MG7: Configurable and scalable 16S metagenomics data analysis – new methods optimized for massive cloud computing

Alexey Alekhin [1] **Evdokim Kovach** [1] **Marina Manrique** [1] **Pablo Pareja** [1] **Eduardo Pareja** [1] **Raquel Tobes** [1] **and Eduardo Pareja-Tobes** [1,*]

[1]*Oh no sequences! Research Group, Era7 Bioinformatics, Granada, Spain*

Correspondence*:
Corresponding Author
Oh no sequences! Research Group, Era7 Bioinformatics, Plaza Campo Verde 3, Granada, 18001, Spain, eparejatobes@ohnosequences.com

2 ABSTRACT

3   No abstract yet. Will be here.

1 INTRODUCTION

5   Metagenomics data analysis is growing at exponential rate during the last years. The increasing throughput
6   of massively parallel sequencing technologies, the derived decreasing cost, and the high impact of
7   metagenomics studies, especially in human health (diagnostics, treatments, drug response, prevention), are
8   crucial reasons responsible for this growth of Metagenomics. There is a growing interest in sequencing
9   all kind of microbiomes (gut, mouth, skin, urinary tract, airway, milk, bladder), in different conditions of
10   health and disease, or after different treatments. Metagenomics is also impacting environmental sciences,
11   crop sciences, agrifood sector and biotechnology in general. This new possibilities for exploring the
12   diversity of micro-organisms in the most diverse environments is opening many new research areas but,
13   due to this wide interest, it is expected that the amount of data will be overwhelming in the short time
14   (Stephens et al., 2015).

15   Genome researchers have raised the alarm over big data in the past nature news add ref but even a more
16   serious challenge might be faced with the metagenomics boom/ upswing. If we compare metagenomics
17   data with other genomics data used in clinical genotyping we find a differential feature: the key role of
18   time. Thus, for example, in some longitudinal studies, serial sampling of the same patient along several
19   weeks (or years) is being used for the follow up of some intestinal pathologies, for studying the evolution
20   of gut microbiome after antibiotic treatment, or for colon cancer early detection (Zeller et al., 2014).
21   This need of sampling across time adds more complexity to metagenomics data storage and demands
22   adapted algorithms to detect state variations across time as well as idiosyncratic commonalities of the
23   microbiome of each individual (Franzosa et al., 2015). In addition to the intra-individual sampling-time
24   dependence, metagenomic clinical test results vary depending on the specific region of extraction of
25   the clinical specimen. This local variability adds complexity to the analysis since different localizations
26   (different tissues, different anatomical regions, healthy or tumour tissues) are required to have a sufficiently

27 complete landscape of the human microbiome. Moreover, reanalysis of old samples using new tools and
28 better reference databases might be also demanded from time to time.

29   During the last years other sciences as astronomy or particle physics are facing the big data challenge
30 but, at least, these science have standards for data processing (Stephens et al., 2015). Global standards
31 for converting raw sequence data into processed data are not yet well defined in metagenomics and
32 there are shortcomings derived from the fact that many bioinformatics methodologies currently used for
33 metagenomics data analysis were designed for a scenario very different that the current one. These are
34 some of the aspects that have suffered crucial changes and advances with a direct impact in metagenomics
35 data analysis. i. The first aspect is related to the sequences to be analyzed: the reads are larger, the
36 sequencing depth and the number of samples of each project are considerably bigger. The first metagenomics
37 studies were very local projects, while nowadays the most fruitful studies are done at a global level
38 (international, continental, national). This kind of global studies has yielded the discovery of clinical
39 biomarkers for diseases of the importance of cancer, obesity or inflammatory bowel diseases and has
40 allowed exploring the biodiversity in many earth environments ii. The second aspect derives from the
41 impressive genomics explosion, its effect being felt in this case in the reference sequences. The immense
42 amount of sequences available in public repositories demands new approaches in curation, update and
43 storage for metagenomics reference databases: current models will or already have problems to face
44 the future avalanche of metagenomic sequences. iii. The third aspect to consider for metagenomics data
45 analysis is related to the appearance of new models for massive computation and storage and to the
46 new programming methodologies (Scala, . . .) and new cloud models and resources. The immense new
47 possibilities that these advances offer must have a direct impact in the metagenomics data analysis. iv.
48 And finally the new social manner to do science, and especially genomic science is the fourth aspect
49 to consider. Metagenomics evolves in a social and global scenario following a science democratization
50 trend in which many small research groups from distant countries share a common big metagenomics
51 project. This global cooperation demands systems allowing following exactly the same pipelines using
52 equivalent cloud resources to modularly execute the analysis in an asynchronous way of working between
53 different groups. This definitively new scenario demands new methods and tools to handle the current and
54 future volume of metagenomic data with the sufficient speed of analysis. Considering all these aspects we
55 have designed a new open source methodology for analyzing metagenomics data that exploits the new
56 possibilities that cloud computing offers to get a system robust, programmatically configurable, modular,
57 distributed, flexible, scalable and traceable in which the biological databases of reference sequences can be
58 easily updated and/or frequently substituted by new ones or by databases specifically designed for focused
59 projects.

2 RESULTS

2.1 Overview

61   To tackle the challenges posed by metagenomics big data analysis outlined in the Introduction,

62   - AWS resources in Scala (??) - A new approach to data analysis specification, management and
63 specification based on working with it in exactly the same way as for a software project, together with the
64 extensive use of compile-time structures and checks. - Parallelization and distributed analysis based on
65 AWS, with on-demand infrastructure as the basic paradigm - fully automated processes, data and cloud
66 resources management. - Static reproducible specification of dependencies and behavior of the different
67 components using *Statika* and *Datasets* - Definition of complex pipelines using *Loquat* a composable

68 system for scaling/parallelizing stateless computations especially designed for Amazon Web Services
69 (AWS) - Modeling of the taxonomy tree using the new paradigm of graph databases (Bio4j). It facilitates
70 the taxonomic assignment tasks and the calculation of the taxa abundance values considering the hierarchic
71 structure of taxonomy tree (cumulative values). - per-read assignment (??)

2.2   16S Reference Database Construction

73 Our 16S Reference Database is a curated subset of sequences from NCBI nucleotide database **nt**. The
74 sequences included were selected by similarity with the bacterial and archaeal reference sequences
75 downloaded from the **RDP database** (Cole et al., 2013). RDP unaligned sequences were used to
76 capture new 16S sequences from **nt** using BLAST similarity search strategies and then, performing
77 additional curation steps to remove sequences with poor taxonomic assignments to taxonomic nodes
78 close to the root of the taxonomy tree. All the nucleotide sequences included in **nt** database has a
79 taxonomic assignment provided by the **Genbank** sequence submitter. NCBI provides a table (available
80 at ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/) to do the mapping of any Genbank Identifier (GI) to its
81 Taxonomy Identifier (TaxID). Thus, we are based on a crowdsourced submitter-maintained taxonomic
82 annotation system for reference sequences. It supposes a sustainable system able to face the expected
83 number of reference sequences that will populate the public global nucleotide databases in the near future.
84 Another advantageous point is that we are based on NCBI taxonomy, the *de facto* standard taxonomic
85 classification for biomolecular data (Cochrane and Galperin, 2010). NCBI taxonomy is, undoubtedly, the
86 most used taxonomy all over the world and the most similar to the official taxonomies of each specific field.
87 This is a crucial point because all the type-culture and tissue databanks follow this official taxonomical
88 classification and, in addition, all the knowledge accumulated during last decades is referred to this
89 taxonomy. In addition NCBI provides a direct connection between taxonomical formal names and the
90 physical specimens that serve as exemplars for the species (Federhen, 2014).

91 Certainly, if metagenomics results are easily integrated with the theoretical and experimental knowledge
92 of each specific area, the impact of metagenomics will be higher that if metagenomics progresses as a
93 disconnected research branch. Considering that metagenomics data interoperability, which is especially
94 critical in clinical environments, requires a stable taxonomy to be used as reference, we decided to rely on
95 the most widely used taxonomy: the NCBI taxonomy. In addition, the biggest global sequence database
96 GenBank follows this taxonomy to register the origin of all their submitted sequences. Our 16S database
97 building strategy allows the substitution of the 16S database by any other subset of **nt**, even by the complete
98 **nt** database if it would be needed, for example, for analyzing shotgun metagenomics data. This possibility
99 of changing the reference database provides flexibility to the system enabling it for easy updating and
100 project-driven personalization.

2.3   Bio4j and Graph Databases

2.4   MG7 Pipeline Description

2.5   Taxonomic Assignment Algorithms

2.5.1   Lowest Common Ancestor based Taxonomic Assignment

105 For each read:

106 1. Select only one BLASTN alignment (HSP) per reference sequence (the HSP with lowest e value) 2.
107 Filter all the HSPs with bitscore below a defined BLASTN bitscore threshold s_0 3. Find the best bitscore
108 value S in the set of BLASTN HSPs corresponding to hits of that read 4. Filter all the alignments with

109 bitscore below p * S (where p is a fixed by the user coefficient to define the bitscore required, e.g. if p=0.9
110 and S=700 the required bitscore threshold would be 630) 5. Select all the taxonomic nodes to which map
111 the reference sequences involved in the selected HSPs: - If all the selected taxonomic nodes forms a line
112 in the taxonomy tree (are located in a not branched lineage to the tree root) we should choose the most
113 specific taxID as the final assignment for that read - If not, we should search for the (sensu stricto) Lowest
114 Common Ancestor (LCA) of all the selected taxonomic nodes (See Figure X)

115 In this approach the value used for evaluating the similarity is the bitscore that is a value that increases
116 when similarity is higher and depends a lot on the length of the HSP

117 ### 2.5.2   Best BLAST hit taxonomic assignment

118 We have maintained the simpler method of Best BLAST Hit (BBH) taxonomic assignment because, in
119 some cases, it can provide information about the sequences that can be more useful than the obtained using
120 LCA algorithm. Using LCA algorithm when some reference sequences with BLAST alignments over the
121 required thresholds map to a not sufficiently specific taxID, the read can be assigned to an unspecific taxon
122 near to the root. If the BBH reference sequence maps to a more specific taxa this method, in that case, gives
123 us useful information.

124 ## 2.6   Using MG7 with some example data-sets

125 We selected the datasets described in [Kennedy-2014] (??)

126 ## 2.7   MG7 availability

127 MG7 is open source, available at https://github.com/ohnosequences/mg7 under an AGPLv3 license.

3   DISCUSSION

128 ### 3.1   What MG7 brings

129 We could summarize the most innovative ideas and developments in MG7:

130 1. Treat data analysis as a software project. This makes for radical improvements in *reproducibility*, *reuse*,
131 *versioning*, *safety*, *automation* and *expressiveness*
132 2. input and output data, their locations and type are expressible and checked at compile-time using our
133 Scala library *datasets* 3. management of dependencies and machine configurations using our Scala
134 library *Statika*
135 3. automation of AWS cloud resources and processes, including distribution and parallelization through
136 the use of *Loquat*
137 4. taxonomic data and related operations are treated natively as what they are: graphs, through the use of
138 *Bio4j*
139 5. MG7 provides a sustainable model for taxonomic assignment, appropriate to face the challenging
140 amount of data that high throughput sequencing technologies generate

141 We will expand on each item in the following sections.

142 ### 3.2   A new approach to data analysis

143 MG7 proposes to define and work with a particular data analysis task as a software project, using Scala.
144 The idea is that *everything*: data description, their location, configuration parameters, the infrastructure

145 used, . . . should be expressed as Scala code, and treated in the same way as any (well-managed) software
146 project. This includes, among other things, using version control systems (`git` in our case), writing tests,
147 making stable releases following semantic versioning or publishing artifacts to a repository.

148   What we see as key advantages of this approach (when coupled with compile-time specification and
149 checking), are

150 • **Reproducibility** the same analysis can be run again with exactly the same configuration in a trivial
151   way.
152 • **Versioning** as in any software project, there can be different versions, stable releases, etc.
153 • **Reuse** we can build standard configurations on top of this and reuse them for subsequent data analysis.
154   A particular data analysis *task* can be used as a *library* in further analysis.
155 • **Decoupling** We can start working on the analysis specification, without any need for available data in
156   a much easier way.
157 • **Documentation** We can take advantage of all the effort put into software documentation tools and
158   practices, such as in our case Scaladoc or literate programming. As documentation, analysis processes
159   and data specification live together in the files, it is much easier to keep coherence between them.
160 • **Expresiveness and safety** For example in our case we can choose only from valid Illumina read
161   types, and then build a default FLASH command based on that. The output locations, being declared
162   statically, are also available for use in further analysis.

163 ## 3.3   Inputs, outputs, data: compile-time, expressive, composable

164 ## 3.4   Tools, data, dependencies and machine configurations

165 ## 3.5   Parallel cloud execution ??

166 ## 3.6   Taxonomy and Bio4j

167   The hierarchic structure of the taxonomy of the living organisms is a tree, and, hence, is also a graph
168 in which each node, with the exception of the root node, has a unique parent node. It led us to model the
169 taxonomy tree as a graph using the graph database paradigm. Previously we developed Bio4j **[Pareja-**
170 **Tobes-2015]**, a platform for the integration of semantically rich biological data using typed graph models.
171 It integrates most publicly available data linked with sequences into a set of interdependent graphs to be
172 used for bioinformatics analysis and especially for biological data.

173 ## 3.7   Future-proof

174 ## 3.8   MG7 Future developments

175   Other possible uses of the general schema: statika, loquat, . . .

176 ### 3.8.1   Shotgun metagenomics

177   It is certainly possible to adapt MG7 to work with shotgun metagenomics data. Simply changing the
178 reference database to include whole genome sequence data could yield interesting results. This could
179 also be refined by restricting reference sequences according to all sort of criteria, like biological function
180 or taxonomy. Bio4j would be an invaluable tool here, thanks to its ability to express express complex
181 predicates on sequences using all the information linked with them (GO annotations, UniProt data, NCBI
182 taxonomy, etc).

183   3.8.2   Comparison of groups of samples

184   3.8.3   Interactive visualizations using the output files of MG7 (Biographika project)

4   MATERIALS AND METHODS

185   ## 4.1   Amazon Web Services

186   ## 4.2   Scala

187   Scala is a hybrid object-functional programming language which runs on Java Virtual Machine. It
188   has support for type-level programming, type-dependent types (through type members) and singleton
189   types, which permits a restricted form of dependent types where types can depend essentially on values
190   determined at compile time (through their corresponding singleton types). Conversely, through implicits
191   one can retrieve the value corresponding to a singleton type.

192   The other key feature for us is Java interoperability, which let us build on the vast number of existing
193   Java libraries; we take advantage of this when using Bio4j as an API for the NCBI taxonomy.

194   MG7 itself and all the libraries used are written in Scala `2.11`.

195   ## 4.3   Statika

196   MG7 uses ohnosequences/statika 2.0.0 for specifying the configuration and behavior of EC2 instances.
197   Statika is a Scala library developed by the first and last authors which serves as a way of defining and
198   composing machine behaviors statically. The main component are **bundles**. Each bundle declares a
199   sequence of computations (its behavior) which will be executed in an **environment**. A bundle can *depend*
200   on other bundles, and when being executed by an environment, its DAG of dependencies is linearized and
201   run in sequence. In our use, bundles correspond to what an EC2 instance should do and an environment to
202   an image (AMI: **A**mazon **M**achine **I**mage) which prepares the basic configuration, downloads the Scala
203   code and runs it.

204   MG7 uses ohnosequences/statika 2.0.0.

205   ## 4.4   Datasets

206   Datasets is a Scala library developed by the first and last authors to declare datasets and their locations.
207   **Data** is represented as type-indexed fields: Keys are modeled as singleton types, and values correspond to
208   what could be called a denotation of the key: a value of type `Location` tagged with the key type. Then a
209   **Dataset** is essentially a collection of data, which are guaranteed statically to be different through type-level
210   predicates, making use of the value type correspondence which can be established through singleton types
211   and implicits. A dataset location is then just a list of locations formed by locations of each data member of
212   that dataset. All this is based on what could be described as an embedding in Scala of an extensible record
213   system with concatenation on disjoint labels, in the spirit of (Harper and Pierce, 1990, Harper and Pierce
214   (1991)). For that *Datasets* uses ohnosequences/cosas 0.7.0.

215   Data keys can further have a reference to a **data type**, which, as the name hints at, can help in providing
216   information about the type of data we are working with. For example, when declaring Illumina reads as a
217   data, a data type containing information about the read length, insert size or end type (single or paired) is
218   used.

219   A **location** can be, for example, an S3 object or a local file; by leaving the location type used to denote
220   particular data free we can work with different "physical" representations, while keeping track of to which
221   logical data they are a representation of. Thus, a process can generate locally a `.fastq` file representing
222   the merged reads, while another can put it in S3 with the fact that they all correspond to the "same" merged
223   reads is always present, as the data that those "physical" representations denote.

224   MG7 uses ohnosequences/datasets 0.2.0.

4.5   Loquat

226   Loquat is a library developed by the first, second and last authors designed for the execution of
227   embarrassingly parallel tasks using S3, SQS and EC2.

228   A **loquat** executes a process with explicit input and output datasets (declared using the *Datasets* library
229   described above). Workers (EC2 instances) read from an SQS queue the S3 locations for both input and
230   output data; then they download the input to local files, and pass these file locations to the process to be
231   executed. The output is then put in the corresponding S3 locations.

232   A manager instance is used to monitor workers, provide initial data to be put in the SQS queue and
233   optionally release resources depending on a set of configurable conditions.

234   Both worker and manager instances are Statika bundles. In the case of the worker, it can declare any
235   dependencies needed to perform its task: other tools, libraries, or data.

236   All configuration such as the number of workers or the instance types is declared statically, the
237   specification of a loquat being ultimately a Scala object. There are deploy and resource management
238   methods, making it easy to use an existing loquat either as a library or from (for example) a Scala REPL.

239   The input and output (and their locations) being defined statically has several critical advantages. First,
240   composing different loquats is easy and safe; just use the output types and locations of the first one as input
241   for the second one. Second, data and their types help in not mixing different resources when implementing
242   a process, while serving as a safe and convenient mechanism for writing generic processing tasks. For
243   example, merging paired-end Illumina reads generically is easy as the data type includes the relevant
244   information (insert size, read length, etc) to pass to a tool such as FLASH.

245   MG7 uses ohnosequences/loquat 2.0.0.

4.6   Type-safe EDSLs for BLAST and FLASH

247   We developed our own Scala-based type-safe EDSLs (Embedded Domain Specific Language) for FLASH
248   and BLAST expressions and their execution.

4.6.1   BLAST EDSL

250   In the case of BLAST we use a model for expressions where we can guarantee for each BLAST command
251   expression at compile time

252   • all required arguments are provided
253   • only valid options are provided
254   • correct types for each option value
255   • valid output record specification

256 Generic type-safe parsers returning an heterogeneous record of BLAST output fields are also available,
257 together with output data defined using *Datasets* which have a reference to the exact BLAST command
258 options which yielded that output. This let us provide generic parsers for BLAST output which are
259 guaranteed to be correct, for example.

260 MG7 uses ohnosequences/blast 0.2.0.

4.6.2   FLASH EDSL

262 In the same spirit as for BLAST, we implemented a type-safe EDSL for FLASH expressions and their
263 execution, sporting features equivalent to those outlined for the BLAST EDSL.

264 MG7 uses ohnosequences/flash 0.1.0.

4.7   Bio4j

266 Bio4j is a data platform integrating data from different resources such as UniProt or GO in a graph data
267 paradigm. We use the module containing the NCBI Taxonomy, and the use their Java API from Scala in the
268 assignment phase.

269 MG7 uses bio4j/bio4j 0.12.0-RC3 and bio4j/bio4j-titan 0.4.0-RC2.

5   ACKNOWLEDGEMENTS

REFERENCES

271 Cochrane, G. R. and Galperin, M. Y. (2010). The 2010 nucleic acids research database issue and online
272   database collection: a community of data resources. *Nucleic acids research* 38, D1–D4
273 Cole, J. R., Wang, Q., Fish, J. A., Chai, B., McGarrell, D. M., Sun, Y., et al. (2013). Ribosomal database
274   project: data and tools for high throughput rrna analysis. *Nucleic acids research* , gkt1244
275 Federhen, S. (2014). Type material in the ncbi taxonomy database. *Nucleic acids research* , gku1127
276 Franzosa, E. A., Huang, K., Meadow, J. F., Gevers, D., Lemon, K. P., Bohannan, B. J., et al. (2015).
277   Identifying personal microbiomes using metagenomic codes. *Proceedings of the National Academy of
278   Sciences* , 201423854
279 Harper, R. and Pierce, B. (1991). A record calculus based on symmetric concatenation. In *Proceedings of
280   the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (ACM), 131–142
281 Harper, R. W. and Pierce, B. C. (1990). Extensible records without subsumption
282 Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., et al. (2015). Big data:
283   Astronomical or genomical? *PLoS Biol* 13, e1002195
284 Zeller, G., Tap, J., Voigt, A. Y., Sunagawa, S., Kultima, J. R., Costea, P. I., et al. (2014). Potential of fecal
285   microbiota for early-stage detection of colorectal cancer. *Molecular systems biology* 10, 766