A dynamic and interactive Javascript components for pydna

Daniel Martins[1] and Bjorn Johansson[2]

[1] University of Minho, Braga, Portugal
[2] CBMA, Campus de Gualtar, University of Minho, Braga, Portugal

Abstract. Pydna is a simulator to DNA unit operations. It can be used to assemble multiple fragments on a single construction and represent it as an output.

To facilitate the interpretation of pydna assembly products, it has been proposed to find a way to represent it on Jupyter Notebook, an IPython interface that enables to create visual representations like graphs, images or figures from python code.

To create the representation class, it has been used an IPython method that generates an HTML document and enables the execution of Javascript code to modify the created document. Within the document it has been codified a 2D container where the representation would be drawn.

The final code, takes a pydna assembly product and creates a rectangular figure for each product fragment. That figure is drawn to depict the relative position and size of a fragment in the product and stores fragment features and information.

The described code allows to represent any assembly product, set its dimensions and get information relative to each fragment by clicking on the respective figure.

It consists on a complement to pydna, the representation will allow a quicker and simpler analysis and interpretation of pydna assembly products.

Keywords: Pydna, Jupyter Notebook, Genetic Assembly.

1 Background

1.1 Introduction

Pydna is a free open source python library. It was developed by professor Bjorn Johansson and co-workers at CBMA (Centro de Biologia Molecular e Ambiental) to use in simulation of synthetic biology experiments. Currently is being adapted to align better with Jupyter Notebook tool. Jupyter is an online open source software that can be used to display, share and run chunks of code.

That was the situation presented as starting point to the project. Therefore, as said on its title, this report seeks to describe the development of a dynamic and interactive Javascript-based representation of pydna outputs. That representation should depict a result of a genetic assembly experiment simulation.

1.2 Objectives

There have been defined some goals to accomplish the desired result and to get the greatest advantages of it.

As project effective goals, is expected an organized and structured representation of biologic constructs to be displayed on Jupyter. This representation shall be interactive, which means that the user shall be able to check details of the components of the construction, such as name, length or sequence. It should also be dynamic, that is, be adaptable to any simulation, representing any given pydna simulation output.

Along with the accomplishment of those primary goals, is also expected the best efficiency on the final representation design, allowing the user to intuitively access all the information as easy and quickly as possible. If possible and useful, is also desirable to add any functionality to the representation, improving analysis quality.

2 State of the Art

About the project's subject, which is entirely pydna, all the useful information can be found in the Open Access article *Pydna: a simulation and documentation tool for DNA assembly strategies using python* (Pereira *et al.*, 2015) [1], on pydna documentation [2] and complete code available on github [3].

The abovementioned article is an introduction to the software with an explanation of some of its features and presentation of a few examples. Both documentation and code itself are very enlightening about pydna operation, the code is vastly commented and some comments even contain examples in them.

2.1 Pydna

As presented by Pereira *et al.* pydna is an open source library, implemented in Python, that rely on packages as Biopython and NetworkX, it is meant to be used as a simulator for DNA unit operations such as restriction, ligation, primer design, assembly and recombination. [1]

Pydna can be used to predict the results of many biologic experiments, in this project, the results and information to be represented can be obtained using the features of just three pydna classes, Dseqrecord, Assembly and Contig.

Dseqrecord. Once pydna functions are focused on DNA, it has a very important class called Dseqrecord, as Biopython's SeqRecord class, it holds most of a DNA molecule's information but Dseqrecord refers to a double stranded molecule and its attributes are used to perform most of pydna functionalities.

The class can take either a string, a Seq, a Dseq, a SeqRecord or another Dseqrecord as initializer and its information is stored as a Dseqrecord object that can be represented as Dseqrecord(-x) or Dseqrecord(oy), which symbolize, respectively, a double stranded sequence with length x and a circular double stranded sequence with length y. [2, 3]

Assembly. This class takes a list of Dseqrecord objects and assembles the respective sequences in three steps, first, all sequences are compared to find homologous subsequences, then is created a graph where the overlapping sequences and the edges of the linear fragments (5' and 3') are stored as nodes connected by edges that store the sequences between the respective overlaps. The third step search through the graph all possible paths establishing the respective assembly product as a class output. [1]

The class allows to access the assembly information, in a list of features that include the input sequences, which of them have homologies, number of overlaps, number of nodes, products length, among others, an output example is shown below. [2]

```
Code exemple 1

Assembly:
Sequences........................: [33] [34] [35]
Sequences with shared homologies.: [33] [34] [35]
Homology limit (bp)..............: 14
Number of overlaps...............: 3
Nodes in graph(incl. 5' & 3')....: 5
Only terminal overlaps...........: No
Circular products................: [59]
Linear products..................: [74] [73] [73] [54]
[54] [53] [15] [14] [14]
```

Contig. An Assembly class product is returned as an instance of the Contig class, whose objects are not meant to be called directly by the programmer, this means that the Contig instances only represents simulations outcomes. [1, 2, 3]

This class have three "repr" methods, `__repr__()` returns a printable representation of the object, `_repr_pretty_()` and `_repr_html_()` are used in specific situations with the same objective as `__repr__()`. [2, 3]

The methods `detailed_figure()` and `detailed_fig()` are synonyms and return the sequences of each fragment of the construction in the same order that are displayed on the assembly result. [2, 3]

Finally, the methods `figure()`, `small_figure()` and `small_fig()` all run the same code. They are perhaps the most important pydna functions to this project, they create a small representation of the assembly construction, and that is what is expected to improve. On that representation, information of each fragment is displayed as the next example: `size of common DNA segment between current and previous fragment | DNA fragment name | size of common DNA segment between current and next fragment.` [2, 3]

Pydna creates a different representation for linear and circular constructions, depicted on code examples 2 and 3 respectively. [2, 3]

```
Code example 2

frag20| 6
       \/
       /\
        6|frag23| 6
                 \/
                 /\
                  6|frag14
```

```
Code example 3

-|2577|61
|       \/
|       /\
|       61|5681|98
|               \/
|               /\
|               98|2389|557
|                       \/
|                       /\
|                       557-
|                          |
 --------------------------
```

2.2 Jupyter

IPython is a comprehensive command shell for interactive and exploratory computing originally developed for Python. IPython runs the user code in a model called REPL (read-eval-print-loop). [4]

Jupyter Notebook provides an online application that extends IPython functionalities, depicting the process apart of executing it. This means that, besides running the code with the IPython Kernel, the notebook stores input and output along markdown notes to comment the code, on an editable document that can be saved as a JSON file with a ".ipynb" extension. Along with the code storing, Jupyter also allows to run chunks of code independently, which makes it an even more attractive tool. [5]

Due to all this, Jupyter can be used as a very helpful way to disseminate and share scientific computing work, it is ideal to represent methods and outcomes in a clean and structured way and allows everyone to test the code and analyze its potentialities.

Jupyter Notebook tool allows the user to interact and manipulate the data through the implementation of widgets, those widgets relies on Backbone.js, a Javascript framework used to provide structure to web applications. [6, 7]

2.3 Existing tools

Pereira *et al*. made a comparison with existing tools such as Gibthon, j5 and RavenCAD on their article [1], Gibthon remains the same as it was when that comparison was made, since the latest updates on github were made 5 years ago [8].

Both RavenCAD and j5, besides performing the assembly simulation, provides a rich graphic representation. RavenCAD shows on a single window, various steps and /or features of the assembly process and let the user to zoom the image [9]. J5 presents the results in a more sophisticated way, it uses DeviceEditor, a BioCAD (Biological Computer-Aided Design) tool to arrange the construction parts and facilitate its selection, it also can show the sequence permanently in a parallel window and highlight the selected parts or subsequences among other functionalities. [10, 11]

It has been found a more recent similar tool, AssemblX is a j5-based web tool that can be used on assembly planning and simulation, it performs a multi-level process, that is, after a first assembly, it can assemble a group of those "level 0" constructions, and then assemble again those products in a third level construct. It has some restrictions regarding the number of units that can be assembled, but is presented as a user-friendly reliable tool. With the implementation of j5 into AssemblX, this tool serves as an interpreter that hands over the inputs to j5 server and takes suggestions from it to compile the units in an adapted strategy leading to a final product. [12, 13]

3 Problem analysis

Pydna is a complete tool for assembly simulations, but the outcome can be a little confusing and difficult to interpret on larger and more complex constructions, so, as it has been said above, this project aims to create a cleaner representation. The easiest way to do it, is implementing the outcome representation on a notebook that allows the usage of alternative ways to create interactive graphics or diagrams among others.

Is expected to get a fully functional and useful tool. It should allow a detailed analysis of the experiment result. All information used in the construction process and the features contained on each fragment should be displayed or accessible.

3.1 Methodologies

On this project, we will look to implement one of the abovementioned widgets to represent the assembly construction, once Backbone.js is the framework on the base of those widgets, it will be the chosen framework to develop the code to create the interactive pydna assembly product representation. [6]

The development will start with an outline of the structure to be represented, it can be linear or circular according to the Contig object to be represented, the second step will be to place and identify the substructures on the created outline, then, create a way to allow the user to access the substructure features such as sequence, length and so on, finally, the representation will be aesthetically improved and refined to give the user a more pleasant usage of the tool.

During the development, whenever possible, it should be added any new functionality if that improves the quality of analysis.

4 Representation Characteristics

The construction has been represented as a rectangular figure with a box and line pattern, the boxes portray the fragments, in turn, the dashed lines depict the overlap between two consecutive fragments. Lastly, in circular products, the representation shows a line that connect the overlap between the last and the first fragments, closing the cycle.

To display the representation, it was necessary to create an area where the fragments could be drawn and organized, the HTML `<canvas>` element suited that need, it is a container for graphics, images, text or figures that can be drawn via Javascript code. That element is created on the body of an HTML document. [14]

The method `_repr_javascript_()` imports an IPython `display_html()` function, it, respectively, triggers the representation creation and displays it when the user runs the cell.

```
#Python code 1

def _repr_javascript_(self):
    from IPython.display import display_html
    html='''<html>
              <head></head>
              <body>
                <canvas id='Prod'></canvas>
              </body>
            </html>'''
    display_html(html, raw=True)
```

The Javascript code constructed with the method `get_js()` works as script for the HTML document created.

4.1 Initialization and display

To create the representation, it has been defined a class `Product_repr(object)` that only requires the assembly product to be represented as argument.

The `__init__()` method sets attributes referent to the product itself, a list of the fragments and the number of fragments to be represented.

It also defines all constant dimensions as default values, those values can be taken as optional arguments.

On this method is also defined the proportion between any fragment figure width and its respective real length. The proportion value has been set in relation to the smaller fragment which will be represented as a 150 pixels width rectangle (default value).

```
#Python code 2
self.proportion = min_size / len (min (self.frag_list, key
=len))
```

4.2 Canvas configuration

The canvas reference has been used to draw the DNA construction representation in a 2D context. [14,15] The canvas is displayed in width and height dimensions suited to the size of the construction.

It also has been defined, text font and alignment, element shadows, width and color of the connecting line.

4.3 Fragment configuration and placing

Construction arrangement. On original pydna, the product fragments were represented in a descending orientation. On the other hand, the created notebook representation portrays a more linear figure, the fragments are alternately displayed above and below a longitudinal axis.

To achieve that, it has been defined two python variables which values are passed to javascript code, `middle_y`, that defines the position of the axis on the canvas, and `delta_y` which is the value of the distance (in pixels) between the axis and the top border of any fragment.

Fragment position. To represent each fragment, the rectangle figure needs the canvas coordinates to be laid on, the vertical coordinate is obtained with the following formula.

$$\text{middle\_y} - ((-1)^{N} * \text{delta\_y}) \qquad (1)$$

Being, "middle_y" and "delta_y" the abovementioned python variables, N is the fragment index. Even N's will subtract delta_y from middle_y and place the rectangle above the axis, odd N's will sum the variables, placing the rectangle below the axis.

The horizontal coordinate is defined by the python variable `start_x` changed and passed to javascript to create each fragment, the `start_x` value of every rectangle apart from the first is obtained with the sum of the previous fragment right border position and the overlap size previously defined.

Creation. Given the coordinates, the rectangle can be created with a simple javascript method `fillRect()` which takes as argument the horizontal position, the vertical position, the width and the height. [14]

It has been associated a text with the fragment name to each rectangle.

4.4 Overlaps

All overlaps had been represented with the same size. This is due to the big discrepancy that can exist between the values of fragment and overlap lengths. To maintain the size proportion, the overlaps would be, in many cases, imperceptible, therefore, it has been set a fixed size to the overlaps.

The overlaps are drawn on each side of every fragment. The right overlap of a fragment and the left overlap of the next fragment are connected by dashed lines which represent the correspondence between them. The size of each overlap is attached to it.

On circular products, the overlap between the last and the first fragment is connected by a line which thus depicts a closed figure.

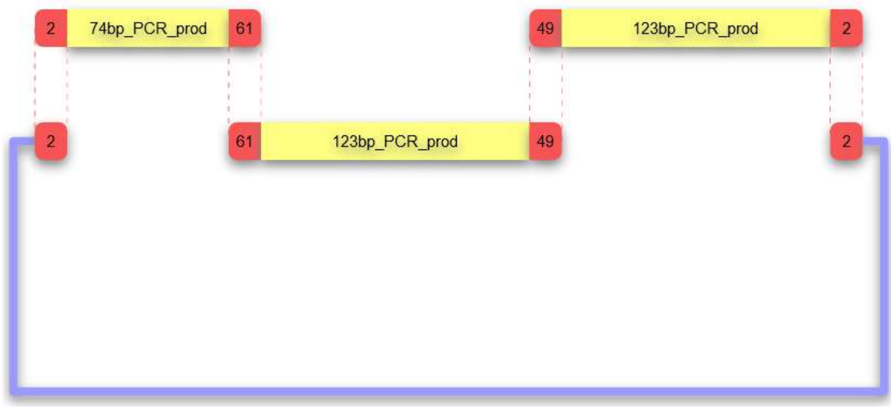


Fig. 1. Example Construction Representation

5 Functionalities

Since the created class takes an assembly product as argument, the code is only meant to be used on pydna assembly products. To run the code, it is not required to install neither jupyter plugins or extensions nor any other library.

The code is contained in a .ipynb file, which can be uploaded and then opened in any jupyter server, therefore, it can be used as it is presented or it can be modified to suit better the needs of the user. To facilitate that modification, all independent dimension variables can be defined on the initializer.

Besides that flexibility, the code creates an index that is appended to each canvas element name, hence creating different elements. That allows to create and display more than a single representation on the same notebook page.

5.1 Information Box

It was desired to display information about a fragment "on click", to achieve that, a click on a fragment figure should be interpreted as an order. Then, the order should search the information relative to the fragment clicked.

Information storage. To search the information with a javascript function, it must be stored on the HTML document.

So, in each iteration (corresponding to a fragment) is created a text that compiles fragment data and appends it to a hidden unique `<div>` HTML element. That unique element enables to get just the fragment information relative to the clicked figure.

Clickable fragments. Javascript allows to obtain the coordinates where the mouse is positioned on the browser with `event.clientX` and `event.clientY` properties.[14, 15]

It also has a built-in method `getBoundingClientRect()` that returns the relative position of an element, which, in this case, is the canvas. [14, 15]

```
// Javascript code 1 - function used to get the click coordi-
nates on the canvas

function getMousePos(canvas, event) {
    var rect = canvas.getBoundingClientRect();
  return {
      x: event.clientX - rect.left,
      y: event.clientY - rect.top
    };
};
```

Taking those relative positions, it's possible to determine the point of the canvas that has been clicked.

Therefore, the defined javascript function `isInside()` verifies if the clicked point coordinates is surrounded by the borders of the rectangle.

The action to be executed when a figure is clicked is stored with the `.addEventListener()` method.

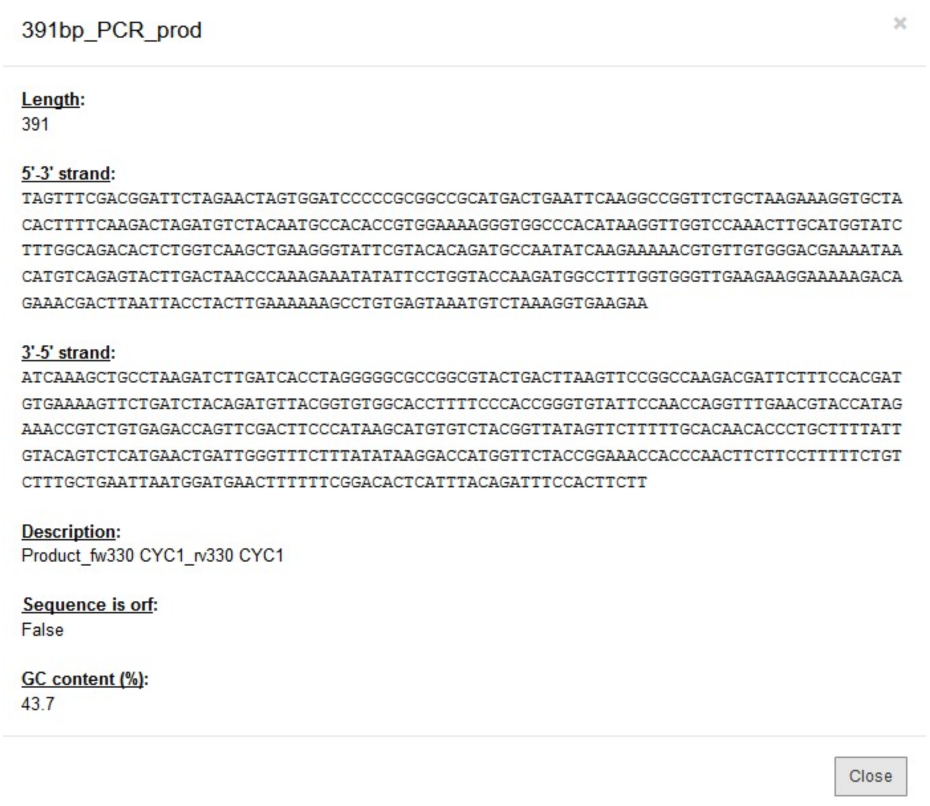
```
// Javascript code 2 - Code executed on click

c.addEventListener('click', function(evt) {
    var mousePos = getMousePos(c, evt);
    debugger;
    if (isInside(mousePos,rect)) {
        require(["base/js/dialog"],
            function(dialog) {
                new.style = "display: block";
                dialog.modal({
                    title: Name,
                    body: info_text,
                    buttons: {
                        'Close': {}
                    }
                });
            }
        );
    }
}, false);
```

Information display. Once it is verified that the fragment figure has been clicked, a popup window is shown and the specific text for that fragment is displayed on it, the name of the fragment is displayed on the window header.

RequireJS is a file and module loader [16] that enables the use of some non-built-in modules in Javascript code.

In this case, it allowed the use of a dialog module, that creates the modal box displayed when the `dialog.modal()` function is executed inside the previously referred `.addEventListener()` method.



391bp_PCR_prod

Length:
391

5'-3' strand:
```
TAGTTTCGACGGATTCTAGAACTAGTGGATCCCCCGCGGCCGCATGACTGAATTCAAGGCCGGTTCTGCTAAGAAAGGTGCTA
CACTTTTCAAGACTAGATGTCTACAATGCCACACCGTGGAAAAGGGTGGCCCACATAAGGTTGGTCCAAACTTGCATGGTATC
TTTGGCAGACACTCTGGTCAAGCTGAAGGGTATTCGTACACAGATGCCAATATCAAGAAAAACGTGTTGTGGGACGAAAATAA
CATGTCAGAGTACTTGACTAACCCAAAGAAATATATTCCTGGTACCAAGATGGCCTTTGGTGGGTTGAAGAAGGAAAAAGACA
GAAACGACTTAATTACCTACTTGAAAAAAGCCTGTGAGTAAATGTCTAAAGGTGAAGAA
```

3'-5' strand:
```
ATCAAAGCTGCCTAAGATCTTGATCACCTAGGGGGCGCCGGCGTACTGACTTAAGTTCCGGCCAAGACGATTCTTTCCACGAT
GTGAAAAGTTCTGATCTACAGATGTTACGGTGTGGCACCTTTTCCCACCGGGTGTATTCCAACCAGGTTTGAACGTACCATAG
AAACCGTCTGTGAGACCAGTTCGACTTCCCATAAGCATGTGTCTACGGTTATAGTTCTTTTTGCACAACACCCTGCTTTTATT
GTACAGTCTCATGAACTGATTGGGTTTCTTTATATAAGGACCATGGTTCTACCGGAAACCACCCAACTTCTTCCTTTTTCTGT
CTTTGCTGAATTAATGGATGAACTTTTTTCGGACACTCATTTACAGATTTCCACTTCTT
```

Description:
Product_fw330 CYC1_rv330 CYC1

Sequence is orf:
False

GC content (%):
43.7

Fig. 2. Information Box example

5.2 Features displayed

As it has been said, the fragment name is displayed on the popup window header, besides that, the window also presents other fragment features. The text

Length. The construction figure is a very good way to observe relative positions between fragments and size proportions, but the information box provides the fragment length to a more accurate analysis.

Sequences, 5'-3' and 3'-5' strands. Once pydna is a tool used to simulate and analyze DNA construction, the fragment sequence is an essential feature, therefore, it is also displayed on the information box.

Description. Some fragments are obtained through pydna methods, in some of those methods, a description is added to the fragment informing about the fragment origin. For instance, it can tell us that the fragment is a PCR product, or that has been cut by a specific enzyme.

Sequence is orf. In various laboratorial procedures, a DNA construction is meant to be introduced on an organism. Given that, the construction must be replicated to keep up with organism multiplication or growth. So, the construction needs to have at least one orf (open reading frame). The information box contains that feature.

GC content (%). Another common laboratorial procedure is PCR, on that amplification method, an important factor to consider is the amount of Guanine and Cytosine on the DNA molecule. The combined percentage of those nucleotides is also displayed on Information Box.

5.3 Improvements to pydna on the user perspective

As it was looked for, the obtained representation creator is a useful complement for pydna.

In addition to the advantage implied in a clearer visual representation, the possibility to modify the dimensions used, principally the fragment minimum and maximum length, allows the user to suit the representation to its needs.

The creator does not need any other imported or installed module besides the ones used to create the assembly product. The fact of being "ready to run" is an attractive characteristic of it, since it can be used by any user, even if he is not familiarized with Command Prompt Commands, conda packages installation, python imports or any other way to enable modules or packages.

The possibility to create multiple representations on the same notebook file, allows the user to easily compare two or more constructions, it is very useful when the user looks for a specific feature on a group of constructions or when it is needed to find out which is the construction more suited to the use on what is meant to apply.

The information Box is perhaps the most useful functionality provided by the representation. Although the complexity of the algorithms involved, pydna provides many features and characteristics of an assembly product and it only requires basic python knowledge of the user to get them as output of the right called method. The representation has no need to call any method, a click displays the compiled information about any fragment, from there, the user can compare and analyze multiple fragment features at the same time, hence improving analysis quality and quickness.

6 Conclusion

The final code fulfills the defined goals. It creates a dynamic representation that can depict any assembly product and display multiple constructions on a single notebook file. The user can set the representation basic dimensions to adjust the representation to what he expects to obtain.

The user can also interact with the representation by clicking on each fragment figure, that action provides compiled information about the fragment, it can be analyzed and used by the user in multiple ways, to, for instance, achieve biologic conclusions or assess the appropriation of a given construction to determined function.

The representation is a useful complement to pydna, making the tool even more attractive and easy to interpret.

7 Supplementary material

This report is accompanied with some additional files.

- Assembly_product_repr.py. Python file containing the developed code
- Assembly_product_repr.ipynb. Jupyter Notebook file with some examples of pydna assembly products and notebook representations of them. Must be opened on Jupyter tool. Installation instructions link:
    - http://jupyter.readthedocs.io/en/latest/install.html [5]
- cyc1.gb, p426GPD.gb and pUG35.gb. Genbank files needed to run example 2.

All files must be on the same directory.

References

1. Pereira, F., Azevedo, F., Carvalho, A., Ribeiro, G., Budde, M., Johansson, B.: Pydna: a simulation and documentation tool for DNA assembly strategies using python. BMC Bioinformatics, 16(142). 2015.
2. Pydna documentation, http://pydna.readthedocs.io/, last accessed 2017/4/27
3. Pydna on github, https://github.com/BjornFJohansson/pydna, last accessed 2017/4/27
4. IPython documentation, http://ipython.readthedocs.io/en/stable/index.html, last accessed 2017/4/27
5. Jupyter documentation, https://jupyter.readthedocs.io/en/latest/index.html, last accessed 2017/4/27
6. Widgets on Jupyter, http://ipywidgets.readthedocs.io/en/latest/index.html, last accessed 2017/4/27
7. Backbone.js documentation, http://backbonejs.org/, last accessed 2017/4/27
8. Gibthon on github, https://github.com/Gibthon/Gibthon, last accessed 2017/4/28
9. RavenCAD online site, https://cidar.bu.edu/ravencad/index.html, last accessed 2017/4/28
10. J5 manual, https://j5.jbei.org/j5manual/pages/1.html, last accessed 2017/4/28
11. Chen, J., Densmore, D., Ham, T., Keasling, J., Hillson, N.: DeviceEditor visual biological CAD canvas. Journal of Biological Engineering, 6(1), 2012.
12. Hochrein, L., Machens, F., Gremmels, J., Schulz, K., Messerschmidt, K., Mueller-Roeber, B.: AssemblX: a user-friendly toolkit for rapid and reliable multi-gene assemblies. Nucleic Acids Research. 2017.
13. AssemblX homepage, http://assemblx.mpimp-golm.mpg.de/, last accessed 2017/4/28
14. W3Schools homepage, https://www.w3schools.com/, last accessed 2017/6/29
15. Mozilla Developer Network homepage, https://developer.mozilla.org/en-US/, last accessed 2017/6/27
16. RequireJS documentation, http://requirejs.org/, last accessed 2017/6/30