problems saving money and time trying to correctly identify the plasmids by digestion. Therefore, it won't work for one sequence, small sequences(<1kb), or sequences that do not share any similarities. The program accepts linear or circular DNA, single or double stranded, and since it uses a brute-force approach it will yield exact results, which shaves off some of its efficiency. It provides a user-friendly interface, with intuitive parameter, function and attribute names, proper documentation, and several warnings and suggestions regarding inputted parameters or sequences. Program usage is explained in a comprehensive way in the Example.ipynb file. The algorithm was fully implemented in Python 3.6.

2.2 Test building and timing

To test the algorithm capabilities and performance, two different sequence sets with different characteristics were chosen.

The first set of sequences consists of 4 variations of the pUC18 cloning vector, with different inserted genes, available at GenBank accession numbers L08752.1, LC129268.1, U07164.1 and U03991, all with sequence sizes around 3kb, and a contiguous sequence of 2240bp between them. Since those sequences do come from different sources, and are made up of circular DNA, they needed to be rotated to share the same origin. Using pydna Dseqrecord objects allowed the use of *synced()* method, with which every sequence was synchronized with pUC18 original sequence, and stored in a text file in fasta format. For this sequence two tests were performed, with different minimum fragment sizes, 50bp and 200bp, to understand what influence it may have over the obtained results.

The second set of enzymes consists of 6 yeast based cloning vectors derived from Mitchell and its associates cloning experiment [10], each one with one different inserted genes. Sequence size is over 14kb for each one of them, and the contiguous sequence has 2.8kb. No synchronizing was needed since all sequences came from the same experiment and therefore share the origin. Three tests were run with this sequence set,

with changes in the fragment size and in the iso parameter. More details available at Results and Discussion section.

To better understand how well the algorithm scaled with sequence size and fragment sizes, and access its usability in real-world situations for each test ran, the algorithm was timed using Python built-in timing methods.

2.3 Code availability and software requirements

All of the code necessary to run the algorithm relies within one Python script which is available at GitHub (https://github.com/JorgeACGomes/Automatic-Restriction-Strategy-Finder). Along with the Restriction_Finder.py module, both sequence sets used for testing are available too, as well as a very comprehensive IPython Notebook file explaining the usage of the program, and all of its functionalities.

All of the changes made to the algorithm and every problem encountered with the algorithm implementation, were also reported in the README file, available at the given GitHub repository.

Running this program requires Python 3.6 as well as the pydna package [12] and all of its dependencies.

3 Results and Discussion

At this point, the produced results will be presented and discussed, in a trial to assess the algorithm strengths and weaknesses.

3.1 Sequence set: seqs.txt

For this sequence set two tests were performed, both of which managed to measure the correct algorithm operation, and timing in seconds. The results will be presented in Table I, which comprises the used parameters and the obtained results, for both timing and restriction enzyme search. The parameters not shown were set to default. All of the info presented could be obtained with simple manipulation of the Restriction_Finder class in which the algorithm is contained. Gel results for the first test are presented in Figure 4, since the second test didn't manage to find any results. The algorithm managed to find three suitable enzymes when the minimum fragment size was set to 50, in under 11 seconds. All of the three possible enzymes are isoschizomers, meaning that any of them will work as intended and produce the same results. For this case, the program prompts a warning, for the user to know that some other result may be retrieved as best the next time the program is ran, since it is an isoschizomer. Using a more restrict fragment size retrieved no results, and took longer time to execute. It is working as intended, and the results were expected. Since the insert size is reduced for most sequences, using such minimum fragment size, causes the algorithm to produce a longer search, but since it found no restriction

enzymes able to digest all of the plasmids and produce fragments bigger than the defined threshold it retrieved no results after 18 seconds. If the fragment size would be an issue and it had to be restrained to a higher value, this program approach wouldn't be able to find results, with its current implementation. The agarose gel resultant from the digestion with PshBI proved that differentiation of the four sequences is possible using only one enzyme. Despite the small difference between the first three lanes, it is noticeable that each one has a unique band that allows the correct identification. An approximation of the results from the gel in Figure 4 can be obtained using Agarose at 1%, and default electrophoresis settings.

Table 1. Results obtained for the first sequence set, using Restriction_Finder capabilities.

Test	Minimum fragment size (bp)	Mean of sequence sizes (bp)	Contiguous sequence size (bp)	Insert sizes (bp)	Best Restriction Enzyme	Other possible	Timing(s)
1	50	3198	2240	446, 568, 2317, 500	PshBI	VspI, AseI	10.753
2	200	3198	2240	446, 568, 2317, 500	-	-	18.541

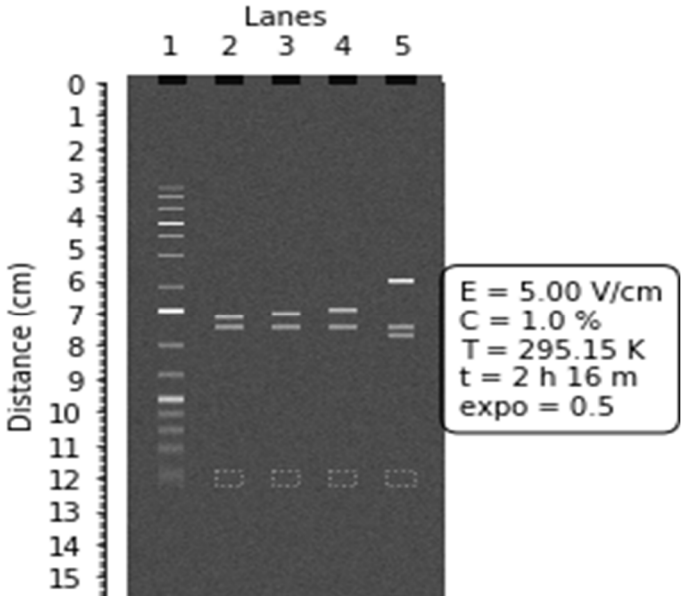

Fig. 4: Simulation of the agarose gel that would be obtained by restriction on the first set of sequences using PshBI at 1% agarose. Lane order after molecular weight ladder: L08752.1, U03991, LC129268.1, U07164.1.

3.2 Sequence set: seqs_vegas.txt

Since the sequences used here came from a real experiment, they fully adjust to what the program has been designed for, and provide a good test-bench for the algorithm capabilities. Besides the restriction enzyme searching function and timing, this sequence set made also possible to test the band optimization step. Five tests were conducted, using different minimum fragment size, different values for the iso parameter and the optim parameter. When set to False, the best enzyme retrieved is assured not to be an isoschizomer, unless all of the possible results are isoschizomers. The placement of this parameter aimed for the search of sub-optimal results, that could produce different gel patterns and used different enzymes, in a trial to adjust to laboratory conditions, where only a small set of enzymes is available at each time. The conditions for the performed tests are summed up in Table 2. This sequence set has a mean length of 15kb, and except for one insert of 3kb all the insert regions are north of 13kb, while the contiguous sequence is made up of 2776 similar nucleotides.

Table 2. Conditions used and results obtained for the tests ran over the seqs_vegas.txt sequence set utilizing the implemented algorithm.

Test	Minimum fragment size (bp)	Isoschizomers parameter	Optimization parameters	Best Restriction Enzyme	Other possible	Timing(s)
1	200	True	True	AanI	PsiI	26.96
2	150	True	True	BbsI	AanI, PsiI, BstV2I, BpiI	41.66
3	150	True	False	BpiI	AanI, BbsI, PsiI, BstV2I	33.28
4	50	True	True	BbsI	NspI, AanI, XceI, BsrFI, PsiI, BstNSI, TatI, and 9 others	61.59
5	50	False	True	TatI	NspI, AanI, XceI, BbsI, BsrFI, PsiI, BstNSI, Alw26I, Cfr10I, and 7 others	57.78

For every single ran test, results were obtained, which is a very positive achievement. Minimum fragment size proved to be no obstacle since both the insert regions and the contiguous sequence were big enough for the finding of restriction enzymes. Also, the obtained results with the optimization parameter set to on showed to be distinguishable when on an agarose gel, as the Figure 5 shows. To test the optimization of the bands one test was performed with that parameter set to False, and as so

the results proved to be less distinguishable, which is also observable in Figure 5. The search for sub-optimal results was also achieved, and retrieved a non isoschizomer enzyme, which was able to still produce good results on a gel. After analyzing these results, and crossing them with the ones from the previous sequence set, a more complete inference about algorithm timings can be done. Time of result retrieval has shown an overall increase when compared with the first sequence set, which was expected, since sequence size is over 5 times larger, and therefore more enzymes are able to find their restriction sites. Should too be considered that this sequence set features 2 more sequences. Also, the time consumed by the algorithm shown to be bigger when the minimum fragment size was smaller, since more enzymes would be accepted and thus more tests need to be run. Should also be considered that for all of the tests with both sequence sets, circular DNA was considered. Using linear DNA is expected to show a little decrease in time, since sequences need to be cut one time less. Disabling the band optimization, was also a manner to try and reduce the time of result retrieval but improvements were smaller than 10 seconds, and since results may not be the optimal ones, in a real-world scenario would not be recommended. Despite the increase in sequence number and size the algorithm shown a very positive performance, while still yielding optimal results, when parametrized to.

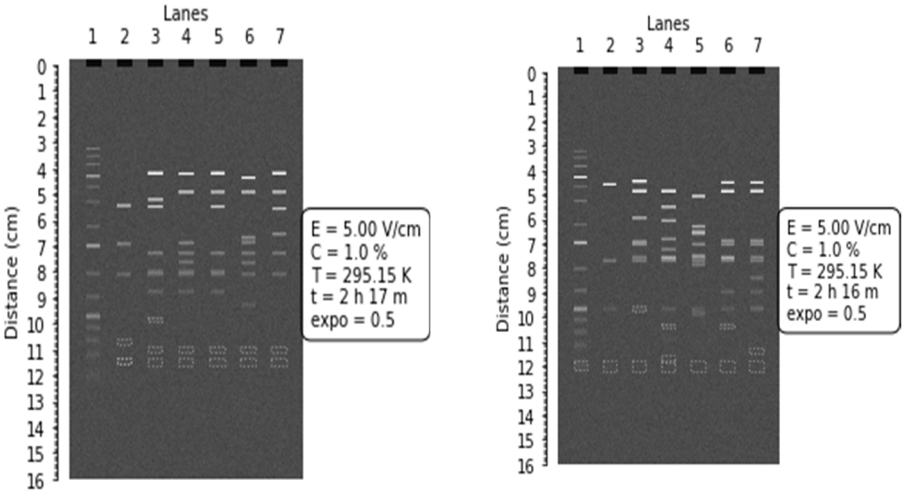

Fig. 5: Gel simulation results obtained using different parameters for the band optimization step (Left: True, Right: False). On the left a restriction with BbsI was performed, as on the right the digestion enzyme was BpiI. It is clear that the lower part of the gel on the left presents bands much closer to each other, which may difficult the correct identification of the sequences in each lane. These results are not optimal , which implies that the band optimization process is working correctly. The presented lanes are respectively: pJC170_pRS416_RF, pJC175_carotenoid, pJC178_carotenoid, pJC181_carotenoid, pJC184_carotenoid, pJC187_carotenoid.

3.3 Algorithms strengths and weaknesses

The obtained results provide a very favorable starting point for this algorithm, since it was able to find results for two very different sequence sets, in a wide variety of circumstances. Both the enzyme search and the band optimization steps of the algorithm proved to be working correctly. Despite using the Commercial enzyme set from Biopython's Restriction package for all test here presented, the algorithm can also use all of the available enzymes to perform the search. It was chosen to omit tests made with those enzymes, since they are not available commercially. The time needed for the result retrieval was very positive across all tests, taking at most 1 minute to retrieve an optimal enzyme, certainly less time than the manual Ad-Hoc search commonly done. Lastly, this algorithm supports both linear and circular DNA, and both single and double stranded DNA. The program provides several functionalities with a simple and very interactive interface. However, this algorithm has some weaknesses too, since it only works for the very specific cases, like the ones mentioned. Despite the time of retrieval being short, it could be even shorter, since it's design focused more on precise results than on efficiency. At a later time, a major overhaul of the code, will allow to the algorithm to run in even less time. The algorithm may contain bugs which may derive from using different test sequences, since it's development take course based on these two sequence sets. Other bugs may be present, which may not have been found over the stress-tests made. Another weakness, is the fact that the program is not, yet, embedded with pydna, and requires some dependencies to be run, which the common user may not have installed.

4 Conclusion

This project had a very strict objective since the start, which was to solve the problem of sequence differentiation, sequences that might derive from cloning and synthetic biology experiments. This objective was achieved, since for both sequence sets used for testing very positive results were obtained. The utilization of the developed program also allows for time and resource saving, since it minimizes the number of restriction enzymes needed, and at the worst tested case took 1 minute to yield results. Another self-proposed objective, was to write a program that would be easy to use by the common biologist. Not only that was achieved, but also several warnings were added to contribute to a more interactive experience for the user. A guide was made to explain all of the functionalities of the program and how it can be used. Reading the guide should provide everything needed to run the program.

On the personal side, this project allowed for the development of my Python, and programming skills, and also to get in touch with the GitHub platform, were all code

was stored, along with the tests and the mentioned guide. Progresses and problems were reported to the README.md file on my repository.

References

1. Agarwal, K. L., Büchi, H., Caruthers, M. H., Gupta, N., Khorana, H. G., Kleppe, K., ... & Sgaramella, V. Total synthesis of the gene for an alanine transfer ribonucleic acid from yeast. Nature, 227, 27-34 (1970).
2. Annaluru, N., Muller, H., Mitchell, L. A., Ramalingam, S., Stracquadanio, G., Richardson, S. M., ... & Cai, Y. Total synthesis of a functional designer eukaryotic chromosome. Science, 344(6179), 55-58 (2014).
3. Boeke, J. D., Church, G., Hessel, A., Kelley, N. J., Arkin, A., Cai, Y., ... & Isaacs, F. J. The genome project-write. Science, 353(6295), 126-127 (2016)
4. Cobb, R. E., Ning, J. C., & Zhao, H. DNA assembly techniques for next-generation combinatorial biosynthesis of natural products. Journal of industrial microbiology & biotechnology, 41(2), 469-477 (2014).
5. Gibson, D. G., Glass, J. I., Lartigue, C., Noskov, V. N., Chuang, R. Y., Algire, M. A., ... & Merryman, C. Creation of a bacterial cell controlled by a chemically synthesized genome. science, 329(5987), 52-56 (2010).
6. Gibson, D. G., Young, L., Chuang, R. Y., Venter, J. C., Hutchison, C. A., & Smith, H. O. Enzymatic assembly of DNA molecules up to several hundred kilobases. Nature methods, 6(5), 343-345 (2009).
7. LaVan, D. A., & Marmon, L. M. Safe and effective synthetic biology. Nature biotechnology, 28(10), 1010-1012 (2010).
8. Li, J. W. H., & Vederas, J. C. Drug discovery and natural products: end of an era or an endless frontier?. Science, 325(5937), 161-165 (2009).
9. Martin, P., Boulukos, K. E., & Pognonec, P. REtools: A laboratory program for restriction enzyme work: enzyme selection and reaction condition assistance. BMC bioinformatics, 7(1), 98 (2006).
10. Mitchell, L. A., Chuang, J., Agmon, N., Khunsriraksakul, C., Phillips, N. A., Cai, Y., ... & Blomquist, P. Versatile genetic assembly system (VEGAS) to assemble pathways for expression in S. cerevisiae. Nucleic acids research, gkv466 (2015).
11. Narnbiar, K. P., Stackhouse, J., Stauifer, D. M., Kennedy, W. P., & Eldredge, J. K. Total synthesis and cloning of a gene coding for the ribonuclease S protein. Science, 223, 1299-1301 (1984).
12. Pereira, F., Azevedo, F., Carvalho, Â., Ribeiro, G. F., Budde, M. W., & Johansson, B. Pydna: a simulation and documentation tool for DNA assembly strategies using python. BMC bioinformatics, 16(1), 142 (2015).
13. Reece, J. B., Taylor, M. R., Simon, E. J., & Dickey, J. L. Campbell biology: concepts & connections. Benjamin Cummings (2012).
14. Shao, Z., & Zhao, H. Construction and engineering of large biochemical pathways via DNA assembler. Synthetic Biology, 85-106 (2013).
15. Shao, Z., Zhao, H., & Zhao, H. DNA assembler, an in vivo genetic method for rapid construction of biochemical pathways. Nucleic acids research, 37(2), e16-e16 (2009).
16. Sismour, A. M., & Benner, S. A. Synthetic biology. Expert opinion on biological therapy, 5(11), 1409-1414 (2005).

17. Tian, J., Ma, K., & Saaem, I. Advancing high-throughput gene synthesis technology. Molecular BioSystems, 5(7), 714-722 (2009).
18. Villalobos, A., Ness, J. E., Gustafsson, C., Minshull, J., & Govindarajan, S. Gene Designer: a synthetic biology tool for constructing artificial DNA segments. *BMC bioinformatics*, *7*(1), 285 (2006).
19. Weeding, E., Houle, J., & Kaznessis, Y. N. SynBioSS designer: a web-based tool for the automated generation of kinetic models for synthetic biological constructs. Briefings in bioinformatics, 11(4), 394-402 (2010).