UNIVERSIDADE DO ALGARVE
Faculdade de Ciências e Tecnologia
Departamento de Ciências Biológicas e Bioengenharia

Automated bioinformatic discovery:

tools that tell you what you should know

A case-study in synthetic biology

Ricardo dos Santos Vidal

Dissertação para obtenção do grau de
Mestrado em Engenharia Biológica

2009

UNIVERSIDADE DO ALGARVE
Faculdade de Ciências e Tecnologia
Departamento de Ciências Biológicas e Bioengenharia

Automated bioinformatic discovery:

tools that tell you what you should know

A case-study in synthetic biology

Ricardo dos Santos Vidal

Dissertação para obtenção do grau de
Mestrado em Engenharia Biológica

Dissertação orientada por:
Dra. Maria Emília Lima Costa
Dra. Reshma Shetty

2009

I hereby declare that the content and execution of all the work produced in this dissertation is my own original work, unless otherwise disclosed.

_____

Ricardo dos Santos Vidal

September 21, 2009

"Scientists discover the world that exists;

Engineers create the world that never was"

Theodore von Kármán

Acknowledgments

This project could not have been made possible without the help of a large group of people, from academics to friends to family.

To Tom Knight, thank you for the interesting idea of building a web application that makes engineering biology simpler, following your view of how working with biology should be, easier.

To Drew Endy and the lab members at MIT, for their support and patience in teaching me all the wonderful things about synthetic biology. Special thanks to Reshma Shetty, Jason Kelly and Barry Canton for their personal views and support.

To Maria Emília Costa for accepting to advise and support me during my dissertation work. Thank you for your patience and time.

To my computer-savvy friends that provided me with their time during this programming journey, thank you Cláudio Gamboa, Tiago Rodrigues and Bill Flanagan. Your help was much appreciated.

To my Mother which has always pushed me forward and supported me with all my decisions during all these years, a special and loving thank you.

And finally, none of this would be possible without the love and support of my beautiful wife, Bárbara, Thank you!

Abstract

The large number of bioinformatics tools currently available for the analysis of genetic sequences and the technical features that each tool presents make the biological engineer's job simultaneously easier and more complex.

The emerging field of synthetic biology is seen as a new approach to engineering biology, where foundational technologies and concepts generally applied to established fields of engineering, such as electrical or software engineering, are implemented in biology to enable the development of methodologies and standards to make engineering biology a predictable, reproducible and efficient task.

Focusing on this new field of synthetic biology, the objective of this thesis was to develop a compound bioinformatics web application to assist in the conception and preparation of standardized biological parts, essential to the progress of this novel field.

Different assembly standards for biological parts, developed and implemented by the community of biological engineers and researchers working in the field of synthetic biology, were analyzed and incorporated into the web application.

The main programming language used for the development of this project was Python, with special usage of the biological computation code library provided by the BioPython project. Within the objective of producing a simple and unique web application, a selection of tools were integrated programmatically so as to generate a comprehensive analysis report of features based on a single input - a DNA or RNA sequence. This automated procedure made for a streamlined user interface with minimal learning requirements or user interaction.

The provided results of the analysis performed by the web application are presented in the form of a report with information regarding the input sequence. These results include information such as for example, the identification of specific restriction sites, thermodynamic stability and secondary structure.

These easily generated results provide biological engineers with information that may allow decisions to be made regarding the eligibility of the initial input sequence to be refined into a standard biological part, which in turn are elementary components in the development of synthetic biological devices or systems.

Key words:

Synthetic Biology; Biobricks; Standards; Biological engineering; Bioinformatics; Python; Web application;

Resumo

O elevado número de ferramentas bioinformáticas disponíveis para a análise de sequências genéticas e as características técnicas de cada uma dessas ferramentas tornam o trabalho do engenheiro biológico simultaneamente facilitado e complexo.

A biologia sintética surge como uma nova forma de abordagem à engenharia biológica, onde a implementação de regras aplicadas a outras áreas de engenharia, como a engenharia electrónica ou informática, permitem desenvolver metodologias e *standards* para tornar a engenharia biologica mais previsivel, reprodutível e eficiente.

Com especial foco na nova área da biologia sintética, este trabalho teve como objectivo o desenvolvimento de uma aplicação bioinformática, online e composta, para assistir na concepção e preparação de componentes biológicos standardizados, essenciais para o progresso de trabalhos nessa mesma área.

Foram analisados os diferentes *standards* de montagem criados e implementados pela comunidade de engenheiros biológicos e investigadores a trabalhar na área da biologia sintética, para a sua incorporação numa ferramenta bioinformática online.

A linguagem de programação principal usada no desenvolvimento deste trabalho foi Python, com recurso a bibliotecas de código especializadas para a computação biológica integradas no projecto BioPython. Diversas ferramentas foram incorporadas de forma a criar uma única aplicação *web* onde se pode obter um conjunto de informações relativos a um único *input* – uma sequência de DNA ou RNA. Este automatismo tornou a ferramenta extremamente fácil de utilizar sem necessidades especiais de aprendizagem por parte do utilizador.

O resultado da análise desta ferramenta bioinformática online é um relatório com informações relativas à sequência fornecida pelo utilizador, tais como a existência de determinadas zonas de restriçao enzimática, estabilidade termodinâmica e estrutura secundária. Estes resultados, obtidos de uma forma simples, fornecem ao engenheiro biológico um conjunto de dados que lhe permitem decidir se está perante uma sequência de interesse, apropriada à preparação de um componente biológico standardizado. Componente esse que é um elemento chave na construção de novos componentes ou sistemas sintéticos.

Palavras chave:

Biologia sintética; Biobricks; *Standards*; Engenharia biológica; Bioinformática; Python; Aplicação *web***.**

Index

Figures

Tables

Glossary

API - Application Programming Interface

BBF – Biobrick Foundation

Biobrick – Standardized biological part

BLAST – Basic Local Alignment Search Tool

DNA – Deoxyribonucleic acid

EMBOSS – European Molecular Biology Open Software Suite

iGEM – International Competition of Genetically Engineered Machines

MIT – Massachusetts Institute of Technology

MVC – Model-View-Controller

MTV – Model-Template-View

Parts Registry – The Registry of Standard Biological Parts

RFC – Request For Comments

RNA – Ribonucleic acid

UNAFold – Unified Nucleic Acid Folding software package

XML – Extensible Markup Language

Objectives

The objective of this work is to build an online, proof-of-concept compound bioinformatics tool that will assist in the creation of standard biological parts, essential components for biological engineers working in the field of synthetic biology. This tool will assist in producing standard compliant parts by performing a selection of sequence analysis tasks.

The proposed web application will provide a straightforward easy-to-use user interface with minimal user interaction and produce an analysis results report web page with information regarding nucleotide sequence statistics, restriction analysis focusing on biological parts standards, secondary structure prediction with graphical representation, and the comparison results between the submitted sequence and a database of standard biological parts from the Registry of Standard Biological Parts.

1 Introduction

1.1 Biological Background

Engineering in its most formal definition, provided by the Engineers' Council for Professional Development (Anonymous 1941), in the United States of America, is stated as the application of "scientific principles to design or develop structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to construct or operate the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation and safety to life and property."

The efficient development and replicability of such structures, machines or processes, is highly dependent on core concepts such as standardization, decoupling and abstraction, without which engineering would be technologically hindered.

The implementation of standards has enabled different fields of science and engineering to progress at a faster and more efficient pace. An example of this was demonstrated by the establishment of standards for screw threads by William Sellers, at the Franklin Institute (Sellers et al. 1864), which provided rapid progress in fields such as mechanical engineering during the American industrial revolution.

Separating a complicated problem into smaller and less complicated ones, that can be worked on independently in a way which can later be combined as a functional whole, is what is referred to as decoupling. This technology enables the separation of tasks for example where a building project can be worked on independently by an architect, an engineer, a constructor, etc. The combined expertise come together to produce a whole, and hopefully functional, project.

Abstraction comes into play as a powerful technology for managing complexity. The ability to separate abstract hierarchies in complex systems is extremely powerful and therefore allows development to be done on different levels without having to fully understand every level.

A simple example of abstraction can be seen in software engineering where lower and higher level programming languages exist. One does not need to know binary code to write a computer program in a high level programming language.

The development and application of these foundational technologies have provided various fields of science and engineering with the means to progress at great pace. The implementations of these technologies are now being seen to enable routine design and construction in synthetic biology, a novel approach to engineering biology (Endy 2005).

1.1.1 Engineering in biology

Biological systems are in most cases complex systems where the orchestration of components such as cells, genes or proteins have been subjects of research over the last century. The inherent complexity brings upon various challenges that limit the ability to engineer biology such as the inability to avoid or manage said biological complexity, the laborious and unreliable construction and characterization of synthetic biological systems, the spontaneous physical variation of biological system behavior, and of course, evolution (Endy 2005).

By applying existing standards for engineering from well established fields, such as software, electrical, mechanical and civil engineering, these aforementioned challenges in biology can be, to some extent, managed and characterized.

1.1.2 Synthetic Biology

Since the discovery of restriction enzymes in 1970, further enabling the manipulation of genetic material, there has been great progress in the field of genetic engineering. The combination of various techniques and technologies such as polymerase chain reaction (PCR) and DNA sequencing technologies has provided ways to, for example, manipulate microorganisms to produce, even at industrial levels, desired metabolites or understanding genetic pathways. These discoveries and developments have provided the necessary tools to read and write genetic material and therefore gave way to the biotechnological boom.

The emerging field of synthetic biology comes as a novel approach to biological engineering and builds upon genetic engineering by including some key technologies such as automated construction or synthesis of genetic sequences, characterization of the constructs through standards and hiding complexity via abstraction.

Consistent with the views of most researchers in the field, including The Royal Academy of Engineering, synthetic biology is currently defined as the field that “aims to design and engineer biologically based parts, novel devices and systems as well as redesigning existing, natural biological systems” (Kitney et al. 2009).

With its engineering vision, synthetic biology aims to overcome the existing fundamental inabilities in systems design and systems fabrication, by developing and implementing the already mentioned foundational technologies to enable systematic forward-engineering of biological systems for improved or novel applications (Figure 1) (Heinemann & Panke 2006).

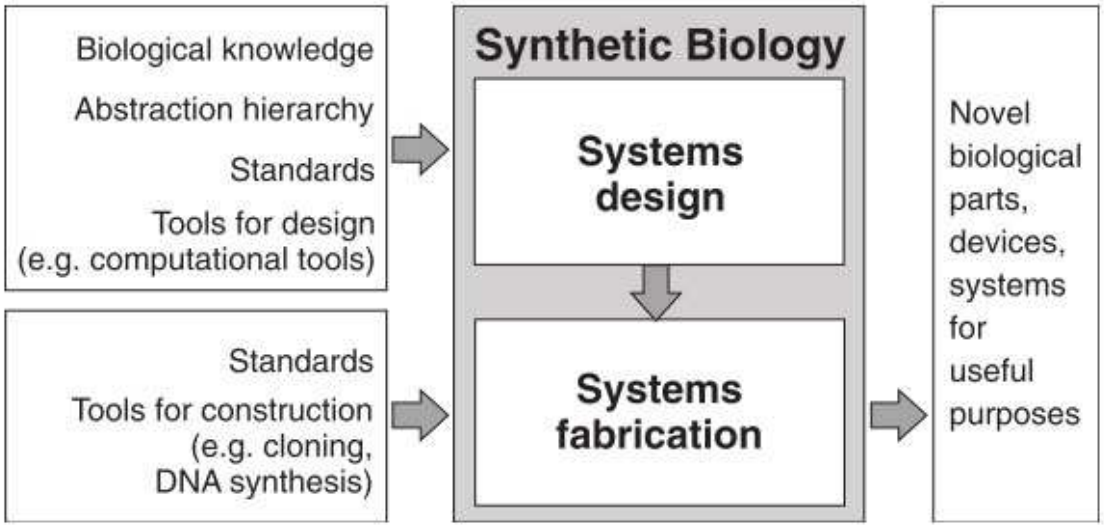


Figure 1 - Synthetic Biology encompasses systems design and fabrication (Heinemann & Panke 2006)

Combining foundational engineering technologies such as *de novo* DNA synthesis, advancements in computational and systems biology, standardization and abstraction, has produced various achievements in the new field of synthetic biology. Among some of the already noteworthy applications are projects in such diverse areas as the design of artificial gene networks (Sprinzak & Elowitz 2005), the refactoring of small genomes (Chan et al. 2005), artificial mammalian oscillators (Tigges et al. 2009) and problem solving with a bacterial computer (Baumgardner et al. 2009).

1.1.3 Biobricks, iGEM and the Parts Registry

The introduction of standardization in assembly techniques for DNA sequences allow DNA assembly reactions to avoid becoming themselves experiments but rather characterized tools for addressing a defined research topic. By replacing the current experimental approach to genetic engineering that is both time consuming and *ad hoc* in nature, with a set of standard, reproducible and reliable engineering mechanisms, it is foreseen that some of the engineering challenges in biology can be overcome and progress made at a more efficient rate.

1.1.4 Biobricks

The implementation of standards in biological engineering, analogous to those established by William Sellers for screw threads for mechanical design in the late 19th century, are expected to enable the interchanging of parts, the assembly of components into systems with predictable behavior, the ability to rely on previously manufactured components and outsourcing the assembly of components to others.

A biological part is defined as a natural nucleic acid sequence that encodes a definable biological function. However, a *standard* biological part is defined as a biological part that has been refined so as to comply with one or more defined technical standards.

Despite previous attempts to implement technical standards in biological parts, it was in 2003 that Thomas Knight, at the Massachusetts Institute of Technology (MIT), proposed the Biobrick standard for physical composition of biological parts (Knight et al. 2003). Parts that conform to these technical standards are called Biobrick standard biological parts.

Contrary to the previous attempts to implement standards in biological parts, Knight's Biobrick standard has been used by multiple groups around the world and adoption of said standard is steadily growing.

The Biobrick standard proposed by Knight comprises the notion that the transformations performed on parts during the DNA sequence assembly reactions are idempotent in terms of structure. This means that each reaction leaves the key structural elements of the component the same. Therefore the output of such transformation is a component that can be used as input in any further manipulations (Figure 3). These components are routinely mentioned as biobricks.

Following Knight's sequence standard, a Biobrick component consists of a circular vector of double stranded DNA containing the component DNA sequence, flanked on the upstream end by EcoRI and XbaI restriction sites, and on the downstream

end by SpeI and PstI restriction sites. Also, the component vector must not contain any of the aforementioned restriction sites.

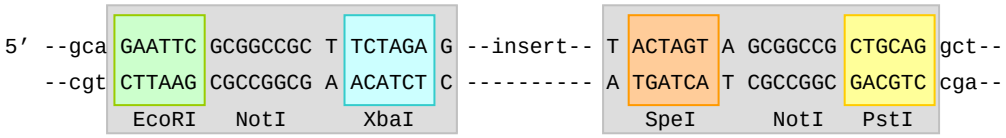


Figure 2 - A typical component vector consists of a sequence of the following form. Upstream flanking EcoRI and XbaI and downstream flanking SpeI and PstI restriction sites. (Knight et al. 2003)

The bases between restriction sites were carefully chosen to eliminate the accidental generation of specific methylation sites which could prevent enzyme cutting within determined strains.

Each complying vector component can be cut in four distinct ways yielding four distinct fragments. By cutting with EcoRI and SpeI, a front insert (FI) is obtained. Cutting with XbaI and PstI creates a back insert (BI). With EcoRI and XbaI, a front vector (FV) is created and finally by cutting with SpeI and PstI creates a back vector (BV) (Knight et al. 2003).

Since XbaI and SpeI recognition sequences have compatible overhangs, it is possible to ligate back inserts with back vectors to add components to the back of existing constructs. The same applies with front inserts and front vectors as regards to adding components to the front of existing constructs. This ligation process results in a mixed SpeI/XbaI site, a scar, that is not recognized by either of the restriction enzymes and can no longer be cut.

The resulting construct is identical in form to the standard components from which it was made. In other words, the resulting construct is flanked by the exact same restriction sites as those flanking the initial “parent” components. This recursive behavior makes it physically possible to “rinse-and-repeat” the process with the newly created construct with any other parts or devices that follow the biobrick standard to form ever more complex constructs or systems.

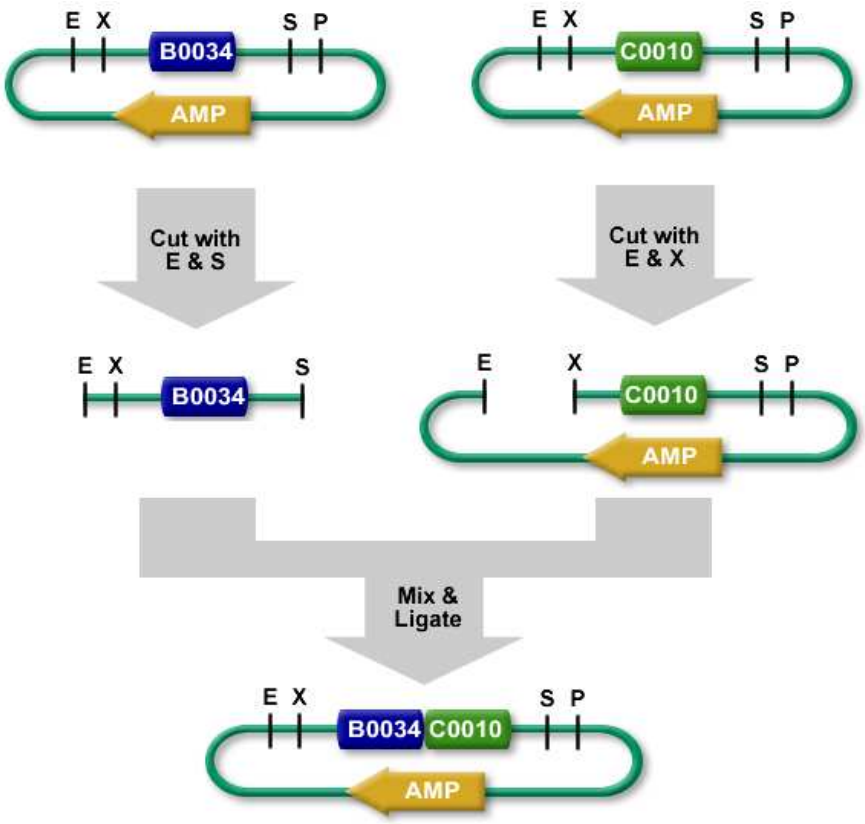


Figure 3- Standard Biobricks Assembly illustrated with the assembly of a "blue" biobrick with a "green" biobrick into a "blue-green" system biobrick (Rettberg 2009a)

One immediate advantage introduced by the biobrick standard is the fact that once a component is constructed, it can then be characterized and stored in a library or registry for use by others that also work according to the same standards.

1.1.5 Standards, plural

There is an anecdote among engineers that goes something like this: "The nice thing about standards is there are so many from which to choose". Many fields of engineering have various concurrent standards. Examples of this are the various co-existing standards for wireless access points and their subtypes in electrical and electronics engineering.

Similarly, there are various standards presented for standard biological parts. These various standards are put together and presented by the BioBricks Foundation, a not-for-profit organization founded by engineers and scientists from MIT, Harvard, and UCSF with significant experience in both non-profit and commercial biotechnology research, that is dedicated to promoting and protecting the open development, sharing, and reuse of BioBrick standard biological parts

Taking inspiration from the Internet Engineering Task Force, which devised and implemented the standard protocols that make the internet what it is today, the BioBricks Foundation has proceeded to implement a Request for Comments process. In other words, short structured documents interestingly named Request for Comments (RFC), are made available with defined standards and open to review and commentary by the BioBricks community.

The list of RFCs put forward by the BioBrick community has been growing with new standards being proposed for different features and techniques relevant to research and work within synthetic biology.

The previously described Biobrick standard for physical composition of biological parts proposed by Knight is described in RFC10 (see 7.2). It is currently the most used and widespread standard. However, other RFCs are available with alternate standards for physical composition and assembly that address specific issues such as fusion proteins.

These other main RFCs regarding physical composition and assembly are RFC23 (see 7.4) and RFC25 (see 7.5). They are extensions to RFC10 and therefore are all compatible with parts that abide by RFC10 standards.

The majority of parts available in the Registry have indications as respects to their compatibility with each of these standards which in many cases are compatible with all four.

1.1.6 International Genetics Engineering Machines

A very good example of the use of biobricks is provided by the yearly International Genetic Engineering Machines (iGEM) competition (Rettberg 2009b), that has been taking place at MIT since 2004, where groups of students from all over the world, work together in teams to design and build biological systems and operate them in living cells based on parts obtained from the registry. New parts produced during the competition are later added to a central registry after characterization and compliance with the biobrick standard, thus providing more parts for others to use in future constructs. This registry is notably called the Registry of Standard Biological Parts (Parts Registry) and is still currently hosted and maintained at MIT.

Projects presented during recent iGEM competitions range from banana and wintergreen scented *E. coli*, to an arsenic biosensor, to a *H. pylori* vaccine.

1.1.7 Registry of Standard Biological Parts

Since its inception, the registry of standard biological parts has grown at a rapid pace and currently contains over 3200 cataloged parts (Figure 4). These parts are available, not only to the iGEM teams, but also academic research facilities or labs that are interested in building synthetic biological systems, and also interested in contributing with new parts or devices to the registry.

Making standard biological parts is crucial to the development and wide establishment of synthetic biology. By creating new parts, the pool of available parts in the Registry for reuse by the original part creator or by any other bioengineer looking to work with a similar biobrick in their constructs grows.

Currently, the Registry is seeing growth of its catalog of parts by a factor of ~2 per year. The enormous number of combinations between standard biological parts and devices that are available in the catalog demonstrates the great potential provided for engineering ever more complex devices and systems (Peccoud et al. 2008).

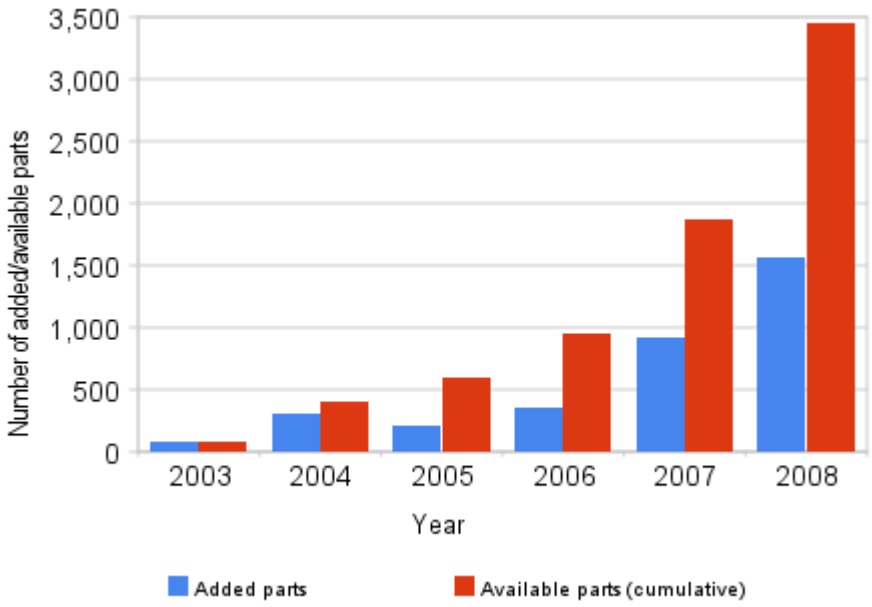


Figure 4- Number of parts added per year and available in the Registry of Standard Biological Parts. (PartsRegistry.org, 2009)

Abstraction levels are an important part of synthetic biology as an approach to engineering biology. As such, the terms: part, device and system, refer to standard levels of hidden complexity at which synthetic biologists work (Figure 5).

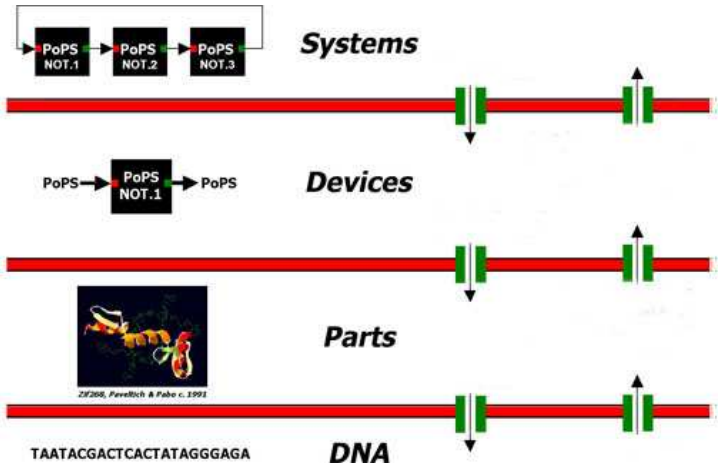


Figure 5 - Abstraction Hierarchy in Synthetic Biology. Abstraction barriers (red) block all exchane of information between abstraction levels. Interfaces (green) enable the limited and principled exchange of information between levels. (Endy 2005)

Working at any of these levels of abstract hierarchy should not require one to understand all the intricacies of each of the other levels. Moreover, there must exist a form of exchanging information between levels, therefore providing each level with sufficient information to interact.

Parts and devices are available in the registry and allow biological engineers to combine these parts or devices into more complex systems without having to start from scratch. In other words, a researcher looking to produce a multi-component system does not have to spend a large part of his or her time refining the biological parts from DNA.

By (re-)using standard compliant, characterized components from the registry, building multi-component biological systems proves to be faster, more efficient and above all, reproducible (Peccoud et al. 2008).

1.1.8 Characterization

Most parts and devices present in the registry have their sequences verified for quality control and, in some cases, are also accompanied by qualitative and quantitative data in the form of a data sheet. This information varies with the type of component but tends to include information on various characteristics such as composition, mechanism, function, specificity, compatibility, stability, among others.

There are various highly characterized components available in the registry of standard biological parts. One such device is a genetically encoded receiver neatly labeled: Bba_F2620 (Canton et al. 2008). This composite device was constructed by using five other standard biological parts. BBa_F2620 is a featured device within the registry due to the level at which it has been characterized (see 7.1).

The characterization of BioBricks is important to enable the reuse of parts by independent researchers across many laboratories. Although Biobricks are standardized in a manner that allows parts to be physically assembled into multi-component systems, it

is important to have standardized tools, techniques and units of measurement that will facilitate the proper characterization of said parts.

1.1.9 Understanding by building

As mentioned in previous chapters, engineering depends upon key concepts such as standardization, abstraction and decoupling, among others. Without these key technologies, engineering ceases to exist by definition and we are left with *ad hoc* experimental work that is in most cases tedious and more than often difficult to reproduce with precision.

By adopting core engineering principles, synthetic biology presents itself as a strong approach to engineering biology. Furthermore, building biological systems from refined standard biological parts becomes an interesting means to understanding the intricate complexity within natural biological systems (Chan et al. 2005).

Although the implementation of engineering principles may enable us to build synthetic biological systems, it is only possible because there are hard working scientists dedicating their lives to understanding how natural biological parts and systems work.

An important field of research that dedicates itself to understanding biology at a higher systems level is promptly named Systems Biology. Where systems biology excels in understanding how natural biological systems work, synthetic biology takes that knowledge and attempts to build artificial or synthetic biological systems to achieve a desired result.

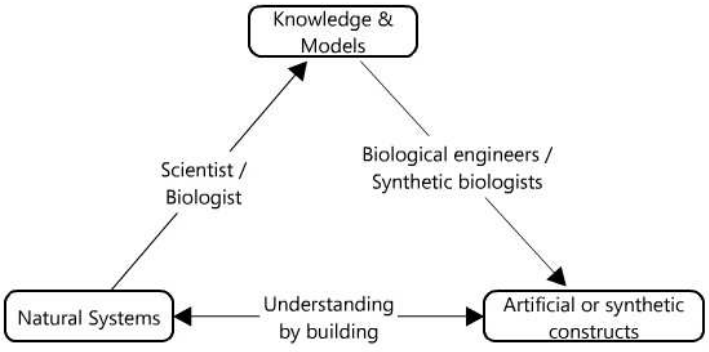


Figure 6 - Understanding natural systems through synthetic biology

1.1.10 Building bricks

In an emerging field of engineering such as synthetic biology, the initial steps are of utmost importance and will play a crucial part in the establishment of this new approach to biological engineering.

The building of multi-component systems is made easier if said components are available, standardized and above all characterized. To make these essential initial biological parts, it is required that they abide to a defined standard, in most cases the BBF RFC10 (see 7.2). The now *standard* biological part should be properly characterized to understand its behavior via defined techniques and measurements.

1.1.11 Before the brick comes the standard

Without the refinement of natural occurring biological parts into standard parts, such as the aforementioned standard Biobricks, the engineering of multi-component devices with expected behavior and precise reproducibility is highly unlikely.

By building new parts and making them available through the Parts Registry, the assembly of multi-component biological devices becomes an engineering task only possible because the majority of the underlying research and testing has already been performed at the parts level, making it possible to design and build systems of higher complexity based on tested and standard compliant components.

Creating standard biological parts is irrefutably one of the most important contributions to be made in order to empower the engineering of novel devices or systems via synthetic biology.

The production of standard biological parts can be referred by, in general terms, the refinement of natural biological parts in conformity with a defined set of standards. This refinement encompasses the selection of a gene of interest, analysis of the genetic sequence for specific characteristics, preparation of said genetic sequence with appropriate upstream prefix and downstream suffix so as to comply with a desired biological standard for physical, and hopefully functional, composability.

Finding the sequence of choice, performing the sequence analysis and the determination of standard compliant pre and suffix primers is made enormously easier with the assistance of biological computational tools and databases.

1.2 Bioinformatics: benefits and obstacles

The goal of bioinformatics is to better understand the living cell and how it functions at the molecular level. By analyzing raw molecular sequence and structural data, bioinformatics research can generate new insights and provide a "global" perspective of the cell.

Currently there exist a large number of bioinformatics tools that allow researchers to analyze specific features of their biological data. If you are analyzing a DNA sequence, you can retrieve such information as the GC content, repeats in your sequence, open reading frames, among many other features. The same happens when analyzing amino acid sequences, with a large number of tools available to retrieve different protein specific data.

By combining a group of tools focused on specific features, such as those pertaining to synthetic biology and the production of standard biological parts, it would be of great utility to biological engineers to have the ability of querying a system that would not only retrieve the data directly related to the sequence but also suggested information of interest supplied by the system in an automated way and further annotate their findings.

Automated discovery related to a given sequence can provide biological engineers with insight into details that would not be easily found without a thorough analysis with a spectrum of different bioinformatics tools.

Access to biological information databases and the use of computational tools is invaluable to anybody attempting to build new standard biological parts or devices. These tools make it possible for biological engineers to analyze desired genetic sequences for specific characteristics that may be of interest to their work.

In general, biological computational tools focus on a specific set of functions and features. The choice of tools is usually related to what the researcher is looking to retrieve

or analyze. May it be DNA, RNA, proteins or any other biological component, there are specialized computational tools for each and every one.

There are many great tools to help researchers obtain and analyze their biological data. The wide range of tools available can be considered beneficial but also hindering since the researcher does not only need to properly interpret the data they obtain via the analysis of their biological target but also is required to understand how to use each software application.

Despite the enormous benefits that bioinformatics applications provide, there are obstacles that researchers have to overcome while using them. These tools or applications can be produced in a wide variety of programming languages, can be operating system specific, require specific input file formats, produce results in proprietary file formats, etc. The benefits, however, are far greater than the obstacles.

The interoperability of bioinformatics services has also been an issue in the past and has made way for centralized web services that provide standardized ways to overcome the differences between biological data-types, databases and data-formats (Wilkinson 2002; Smedley et al. 2009; Pillai et al. 2005).

2 Computational tools for synthetic biology

Although synthetic biology is still an emerging field, the number of related computational tools currently available is quickly growing. There are tools focused on a variety of different objectives which range from tasks such as assessing physical composability to constructing genetic networks to organizing libraries of standard biological parts.

A review of available computational tools for synthetic biology in the literature brings a few exemplary applications forward that are noteworthy and are seeing great development.

2.1 Clotho

Following a platform-based design paradigm, Clotho is a promising attempt at producing an integrated and extensible toolbox. Based on a core-and-hub network system, Clotho manages multiple connections (hubs) that perform self-contained tasks while maintaining the ability to connect to any other connection via the core. The decentralized nature of this tool allows connections to be created by computational biologists which can be integrated into Clotho. These hubs can be kept locally in a lab or shared with the community of users (Densmore et al. 2009).

2.2 Tinkercell

TinkerCell, previously known as Athena, is a visual tool which enables building, simulation and analysis of genetic networks. A family tree of biological parts can be loaded from a database and can then be assembled into multi-component systems in a graphical working environment (Chandran et al. 2009).

2.3 BioJade

BioJADE is a design and simulation tool for synthetic biological systems written in Java, and makes interactive use of BioBrick Repositories such as the Registry of Standard Biological Parts. BioJADE enables system designers to specify a system abstractly, tune it, simulate its behavior using a variety of simulators, and finally package the part for use by either the designer or the public (Goler 2004).

2.4 GeneDesign

GeneDesign is a collection of algorithms written in Perl that allow users to perform several individual tasks such as random nucleotide sequence generation, reverse translation, silent site insertion or removal, restriction enzyme recognition site presence. (Richardson et al. 2006).

2.5 GeNetDes

GeNetDes is a tool to design transcriptional networks with targeted behavior that could be used to better understand the design principles of genetic circuits. GeNetDes is currently extending to design networks by assembling standardized biological part models. The models contain data obtained from part characterizations (Rodrigo et al. 2007).

2.6 SynBioSS

SynBioSS (Synthetic Biology Software Suite) is a software suite for the quantitative simulation of biochemical networks using hybrid stochastic algorithms.

SynBioSS provides a graphic user interface that enables biologists to perform chemical network reaction simulations without the need to understand the sophisticated underlying algorithms. SynBioSS can accurately simulate any system modeled as a network of reactions. The software suite handles input data, runs the simulations and

vividly visualizes simulation results, without requiring any programming background from the user (Hill et al. 2008).

Table 1 - Computational tools in synthetic biology

Software tool	**Description**	**References**	**Website**
BioJade	A design and simulation tool for synthetic biology	(Goler 2004)	http://web.mit.edu/jagoler/www/biojade/
Clotho	Synthetic biology design environment	(Densmore et al. 2009)	http://biocad-server.eecs.berkeley.edu/wiki/index.php/Clotho_Development
GeneDesign	A suite of algorithms for the design of synthetic genes	(Richardson et al. 2006)	http://www.genedesign.org/
GeNetDes	A tool to design transcriptional networks with targeted behavior	(Rodrigo et al. 2007)	http://soft.synth-bio.org/genetdes.html
SynBioSS	A software suite for the quantitative simulation of biochemical networks using hybrid stochastic algorithms	(Hill et al. 2008)	http://synbioss.sourceforge.net/
Tinkercell	A tool for building, simulating and analyzing genetic circuits	(Chandran et al. 2009)	http://www.tinkercell.com/

3 Material and Methods

The following sections enumerate the web application **development technologies** and their subsequent **implementations** for the development of the proposed bioinformatics web application.

The first section gives insight into the programming languages, applications and frameworks used during the development this project. Utilized technologies are each specified and followed by a detailed description regarding their function and the motive for selection.

The second section describes the implementations of the previously mentioned technologies used in the developing of the proposed web application. The internal application work-flow is described sequentially from the initial input sequence to the finalized web page output and available files.

3.1 Development technologies

3.1.1 Python

The main programming language used to develop this project was Python.

Python is a very high-level, interpreted, object-oriented and extensible programming language. It is free to use and has demonstrated enormous stability and continuous development by a large and active community of contributing open source developers (www.python.org).

Despite the existence of various other programming languages with similar characteristics as those just described, Python was selected as the main programming language for this project for a few other reasons. Among such reasons are the versatility as a scripting language for web applications, the availability for all major operating systems, the proven performance and, especially for this project, the existence of a freely available library of tools and applications for biological computation aptly named BioPython.

3.1.2 BioPython

The BioPython project is an international open source collaboration of developers providing Python libraries for bioinformatics. The libraries include a wide number of modules that perform specific tasks such as reading sequence files in various formats, interacting with tools such as BLAST, ClustalW and EMBOSS, interacting with online databases (Cock et al. 2009).

Using BioPython played an elemental part in the development of this project since it allowed for the rapid development of specific features with recourse to the built-in modules.

3.1.2.1 BioPython Modules

- **Bio.Seq**
    - Biological sequences are arguably the central object in bioinformatics and therefore a mechanism for dealing with sequences is essential. `Bio.Seq` provides the `Seq` object that combines the biological sequence as string with an alphabet attribute that defines what type of sequence currently being manipulated. The `Seq` object supports biologically relevant methods such as complement, reverse complement, transcribe and translate that perform the analogous biological transformations to the working genetic sequence.

- **Bio.SeqUtils**
    - This module provides a miscellaneous group of functions for working with biological sequences such as GC content percentage calculation, sequence molecular weight determination or subsequence search capabilites. Included in `Bio.SeqUtils` are also submodules for working with protein sequences and melting temperature calculations, among others.

- **Bio.Restriction**
  - Restriction enzymes play a very important role in genetic engineering. `Bio.Restriction` provides the tools to perform restriction analysis on biological sequences. This package includes facilities provided by Rebase and contains information for over 600 restriction enzymes. Given the importance of restriction enzymes in the assembly of standard biological parts, this module provided numerous tools which facilitated the restriction analysis of the input sequence.

- **Bio.Alphabet.IUPAC**
  - Biological sequences are comprised of nucleotides or proteins (amino acids). The `Bio.Alphabet.IUPAC` module provides the means to identify the constituting elements within a sequence according to IUPAC defined nucleotide or protein alphabets.

- **Bio.NCBIXML**
  - The `Bio.NCBIXML` module included in BioPython enables the XML output of a BLAST to be parsed. BLAST enables the comparison of biological sequences with an appropriately formated database and the results of said comparison are stored in a XML file or object, which can then be parsed using this module.

3.1.3 BLAST

The Basic Local Alignment Search Tool, or BLAST, is an algorithm for comparing primary biological sequence information. BLAST enables researchers to compare a query sequence with a database or library of sequences and identify similarities between them within a specified threshold.

Comparison is just one of the usage purposes for BLAST. Other usage purposes include identification of species, establishing phylogeny, DNA mapping and locating domains.

BLAST was developed at the US National Institute of Health (NIH) and is currently freely available to the public as a standalone command-line program or as a web-based tool at the National Center for Biotechnology Information (NCBI) (Altschul et al. 1990). There are various versions of BLAST that compare different biological databases with respect to specific input query and biological sequence of interest, be it protein or DNA.

A standalone command-line version of BLAST was programmatically incorporated into the web application so as to enable the comparison of the input sequence with a local biological database of sequences provided from the Parts Registry.

3.1.4 UNAFold

UNAFold, derived from "Unified Nucleic Acid Folding", is an integrated collection of programs that simulate folding, hybridization, and melting pathways for one or two single-stranded nucleic acid sequences (Markham & Zuker 2008). Folding prediction for single-stranded RNA or DNA combines free energy minimization, partition function calculations and stochastic sampling. The software package works on all major operating systems and given the fact that it is "command line" driven, it was programmatically integrated into the Python code developed for the web application.

One specific feature of interest provided by the UNAFold software package, through the build-in `mfold_utils`, was the ability to produce images of secondary structures based on the predicted folding of single-stranded RNA or DNA.

3.1.5 EMBOSS

The European Molecular Biology Open Software Suite, normally referred to as EMBOSS, is a software package targeted for the molecular biology and bioinformatics community. It is comprised of a large selection of analysis tools that support a wide range of file formats and biological data (Rice 2000).

The extensive library of tools available within the EMBOSS provides itself as a platform for developers of biological analysis tools to build upon. The ease of use, the

extensive number of tools available within the suite and the built-in interaction within the BioPython package proved to be sufficient reasons to choose to work with the EMBOSS.

3.1.5.1 Backtranseq

The EMBOSS includes a wide range of applications for biological sequence analysis and manipulation. `backtranseq` is a particular application that takes a protein sequence of amino acids and makes a best estimate of the likely nucleic acid sequence that could have generated the initially provided amino acid sequence. This estimation is based on codon frequency tables for determined species, thus selecting the most frequent occuring codon in that species when constructing the nucleic acid sequence, thereby providing a programmatic way of performing an estimated codon-optimized reverse translation.

3.1.6 Django

Django is a high-level Python web framework that follows the model-view-controller (MVC) architectural pattern.(Holovaty & Kaplan-Moss 2007) This pattern is mentioned within Django documentation as a model-template-views (MTV) pattern but is based on the same principles of decoupling between programming logic and presentation.

The large community of users, extensive documentation, rapid deployment and emphasis on reusability proved Django as good choice to bring this bioinformatics project to web application status.

3.1.7 Javascript

Javascript is a scripting language used to enable programmatic access to objects within a client application, in this case the web browser and the proposed web application. Moreover, enabling dynamic interaction with the content presented in the web browser, Javascript provided the means with which specific strings within the

presented biological sequence could be highlighted, thus providing a visual indication of the location of restriction sites through a toggle switch per restriction enzyme selected.

3.1.8 Google Charts API

The Google Charts API (application programming interface) is a web service provided by Google that enables programmers to dynamically generate charts of various types (http://code.google.com/apis/chart/).

3.2 Implementations

The bioinformatics web application development was enabled by the implementation of specific methods and technologies already mentioned in the previous section. The programmatic work-flow that follows elaborates on the steps and methods implemented to achieve the working proof-of-concept web application.

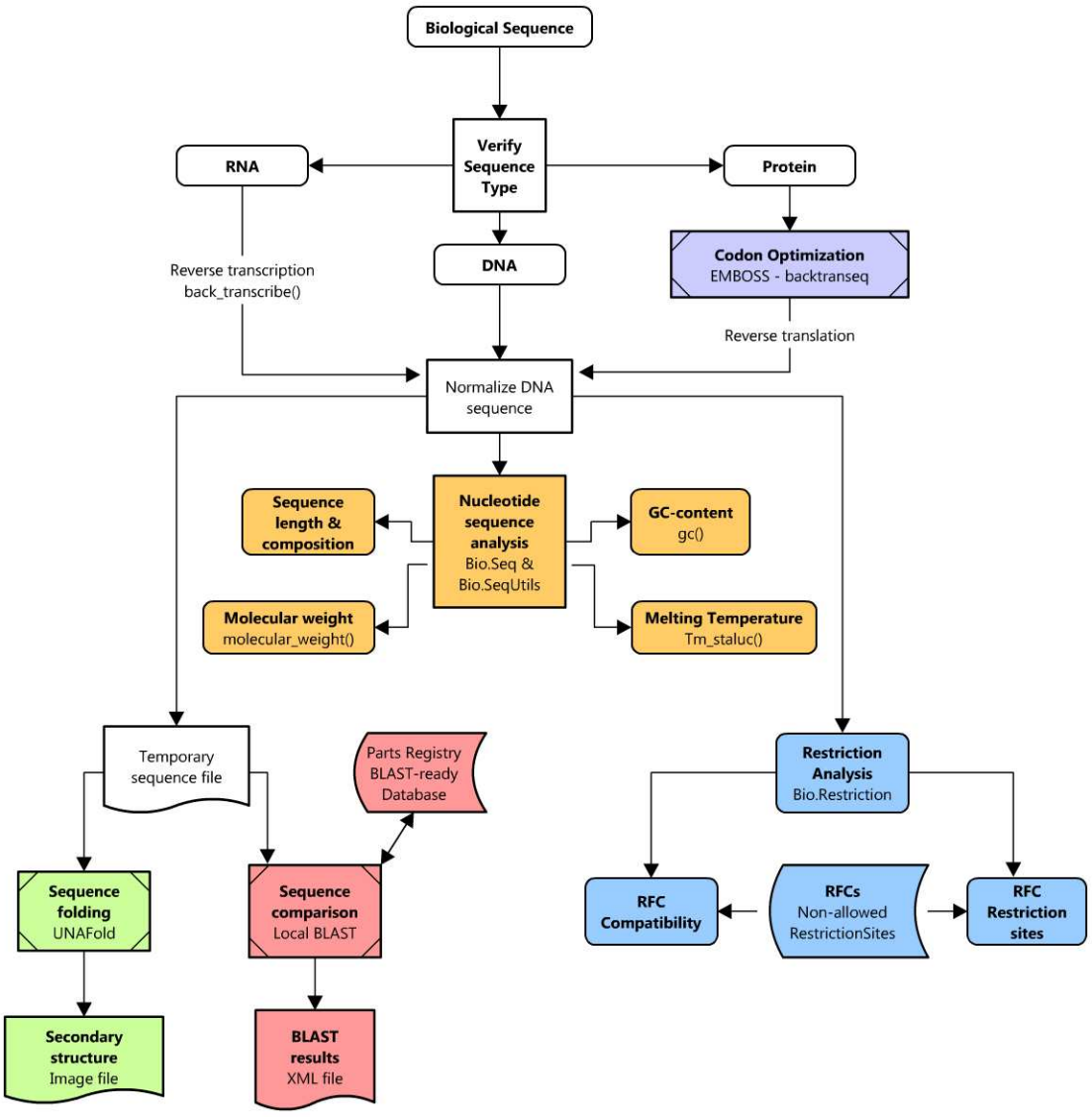


Figure 7 – Overview of the web application work-flow. Colors represent the different implementation steps: Grey: Reverse translation with EMBOSS; Orange: Nucleotide sequence analysis; Blue: Restriction analysis; Magenta: Sequence comparison with BLAST; Green: Sequence folding with UNAFold;

3.2.1 Biological sequence input

As previously mentioned, biological sequences are arguably the central object in bioinformatics. It is no different in this case. The input biological sequence is the working material for the entire web application work-flow. All the results presented are estimated, calculated and displayed based on the original input sequence.

To keep the web application as simplified as possible, a form is presented to the user with a simple text area to introduce the biological sequence of interest along with a submit button. The only case in which there are extra options to be selected is in the case where the sequence being submitted is a protein sequence.

Once submitted, the biological sequence goes through various steps that involve sequence analysis, calculations, comparison, and estimation. These steps are here described.

3.2.2 Sequence normalization

Biological sequences can be retrieved from a wide variety of locations and in various formats. Therefore, the initial step implemented in the work-flow was the normalization of the input biological sequence to a gap-less and fully capitalized sequence of DNA nucleotides (ACGT).

The currently allowed input must be an unambiguous DNA, RNA or amino acid sequence. This input is first verified for it's content and identified as DNA, RNA or protein. Once the sequence type is identified, any necessary functions are called to produce the analogous DNA sequence.

In other words, if the submitted biological sequence is DNA, it is properly formated to uppercase with numbers and gaps removed. If the provided sequence is RNA, the sequence is first reverse transcribed to DNA and then formatted like-wise.

In the case where the input is a protein sequence of amino acids, an estimated reverse translation to DNA is provided based on codon frequency tables for specific species selected by the user (Human, *E. coli* or *S. cerevisiae*). This reverse translation is

an estimated codon optimized DNA sequence for a target species produced using the EMBOSS package and formatted as mentioned for the DNA and RNA input cases.

3.2.3 Temporary sequence file

As already described, the initial input biological sequence is converted to the analogous DNA sequence, independently of the original input having been DNA, RNA or protein. This converted and normalized sequence is used in a group of functions throughout the work-flow of the proposed web application but is also stored in a file that will serve as an input file for a few tools that require so. This file is generated and stored with a random filename and is kept on the system in a temporary location.

3.2.4 Nucleotide sequence analysis

Once obtained, the normalized DNA sequence is analyzed using defined BioPython modules and submodules.

Complement and reverse complement sequences

Among the first results generated from the sequence, are the complement and reverse complement sequences provided by the `complement()` and `reverse_complement()` functions built into the `Bio.Seq` module. These sequences will be used at later stages for tasks such as forward and reverse primer preparation.

GC Content and other nucleotide information

Information based on the sequence and length of the nucleotide string is calculated. The length of the sequence is determined and also the number and total percentage of each nucleotide species is obtained via simple arithmetic:

$$\frac{A}{A+C+G+T}\times 100 \qquad \textit{Equation 1}$$

A useful indicator regarding the thermostability of a double stranded DNA sequence is provided by the GC-content. The percentage of nitrogenous bases in a DNA molecule which are either guanine (G) or citosine (C) determines the GC-content.

The GC pair is bound by three hydrogen bonds, one more than those that binding the AT pair. Therefore, DNA that contains a higher percentage of GC-content is generally considered to be more stable than DNA with low GC-content.

As already mentioned, GC-content is generally represented as a percentage and can be calculated as:

$$\frac{G+C}{A+C+G+T}\times 100 \qquad \textit{Equation 2}$$

Despite the simplicity of the arithmetic, the `Bio.SeqUtils` module contains a built-in `gc()` function which was used to obtain the desired statistic.

Melting temperature

Melting temperature (T_m) is defined as the temperature at which half of the DNA strands are in the double-helical state and half are in the "random-coil" states. The melting temperature depends on the length of the sequence of nucleotides and the nucleotide composition.

Using the built-in `MeltingTemp` submodule from `Bio.SeqUtils`, a function named `Tm_staluc()`enables the calculation of the thermodynamic melting temperatures of nucleotide sequences according to the nearest neighbor method (SantaLucia 1998).

This method of calculating the melting temperature takes into account the stacking energies between adjacent nucleotides along the double helix. Each combination of two adjacent bases has an enthalpic (ΔH) and entropic (ΔS) parameter. These parameters along with the concentration of the strands (single C_1 and complementary C_2) and the universal gas constant (R) come together in an equation which allows the concentration of the melting temperature (T_m) to be calculated as (SantaLucia 1998):

$$T_m = \frac{\Delta H}{\Delta S + R \ln\left(C_1 - \frac{C_2}{2}\right)} - 273{,}15°C \qquad \textit{Equation 3}$$

3.2.5 Restriction analysis

Restriction enzymes play an important role in biological engineering. The ability to manipulate genetic material with precision provides synthetic biologists with the necessary tools to create standard biological parts and even construct multi-component devices or systems.

Given the importance that standardization has in synthetic biology, a set of specific functions were created to perform restriction analysis of the input sequence.

RFC compatibility

A function to verify whether if the input sequence is compatible with the main biological part assembly standards, RFC10, RFC21, RFC23 and RFC25, was developed.

These standards contain a set of restriction enzyme sites that must be avoided in the target insert so as to enable the creation of a compliant standard biological part.

By providing the function with the list of non-allowed restriction enzymes to be aware of, the function then analyses the sequence and provides a true or false output, indicating if the target sequence is compatible to the defined RFCs, or not.

Restriction sites localization

In the case where the provided sequence does not comply with one, or more, of the RFCs provided, non-allowed restriction sites are identified.

The identification of restriction sites is performed using the BioPython module `Bio.Restriction`. This module contains a set of functions that allow a full restriction analysis to be performed based on the sequence provided and a set of restriction enzymes of interest. The non-allowed restriction enzymes defined by the four different biological part assembly standards (RFCs) where used in this case.

Table 2- List of non-allowed restriction sites (represented by the corresponding restriction enzyme) for each biological part assembly standard.

RFC	Non-allowed Restriction Enzyme sites*
RFC 10	**EcoRI, XbaI, SpeI; PstI**, PvuII, XhoI, AvrII, NheII, SapI
RFC 21	**EcoRI, BglII, BamHI, XhoI**
RFC 23	**EcoRI, XbaI, SpeI, PstI**, PvuII, XhoI, AvrII, NheII, SapI
RFC 25	**EcoRI, XbaI, NgoMIV, NgoMI**, **AgeI, SpeI, PstI**, PvuII, XhoI, AvrII, NheII, SapI

* - Restriction enzymes sites that are not-allowed are in bold. Others are not recommended.

Primers for Biobrick creation via PCR

If the provided sequence of interest is free of non-allowed restriction sites it may be suitable to be made into a standard biological part. The most used standard for producing parts is known as the Biobrick standard (RFC10).

In order for the provided sequence to become a Biobrick, an appropriately added prefix and suffix need to be applied. This can be done via PCR construction, where the forward and reverse primers follow and contain the restriction enzyme sites defined by the Biobrick standard. The forward and reverse primers are generated via a function that follows the defined specifications in Biobrick standard (RFC10) (see 7.2).

3.2.6 Local BLAST against Parts Registry

The registry of standard biological parts (Parts Registry) has seen the number of added and available standard parts grow by a factor of two per year.

The sequence information of the over 3200 available parts is currently made available via a large Fasta file that can be downloaded at the Parts Registry.

This file is downloaded to the web application server on a regular basis via a `cron job` and converted to the BLAST compatible format, thus serving as a local database of standard biological part sequence information against which a local BLAST run will attempt to compare the user provided sequence.

Using the temporarily kept sequence file previously mentioned (see 3.2.3), the local BLAST command-line tool is called to compare the sequence contained in the sequence file and the 3200 plus sequences represented in the Parts Registry. The resulting output is an XML file that is stored in the same location as the sequence file with the exact same filename. This XML file contains the complete set of BLAST results, if any.

The BioPython module `Bio.NCBIXML` enables the necessary commands to be forwarded to the local BLAST tool and also provides the tools to parse the XML results file.

The full BLAST results XML file is made available for download to the user and a selection of the top scoring results with acceptable e-value below the defined threshold are presented in the web application results interface.

3.2.7 Secondary Structure prediction with UNAFold

Secondary structures aid in predicting mRNA regulation and ribosome binding site strengths. To produce secondary structure estimations based on an input sequence, a Python function was created which uses the temporarily stored sequence file (see 3.2.3) as an input file for the UNAFold command-line tool.

A set of add-on tools called `mfold_utils` were used in conjunction with UNAFold to produce a graphical representation of the secondary structure. This resulting image file is temporarily stored in the same location as the sequence file (see 3.2.3) and the local BLAST results XML file (see 3.2.6), also with the same filename and made available for download through the proposed web application.

3.2.8 Bringing the application to the web

The implementation of the various methods and functions previously described produces data that is either stored in files or stored in memory waiting to be presented.

By using the powerful Python web framework, Django, the data can be called, manipulated once again and presented in a web browser, thus bringing the full potential of the written Python code to the web, in the form of a working web application.

Django follows the MTV (model-template-view) architectural pattern, thus separating presentation from programming logic. Therefore, all the results from the preceding functions and any necessary manipulation still required before the presentation of the results were performed in the View. The results are passed from the View to the Template, where the results are finally seen by the user in the web browser.

Taking advantage of the Javascript language, a script was created to enable highlighting of restriction sites via toggle links per restriction enzyme. This client-side script provides users with a visual cue as to the location of the restriction sites within the initially provided sequence.

4 Results and Discussion

The following sections describe the proposed web application as a practical result of the implemented development technologies. The sections are ordered in terms of their interaction and presentation to the user and discussed in regards to their function and importance.

4.1 Biological sequence input interface

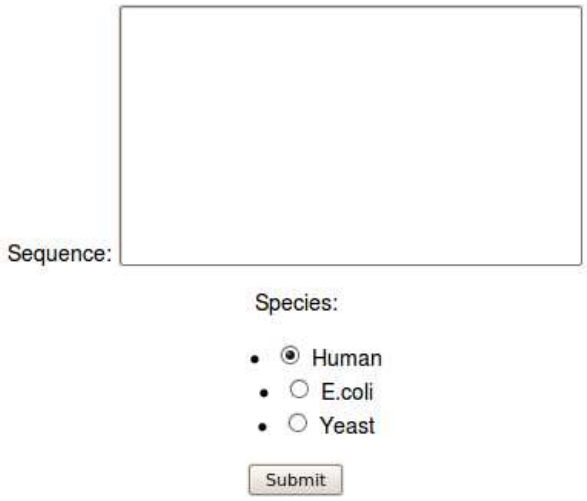


Figure 8 - Web application user interface form. Text area is for biological sequence insertion. When sequence is protein, species selection options are required to perform codon optimization.

4.1.1 Biological sequence input

Initial interaction with the web application comes through a simple form that requests the introduction of a biological sequence in the form of DNA, RNA or Protein. For each type of sequence introduced, the resulting sequence with which the web application will perform analysis and calculations will be a DNA sequence.

The input field allows biological sequences to be introduced with numbers, spaces, multiple lines or any other characters that are generally copied along with the sequence of interest from biological databases like GenBank (Benson et al. 2008). The non-

biologically relevant characters are automatically removed, the sequence is then formatted to uppercase and is verified for its type. DNA sequences remain DNA, RNA sequences are reverse transcribed.

Currently, the introduced sequences may not contain ambiguous DNA, RNA or amino acids.

In the case where the introduced sequence is a protein, the amino acid sequence is reverse translated to DNA based on one of the codon optimization options listed below the input field. These options include Human, *E. coli* or *S. cerevisiae* optimized codon sequences. The reverse translation of proteins to DNA with species optimized codon sequences is of special interest for synthetic biologists interested in expressing proteins across species.

4.2 Sequence analysis results

Once the biological sequence of interest is submitted, the resulting DNA sequence is then run through various functions that return a variety of data. The data are presented to the user in a single generated web page of results divided in sections.

4.2.1 Biological sequence and reverse complement

The initial section of the results web page displays the type of sequence submitted by the web application user before conversion to DNA, if applicable.

The DNA sequence with which the web application performs all the required analysis is presented in uppercase letters and spaced as codon triplets. The reverse complement of the sequence is similarly presented.

The sequences are displayed at the top of the results page providing the user with an overview of the submitted sequence and enabling the visualization of highlighted restriction sites when present. These highlighted sites are toggled on or off by Javascript

enabled links listed in the restriction sites and compatibility section of the results (see 4.2.3).

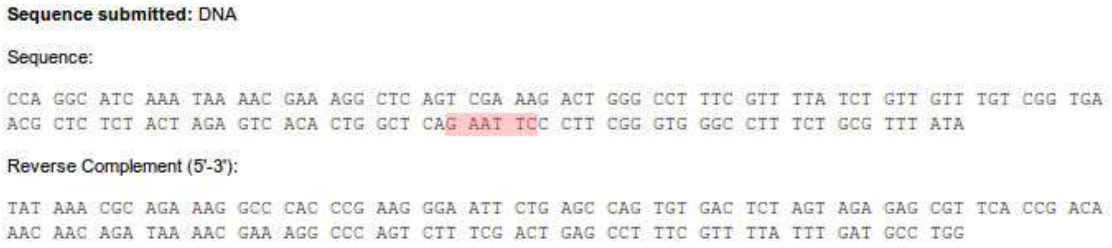


Figure 9 – Information regarding the type of sequence originally submitted to the web application and the layout of the DNA sequence and reverse complement sequence. Non-desired restriction site marked in red.

4.2.2 Nucleotide statistics

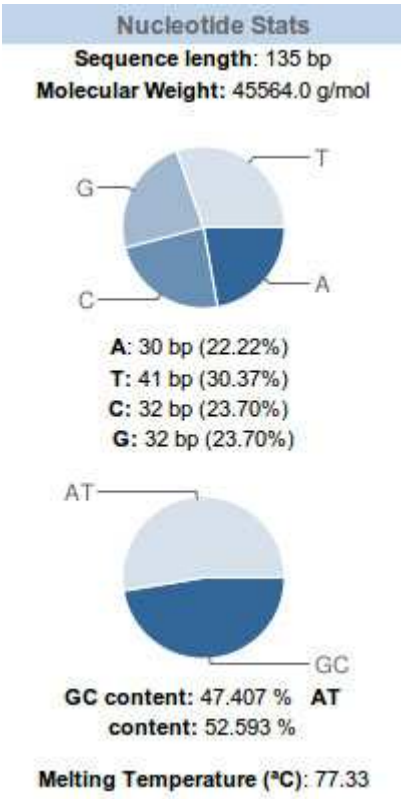


Figure 10 - Various nucleotide generated data presented in numerical form or via dynamic generated pie-charts.

Simple pie-charts are generated on-the-fly using the Google Chart API providing a visual representation of the composition and the GC-content of the submitted sequence.

Other data obtained through nucleotide analysis are presented such as the sequence length, molecular weight and melting temperature.

Data such as GC-content and melting temperature are closely related and both influence DNA strand thermostability (SantaLucia & Hicks 2004; SantaLucia 1998).

4.2.3 Restriction sites and compatibility

As already mentioned, the proposed web application is presented as a case-study in the field of synthetic biology.

The results in the restriction sites and compatibility section are of great interest to synthetic biologists and biological engineers since they provide very specific information regarding the compatibility of the provided sequence to become a standard biological part.

In the cases where the sequence is not compatible, thus not ready to become a standard biological part according to any of the analyzed standards, the non-allowed restriction sites are identified by the specific base pair location of the unwanted cut site.

Javascript enabled links provide an interactive and visual means to identify the location of a specific restriction site.

The information included in this section provides the means to determine if the sequence is eligible to become a standard biological part and conform to physical composability standards.

If so, forward and reverse primers are provided to create a BioBrick part via PCR reaction according to specifications in the Biobrick standard (RFC10).

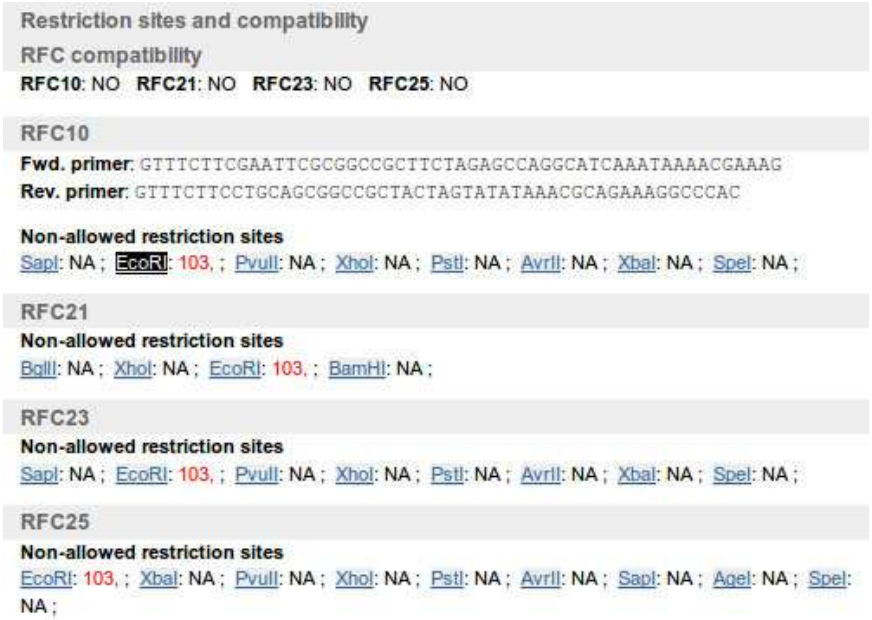


Figure 11 - Restriction sites and standards compatibility

4.2.4 Local BLAST results

The top 10 parsed results of a local BLAST run of the submitted sequence against the Parts Registry sequence database are presented as tabular data with links to the Biobrick part page at the Parts Registry official website, the provided description and the e-value obtained via the sequence comparison. Results with an e-value below a specified threshold are displayed.

The local BLAST results provide a way to verify if there already exists a Biobrick similar to the sequence currently being searched for and if so, provides more information at the distance of a simple click of a link to the Parts Registry.

Local Blast: Results (against Parts Registry)

BioBrick	Description	e-value
pSB1A7	Plasmid_Backbone "Transcriptionally insulated high copy BioBrick	1.15982e-68
pSB1A7	Plasmid_Backbone "Transcriptionally insulated high copy BioBrick	1.15982e-68
pSB1A7	Plasmid_Backbone "Transcriptionally insulated high copy BioBrick	0.00128803
pSB1A7	Plasmid_Backbone "Transcriptionally insulated high copy BioBrick	0.00128803
pSB1A10	Plasmid_Backbone "Screening Plasmid	1.15982e-68
pSB1A10	Plasmid_Backbone "Screening Plasmid	0.00128803
BBa_Z52935	Generator "protein generator for	1.15982e-68
BBa_Z52935	Generator "protein generator for	0.00128803

Figure 12 - Local BLAST results presented in a table with BioBrick part name linked to offical Parts Registry page for the part, short description of part and e-value obtained from sequence comparison

4.2.5 Secondary Structure

Unlike most of the information presented in the previous sections, the image displayed in this section provides insight into the functional composibility of the standard biological part, if it is to be made into a Biobrick.

Secondary structures aid in predicting mRNA regulation and ribosome binding site strengths. In cases where unwanted secondary structures such as long hairpins or loops are estimated for the sequence provided, the synthetic biologist may need to take these predictions into account.

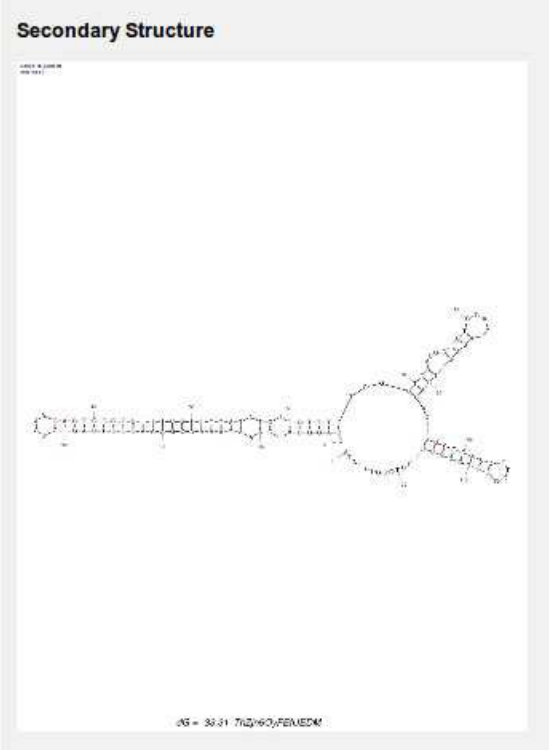


Figure 13 - Estimated secondary structure for submitted sequence

4.3 Overview of results

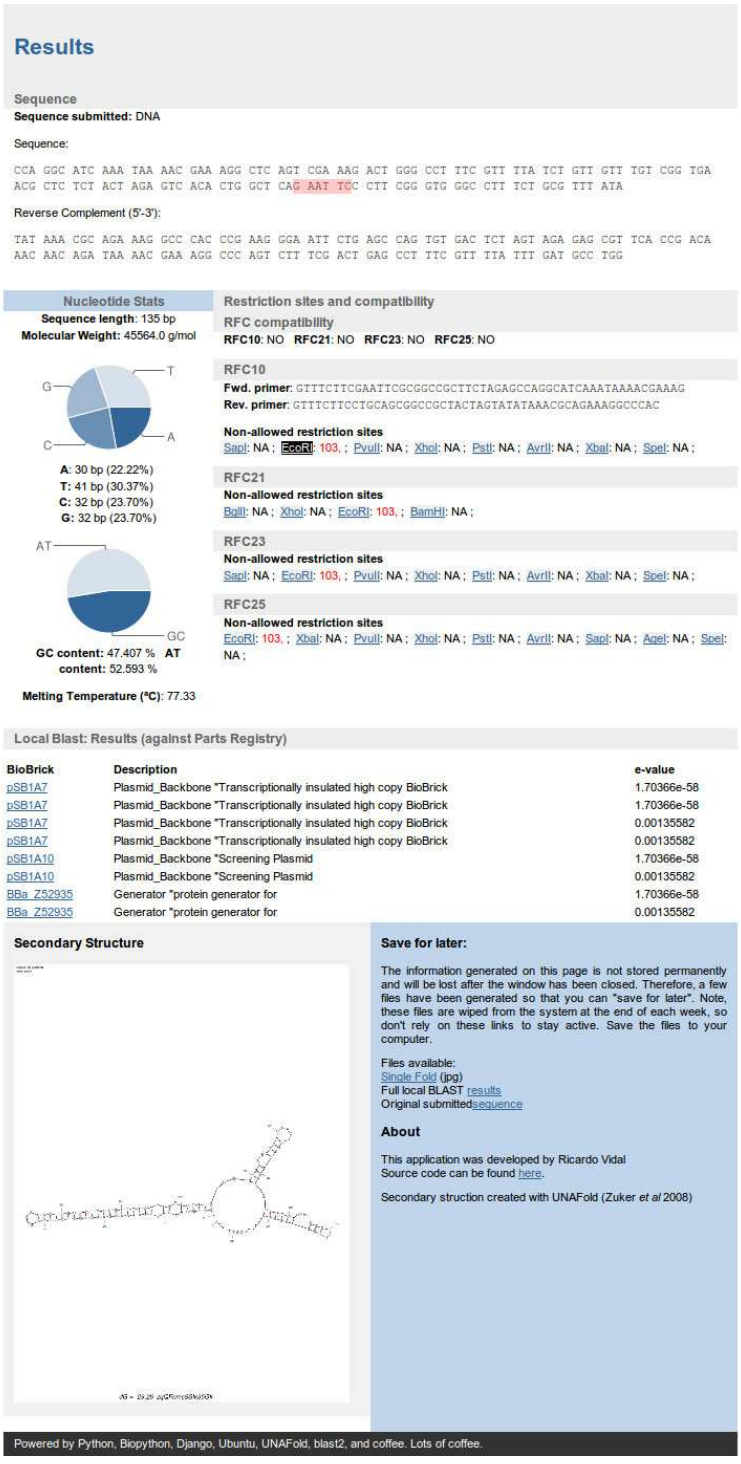


Figure 14 - Full overview of the results page generated on demand via user submitted biological sequence

This program was developed as a proof-of-concept bioinformatics tool demonstrating that a compound web application can provide a wide variety of data based on limited input and interaction on behalf of the user.

The objective of this project was to develop an intuitive web application that would require minimal user input to perform a task that would in other circumstances require multiple tools, knowledge of how they work and in some cases programming skills.

By focusing on a specific field of research, such as synthetic biology, it was possible to assess the required tools and information that synthetic biologists would use in their research. Refining natural biological parts into standardized ones is made easier by using the proposed web application. It provides biological engineers with a one-stop location to analyze biological sequences and obtain a set of results that not only characterize the target sequence but also provide preliminary information regarding the potential for said sequence to be made into a working Biobrick.

The information is provided via a single web form that runs the biological sequence of interest through a series of Python functions and robust bioinformatics tools such as BLAST, EMBOSS and UNAFold to provide biologically relevant information to the synthetic biologist without any required programming or bioinformatics knowledge whatsoever.

5 Future implementations

As most software or web applications, there is always space for improvement or adding new functions or features.

Here are listed a few future implementations that may be incorporated into the existing web application which may provide more information or tools for synthetic biologists attempting to create novel standard biological parts:

- **Generation of primers for each of the assembly standards provided, not only for the BioBrick standard (RFC10)**.
    - Since RFC23 and RFC25 assembly standards are compatible with the widely used Biobrick standard, primer generation for these alternative RFCs would assist many synthetic biologists working with fusion proteins.

- **Provide an estimated cost to synthesis the desired BioBrick and provide an easy gateway to selected DNA synthesis enterprises.**
    - With the cost of nucleotide sequence synthesis rapidly decreasing, it is now cost-effective to have sequences synthesized instead of constructing them via PCR. Providing information and a gateway to the companies that currently offer such services could provide useful for synthetic biologists.

- **Implement export options to online locations such as OpenWetWare or the Parts Registry.**
    - Once the results are displayed, information on the current web application is lost upon closing of the browser window. All information is generated on-the-fly, files are made available for download if necessary. However, an export option to a synthetic biology related wiki such as OpenWetWare or directly to the Parts Registry as a potential new Biobrick would prove to be useful.

- **Suggested sequence refinement for standard compliance.**
    - Once the submitted sequence has been analyzed and in the cases where the biological sequence is not compatible with any one of the RFCs, it would be of great interest to have the web application suggest point mutations so as to remove any unwanted restriction sites.

6 Bibliography

Altschul, S.F. et al., 1990. Basic local alignment search tool. *Journal of molecular biology*, 215(3), 403-10. Available at: http://www.ncbi.nlm.nih.gov/pubmed/2231712.

Anonymous, 1941. THE ENGINEERS' COUNCIL FOR PROFESSIONAL DEVELOPMENT. *Science (New York, N.Y.)*, 94(2446), 456. Available at: http://www.ncbi.nlm.nih.gov/pubmed/17744800.

Baumgardner, J. et al., 2009. Solving a Hamiltonian Path Problem with a bacterial computer. *Journal of biological engineering*, 3, 11. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19630940.

Benson, D.A. et al., 2008. GenBank. *Nucleic acids research*, 36(Database issue), D25-30. Available at: http://www.ncbi.nlm.nih.gov/pubmed/18073190.

Canton, B., Labno, A. & Endy, D., 2008. Refinement and standardization of synthetic biological parts and devices. *Nature biotechnology*, 26(7), 787-93. Available at: http://www.nature.com/nbt/journal/v26/n7/abs/nbt1413.html.

Chan, L.Y., Kosuri, S. & Endy, D., 2005. Refactoring bacteriophage T7. *Molecular systems biology*, 1, 2005.0018. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16729053.

Chandran, D., Bergmann, F.T. & Sauro, H.M., 2009. TinkerCell: Modular CAD Tool for Synthetic Biology. *DNA Sequence*, 23. Available at: http://arxiv.org/abs/0907.3976.

Cock, P.J. et al., 2009. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics (Oxford, England)*, 25(11), 1422-3. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19304878.

Densmore, D. et al., 2009. A platform-based design environment for synthetic biological systems. In *TAPIA '09: The Fifth Richard Tapia Celebration of Diversity in Computing Conference*. Portland, Oregon: ACM, pp. 24-29. Available at: http://portal.acm.org/citation.cfm?id=1565806.

Endy, D., 2005. Foundations for engineering biology. *Nature*, 438(7067), 449-53. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16306983.

Goler, J., 2004. Biojade: A design and simulation tool for synthetic biological systems. *DSpace. MIT*, (May), 56. Available at: http://hdl.handle.net/1721.1/30475.

Heinemann, M. & Panke, S., 2006. Synthetic biology--putting engineering into biology. *Bioinformatics (Oxford, England)*, 22(22), 2790-9. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16954140.

Hill, A.D. et al., 2008. SynBioSS: the synthetic biology modeling suite. *Bioinformatics (Oxford, England)*, 24(21), 2551-3. Available at: http://www.ncbi.nlm.nih.gov/pubmed/18757873.

Holovaty, A. & Kaplan-Moss, J., 2007. *The Definitive Guide to Django: Web Development Done Right* 1 ed., Apress. Available at: http://www.apress.com/book/view/1590597257.

Kitney, R. et al., 2009. Synthetic Biology: scope, applications and implications. *Current*, 61.

Knight, T. et al., 2003. Idempotent Vector Design for the Standard Assembly of Biobricks. , 11. Available at: http://hdl.handle.net/1721.1/21168.

Markham, N.R. & Zuker, M., 2008. UNAFold: software for nucleic acid folding and hybriziation. *Bioinformatics*, II(453), 3-31. Available at: http://www.springerlink.com/index/10.1007/978-1-60327-429-6.

Peccoud, J. et al., 2008. Targeted development of registries of biological parts. M. Thattai. *PLoS ONE*, 3(7), e2671. Available at: http://www.ncbi.nlm.nih.gov/pubmed/18628824.

Pillai, S. et al., 2005. SOAP-based services provided by the European Bioinformatics Institute. *Nucleic acids research*, 33(Web Server issue), W25-8. Available at: http://www.ncbi.nlm.nih.gov/pubmed/15980463.

Rettberg, R., 2009. International Genetic Engineering Machines. Available at: http://www.igem.org.

Rettberg, R., 2009. Parts Registry. Available at: http://www.partsregistry.org.

Rice, P., 2000. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*, 16(6), 276-277. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0168952500020242.

Richardson, S.M. et al., 2006. GeneDesign: rapid, automated design of multikilobase synthetic genes. *Genome research*, 16(4), 550-6. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16481661.

Rodrigo, G., Carrera, J. & Jaramillo, A., 2007. Genetdes: automatic design of transcriptional networks. *Bioinformatics*, 23(14), 1857-1858. Available at: http://www.bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/btm237.

SantaLucia, J. & Hicks, D., 2004. The thermodynamics of DNA structural motifs. *Annual review of biophysics and biomolecular structure*, 33, 415-40. Available at: http://www.ncbi.nlm.nih.gov/pubmed/15139820.

SantaLucia, J., 1998. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences of the United States of America*, 95(4), 1460-5. Available at: http://www.ncbi.nlm.nih.gov/pubmed/9465037.

Sellers, W. et al., 1864. Proceedings of the Franklin Institute. 343. *Preservation*.

Smedley, D. et al., 2009. BioMart--biological queries made easy. *BMC genomics*, 10, 22. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19144180.

Sprinzak, D. & Elowitz, M.B., 2005. Reconstruction of genetic circuits. *Nature*, 438(7067), 443-8. Available at: http://www.ncbi.nlm.nih.gov/pubmed/16306982.

Tigges, M. et al., 2009. A tunable synthetic mammalian oscillator. *Nature*, 457(7227), 309-12. Available at: http://www.ncbi.nlm.nih.gov/pubmed/19148099.

Wilkinson, M.D., 2002. BioMOBY: An open source biological web services proposal. *Briefings in Bioinformatics*, 3(4), 331-341. Available at: http://bib.oxfordjournals.org/cgi/doi/10.1093/bib/3.4.331.

7 Appendix

7.1 Bba_F2620

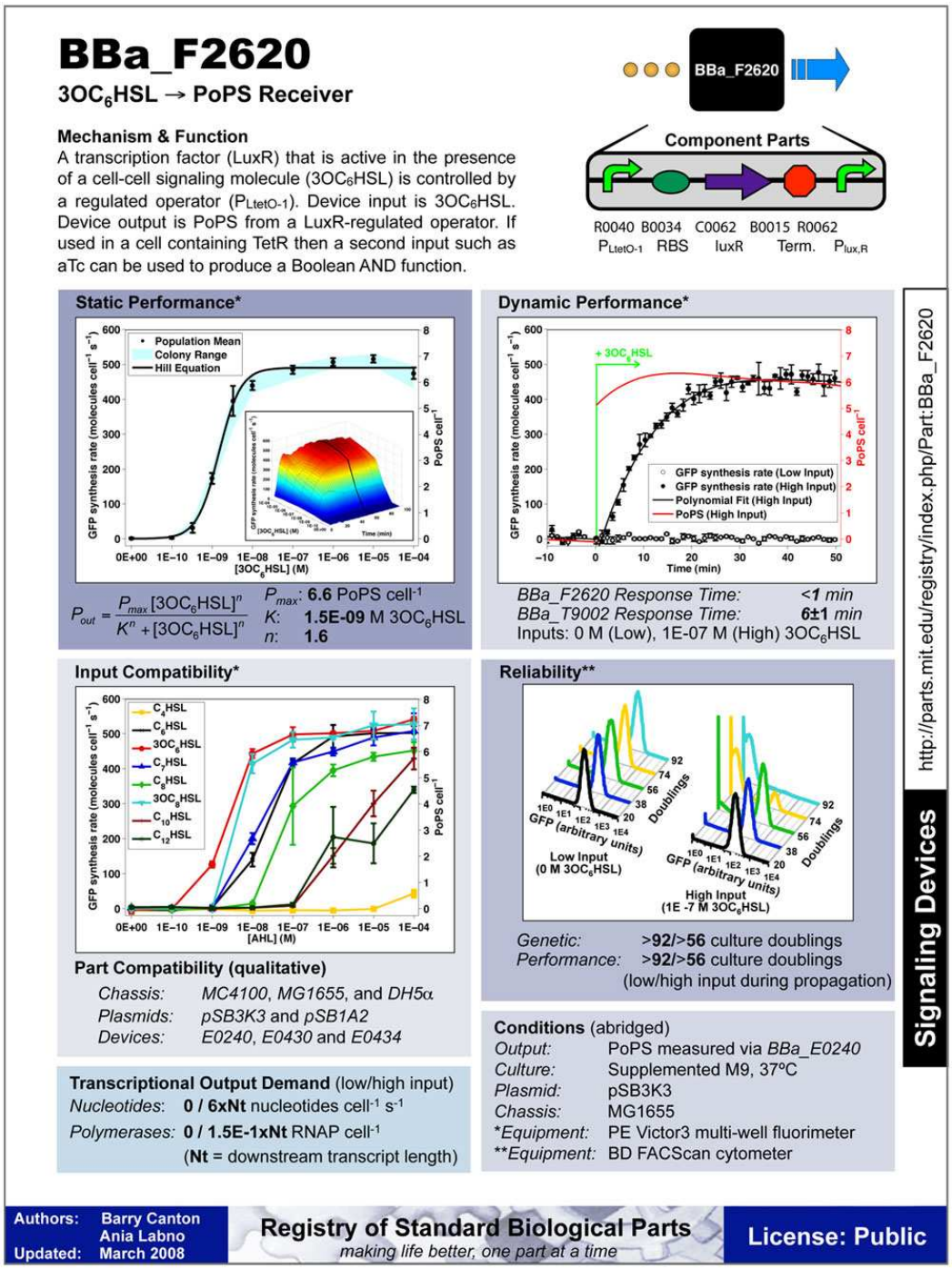


Figure 15 - Data sheet for standard biological part BBa_F2620 (Canton et al. 2008)

7.2 RFC10

Draft Standard for Biobrick Biological Parts

Tom Knight
3 May 2007

This standard defines the required sequence properties for a Biobrick(tm) standard biological part. It does not define any functional characteristics of the parts, nor does it motivate any aspect of these standards. All sequences defined herein are specified in the 5' to 3' direction.

0. A Biobrick compatible standard biological part consists of a DNA fragment potentially conveying informational or functional properties to a composite structure assembled from multiple parts. The current assembly process requires certain sequence properties for the part and the surrounding DNA.

1. Allowed sequences within Biobrick parts include any DNA sequence which does not contain the following subsequences:

   EcoRI site: GAATTC
   XbaI site: TCTAGA
   SpeI site: ACTAGT
   PstI site: CTGCAG
   NotI site: GCGGCCGC

   Additionally, there are a set of sites which, if convenient, should also be eliminated. Parts containing these sites qualify as fully Biobrick standard compliant, but future assembly and advanced uses of the parts may be compromised. These include:

   PvuII site: CAGCTG
   XhoI site; CTCGAG
   AvrII site: CCTAGG
   NheI site: GCTAGC
   SapI site: GCTCTTC and GAAGAGC

2. Biobrick Suffix

   Each Biobrick part must contain precisely this sequence immediately following the 3' end of the part:

   T ACTAGT A GCGGCCG CTGCAG
   (note: if constructing a primer, this sequence must be reverse complemented.)

3. Biobrick Prefix:

   To allow the construction of ribosomal binding sequences 5' of coding regions, the prefix used for coding regions is distinguished

from the sequence for non-coding Biobrick parts.

a. For non-coding Biobrick parts (the default) the Biobrick part must contain precisely the following sequence immediately 5' of the part:

   GAATTC GCGGCCGC T TCTAGA G

b. For Biobrick parts coding for proteins, the Biobrick part must contain precisely the following sequence immediately 5' of the ATG start of the coding region:

   GAATTC GCGGCCGC T TCTAG

   Parts containing start codons other than ATG must be modified to use ATG as the start codon. We highly recommend, but do not require, that all coding regions terminate with in-frame TAATAA stop codons, replacing other stop codons (TGA, TAG).

4. Plasmid context

Biobrick parts must be supplied in plasmids compatible with registry and assembly procedures. The pSBxxxx series of plasmids comply with these requirements. Other plasmids may be compliant with the following constraints:

a. Antibiotic resistance:

   All plasmids must carry at least one of the following antibiotic resistance markers:

   Ampicillin
   Chloramphenicol
   Kanamycin
   Tetracycline

   It is acceptable for a plasmid to convey both ampicillin resistance and no more than one additional antibiotic resistance markers.

   Note that certain strains convey resistance as well. Parts delivered to the registry must be in strains which do not convey resistance to these markers.

b. Sequencing primers

   The registry uses the following primers in sequencing all parts. Plasmids which omit or misplace these primers cannot be sequenced:

   VF2: TGCCACCTGACGTCTAAGAA
   VR: ATTACCGCCTTTGAGTGAGC

5. Strains

The registry maintains parts libraries in frozen bacterial cultures, and encourages submissions in this form. The bacterial strain must be a K-12 cloning strain (endA-). Under no circumstances will the registry accept submissions in any strain which is not BSL-1. We recommend strains such as Top10, DH10B, and DH5a. We do not recommend submissions in MC4100, BL21 and similar strains.

6. PCR contruction of Biobrick parts

Biobrick parts can be constructed by PCR from naturally occurring coding regions or other long DNA sequences. The recommended primer sequences for PCR of these fragments are:

Biobrick prefix:
GTTTCTTC GAATTC GCGGCCGC T TCTAGA G <18-24 bp of matching primer>

Coding region biobrick prefix:
GTTTCTTC GAATTC GCGGCCGC T TCTAG <18-24 bp of matching primer, beginning with ATG>

Biobrick suffix:
GTTTCTTC CTGCAG CGGCCGC T ACTAGT A <18-24 bp of matching primer (reverse complement)>

The reverse complement of this suffix sequence is:
3' T ACTAGT A GCGGCCG CTGCAG GAAGAAAC 5'

7.3 RFC21

Researchers at UC Berkeley have developed the BglBrick assembly standard, or Assembly standard 21, based on idempotent assembly with BamHI and BglII restriction enzymes. In a nutshell, most parts look like this:

```
        Prefix                          Suffix
5' GAATTC atg AGATCT ...part... GGATCC taa CTCGAG 3'
   EcoRI      BglII              BamHI   *   XhoI
```

Fusing two parts leaves the following scar:

```
5' [part A] GGATCT [part B] 3'
            G  S
```

Note, however, that Assembly standard 20 is intended as a minimal physical assembly standard, and only those features needed for interconversion of BglBrick assembly standard plasmids are formally defined. Therefore, `atg` and `taa` spacers are not core definitions of the standard.

See The BioBricks Foundation wiki for a discussion and comparison of different technical standards.

Formal Definition:

- A Berkeley assembly standard part is a DNA sequence flanked on the 5' end by `GATCT` and on the 3' end by `G` lacking BglII, BamHI, EcoRI, and XhoI restriction sites
- A Berkeley assembly standard vector is a DNA sequence flanked on its 5' end by `GATCC` and on its 3' end by `A`
- A Berkeley assembly standard entry vector has a unique EcoRI site, no BamHI or BglII restriction sites, and at most one XhoI site 5' to the EcoRI site
- A Berkeley assembly standard plasmid is represented as <vector_name>-<part_name> and has the sequence obtained by concatenating the vector and part sequences
- Further definition constraints are "sub-standards" of the Berkeley assembly standard format

Advantages

- in-frame fusion of protein parts
- benign protein scar
- enzymes selected for efficient cutting

Disadvantages

- BglII cannot be heat-inactivated therefore the current 3A standard assembly procedures won't work
- incompatible to original BioBrick assembly standard format
- incompatible to BioFusion format

7.4 RFC23

A New Biobrick Assembly Strategy Designed for Facile Protein Engineering

Ira E. Phillips* and Pamela A. Silver*

April 18, 2006

Abstract

The existing biobrick assembly technique[1] provides a straightforward way to combine standardized biological components, termed biobricks. This system, however, is limited in that each protein-encoding biobrick must contain a complete translated region. Signal sequences or other protein domains often convey a specific function to the protein to which they are attached; hence, each domain should be considered an independent biological part. With the current assembly technique, assembling such parts is not possible. This paper presents a revised assembly strategy that is compatible with the current biobrick definition and permits the construction of fusion proteins.

Introduction

The standard biobrick assembly technique was created in an attempt to simplify the construction of long concatemeric pieces of DNA often used in synthetic biology [1]. Previously existing techniques were limiting in that one often needed to design a unique cloning strategy for each desired construct. Moreover, the intermediates in the cloning of one construct would be of little value in creating other constructs.

As a solution to these problems, T. Knight developed a novel cloning strategy that permits standardization of biological parts or biobricks [1]. Each biobrick can be placed in front of or behind another biobrick using a standard protocol that is independent of the content of the parts involved. (To insert a biobrick in front of another, simply digest the upstream biobrick with EcoRI and SpeI. Cut out this insert and ligate into a vector containing the second part cut with EcoRI and XbaI. Conversely, to insert a part behind another, digest the insert [downstream part] with XbaI and PstI. Then, ligate into a vector containing the upstream part that has been cut with SpeI and PstI.) Most importantly, the product of such a combination has the same combination properties as the

*Department of Systems Biology, Harvard Medical School, Boston, MA 02115.

```
5' -GAATTC GCGGCCGC T TCTAGA G part T ACTAGT A GCGGCCG CTGCAG- 3'
3' -CTTAAG CGCCGGCG A ACATCT C part A TGATCA T CGCCGGC GACGTC- 5'
    EcoRI    NotI      XbaI            SpeI     NotI    PstI

       5' -TCTAGA G Part_1 T ACTAGA G Part_2 T ACTAGT- 3'
       3' -AGATCT C Part_1 A TGATCT C Part_2 A TGATCA- 5'
            XbaI           Mixed Site           SpeI
```

Figure 1: Standard definition of the biobrick ends (top) and the result from biobrick assembly (bottom). The mixed site is eight base pairs, resulting in a frame shift if translated.

originals. Hence, component biobricks can, in turn, be assembled together, creating long concatemeric sequences.

There is, however, a limitation in the current biobrick assembly technique–it does not permit the creation of fusion proteins. This limitation, which is especially problematic for eukaryotic systems, stems from two problems: translation over the mixed site (that results from fusing two parts) creates a frame shift, and coding region parts end with stop codons.

Initially, this shortcoming was handled by creating various versions of each major coding region part. For example, at least four different versions of ECFP (BBa_E0020, BBa_E0022, BBa_E0024, BBa_E0026) were synthesized using non-biobrick assembly techniques [2]. While such a solution is manageable on a small scale, it would be advantageous to harness the power of the biobrick assembly strategy to permit combination of the basic ECFP reporter to the desired signal sequences. Such a technique becomes critical when multiple fusion protein sequences are desired from the combination of multiple protein domains.

Improvements

We wanted to create an improved assembly strategy that allows for the creation and assembly of protein domain parts and that is compatible with existing biobricks. Moreover, we desired a strategy that would require relatively little testing. Therefore, we decided to use the restriction enzyme system described by T. Knight [1], since this system had already been proven to work. We, however, needed to change the biobrick definition to allow translation over the mixed site, which results from the fusion of two parts. Previously, this mixed site consisted of eight base pairs, which would result in a frame shift when this region is translated.

While it is necessary to maintain the frame over the mixed site, the translation frame for the restriction-ligation scar (ACTAGA) can be changed by adding different numbers of spacer nucleotides on either side of the part sequence. All three frames were considered, but only two were viable (Figure 2). The first (left) frame was chosen due to its shorter length and higher hydrophilicity. (It is important to note that each of the two possible frames results in the incor-

```
ACT AGA        xAC TAG Axx       xxA CTA GAx
Thr Arg            STOP              Leu D/E
```

Figure 2: Possible translation frames over the mixed site. Two frames are viable.

```
5' -GAATTC GCGGCCGC T TCTAGA part ACTAGT A GCGGCCG CTGCAG- 3'
3' -CTTAAG CGCCGGCG A ACATCT part TGATCA T CGCCGGC GACGTC- 5'
    EcoRI    NotI      XbaI        SpeI     NotI    PstI

          5' -TCTAGA Part_1 ACTAGA Part_2 ACTAGT- 3'
          3' -AGATCT Part_1 TGATCT Part_2 TGATCA- 5'
               XbaI       Mixed Site       SpeI
```

Figure 3: New definition of the biobrick ends (top) and the result from biobrick assembly (bottom). The mixed site is six base pairs, thereby maintaining the reading frame when translated.

poration of a charged amino acid. Each could have unintended consequences if placed beside charge-sensitive sequences.)

When two parts are combined using the new strategy, the reading frame is maintained, and a threonine and arginine are inserted between the coding parts (Figure 3).

It is important to note that the spacer nucleotides between the part and the XbaI and SpeI sites were originally incorporated in order to prevent DNA methylation and its consequential inhibition of the required restriction enzymes. These spacer nucleotides are not present in the assembly strategy presented here. Hence, it is required to prescreen parts to ensure that methylation sites are not incorporated. Dam methylation of the XbaI site is the most critical. Parts beginning with the sequences TCx create a site capable of inhibiting XbaI. If a desired part naturally begins with this sequence (coding for serine), it is necessary to change the first codon to either AGT or AGC (both also encoding serine).

Additionally, several modifications to the part specifications are required. First, each coding region should start and end in frame and not contain a stop codon. Also, it may be desirable to omit the start codon as well in order to reduce aberrant translation starting points [3].

Since parts containing protein domains have neither a start or stop codon, it is necessary to create a part category for each of these. Stop codon parts are straightforward. They should contain one or more stop codons to halt translation.

Start codon containing parts, however, prove more interesting. A limitation of the current biobrick assembly strategy is that ribosome binding sites contain only untranslated regions and a fraction of the start codon. It is known in eukaryotic systems, however, that the first nucleotide following the ATG of the start codon can have a significant effect on translational efficiency [4]. It

is possible with the improved biobrick strategy to incorporate another codon downstream of the start codon to control this variable. This permits the creating of a part of the Kozak consensus sequence, which is the consensus translation start codon sequence for highly expressed eukaryotic genes. Doing so should make the translation initiation site strengths more predictable.

In summary, the following modifications have been made to the original biobrick definition.

- The biobrick ends have changed so that the XbaI and SpeI restriction sites are adjacent to the biobrick part.
- Coding parts begin and end "in frame" and do not contain start or stop codons.
- Coding parts should be fused to both promoter and translation initiator parts and stop codon and transcriptional terminator parts.
- Parts must not begin with the nucleotides TC.
- Non-translation-related parts created with T. Knight's definition can be mixed with the parts proposed here.
- Previously built coding parts can be easily converted by PCR to meet the fusion biobrick specification.

Possible Protein Domain Parts

This new assembly technique permits the creation of a wide new list of part categories, including some interesting eukaryotic-specific parts. An incomplete list of such possible new parts is included below.

- Degradation Signals
- DNA Binding Domains
- Introns: Increase nuclear export; possibly delay translation.
- Localization Signals: Export signals, nuclear import signals, etc.
- Protein Domain Linkers
- Reporters
- Start Codons: Ribosome binding sites, consensus Kozak sequence, etc.
- Stop Codons
- Tags: His, FLAG, maltose binding protein, etc.
- Transcriptional regulatory domains

Figure 4: Example of a yeast fusion protein created with the described assembly method. The CTS1 promoter is followed by the Kozak sequence, two YFP proteins fused to an SV40 NLS, a stop codon, and a terminator from ADH1.

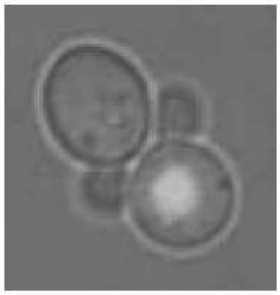


Figure 5: Fluorescent image overlaid with Nomarski of yeast with the construct in Figure 4 integrated in the genome. The YFP fusion protein is expressed in the daughter cell and localizes to the nucleus as expected.

Results

This modified cloning strategy has been used in our lab to produce approximately 100 protein fusion constructs. An example of such a fusion construct is shown in Figure 4. This construct contains a yeast promoter from CTS1, a gene expressed specifically in daughter cells [5], followed by the Kozak consensus sequence, two copies of yeast-codon optimized YFP, an SV40 nuclear localization signal, and finally a stop codon and transcriptional terminator from the yeast ADH1 gene. If properly expressed, this fusion protein should be present primarily in the nucleus of daughter cells. The predicted result is confirmed in Figure 5, where the presence of the fluorescent YFP in the nucleus confirms that the full-length fusion protein has been produced.

In summary, this modified version of the biobrick assembly technique permits the creation and assembly of a more diverse set of biological parts.

Acknowledgements

This work was supported by funds from the Keck Foundation.

References

[1] Knight T. "Idempotent Vector Design for Standard Assembly of Biobricks." DSpace http://hdl.handle.net/1721.1/21168 (2003).

[2] Registry of Standard Biological Parts. MIT. Release 11.21.05.

[3] Kozak M. "Regulation of translation via mRNA structure in prokaryotes and eukaryotes." Gene. 361: 13-37 (2005).

[4] Kozak M. "An analysis of 5-noncoding sequences from 699 vertebrate messenger RNAs." Nucleic Acids Res. 15(20): 8125-48 (1987).

[5] Colman-Lerner A, Chin TE, and Brent R. "Yeast Cbk1 and Mob2 activate daughter-specific genetic programs to induce asymmetric cell fates." Cell. 107: 739-750 (2001).

7.5 RFC25

BBF RFC 25: Fusion Protein (Freiburg) Biobrick assembly standard

Kristian M. Müller, Katja M. Arndt, iGEM 2007 Team Freiburg, Raik Grünberg

31.3.2009

1. Purpose

This Request for Comments (RFC) describes an extension to the original BioBrick assembly standard (BBF RFC 10). The Fusion Assembly strategy described here is fully backwards compatible with RFC 10 and supports the construction of in frame fusion proteins.

2. Relation to other BBF RFCs

RFC 25 extends RFC 10. RFC 25 can be used instead of RFC 23.

3. Copyright Notice

4. Motivation

The original BioBrick prefix and suffix (BBF RFC 10) with its easy cloning strategy is an excellent and universal way to combine various parts, e.g. promoter region, gene of interest, terminator etc. However, also individual proteins can be composed of different parts and the "mix-and-match" combination of fusion proteins is thus an important task in Synthetic Biology. However, the original assembly strategy does not allow for the cloning of protein fusions due to the generation of a stop codon at the SpeI/Xba scar.

According to RCF 10 the suffix of the first part (part in gray, PstI, NotI, SpeI):

...NNtactagtagcggccgctgcag

is combined with prefix of the second part (EcoRI, NotI, XbaI, part in gray):

gaattccgcggccgcttctagagNN...

resulting in the SpeI/Xba combination/scar:

...NNtactagagNN...

This scar encodes for the amino acids Tyr (codon: tac), STOP (codon: tag) and Ser or Arg (codon agn).

Consequently, according to RFC 10 proteins can only be designed as one part but not in a modular fashion. This request for Comments proposes a backward compatible extension to RFC 10 which allows for the construction of fusion proteins from modular BioBrick "FusionParts".

5. Detailed Description

5.1 Formal requirements

FusionParts MUST have the following **Prefix:**

gaattccgcggccgcttctagatggccggcNN...

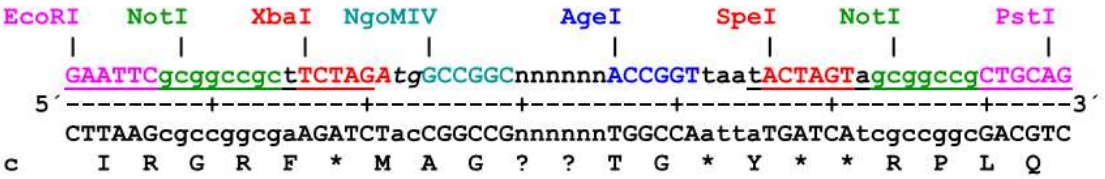
EcoRI, NotI, XbaI, NgoMIV, part in gray. The underlined sequence corresponds to the original (RFC 10) BioBrick prefix for protein coding sequences (i.e. 'expression parts').

Fusion Parts MUST have the following **Suffix:**

...NNaccggttaatactagtagcggccgctgcag

part in gray, AgeI, SpeI, NotI, PstI. The underlined sequence corresponds to the original (RFC 10) BioBrick suffix.

According to RFC 25, a complete FusionPart has thus the following format (part sequence is given as nnnnnn):



Fusion parts MUST NOT contain recognition sites for AgeI (ACCGGT) or NgoMIV (GCCGGC) anywhere in their sequence (outside prefix and suffix). The previous restrictions of RFC 10 apply as well. That means, fusion parts, furthermore, MUST NOT contain EcoRI, XbaI, SpeI, or PstI restriction sites. It is RECOMMENDED to also remove further restriction sites. See RFC 10 for details.

5.2 Application of the FusionPart Standard

Fusion parts can be used like standard Biobrick parts according to RFC 10. In addition, fusion proteins can be constructed using the restriction sites NgoMIV and AgeI following the same cloning principles. The NgoMIV and AgeI recognition sites are placed in frame with the protein coding sequence. The NgoMIV overhang (G'CCGG,C) is compatible with the AgeI overhang (A'CCGG,T) and ligation results in the scar ACCGCC coding for Thr and Gly.

For example, in order to fuse Part A N-terminally to part B, in the classic serial strategy, Part A SHOULD be digested with EcoRI and AgeI, and the vector containing Part B SHOULD be digested with EcoRI and NgoMIV. Ligation of the Part A into the vector B results in the FusionPart AB.

The 3A Assembly strategy is also supported. In this case, Part A SHOULD be digested with EcoRI and AgeI and Part B SHOULD be digested with NgoMIV and PstI and the construction vector SHOULD be digested with EcoRI and PstI followed by triple ligation. In the latter case, OPTIONALLY, either EcoRI can be replaced by XbaI or PstI by SpeI.

5.3 N-part

Some protein parts do not tolerate modifications of their N-terminus. However, combining Fusion Parts with existing RFC 10 RBS parts adds the sequence Met-Ala-Gly to the N-terminus of the protein. In cases where the native N-terminus needs to be preserved, FusionParts MAY be constructed in a hybrid "N-part" format. The Fusion N-part Prefix is identical to the BioBrick RFC 10 prefix for coding sequences (EcoRI, NotI, XbaI, part in gray):

gaattccgcggccgcttctagATG...

An N-part MUST have the following format:

```
EcoRI   NotI    XbaI          AgeI      SpeI    NotI      PstI
  |      |       |             |         |       |         |
  GAATTCgcggccgctTCTAGAtgnnnnnnACCGGTtaatACTAGTagcggccgCTGCAG
  ---------+---------+---------+---------+---------+---------
  CTTAAGcgccggcgaAGATCTacnnnnnnTGGCCAattaTGATCAtcgccggcGACGTC
    I  R  G  R  F  *  M  ?  ?  T  G  *  Y  *  *  R  P  L  Q
```

The underlined sequence corresponds to the RFC 10 proposal.

The classic RFC 10 assembly is used to combine Fusion N-parts with RFC 10 compliant

RBS parts. That means, the N-part SHOULD be digested with XbaI + PstI, and the RBS part with EcoRI and SpeI. FusionParts can be fused to the N-part by digesting the N-part with AgeI/SpeI and the FusionPart with NgoMIV/SpeI. Any number of FusionParts can be combined and optionally fused to the N-part.

6. Discussion

The Fusion part format meets the following design criteria:

(1) It enables protein fusions.

(2) RFC 25 is an extension rather than modification of RFC 10. Both FusionParts and Fusion N-parts are fully compatible with standard (RFC 10) BioBricks. They feature the RFC 10 BioBrick prefix for coding sequences and the standard BioBrick suffix. A FusionPart is equivalent to a RFC 10 stand-alone protein part as it has a start codon after the XbaI site (as defined for RFC 10 coding sequences) with two additional amino acids (Ala-Gly) encoded before the sequence of the protein of interest. The Fusion suffix sequence contains 3 stop codons to prevent accidental read-through.

(2) These fusion constructs are biochemically well behaved – the "scar sequence" between protein parts does not contain a frame-shift or stop codon. It does not contain any codon that is rarely used in E. coli. It translates to the sequence Thr-Gly, both of which are small uncharged amino acids that are often found in natural and synthetic linker peptides.

(3) The Ala-Gly sequence coded by the NgoMIV site satisfies the N-end rule [1]. That means that that this sequence does not promote degradation of the protein. Transcription start Biobricks (for example, RBS+start codon) can therefore also be designed as FusionParts and assembled without switching back to RFC 10 assembly. This improves over the BioFusion format described in BBF RFC 23. (4) Where needed, RFC 25 allows for preserving a protein's natural N-terminus. This can be important for some proteins..

(4) The Fusion format is compatible with the established BioBrick cloning protocols. All enzymes used can be inactivated by heating.

(5) In addition, the authors provide 3A (three antibiotic resistance) construction vectors that adhere to the presented format and allow for three fragment ligations with selection for insertion.

RFC 25 improves over RFC 23, in particular, regarding scar composition, N-end rule safety, and compatibility. This comes at the expense of two additional restriction sites which lengthen prefix and suffix and put additional constrains on part sequences.

There are different application scenarios for the Fusion Biobrick format. "Casual" protein

engineers can first assemble their full-length protein from FusionParts using the Fusion assembly, and then switch back to the classic RFC 10 assembly in order to embed this synthetic protein into larger circuits. In this way, novel protein fusions can be combined with the large set of transcription regulators or other devices that are already available in the Registry of Standard Biological Parts. Note though, that the assembly MUST be finished with RFC 10 enzymes, once the switch has been made.

Users with a focus on protein engineering may prefer to construct all of their parts, including non-coding ones, in Fusion format. This avoids the need to switch assembly strategies.

BioFusion parts (RFC 23) can be coupled to FusionParts as long as we are dealing with independent proteins or non-coding elements. On the other hand, it is not possible to create protein fusions directly from the two different formats. However, BBF RFC 24 describes an indirect strategy that involves one additional sub-cloning step.

Experience

At the time of writing, the Fusion format has been thoroughly tested by two student teams [2, 3] as well as several more senior investigators in different labs. Fusion parts perform robustly both in classic as well as in 3A assemblies without gel-purification. Customized protocols are available on OpenWetware.org [4].

7. Parts and Materials

As of 31.3.2009 the following construction vectors are fully compatible with the FusionPart proposal. Many other vectors work as well.

- pSB1AC3F (BBa_J18901)
- pSB1AK3F (BBa_J18902)
- pSB1AT3F (BBa_J18903)

Several FusionParts in the registry are shipped in the following vector (provided by a gene synthesis company).

- pMA (BBa_K157000)

Note: 3A assemblies starting from parts in this vector backbone show consistently low yields that require extensive screening of transformants. The reason for this remains unclear. We advise to sub-clone parts into pSB1* vectors before channeling them into 3A assemblies or to gel purify the insert before ligation.

The following restriction enzymes are used:
NgoMIV (also named NgoMI)

- As of spring 2009 this enzyme is available from the companies New England Biolabs and Promega
- Isoschizomers are NaeI, MroNI, NgoAIV, and PdiI.

- Activity in NEB buffers as of spring 2009: 1 - 100%, 2 - 50%, 3 – 10%, 4 – 100%.
- Heat Inactivation: 80°C for 20 minutes
- Methylation sensitivity: not sensitive to dam methylation and dcm methylation, blocked by CpG methylation.
- Number of recognition sites in DNA molecules are (adapted from Fermentas):

Lambda	PhiX 174	M13 mp18/19	pBR 322	pUC 18/19	pUC 57	pTZ19R/U	pBluescript IIKS(-/+)	pBluescript IISK(-/+)	pACYC 177	pACYC 184
1	0	1	4	0	0	1	1	1	0	5

AgeI

- As of spring 2009 this enzyme is available from New England Biolabs, Nippon Gene Co, and Promega.
- Isoschizomers are AsiAI, AsiGI, BshTI, CspAI, PinAI
- Activity in NEB buffers as of spring 2009: 1 - 100%, 2 - 50%, 3 – 10%, 4 – 75%.
- Methylation sensitivity: not sensitive to dam methylation and dcm methylation, blocked by CpG methylation.
- Heat inactivation: 65°C for 20 minutes
- Number of recognition sites in DNA molecules are (adapted from Fermentas):

Lambda	PhiX 174	M13 mp18/19	pBR 322	pUC 18/19	pUC 57	pTZ19R/U	pBluescript IIKS(-/+)	pBluescript IISK(-/+)	pACYC 177	pACYC 184
13	0	0	0	0	0	0	0	0	2	4

8. Author's Contact Information

Dr. Kristian M. Müller
Assistant Professor for Molecular Biology and Protein Engineering
Institute for Biology III
University of Freiburg
Schaenzlestr. 1
D-79104 Freiburg
Germany
e-mail kristian@biologie.uni-freiburg.de
Tel. ++49-(0)761-203-2748 (office)
Tel. ++49-(0)761-203-2763 (lab)
Fax. ++49-(0)761-203-2745

Dr. Katja M. Arndt
Fellow of the Freiburg Institute for Advanced Studies
Institute for Biology III
University of Freiburg
Schaenzlestr. 1

D-79104 Freiburg
Germany
email Katja@biologie.uni-freiburg.de
Tel. ++49-(0)761-203-2748 (office)

Dr. Raik Grünberg
EMBL-CRG Systems Biology Unit
CRG – Centre de Regulacio Genomica
Dr. Aiguader 88
08003 Barcelona
e-mail raik.gruenberg@crg.es

References

[1] Varshavsky A "The N-end rule: functions, mysteries, uses." Proc Natl Acad Sci U S A. 1996 Oct 29;93(22):12142-9.

[2] Original Proposal for Fusion parts at iGEM 2007: http://parts.mit.edu/igem07/index.php/Freiburg .

[3] Use of the standard for synthetic receptors at iGEM 2008: http://2008.igem.org/Team:Freiburg .

[4] The PrbbBioBrick community: Biobrick assembly protocols: http://openwetware.org/wiki/PrbbBB:Protocols .

7.6 Programming code

The following files are presented in this nested order, representing their location within the file structure. Folders are in bold.

- **bbhelper**
  - **media/seqfiles**
  - **templates**
    - myform.html
    - results.html
  - cleanbbfasta.py
  - forms.py
  - lblast.py
  - rfcs.py
  - seqchecker.py
  - views.py

The following program files are available for download at http://bbhelper.biotechlife.net.

7.6.1 myform.html

```
<html>
<head>
<title>bbhelper</title>
</head>
<body>
<center>
<form action="." method="POST">
{{ form.as_p }}
<input type="submit" value="Submit" />
</form>
</center>
</html>
```

7.6.2 results.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Seq results</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<meta name="generator" content="HAPedit 3.1">
<style type="text/css">
html,body{margin:0;padding:0}
body{font: 76% arial,sans-serif;text-align:center}
p{margin:0 10px 10px}
a{color: #336699;}
div#header h1{height:80px;line-height:80px;margin:0;
  padding-left:10px;background: #EEE;color: #336699}
div#container{text-align:left}
div#content p{line-height:1.4}
div#content img{border:0px solid #CCCCCC;}
div#content h3{height:20px;line-height:20px;margin:0;
  padding-left:10px;background: #EEE;color: #666666}
div#navigation{background:#C0D5EB}
div#extra{background:#f1f1f1}
div#footer{background: #333;color: #FFF}
div#footer p{margin:0;padding:5px 10px}
.bad {color:red;}
div#container{width:700px;margin:0 auto}
div#navigation{float:right;width:350px}
div#navigation h3{padding-left:10px;}
div#extra{float:left;width:350px}
div#extra h3{padding-left:10px;}
div#footer{clear:both;width:100%}

p#seq {font-family: monospace}
p#revseq {font-family: monospace}

.my_class_highlighted {
    background-color: #FFCBCB;
    color: #750F0F;
}
.my_class_highlighted2 {
    background-color: red;
    font-weight:bold;
}

.unselected {
    background-color: #EEEEEE;
}
.selected {
    background-color: #000000;
    color: #FFFFFF;
}
</style>

<script type="text/javascript">

function toggleHighLight(buttonElm, elmId, strMatch, classNameString)
{
    if(typeof(buttonElm) == 'undefined' || typeof(elmId) == 'undefined' ||
typeof(strMatch) == 'undefined' || typeof(classNameString) == 'undefined') {
        throw 'Error: missed params... ';
```

```
        return;
    }
    var elm = document.getElementById(elmId);
    var lighted = buttonElm.getAttribute('lighted');

    if(lighted == null || lighted == 'false') {

        buttonElm.setAttribute('lighted', 'true');
        buttonElm.className = 'selected';

        var content = elm.innerHTML;

        // replace strMatch and put "\s?" between all chars
        var aStrMatch = strMatch.split('');
        var reStrMatch = aStrMatch.join('\\s?');

        var reFind = new RegExp("("+reStrMatch+")", "gm");

        elm.innerHTML = content.replace(reFind, "<span
class=\""+classNameString+"\">$1</span>")
    } else {
        var aLightSpans = elm.getElementsByTagName('span');
        var spanContent = false;
        for(var i = (aLightSpans.length - 1); i >= 0; i--) {
            if(aLightSpans[i].className == classNameString) {
                spanContent = aLightSpans[i].childNodes[0];
                //alert(spanContent.textContent);

                aLightSpans[i].parentNode.replaceChild(spanContent,
aLightSpans[i]);
            }
        }
        buttonElm.setAttribute('lighted', 'false');
        buttonElm.className = 'unselected';
    }

}

</script>
</head>
<body>
       <div id="container">
              <div id="header"><h1>Results</h1></div>
              <div id="wrapper">
              <div id="content">
                     <h3>Sequence</h3>
                     <p><strong>Sequence submitted:</strong> {{ params.seqtype
}}</p>
                     <p>Sequence:</p>
                     <p id="seq">{{ params.fwd_codons }}</p>
                     <p>Reverse Complement (5'-3'):</p>
                     <p id="revseq">{{ params.rev_codons }}<br/><br /></p>
       <div id="nucstats"  style="float: left; width: 200px; text-
align:center;">
                     <h3 style="background:#C0D5EB;" >Nucleotide Stats</h3>
                     <p><strong>Sequence length</strong>: {{ params.seqlen }}
bp<br />
                     <strong>Molecular Weight:</strong> {{ params.seqmw }}
g/mol</p>
                     <img
src="http://chart.apis.google.com/chart?chs=150x100&amp;chd=t:{{ params.per_a
}},{{ params.per_c }},{{ params.per_g }},{{ params.per_t
}}&amp;chco=336699&amp;cht=p&amp;chl=A|C|G|T" alt="Bases (Percentage)"/>
```

```
                    <p><strong>A</strong>: {{ params.num_a }} bp ({{
params.per_a }}%) <br />
                    <strong>T:</strong> {{ params.num_t }} bp ({{ params.per_t
}}%) <br />
                    <strong>C:</strong> {{ params.num_c }} bp ({{ params.per_c
}}%) <br />
                    <strong>G:</strong> {{ params.num_g }} bp ({{ params.per_g
}}%)</p>
            <img
src="http://chart.apis.google.com/chart?chs=150x100&amp;chd=t:{{ params.gc
}},{{ params.at }}&amp;chco=336699&amp;cht=p&amp;chl=GC|AT" alt="GC/AT Content"
/>
                    <p><strong>GC content:</strong> {{ params.gc }} %  
                    <strong>AT content:</strong> {{ params.at }} %</p>
                    <p><strong>Melting Temperature (ªC)</strong>: {{ params.tm
}}</p>
       </div>
       <div id="rsites" style="float: right; width: 500px;">
                    <h3>Restriction sites and compatibility</h3>
                    <h3>RFC compatibility</h3>
                    <p><strong>RFC10</strong>: {{ compat.rfc10 }}  
                    <strong>RFC21</strong>: {{ compat.rfc21 }}  
                    <strong>RFC23</strong>: {{ compat.rfc23 }}  
                    <strong>RFC25</strong>: {{ compat.rfc25 }} </p>
                    <h3>RFC10</h3>
                    <p><strong>Fwd. primer</strong>: <code>{{
rfc10primers.fwd_primer }}</code><br />
                    <strong>Rev. primer</strong>: <code>{{
rfc10primers.rev_primer }}</code><br /></p>
                  <p><strong>Non-allowed restriction sites</strong><br />
                  {% for renz,cuts in cutsites.rfc10.items %}
                      <a href="#" class="unselected"
onclick="toggleHighLight(this, 'seq', '{{ cuts.rs_site }}',
'my_class_highlighted'); return false;">{{ renz }}</a>:
                          {% if not cuts.sites %}
                           NA
                          {% else %}
                              {% for pos in cuts.sites %}
                                  <span class="bad">{{ pos }}, </span>
                              {% endfor %}
                          {% endif %}; 
                  {% endfor %}</p>

                  <h3>RFC21</h3>
                  <p><strong>Non-allowed restriction sites</strong><br />
                  {% for renz,cuts in cutsites.rfc21.items %}
                      <a href="#" class="unselected"
onclick="toggleHighLight(this, 'seq', '{{ cuts.rs_site }}',
'my_class_highlighted'); return false;">{{ renz }}</a>:
                          {% if not cuts.sites %}
                           NA
                          {% else %}
                              {% for pos in cuts.sites %}
                                  <span class="bad">{{ pos }}, </span>
                              {% endfor %}
                          {% endif %}; 
                  {% endfor %}</p>

                  <h3>RFC23</h3>
                  <p><strong>Non-allowed restriction sites</strong><br />
                  {% for renz,cuts in cutsites.rfc23.items %}
```

```
                    <a href="#" class="unselected"
onclick="toggleHighLight(this, 'seq', '{{ cuts.rs_site }}',
'my_class_highlighted'); return false;">{{ renz }}</a>:
                        {% if not cuts.sites %}
                         NA
                        {% else %}
                            {% for pos in cuts.sites %}
                                <span class="bad">{{ pos }}, </span>
                            {% endfor %}
                        {% endif %}; 
                {% endfor %}</p>

                <h3>RFC25</h3>
                <p><strong>Non-allowed restriction sites</strong><br />
                {% for renz,cuts in cutsites.rfc25.items %}
                    <a href="#" class="unselected"
onclick="toggleHighLight(this, 'seq', '{{ cuts.rs_site }}',
'my_class_highlighted'); return false;">{{ renz }}</a>:
                        {% if not cuts.sites %}
                         NA
                        {% else %}
                            {% for pos in cuts.sites %}
                                <span class="bad">{{ pos }}, </span>
                            {% endfor %}
                        {% endif %}; 
                {% endfor %}</p>
    </div>
    <div style="clear:both; width: 100%;"> </div>
                <h3>Local Blast: Results (against Parts Registry)</h3>
                <p><table id="blasttable" width="100%">

<tr><td><strong>BioBrick</strong></td><td><strong>Description</strong></td><td>
<strong>e-value</strong></td></tr>
                {% for item in params.blast_list %}
                    <tr><td><a href="http://partsregistry.org/Part:{{
item.biobrick }}" >{{ item.biobrick }}</a></td><td>{{ item.seqtitle
}}</td><td>{{ item.e_value }}</td></tr>
                {% endfor %}</table></p>
            </div>
       </div>
       <div id="navigation">
              <h3>Save for later:</h3>
              <p style="text-align:justify;">The information generated on this
page is not stored permanently and will be lost after the window has been
closed.
              Therefore, a few files have been generated so that you can "save
for later". Note, these files are wiped from the system at the end of each
week, so don't rely on these links to stay active. Save the files to your
computer.</p>
              <p>Files available:<br /><a href="/media/seqfiles/{{
params.seqfile }}.jpg">Single Fold</a> (jpg)<br />
              Full local BLAST <a href="/media/seqfiles/{{ params.seqfile
}}.xml">results</a><br />
              Original submitted<a href="/media/seqfiles/{{ params.seqfile
}}.seq">sequence</a></p>
              <h3>About</h3>
              <p>This application was developed by Ricardo Vidal<br />
              Source code can be found <a href="">here</a>.</p>
              <p>Secondary struction created with UNAFold (Zuker <em>et al</em>
2008)</p>
              <p> </p>
       </div>
       <div id="extra">
```

```
        <h3>Secondary Structure</h3>
             <p><img src="/media/seqfiles/{{ params.seqfile }}.jpg"
width="320px" /></p>
       </div>
       <div id="footer"><p>Powered by Python, Biopython, Django, Ubuntu,
UNAFold, blast2, and coffee. Lots of coffee.</p></div>
       </div>
</body>
</html>
```

7.6.3 cleanbbfasta.py

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Alphabet import IUPAC
from Bio import SeqIO
handle = open("All_Parts")
all_bbs = []

bb_dict = SeqIO.to_dict(SeqIO.parse(handle, "fasta"))
handle.close()


def sortDictKeys(adict):
       items = adict.items()
       items.sort()
       return [key for key, value in items]


mybbs = []
for b in sortDictKeys(bb_dict):
    if str(bb_dict[b].seq) == '':
        pass
    else:
        mybbs.append(bb_dict[b])



f = open("clean_bbs.faa", "w")
SeqIO.write(mybbs, f, "fasta")
f.close()
```

7.6.4 forms.py

```
from django import forms

class SeqForm(forms.Form):
    message = forms.CharField(label='Sequence', widget=forms.Textarea)
    species    =    forms.ChoiceField(choices=(('human',    'Human'),    ('ecoli',
'E.coli'), ('yeast', 'Yeast')),
                               widget=forms.RadioSelect)
```

7.6.5 lblast.py

```
from Bio.Blast import NCBIStandalone
from Bio.Blast import NCBIXML

#my_blast_file = "/home/rvidal/NetBeansProjects/bbhelper/src/testseq.fasta"
SEQ_FOLDER = "/path/to/code/folder/bbhelper/media/seqfiles/"
my_blast_db = "/path/to/registry/cleanbb/forBLAST/clean_bbs.faa"
my_blast_exe = "/usr/bin/blastall"
E_VALUE_THRESH = 0.04


def local_blast(rfname):
    my_blast_file = SEQ_FOLDER+rfname+".seq"
    result_handle,    error_handle    =    NCBIStandalone.blastall(my_blast_exe,
"blastn",
                                                    my_blast_db, my_blast_file)
    ''' Save results to xml file '''
    blast_results = result_handle.read()
    xmlfile = SEQ_FOLDER+rfname+'.xml'
    save_file = open(xmlfile, "w")
    save_file.write(blast_results)
    save_file.close()

    ''' open the new xml file to be parsed '''
    result_handle = open(xmlfile)
    blast_records = NCBIXML.parse(result_handle)
    blast_record = blast_records.next()
    results_list = []
    if blast_record.alignments == []:
        pass
        #print "No relevant results in Registry"
    else:
        if len(blast_record.alignments) > 3:
            bresults = 3
        else:
            bresults = len(blast_record.alignments)
        for i in range(bresults):
            for hsp in blast_record.alignments[i].hsps:
                if hsp.expect < E_VALUE_THRESH:
                    parts = blast_record.alignments[i].title.split(' ')
                    lb_result = {}
                    lb_result['biobrick'] = parts[1]
                    lb_result['seqtitle'] = ' '.join(parts[4:-1])
                    lb_result['length'] = blast_record.alignments[i].length
                    lb_result['e_value'] = hsp.expect
                    results_list.append(lb_result)
                else:
                    pass
                    #print 'e value out of range.'
    return results_list
```

7.6.6 rfcs.py

```
from Bio.Restriction import *
from Bio.Seq import Seq

spacer = 'GTTTCTTC'


def overhang(seq):
    """ Select the first 20+ nucleotides from from a dna sequence
        make sure it ends in G or C
    """
    seq = Seq(seq)
    seq = str(seq)
    seq_chunk = seq[:20]

    for i in range(20,30):
        if seq[i:i+1] == 'C':
            seq_chunk += seq[i:i+1]
            break
        elif seq[i:i+1] == 'G':
            seq_chunk += seq[i:i+1]
            break
        else:
            seq_chunk += seq[i:i+1]
    return seq_chunk


def rfc10_primers(seq):
    """  Generate  the  forward  and  reverse  primers  according  to  the  RFC10
assembly standard
        http://openwetware.org/wiki/The_BioBricks_Foundation:BBFRFC10
        Coding sequence or non-coding sequence specific primers generated.
    """
    primers_rfc10 = {}
    fwd_chunk = overhang(str(seq))
    rev_chunk = overhang(str(seq.reverse_complement()))
    if str(seq[:3]) == 'ATG':
        fwd_primer = spacer + EcoRI.site + NotI.site + 'T' + XbaI.site[:-1] +
fwd_chunk
        primers_rfc10['fwd_primer'] = fwd_primer
        rev_primer = spacer + PstI.site + NotI.site[1:] + 'T' + SpeI.site + 'A'
+ rev_chunk
        primers_rfc10['rev_primer'] = rev_primer
    else:
        fwd_primer = spacer + EcoRI.site + NotI.site + 'T' + XbaI.site + 'G' +
fwd_chunk
        primers_rfc10['fwd_primer'] = fwd_primer
        rev_primer = spacer + PstI.site + NotI.site[1:] + 'T' + SpeI.site + 'A'
+ rev_chunk
        primers_rfc10['rev_primer'] = rev_primer
    return primers_rfc10
```

7.6.7 seqchecker.py

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
from Bio.Restriction import *
from Bio.SeqUtils import *
from Bio.SeqUtils.MeltingTemp import *
import string
import random
import os
import time

SEQ_FOLDER = “/path/to/code/folder/bbhelper/media/seqfiles/"
UNAFOLD_DATA = '/path/to/store/unafoldtempfiles/'
# RFCs for standard biological parts
rfcdict = {'rfc10': EcoRI+XbaI+SpeI+PstI+PvuII+XhoI+AvrII+SapI,
             'rfc21': EcoRI+BglII+BamHI+XhoI,
        'rfc23': EcoRI+XbaI+SpeI+PstI+PvuII+XhoI+AvrII+SapI,
        'rfc25': EcoRI+XbaI+AgeI+SpeI+PstI+PvuII+XhoI+AvrII+SapI}

''' dictionary of restriction enzymes and corresponding recognition sites '''
rs_sites = {'EcoRI': EcoRI.site,
            'XbaI': XbaI.site,
            'SpeI': SpeI.site,
            'PstI': PstI.site,
            'PvuII': PvuII.site,
            'XhoI': XhoI.site,
            'AvrII': AvrII.site,
            'SapI': SapI.site,
            'BglII': BglII.site,
            'BamHI': BamHI.site,
            'AgeI': AgeI.site
            }

dna_bases = IUPAC.unambiguous_dna.letters
rna_bases = IUPAC.unambiguous_rna.letters
amino_acids = IUPAC.protein.letters

class SeqChecker (object):
    def __init__(self, seq, species="human"):
        myseq = Seq(seq)
        self.myseq = myseq
        self.species = species

    def seqType(self):
        """ Find and return what type of sequence you have. DNA, RNA or AA """
        stype = 'DNA'
        self.stype = stype
        for b in self.myseq:
            if b not in dna_bases:
                stype= ''
                if b not in rna_bases:
                    stype= ''
                    if b not in amino_acids:
                        stype= ''
                    else:
                        stype= 'Protein'
                else:
                    stype= 'RNA'
            else:
                pass
```

```
        return stype

    def rfcCompatible(self, rfcdict):
        """ Checks the dna sequence for RFC compatibility
            This is done by verifying if there exist any non-wanted
            restriction sites within the sequence provided.
        """
        myseq = self.convToDNA()
        comp_reg = {}
        for rfc_num in rfcdict:
            compat = 'YES'
            comp_reg[rfc_num] = ''
            cutsdict = Analysis(rfcdict[rfc_num], myseq).full()
            for v in cutsdict.values():
                if v != []:
                    compat = 'NO'
                else:
                    pass
            comp_reg[rfc_num] = compat
        return comp_reg # return dict {'rfcx': 1 or 0}

    def rfcCutSites(self, rfcdict):
        ''' Analyse the sequence according to the rfc(s) supplied'''
        myseq = self.convToDNA()
        self.rfcdict = rfcdict
        site_reg = {}
        self.site_reg = site_reg
        for rfc_num in rfcdict:
            cutsdict = Analysis(rfcdict[rfc_num], myseq)
            site_reg[rfc_num] = cutsdict.full()
        ''' returns dictionary {rfc_num: {EcoRI:[12], SepI:[]}} '''
        return site_reg

    def convToDNA(self):
        dna_seq = ''
        self.dna_seq = dna_seq
        ''' convert to DNA if provided dna or rna. alert error if protein '''
        if self.seqType() == "Protein":
            dna_seq = Seq(self.rev_translate(str(self.myseq), self.species))
        elif self.seqType() == "RNA":
            dna_seq = self.myseq.back_transcribe()
        else:
            dna_seq = self.myseq
        ''' Save the final sequence, back_transcribed if necessary '''
        return dna_seq

    def seq2codons(self, sequence):
        codons = []
        for i in range(0,len(sequence), 3):
            codons.append(sequence[i:i+3])
        return ' '.join(codons)

    def makeSeqFile(self):
        '''
            Create a file with a random name with the sequence.
            Required for UNAFold and local BLAST
        '''
        dna_seq = str(self.convToDNA())
        rfname = "".join([random.choice(string.letters+string.digits) for x in
range(1, 16)])
        newfile = rfname+".seq"
        fpath = SEQ_FOLDER+newfile
        seqfile = open(fpath, 'w')
```

```
        seqfile.write(dna_seq)
        seqfile.close()
        self.rfname = rfname
        return rfname

    def single_fold(self):
        filename = self.makeSeqFile()
        seqfile = SEQ_FOLDER+filename
        resfile = UNAFOLD_DATA+filename
        ''' run UNAFold '''
        command_string = 'UNAFold.pl --NA=DNA --max=1 --mode=bases --label=10 -
-run-type=html ' + seqfile + '.seq'
        os.system(command_string)

        ''' convert .ps file to .jpg '''
        command_string = 'convert ' + resfile + '_1.ps ' + seqfile + '.jpg'
        os.system(command_string)

        command_string = 'rm ' + resfile + '*'
        os.system(command_string)
        return filename

    def rev_translate(self, sequence, species):
        """
            Reverse translation of amino acid sequence to DNA sequence
            using EMBOSS backtranseq (human, ecoli and yeast .cut files)
        """
        sequence = str(sequence)
        species = self.species

        tmp_aainput = "aainput.txt"
        tmp_aaoutput = "aaoutput.fasta"

        """ Use the appropriate .cut file for EMBOSS backtranseq """
        if species == "human":
            s_file = "Ehum.cut"
        elif species == "ecoli":
            s_file = "Eecoli.cut"
        elif species == "yeast":
            s_file = "Eyeast.cut"


        tmp_inputfile_loc = SEQ_FOLDER + tmp_aainput
        tmp_outputfile_loc = SEQ_FOLDER + tmp_aaoutput

        tmp_f = open(tmp_inputfile_loc, 'w')
        tmp_f.write(sequence)
        tmp_f.close()

        """ Run EMBOSS backtranseq """
        emboss_cl = 'backtranseq -sequence ' + tmp_inputfile_loc + ' -cfile ' +
s_file + ' -outfile ' + tmp_outputfile_loc
        os.system(emboss_cl)


        """ read newly created file with reverse translated sequence """
        #filename = 'seqecoli.fasta'
        out_f = open(tmp_outputfile_loc, 'r')
        return re.sub('[0-9\>\<\n\t\r]+','', ''.join(out_f.readlines()[1:]))

    def getParams(self):
        ''' Returns a dictionary with following data
            - GC content (percentage)
```

```
            - AT content (percentage)
            - Number of A, C, T, G (int)
            - Percentage of A, C, T, G (percentage)
            - Reverse Complement Sequence (string)
            - Sequence Molecular Weight (float)
        '''
        dna_seq = self.convToDNA()
        rfname = self.single_fold()
        fwd_codons = self.seq2codons(str(dna_seq))
        rev_codons = self.seq2codons(str(dna_seq.reverse_complement()))

        time.sleep(2)
        params = {}
        params['seqtype'] = self.seqType()
        params['gc'] = format(GC(dna_seq),'.3f')
        params['at']                                                    =
format((float(dna_seq.count("A"))+float(dna_seq.count("T")))*100/float(len(dna_
seq)), ".3f")
        params['num_a'] = dna_seq.count("A")
        params['per_a']                                                 =
format((float(dna_seq.count("A")))*100/float(len(dna_seq)), ".2f")
        params['num_c'] = dna_seq.count("C")
        params['per_c']                                                 =
format((float(dna_seq.count("C")))*100/float(len(dna_seq)), ".2f")
        params['num_g'] = dna_seq.count("G")
        params['per_g']                                                 =
format((float(dna_seq.count("G")))*100/float(len(dna_seq)), ".2f")
        params['num_t'] = dna_seq.count("T")
        params['per_t']                                                 =
format((float(dna_seq.count("T")))*100/float(len(dna_seq)), ".2f")
        params['seqlen'] = len(dna_seq)
        params['seq'] = dna_seq
        params['fwd_codons'] = fwd_codons
        params['rev_codons'] = rev_codons
        params['seqcomp'] = dna_seq.complement()
        params['seqrevcomp'] = dna_seq.reverse_complement()
        params['seqmw'] = molecular_weight(dna_seq)
        params['seqfile'] = rfname
        params['rs_sites'] = rs_sites
        params['tm'] = format(Tm_staluc(str(dna_seq)), ".2f")
        return params


# Create a list of sorted dictionary keys
def sortDictKeys(adict):
    items = adict.items()
    items.sort()
    return [key for key, value in items]
```

7.6.8 views.py

```
from seqchecker import *
from forms import SeqForm
from django.shortcuts import render_to_response
from lblast import *
from rfcs import *
import re

def seqread(request):
    if request.method == 'POST':
        form = SeqForm(request.POST)
        if form.is_valid():
            msg = form.cleaned_data['message']
            species = form.cleaned_data['species']
            msg   =   re.sub('[0-9\>\<\n\t\r]+','',   msg.encode()).replace('   ',
'').upper()
            s = SeqChecker(msg, species)
            params = s.getParams()
            rfc10primers = rfc10_primers(params['seq'])
            lblast_list = local_blast(params['seqfile'])
            params['blast_list'] = lblast_list

            rs_sites = params['rs_sites']
            cutsites = s.rfcCutSites(rfcdict)

            ''' Reorganize rfcCutSite dictionary to include R.Enz. sites '''
            all_cutsites = {}
            renz_list = {}
            renz_info = {}
            for rfcs in cutsites:
                #print rfcs
                renz_list = {}
                for renz in cutsites[rfcs]:
                    #print renz
                    allcuts = []
                    for site in cutsites[rfcs][renz]:
                        allcuts.append(site)
                    renz_info = {}
                    renz_info['sites'] = allcuts
                    renz_info['rs_site'] = rs_sites[str(renz)]
                    renz_list[renz] = renz_info
                    all_cutsites[rfcs] = renz_list

            compat = s.rfcCompatible(rfcdict)
            return render_to_response('results.html', {
                'params':params,
                'cutsites':all_cutsites,
                'compat':compat,
                'rfc10primers':rfc10primers,

            })
    else:
        form = SeqForm()

    return render_to_response('myform.html', {'form':form,})

def serve(request, path, document_root, show_indexes=False):
    pass
```