IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-3, Special Issue-1 E-ISSN: 2347-2693

Performance Evaluation of BioPerl, Biojava, BioPython, BioRuby and BioSmalltalk for Executing Bioinformatics Tasks

Dipanjan Moitra*
Dept. of Management,
University of North Bengal, Darjeeling-734013.
tataijal@gmail.com

R. K. Samanta
Dept. of Computer Science & Application
University of North Bengal, Darjeeling–734013.
rksamantark@gmail.com

***Abstract*—In the recent years, Bioinformatics and computational biology are two of some important and active research disciplines. Finding insights into biology, information technology tools in the form of programming languages suitable for biology along with data mining tools and techniques are deployed. The open source programming languages used in bioinformatics are informally called Bio* projects. This work explores the performances of BioPerl, Biojava, BioPython, BioRuby, BioSmalltalk under Bio* projects for executing bioinformatics tasks.**

Keywords-Bioinformatics, Bio* projects , BioPerl, Biojava, BioPython, BioRuby, BioSmalltalk.

I. INTRODUCTION

Bioinformatics, a rapidly evolving discipline, is the application of computational tools and techniques to the management and analysis of biological data [1]. Common tasks in bioinformatics are parsing the results of an analysis program, sequence similarity searching, functional motif searching, sequence retrieval, multiple sequence alignment, restriction mapping, secondary and tertiary structure prediction, DNA analysis, literature searching, protein analysis, sequence assembly, etc. [2], [3].

Wide range of bioinformatics tasks can be accomplished by using different programming languages. These languages may be broadly classified as *script* languages (Perl, Python, Ruby, etc.) and *non-script* languages (C, C++, Java, etc.). The *non-script* group may further be classified as compiled (C, C++, etc.) and *semi-compiled* (Java, C#, etc.) languages [5]. Several studies have been carried out in order to compare these languages and such studies have revealed that the *script* group often turns out to be more productive than conventional languages [4].

Script languages and a few *semi-compiled* languages available for performing various bioinformatics tasks mostly belong to the group of Open Source Projects. It is less expensive than its commercial peer and anybody can contribute towards the development of the project [6]. The open source programming languages used in bioinformatics are informally called Bio* projects, typically pronounced with a *Bio* prefix, e.g., BioPerl, BioJava, BioPython, etc. [2], [7], [8].

Section 2 describes some leading Bio* projects. Literature review is done in section 3. Section 4 describes the objective of the study. Research methodology is described in section 5. Section 6 presents the codes used in the study. Results and discussion are presented in section 7. Limitations of the study and future scope are discussed in section 8.

II. DESCRIPTION OF LEADING BIO* PROJECTS

Bioperl, perhaps the oldest of the Bio* projects, is a group of more than 500 Perl modules having numerous bioinformatics utilities and have been written and maintained by an international group of volunteers [1], [11]. The

bioperl-live repository contains the core functionality and additional packages are for creating graphical interfaces (bioperl-gui), setting up persistent ORM storage in RDMBS (bioperl-db), running and parsing the results from hundreds of bioinformatics applications (bioperl-run), and software to automate bioinformatics analyses (bioperl-pipeline) [12], [13]. It also has data models and operations for ontologies, phylogenetic trees, genetic maps and markers and population genetics [2].

The BioJava is an open source bioinformatics project. The BioJava API's have capabilities for manipulating biological sequences, parsing common file formats, accessing to BioSQL and other databases, performing statistical analysis, and other tasks [13].

The Biopython Project is an international association of developers of freely available Python tools for computational molecular biology and life science research [14].

The BioRuby project is an open source class library for bioinformatics written in the object oriented scripting language Ruby [15]. The BioRuby library provides various methods for manipulating biological sequences, accessing biological databases, parsing database entries, executing biological analysis applications and parsing their results [13].

BioSmalltalk is a library for doing pure object-oriented bioinformatics with the Smalltalk programming system [10].

III. LITERATURE REVIEW

While coding a bioinformatics algorithm, programmers or biologists often select any of the so called Bio* projects depending upon the familiarity of the language. Several efforts have been conducted towards the benchmarking of programming tools or languages. They are mostly speculative in nature. Although a few empirical studies have been conducted so far, in order to measure the efficiencies of bioinformatics languages, they have their respective limitations. Some of them have studied the efficiency of both the script and non-script groups [4], [5]. Another few have studied the popularity of a handful of Bio* projects [8]. Some of them have a lack of homogeneity [9] and some have considered a narrow range of languages [10] and thus may create confusion in the mind of the user that which language he/she should select. These works are undoubtedly useful from their respective standpoints, there is a need to empirically study the leading Bio* projects in order to find out their merits, popularity as well as their limitations. To the best of our knowledge, there is no study comparing these five languages we consider in this paper.

IV. OBJECTIVE OF THE STUDY

The primary objective of the paper is to study the merits and limitations of the leading Bio* projects: BioPerl, Biojava, BioPython, BioRuby and BioSmalltalk. In order to accomplish the aforementioned objective, the following quantitative metrics have been considered: Lines of Code (LOC), Execution Time and Memory Usage for carrying out basic bioinformatics tasks. Another objective of the paper is to find out the popularity and maintainability of these open source projects and the following qualitative parameters have been considered in this regard: Community Support, Maintainability and User Interest.

V. RESEARCH METHODOLOGY

A basic bioinformatics task was selected in order to measure the merits of the selected Bio* languages. The task was to read a local GENBANK file having a single sequence and to convert it into an equivalent FASTA file. The task is disk I/O bound and secondly, it is independent of any external factor and it is a simple text parsing job. These characteristics make the task ideal for benchmarking. The task was then implemented using each of the concerned programming languages under same platform and configuration which were as follows:

- Operating System – Windows 8 Single Language
- Processor – Intel ® Core ™ i5-3230M CPU @ 2.6 GHz
- Installed Memory – 4 GB

- System Type – 64-bit OS, x64-based processor

In this paper, the following languages/tools and versions have been used:

i) BioPerl 1.6 with Strawberry-perl-5.18.2.2-64bit
ii) BioJava 1.3 with bytecode-0.92
iii) BioPython 1.64 with Python version 3.4.1
iv) BioRuby 1.4.3 with Ruby 2.0.0-x64
v) Pharo (BioSmalltalk 0.5)/BioPharo

In order to avoid adverse effects of other running processes, each program was executed several times and the average values regarding execution times, memory usage along with the respective LOC were recorded.

Google Trends measures how often people search for the given term relative to the total search-volume across various regions of the world [16]. The horizontal axis of the resulting graph represents time, and the vertical is how often a term is searched for relative to the total number of searches, globally. This indicates the demand for information about the particular language [18]. The scaled search volume for each language from 2004 to 2014 along with a prediction for 2015 had been measured using Google Trends [17]. This shows the interest of users for these languages and thus depicts the popularity of the concerned language.

Open Source projects can hardly sustain without a consistent Community Cooperation [20]. This qualitative metric can be indirectly measured by using the following quantitative metrics:

- Number of Committers - a committer is a developer who is able to modify the source code of a particular piece of open-source software [21].
- Number of Commits - commits record changes to the software system [22].

Another significant qualitative metric is the Maintainability [19] which may be indirectly measured by using the following quantitative metrics:

- Total Lines of Code – typically the size of the project or total physical lines.
- Comment Density - comment lines divided by total lines of code.

VI. GLIMPSES OF CODES

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your word processor, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

A. *BioPerl*

```
use Benchmark::Timer;
use Bio::Perl;
# forces genbank format
my $infilename = 'AF165912.gbk';
my $outfilename = 'outPerl.fa';
my $t = Benchmark::Timer->new();
$t->start('my_tag');
#reads an array of sequences
@seq_object_array = read_all_sequences($infilename,'genbank');
write_sequence(">$outfilename", 'fasta', @seq_object_array);
$t->stop;
print $t->report;
```

```
$waitVar = <STDIN>;
```

B. BioJava

```
import java.io.*;
import org.biojava.bio.*;
import org.biojava.bio.seq.*;
import org.biojava.bio.seq.io.*;
public class RWBiojava {
public static void main(String[] args) {
long start = System.currentTimeMillis();
try {
BufferedReader br = new BufferedReader(new FileReader("AF165912.gbk"));
String format = "GENBANK";
String alpha = "DNA";
SequenceIterator iter =
(SequenceIterator)SeqIOTools.fileToBiojava(format, alpha, br);
SeqIOTools.writeFasta(new FileOutputStream("outBiojava.fa"), iter);
long stop = System.currentTimeMillis();
System.out.println("total =\t" + (stop - start) + "(msec)");
while(true){}
}
catch (FileNotFoundException ex) {
//can't find file specified by args[0]
ex.printStackTrace();
}catch (BioException ex) {
//invalid file format name
ex.printStackTrace();
}catch (IOException ex){
//error writing to fasta
ex.printStackTrace();
}
}
}
```

C. BioPython

```
from Bio import GenBank
from Bio import SeqIO
from sys import *
import time
start = time.clock()
gb_file = "AF165912.gbk"
gb_handle = open(gb_file, 'r')
SeqIO.convert("AF165912.gbk", "genbank", "outBiopython.fa", "fasta")
end = time.clock()
print ("%f" % (end-start))
```

D. BioRuby

```
#!usr/bin/env ruby
require 'bio'
```

```
require 'benchmark'
result = Benchmark.measure do
ff=Bio::FlatFile.new(Bio::GenBank,ARGF)
f = File.new("myfile.fa", "w")
while gb=ff.next_entry
        f.puts(gb.seq.to_fasta("gb:#{gb.entry_id}
#gb.definition}",70))
end
f.close
end
puts result
sleep(10)
```

E. BioSmalltalk

```
| file x y m t l s p q e d c f b stream working |
x:=Time millisecondClockValue .
file := BioFile on: (FileStream readOnlyFileNamed: BioObject testFilesDirectoryName asFileReference /
'AF165912.gbk').

e:=file contents.
m:=e asString.
s:=m size.
t:= m findString: 'ACCESSION' startingAt: 1 caseSensitive: true.
c:=m findString: (String cr) startingAt: t caseSensitive: true.
d:=m copyFrom: t+9 to: c.
f:=d trimBoth.
l:=m findString: 'ORIGIN' startingAt: 1 caseSensitive: true.
p := m copyFrom: l+6 to: s.
q:= p asUppercase select: [:a | a isLetter or: a==(Character cr)].
q:= q trimBoth.
b:= '>',f,String crlf,q.
working := FileSystem disk workingDirectory.
stream := (working / 'test.fa') writeStream.
stream nextPutAll: b.
stream close.
y:=Time millisecondClockValue.
Transcript open.
Transcript show:(y-x);cr.
```

VII. RESULTS AND DISCUSSION

Table 1 shows the memory usage, execution time and Lines of Code (LOC) for the bioinformatics tasks as described in section 5.

Table 1: Comparison of Memory Usage, Execution Times & LOC

Language	Memory Usage	Execution Time	LOC
BioPerl	24.5 MB	295.76 ms	13
BioJava	19.0 MB	1047.00 ms	30

BioPython	11.0 MB	279.68 ms	10
BioRuby	7.7 MB	640.47 ms	14
BioSmalltalk	54.0 MB	12.00 ms	20

From the table 1, it was evident that BioRuby has had the lowest memory usage and BioPython was very close to it. BioPerl and BioJava had performed moderately. BioSmalltalk had the highest memory usage. BioSmalltalk has had the lowest execution time and no other language was a close match for it. Performances of BioPyhton and BioPerl were average. BioRuby was lagging far behind them. BioJava had the maximum execution time. BioPython had consumed the lowest LOC. BioPerl and BioRuby were close to BioPython. BioSmalltalk and BioJava had much larger LOC and among them BioJava was the largest.

From the above results, it may be said that the performance of BioPython was very consistent and it may be considered to be one of the most preferable languages for performing basic bioinformatics tasks.

It was noted that, in spite of having a larger memory usage and LOC, Biosmalltalk had the lowest execution time which was considerably less than other languages. The reason behind having a hefty size and memory consumption was that, there was no direct API support to convert a local GENBANK file to its equivalent FASTA format. However, there is an API support for online conversion and for a disk I/O based conversion only if the source file is in XML format. This fact had compelled the experiments to continue with primitive Smalltalk operations. If there were API support, there would have been a much lesser memory consumption and size.

Fig.1. shows the trends of interest of users for these languages and thus depicts the popularity of the concerned language which was obtained using Google Trends.

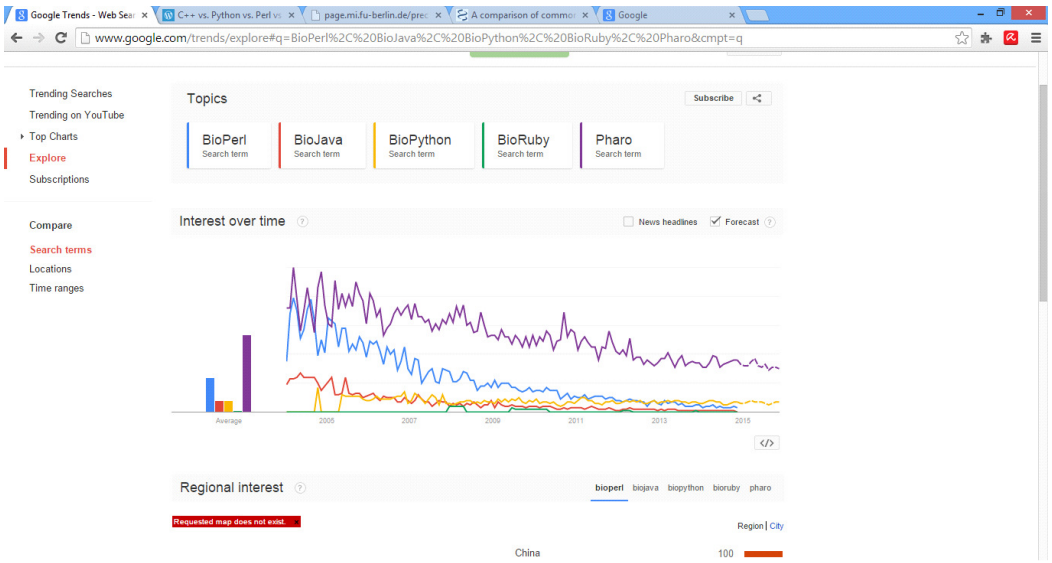

Fig 1

From fig. 1, it is found that Pharo (BioSmalltalk) had the most dominant presence in the sphere of Bio* community. Users' interest to BioPerl was also consistent, but that is quite natural for an old language like it. It was also clear that the search volume for BioJava, BioRuby was also gradually decreasing. After considering the prediction part of fig. 1, it could be said that only BioSmalltalk and BioPython might be there as far as users' interest or popularity is concerned.

Table 2: Repository Metrics Data (OpenHub)

Qualitative Parameters	Qualitative Parameters	Bio Perl	Bio Java	Bio Python	Bio Ruby	Bio Smalltalk
Community	Average no. of	17	20	27	3	15

Cooperation	Contributors per month					
	Average Commit frequency per month	301	564	606	22	687
Maintainability	Comment Density	32.8	28.8	14.6	21.1	12.7
	Total LOC as on 2014	819477	538232	368708	1279542	588822
	Percentage of blank lines to the total LOC	20.3	13.8	9.1	12.8	13.4
	Percentage of code lines to the total LOC	46.9	57.4	76.3	66.3	73.8

From the above discussion, it may be concluded that, if there is a proper API support, BioSmalltalk may be considered for performing basic bioinformatics tasks. Otherwise, BioPython may be a good alternative, at least for the beginners.

VIII. LIMITATIONS & FUTURE SCOPE

In the study, only one basic bioinformatics task has been considered. Experiments may be conducted with other bioinformatics tasks like performing disk I/O with GENBANK file having multiple sequences, finding a sequence in a GENBANK file with a locus name [9], computing the reverse complement of a DNA sequence, translating a DNA sequence to Protein, etc. The complicated bioinformatics tasks will also be considered in our future work.

REFERENCES

[1] James Tisdall, Beginning Perl for Bioinformatics, First Edition, O'Reilly, October 2001, ISBN: 0-596-00080-4

[2] Jason E.Stajich, Hilmar Lapp, "Open Source tools and toolkits for bioinformatics: significance, and where are we?", Briefings in Bioinformatics, Oxford University Press, Vol. 7 No. 3, Pp. 287-296, 2006

[3] Dat Tran, Christopher Dubay, Paul Gorman, William Hersh, "Applying Task Analysis to Describe & Facilitate Bioinformatics Tasks", MEDINFO, 2004, M. Fieschi et al. (Eds), Amsterdam: IOS Press, IMIA

[4] Lutz Prechelt, "An empirical Comparison of C, C++, Java, Perl, Python, Rexx and Tcl for a search/string-processing program", University at Karlsruhe, Technical Report 2000-5, March 10, 2000

[5] Mathieu Fourment, Michael R Gillings, "A comparison of common programming languages used in bioinformatics", *BMC Bioinformatics,* 9:82, 2008

[6] Open Source Initiative, http://www.opensource.org/, accessed December, 2014

[7] Open Bioinformatics Foundation, http://www.open-bio.org/, accessed December, 2014

[8] M. Rahmania, D. Bastola, L. Najjar, "Comparative Analysis of Software Repository Metrics in BioPerl, BioJava and BioRuby**",** International Conference on Computational Science, ICCS 2012, Procedia Computer Science 9 (2012) 518 – 521, 1877-0509, Published by Elsevier Ltd.

[9] T. Ryu, "Benchmarking of BioPerl, Perl, BioJava, Java, BioPython, and Python for primitive bioinformatics tasks and choosing a suitable language", International Journal of Contents, Vol.5, No.2, June 2000

[10] Hernán F. Morales, Guillermo Giovambattista, "BioSmalltalk: A pure object system and library for bioinformatics**",** Bioinformatics Application Note, Oxford University Press, vol. 29, No. 18, Pp. 2355-2356, 2013

[11] K. Cara Woodwark, "Meeting Review: The Bioinformatics Open Source Conference 2001 (BOSC 2001)", Comparative and Functional Genomics, 2001; 2: 327–329

[12] BioPerl Web Information, http://bioperl.org, accessed December, 2014

[13] The Open Source Network, http://openhub.net, accessed December, 2014

[14] BioPerl Web Information, http://www.biopython.org, accessed December, 2014

[15] Ruby Programming Language, http://www.ruby-lang.org, accessed December, 2014

[16] IEEE Spectrum, http://spectrum.ieee.org/ns/IEEE_TPL/methods.html, accessed December, 2014

[17] Wiki Definition of Google Trends, http://en.wikipedia.org/wiki/Google_Trends, accessed December, 2014

[18] Hyunyoung Choi, Hal Varian "Predicting the Present with Google Trends", *Google Inc, April 10, 2009*

[19] Liguo Yu, Stephen R. Schach, Kai Chen, "Measuring the Maintainability of Open-Source Software", 0-7803-9508-5/05, IEEE

[20] Wiki Definition of Open Source Community Support, http://en.wikipedia.org/wiki/Open-source_movement, accessed December, 2014

[21] Carsten Kolassa, Dirk Riehle, Michel A. Salim, "The Empirical Commit Frequency Distribution of Open Source Projects", ACM 978-1-4503-1852-5/13/08

[22] Carsten Kolassa, Dirk Riehle, Michel A. Salim, "A Model of the Commit Size Distribution of Open Source", Proceedings of the 39th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2013), LNCS 7741. Page 52-66. Springer Verlag, 2013