Summer Internship: Migrating Integral Darwin Functionalities to Python

Ferenc Galkó
ferenc.galko@gmail.com

September 3, 2014

Contents

1 Introduction

The nascent field of bioinformatics is expanding at unprecedented rate, demanding more and more efficient implementations of a variety of algorithms, especially associated with sequence alignment.

In many cases hundreds of billions of alignments have to be done to investigate relations between publicly available genes. Such an immense amount of operations require an efficient low level - possibly vectorised - implementation, for which usually C/C++ is used. However, it is usually not really convenient to develop the applications that use the alignment algorithm in these low level languages.

PyOPA makes it possible to carry out various operations on sequences with the efficiency of a vectorised C and the convenience of a Python code, in a platform independent way.

The C core, that is wrapped by using NumPy and Cython, is already broadly used from Darwin, so in order to make a platform-switching possible, all results must conform to the results coming from Darwin, which is assured by the unit tests coming with PyOPA.

In this document, we concisely describe the main functionalities offered by PyOPA as well as a brief comparison to PyCogent. In Section 2 we show how to install PyOPA, while Section 3 outlines the main features of the package, followed by the comparison to PyCogent in Section 4.

2 Installing PyOPA

PyOPA draws on features offered by *NumPy* and *Cython* and it is strongly recommended to properly install these packages before installing PyOPA, although these two packages are defined as dependencies and should be automatically downloaded and installed during the installation.

There are two simple ways to install PyOPA, the first, probably more convenient, is by using *pip*:

```
pip install PyOPA
```

This simple code snippet should automatically download PyOPA from *PyPi* and compile and install the necessary files according to your operating system. This means, that on UNIX-based systems *gcc*, and on Windows-based systems *Visual Studio Compiler* must be available.

If you are able to install *NumPy* and *Cython* without any problems by using *pip*, which means that you have the necessary means and permissions to download, compile and install a package from *PyPi*, no errors should occur during the installation of PyOPA.

Another possibility is to download, extract and install the package manually. After extracting the project into a separate folder, you should navigate into the directory and use

```
python install setup.py
```

to install PyOPA. This should compile and install all the necessary files without any problems.

3 Features

3.1 Byte and short estimations

Instead of calculating the actual double score for two given sequences, we can rapidly compute an upper bound of the score. If this upper bound score is lower than the threshold of interest, we can omit the calculation of the double precision score for these particular sequences.

If the upper bound score is over the threshold, however, we have do another alignment to get the actual score, which can still be lower than the threshold. Since the vast majority of the scores is usually lower than the threshold, a substantial amount of time can be saved by using the byte or short estimation, because an estimation runs much faster than a double alignment.

To produce an upper bound by the short (or the byte) method, we have to transform the double distance matrix and the gap costs to shorts (or bytes). The transformation has to be done in a way that the future calculations will surely produce an upper bound score. Such a transformation could be $ceil(x)$ for every element, but this would result in a really imprecise upper bound score. Instead, we use $ceil(x \cdot factor)$ with $factor = UNSIGNED\_SHORT\_MAX/threshold$ for the short version, and a bit more complicated formula for the byte version. The factor should be designed in a way that it reduces the relative rounding errors, for a more precise estimation.

After the transformations, we can cram multiple elements into a single register (8 for the short version and 16 for the byte version, assuming 128-bit

SSE registers) and perform the calculation. Of course the result score has to be scaled back by the used $factor$.

To hasten the estimation further, we can create so-called profiles for a given S sequence and matrix combination. After a profile is created, we can use it for multiple alignments to the same S and matrix combination.

4 Comparison with PyCogent