Prediction files merging script and proteins similarities search

Axel Giottonini

1 Introduction

From the invention of the first *DNA sequencing method* in the seventies, a lot of progress have been made to reach nowadays high-throughput methods, which allows the sequencing of up to billion bases sequences in a reasonable time. The biologists thus have been able to produce a massive amount of data. [5]
When thinking about this data, two major problematic appear. The first one is to answer the question *"How to store the data?"*. To face this problem, various file formats were created allowing the storage of a simple sequence, for example with *FASTA files*, up to a complete genome with its annotations, with the *EMBL* or *GenBank* specification. Along with the file formats, different databases were founded allowing searching with simple queries what interested us.
The second issue to face is the analysis of the data. As it would be impossible to analyse all the raw data with classical biological methods, tools had to be created to produce descriptions of the sequences. Thus programs predicting the gene position were developed, such as *Maker*, and once one was able to predict a gene other software, such as *Pannzer*, permitting the gene annotation were produced. [11] [3] [10]

In this project, we focused on the output of both programs mentioned above. From a *FASTA file* containing the draught genome of a species, *Maker* first predicts the genes present after what *Pannzer* provides a functional description coupled with GO terms of the genes. The GO terms are structured in the GO database, where they are divided into three sub-ontologies : *Molecular Function*, *Cellular Component* and *Biological Process*. Not only can a hierarchical relationship be observed between the terms, but also specific relationships with each other such as *is a*, *part of*, *has part* or *regulates* [9]

The project was divided in two parts. We first aimed to provide a script for merging the output of *Maker* and *Pannzer* into a single *EMBL entry*. Once this step was done, we tried to figure out a method to find proteins with similar function between two species.

1.1 Files merging

1.1.1 Aims

As mentioned previously, we aimed to create a tool for merging the initial FASTA file with the outputted files from *Maker* and *Pannzer* into an *EMBL entry* file.

Tools for converting *GFF3 files*, which is the specification of the *Maker* output, into *EMBL entries* already exist. [20] We thus had the choice between two methods. The first approach would have been to convert the *Pannzer* output into a *GFF3 file*, afterward we could have merged both outputted files into a single one and converted it into an *EMBL entry*. The second approach, which was chosen, was to use the *Biopython* module to create a script parsing and reading both outputted files to create a step-by-step *EMBL* record.

1.1.2 Files Format

In the last paragraphs various file formats were evoked. We provide below a summary of their specification focusing on the elements that were used to create the script.
As the writing of the *EMBL entries* is managed by *Biopython* and as its specification would probably require a whole report just to introduce it, we will not develop this point. For anyone that could still be interested in it, its documentation could be found online [21].

FASTA Format As evoked in the beginning of the introduction, the *FASTA format* serves to store sequences with a single line description. It is a really basic format where each entry is composed of two lines. The first line prefixed with a greater ">" char contains a minimalist description of the sequence when the second line contains the sequence. It is also possible to split the second line in multiple lines to produce a file easier to read for humans. [18]
Below is an example of an entry to illustrate the file format.

```
>seq1
ATGCTTAGATATTGAGATGGTGGTGAATGATATTG...
```

```
>seq2 (human friendly)
GGCTAGTGTCAAAACGTACGTATACTATGAACCTAATC
AGCTAAAAAAACGCGCGATCGTGCGATGCAAGATCAAG
GCGATTGAGATGGTGGATGCAAGATACTGGCTAGTAAA
...
```

Maker Output (GFF3) The *GFF3 format* serves to identify features, e.g. exons or CDS, in a sequence. It is a tabular format with nine columns: the sequence identifier (0), the source (1), the feature type (2), the start position (3), the end position (4), the score (5), the strand (6), the phase (7) and finally an array of attributes (8). In order to filter the data, the sequence identifier and a more precise identifier in the attributes array are required. The columns necessary for the creation of the EMBL entry are the start and end position, which takes numerical values, and the strand column, which takes a plus sign or minus sign depending on the strand orientation. Dots may be present in columns where the field value has not been specified. [13] Here is a example of such file structure, with in the first line the column referenced in our previous explanation.

```
0   1 2      3 4   5 6 7 8
seq . contig 1 112 . . . ID=seq...
seq . gene   16 70 . + . ID=gen...
seq . mRNA   16 70 . + . ID=mRN...
seq . exon   16 20 . + . ID=exo...
seq . CDS    16 20 . + 0 ID=cds...
```

PANNZER Output Pannzer produces two principal files. One with the functional description of the coding region and the other one with the GO terms related to the regions.
On the first attempt for creating the conversion script, we tried working with both files, which resulted in script creating an *EMBL file* with redundant and incoherent data. After a more precise analysis of the inputted files, we observed that they contained different description for the same region with different likelihood. Even if it had been possible to filter the data and thus remove the redundant data, Pannzer provides us a summarised file containing the likeliest annotation between the two main files, removing an unnecessary filtering task.
The summary file has a tabular structure with six columns : the gene identifier, the annotation type, the annotation score, the positive predictive value, the annotation identifier and the description of the annotation. We can enumerate six types of annotation : the original description (*original_DE*), which contains general information about the output of the gene and is unique for the gene, the sequence type (*qseq*), which give us the translated sequence of the protein, the three *GO* sub-ontology types (*MF_ARGOT*, *CC_ARGOT* and *BP_ARGOT*), which provides the *GO terms* linked to the gene and finally an enzymatic entry (*EC_ARGOT*), which provides the *KEGG* identifier related to genes.

```
seq original_DE euk n.d. -0.53 protein ...
seq qseq n.d. n.d. n.d. MSSKATKNAPEGKKS...
seq DE  1.50 0.70 0.7 Very-long-chain (...
seq MF_ARGOT 14.44 0.84 0102345 3-hydro...
seq EC_ARGOT 14.44 0.84 EC:4.2.1.134 GO...
seq CC_ARGOT 7.48 0.70 0005789 endoplas...
seq BP_ARGOT 7.18 0.69 0006633 fatty ac...
```

1.2 Protein homology

1.2.1 Aims

As mentioned earlier, the second part of the project was to search for protein with an analogous role in two mechanisms related to blood feeding.
The first element to demonstrate an analogy between two proteins is a sequence similarity search, which shows distantly related proteins. The second element we tried to use was the *GO term* array linked to proteins. [1]
In order to use the annotation of the protein, we aimed to create a workflow composed of different scripts and tools to link one species proteins found in the literature to proteins present in the other species.

1.2.2 Species of Interest

For this step, we took our interest on two *Diptera* : *Phlebotomus papatasi*, commonly named sand fly, and *Glossina morsitans*, which is also known as Tsetse fly.
From the literature about *Phlebotomus papatasi*, we created a list of proteins related to its salivary secretion when for *Glossina morsitans*, we focused on the protein having a role in the iron metabolism. [7]

1.2.3 Blood feeding

Iron metabolism In order to minimise the number of visits to their hosts, *Diptera* commonly ingests blood volume several times their weight. The degradation of blood proteins generate molecules like amino acids, iron and haeme in huge concentrations, huge enough to become toxic. [6] Iron homoeostasis has thus a key role in the survival of blood-feeding arthropods. Thus, iron regulatory proteins, which binds to the IRE of the UTR of mRNA coding for proteins that takes part in the iron metabolism, are important for detoxification of the insects. [4] For our research, we used a set of proteins emphasised by *Jalali Sefid Dashti et al.*, as known to have orthologs in various other insect species.

	Protein ID	AC number
Iron metabolism proteins	GMOY000584	D3TN42
	GMOY000853	A0A1B0FBD9
	GMOY001045	D3TML8
	GMOY001046	A0A1B0FBW3
	GMOY001347	A0A1B0FCQ2
	GMOY001475	A0A1B0FD25
	GMOY001551	A0A1B0FD91
	GMOY001601	A0A1B0FDD6
	GMOY002533	A0A1B0FFY8
	GMOY003206	A0A1B0FHF5
	GMOY003449	A0A1B0FI66
	GMOY003491	A0A1B0FI81
	GMOY004282	A0A1B0FKD9
	GMOY004296	A0A1B0FKA5
	GMOY004905	A0A1B0FM38
	GMOY005336	A0A1B0FN70
	GMOY005442	A0A1B0FNK0
	GMOY005513	A0A1B0FNN8
	GMOY005545	A0A1B0FNR9
	GMOY006327	A0A1B0FR36
	GMOY006724	A0A1B0G0F8
	GMOY006808	A0A1B0G0N3
	GMOY007858	A0A1B0G3F7
	GMOY007975	A0A1B0G3S6
	GMOY008151	A0A1B0G4A4
	GMOY008502	Q2PYZ6
	GMOY008535	A0A1B0G5D9
	GMOY008920	Q0QHL5
	GMOY010282	A0A1B0GAE7
	GMOY011648	A0A1B0GEC4
	GMOY011720	A0A1B0GEJ7
	GMOY000357	A0A1B0FA33
	GMOY001186	A0A1B0FC91
	GMOY003300	A0A1B0FHP0
	GMOY005208	A0A1B0FMY4
	GMOY006619	A0A1B0FRL3
	GMOY006809	A0A1B0G0N4
	GMOY007718	A0A1B0G316
	GMOY008670	A0A1B0G5S4
	GMOY009423	A0A1B0G7Y4
	GMOY009591	D3TRE3
	GMOY010018	A0A1B0G9N1
	GMOY011894	A0A1B0GF23

Salivary proteins To optimise the blood intake, blood-feeding insects have developed proteins affecting various mammalian phenomenons such as coagulation cascade and platelet activity, vasodilatation . [7] The set of proteins used in our research was highlighted by *Flanley et al.* research on the salivary protein diversity. [12]

	Protein ID	AC number
Salivary proteins	SP12	Q95WE5
	SP14	Q95WE4
	SP28	Q95WE2
	SP29	M1JB47
	SP32	Q95WE0
	SP36	M1JB52
	SP42	Q95WD9
	SP44	Q95WD8

2 Tools and Methods

2.1 Unix Commands

Although the project was mainly coded in *Python*, we decided to use *Unix command grep* to filter the data. This choice was made under the fact that programming something similar to grep or even various filters responsible for each type of file would have taken much more time and could probably not be as efficient as the *grep command*. Another option could have been to read and parse all the data to create an array of objects. Although this option may have been much more elegant, we may have faced memory overflow that is why we renounced to it. [24]
To take advantage of the power of *Unix*, we used the *subprocess* package from *Python* which allows executing command with a single line of code such as

```
subprocess.call(cmd, shell=True)
```

2.2 Biopython

Biopython is a rich library that provides multiple tools for bioinformatics. On our project we worked with the classes related to sequence annotation such as *SeqRecord*, *SeqFeature* and *SeqIO*. [17] [2]

2.2.1 SeqRecord

In order to create a *SeqRecord* object, two information are required: a sequence and an identifier, which can both be found in a *FASTA entry*. The *SeqRecord* class allows linking these elements with various secondary data such as a name, a description, features, etc. Only the features, stored as a list of *SeqFeature*, will be required for our script. [16] We present here an example of the initialisation of the *SeqRecord object*.

```
myRecord = SeqRecord(
    seq = Seq("ATCG..."),
    id = "my_id"
)
```

2.2.2 SeqFeature

As mentioned above, the SeqRecord contains a list of *SeqFeature*. This object is the main part of a sequence description as it aims to encapsulate as much information as possible about parts of the sequence. A *SeqFeature* object is composed of three fields: the type, the location and a dictionary of qualifiers.
The type field can take different values. In our case we worked with the *gene*, *mRNA*, *CDS*, *3'UTR*, *5'UTR* and *exon* types. The location field takes a *Featurelocation* or *CompoundLocation* object as value, depending on whether the sequence is continuous or not. Both the type field and the location field can be initialised with the Maker output. Finally, the qualifiers field, which is a dictionary containing multiple possible fields may be filled with the Pannzer summary file, where for example the *GO terms* will be inserted in a list accessible with the *db_xref* key. [14]
Here is an example of the initialisation of a *SeqFeature* object and its appending to the previous *SeqRecord* object.

```
myFeature = SeqFeature(
    location = FeatureLocation(
        1010,
        2424,
        -1,
    ),
    qualifiers = {
        "gene":"my_gene",
        "note":list()
    },
    type = "gene"
)
myRecord.features.append(myFeature)
```

2.2.3 SeqIO

The last relevant module of *Biopython* library is the *SeqIO* package. This package contains modules to read specific format but also to print records in a desired format. We thus avoided creating a writer for the *EMBL* format, what would have been a complex task considering the format specification.
As one may observe, the SeqIO package does not apply a strict validation of the output. To remedy this weakness we used an *EMBL validator* to verify each printed record. [15]

2.3 Merging Workflow

Our script requires four input files. The first one is the FASTA file containing all the sequences analysed by Pannzer and Maker. The second and third are the files outputted by the previously evoked programs and the last file is a description of the project. This file is used to provide the data that cannot be retrieved from the previous files. It is a tabular file using colon as field delimiter written as follows :

```
PROJECT:<project>
DIVISION:<division>
TAXONOMY:<taxonomy>
ORGANISM:<organism>
MOLECULE_TYPE:<molecule_type>
TOPOLOGY:<topology>
DESCRIPTION:"<description>"
TRANSL_TABLE:<transl_table>
```

The main behaviour of our script is to loop through the *FASTA file* provided to create its corresponding *SeqRecord* object, to retrieve all the information of the record in the *Pannzer* and *Maker* output and finally to print the resulting record in the *EMBL format* in an output file. As mentioned before, we loop through the *FASTA file*. To do so, we have to read the couple of lines containing the sequence identifier and the sequence. With the information contained in the description of the project and the *FASTA entry*, we initialise the *SeqRecord* object. Then, with the sequence identifier and the *grep* command we proceed to the creation of four files related to our sequence: a file containing all the predictions from *Maker*, a file containing only the exon predictions, a file containing all the annotations from *Pannzer* and finally a file containing all the subsequence identifier. We then iterate through the subsequence file so that we create new record features. When the iteration begins, we aim to create a more accurate filtering of the previously created files. We thus create a file containing the predictions for the subsequence and one containing the annotations.
Following the creation of the files, we iterate through them. The subsequence predictions allow us the creation of various features, such as *gene*, *mRNA*, *CDS* and the 3 prime and 5 prime *UTR*, when the annotation file allows us to complete the qualifiers for the *CDS* features.
The features are then added to the record and we escape the second loop. Following this step, an iteration through the exon prediction allows us to append the exon features to the record.
The final step consists of outputting the record with the help of the *SeqIO* module. As a last step, we verify that the outputted file respects the EMBL validation with an EMBL validator [8].

To get more in the details, the whole script is accessible on github at this link : https://bit.ly/3rMY3Iq.

2.4 BLASTP

BLAST provides a library of tools to search similar regions between sequences, so that genes can be linked

with functional and evolutionary relationships.
BLASTP allows the research between protein sequences. We used the web interface with an expected threshold of 10^{-5} and a word size of 3. For the scoring parameters, we used the *BLOSUM62* matrix and set the gap cost parameter to *existence:1 extension: 1* as it was the default parameters. [19]

2.5 Psi-Search

Such as *BLAST* tools, *Psi-Search* is a tool to find out relationships between protein sequences. [22]
We used the web client with the *UniProtKB Arthropoda* database and a cut-off of 10^{-3}

2.6 Protein Research Workflow

Starting from the list of proteins of interest from the literature, we imagined two search methods to link the proteins from one genome to the other one.

In a first time, we searched for related proteins between our two species with a sequence similarity search using the *Psi-Search* engine. Once the *Psi-Search* was made we used *BLASTP* to link the proteins from initial list to the one predicted in our genome and the protein predicted by *Psi-Search* to the other genome.

In a second time, we searched for links between proteins using the *Go terms*. We started using BLASTP to link the list of proteins of interest from the literature with the one predicted in our genome. This step gave us a graph between our list of protein of interest and the whole list of protein from the initial genome.
Once we retrieved all the proteins aimed to create a list containing each *GO term* related to our protein. For this step, we started with the list of protein previously obtained and the genome related to this list. Reading the genome file with *BioPython*, we selected the features corresponding to the proteins of interest and then extended our list with the *GO terms* linked to it. Here we obtained another graph linking proteins with their *GO terms*.
From the dictionary of *GO terms*, we then wanted to link each term to the proteins of the other genome which were described by it. To do so, we once again read the genome file with *BioPython* and for each feature, we searched if its *GO terms* were found in our lists. We thus got a third graph putting in relation the *GO terms* found in one species with the proteins predicted in another species. As mentioned in the three steps, we obtained three graphs that linked various elements of the genomes. Put together, these graphs should allow linking protein from one species to another species based on their description.
In order to visualise the results, we summarised all the data obtained in a *JSON object* that contained maps linking proteins and *GO terms* to identifiers and the previously evoked graphs. The choice of the *JSON object* was motivated by our better ability to work with it and the fact its usage could fit with multiple languages.

3 Results and interpretation

3.1 Files merging

We analysed our script with the *cProfile* module from *Python* and summarised the result in the table below.

Testing the script with *Phlebotomus papatasi* prediction and annotation files using one *CPU* required *4559.7* seconds but it would run faster without using the *cProfile* module. [23]
Looking at the various iterator, we see that the *fastaIterator* calls are equal to the number of entries in the input *FASTA file* (1554 lines for 777 entries). The *gffIterator* is called 184754 times when the number of entries is nearly ten times higher, what is the result of ignoring various types of entry, such as *match_part* or *protein_match*, which are not required for the creation of the record. Finally when observing the *annotationIterator* function, we see that we have a slightly higher number of calls than the number of entries (565498). That point could be improved by trying to keep in memory the data instead of the file creation.
We observe that the most time-consuming function is *cmd*, what was to be expected. In fact, this function is used to take benefits from the *Unix commands* to produce the various filtering which is the most time-consuming part of the script. We should also pay attention to the fact that the *cmd* function is not only called for filtering but also for the creation of folders, the verification of the output file, etc. Thus we may assume that the filtering time may be a bit higher than the one given for the *cmd* function.

ncalls	cumtime	function
1	4559.658	main
15753	0.774	lineIterator
778	2.324	fastaIterator
184754	16.576	gffIterator
580461	20.928	annotationIterator
777	0.013	FASTA entry
169786	5.080	GFF entry
565493	8.921	Annotation entry
49410	1.409	exonId
36054	0.898	keggId
29936	1.252	mergeLocations
51125	4206.897	cmd

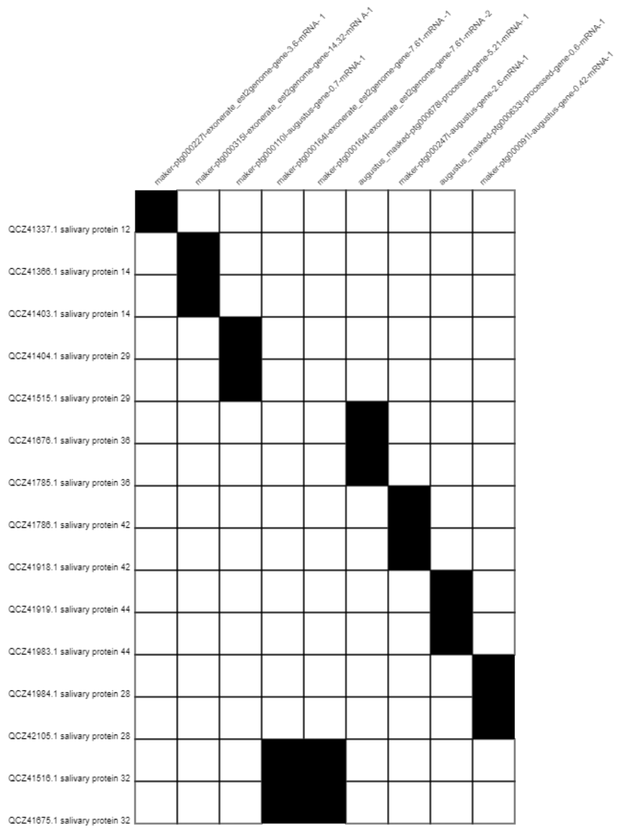
3.2 Homologous proteins

3.2.1 Salivary proteins

With the selected protein, we were able to link five of the eight proteins to possible orthologs in *Glossina morsitans* with the help of the *Psi-Search tool*. We successfully linked our reference protein to the one predicted in the *Phlebotomus papatasi* genome with *BLASTP* but we could not link the proteins from *Glossina morsitans* with the one predicted in their genome. The result is summarised in the following table:

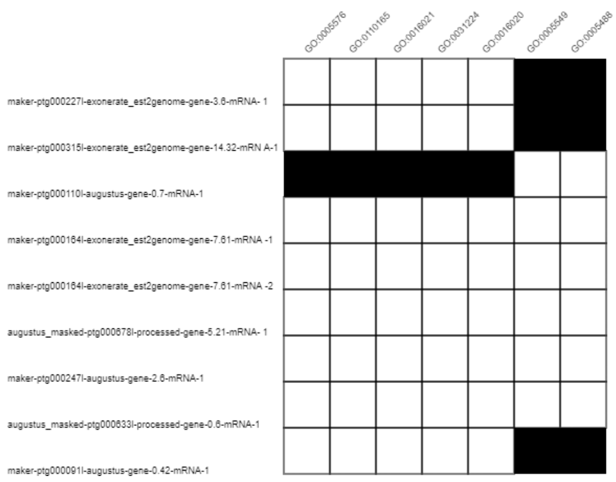
Focusing on the orthologous proteins, we see that SP28 was linked to an odorant binding protein, SP29 to a salivary antigen, SP36 to an apyrase precursor, SP42 to an mischaracterised protein and SP44 to a Yellow-e protein. Then if we look in detail in the predicted proteins in Phlebotomus papatasi genome, we observe that SP36 and SP44 corresponding proteins had no annotations.
For three proteins, corresponding to SP12, SP14 and SP28, the annotation was composed of two *GO terms*: *GO:0005549* and *GO:0005488*, which both refer to binding molecular function, *GO:0005549* being more precise as it defines an odorant binding function.
The last observation focusing on the predicted proteins was that for three proteins we had a more precise description. The protein linked to SP29 was linked to a salivary antigen, the one corresponding to SP32 was linked to a fragment of a salivary protein and SP42 was linked to a major royal jelly protein.
Considering the predicted proteins in *Glossina morsitans* as explained previously, we could not obtain corresponding protein in the predicted genome to the one retrieved by the *Psi-Search* tool. In fact, when searching for the sequence similarities with *BLASTP*, we never obtained a high identity score making meaning probably that the proteins from the databases did not correspond to our predicted proteins.

As explained in the method description, we could obtain three graphs from the *GO terms* approach. Searching for our predicted proteins corresponding to the literature proteins, we obtained the following graph.



On the graph we observe that for most proteins, we had two reference sequences which allowed obtaining only one protein in our predicted genome, but that for *SP32* two proteins were assigned but that they may different possible protein for the same locus. We thus can say that the comparison with *BLASTP* allows obtaining a graph with a good data integrity.

The next graph gives us the relations between our predicted proteins and the *GO terms*. We observe the same result as in the previous approach, where three proteins are linked to two *GO terms*. We also observes that five proteins are not linked to any *GO terms* and finally one protein is linked to five *GO terms*. Unfortunately when looking at the description of the corresponding *GO terms* we observe that they describe a membrane component and thus are not that much more precise. We could thus at this step already expect not to get any interesting result from our method.



We still proceeded with the research of the proteins linked to the *GO terms* where we obtained a list of 167 proteins, manly being linked to membrane components. We could still find out three proteins related to the odorant binding molecular function: GMOY011399, GMOY012255 and GMOY011902. We thus proceeded to link 8 protein from *Phlebotomus papatasi* with 167 in *Glossina morsitans* genome.But when we observe the relationship between this term, we observe that very few proteins from *Glossina morsitans* are linked with *GO terms* related to multiple proteins from *Phlebotomus papatasi*. This can be explained by a small number of GO terms coming from an also small amount of initial protein. Thus, working with more proteins could probably increase the number of "bridges" between *Phlebotomus papatasi* and *Glossina morsitans* proteins.

3.2.2 Iron Metabolism Proteins

Starting with our previous observation from salivary proteins, we decided to increase the size of the set of proteins of interest for our research. We started with 46 proteins from the literature, listed in the table ....

When we searched for the corresponding proteins in *Glossina morsitans* genome, only three proteins could be linked with the predicted genome. As a similar issue happened when we were working with the salivary proteins, we thought that the problem may come from the predicted genome.
With the issue we were facing, it was thus impossible to try again the method with the *GO terms* as they were inaccessible.

4 Discussion

Considering the file merging, we managed to create an efficient script to merge the different types of data into *EMBL entries*. What is considered here as efficient is the ability to merge the script on a local machine within a few hours.
Thinking of how to improve the quality of the script, two main things would be important. First, we could analyse the speed of the script in function of the level of parsing. In our script, we see that we created a lot of sub-files but is it really the best solution ? The other option would be to parallelisation of various tasks so that we can take benefits from the whole computer power.
Finally, one main weakness was reported using the script. As it requires *Python 3.6* and *Biopython 1.78*, its compatibility is limited. Thus the development of compatible version with an older system could at the time this report is written be a good thing but it also points out that even for such a small script the compatibility as to be maintained.

Our research revealed two predominant weaknesses in our approaches. The first one is related to the usage of predicted genes to create an EMBL entry, when the second one concern the amount of data required to search for proteins with similar functions.
A solution to our first issue may be to check our resulting genome with what is already existing. We thus may counter the lack of genes and proteins similitude by searching what already exists. We could then validate not only the *EMBL specification* but also the accuracy of our entry.
For the second problematic, improving the approach may be more complex than just by increasing the size of the input. First of all, even if in general the more proteins we have, the larger the number of GO terms will be, not in every case does a protein have GO terms attached to it and the number of GO terms vary a lot between proteins, what may influence the size of. Then, even if we can create a set of GO terms large enough, we may probably retrieve to much protein in the other species. For example, in the research with salivary proteins, for only 7 GO terms 167 proteins were linked to them, meaning that a lot could be protein that does not interest us. If we have this problem, a solution would be to add weights between proteins and GO terms, meaning that we need a better comprehension of the GO tree.

References

[1] Z Zhang. "Protein sequence similarity searches using patterns as seeds". In: *Nucleic Acids Research* 26.17 (Sept. 1998), pp. 3986–3990. DOI: 10.1093/nar/26.17.3986. URL: `https://doi.org/10.1093/nar/26.17.3986`.

[2] P. J. A. Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (Mar. 2009), pp. 1422–1423. DOI: 10 . 1093 /

bioinformatics/btp163. URL: https://doi.org/10.1093/bioinformatics/btp163.

[3] Carson Holt and Mark Yandell. "MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects". In: *BMC Bioinformatics* 12.1 (Dec. 2011). DOI: 10.1186/1471-2105-12-491. URL: https://doi.org/10.1186/1471-2105-12-491.

[4] Zahra Jalali Sefid Dashti, Junaid Gamieldien, and Alan Christoffels. "Computational characterization of Iron metabolism in the Tsetse disease vector, Glossina morsitans: IRE stem-loops". In: *BMC Genomics* 17.1 (Aug. 2016). DOI: 10.1186/s12864-016-2932-7. URL: https://doi.org/10.1186/s12864-016-2932-7.

[5] Jay Shendure et al. "DNA sequencing at 40: past, present and future". In: *Nature* 550.7676 (Oct. 2017), pp. 345–353. DOI: 10.1038/nature24286. URL: https://doi.org/10.1038/nature24286.

[6] Marcos Sterkel et al. "The Dose Makes the Poison: Nutritional Overload Determines the Life Traits of Blood-Feeding Arthropods". In: *Trends in Parasitology* 33.8 (Aug. 2017), pp. 633–644. DOI: 10.1016/j.pt.2017.04.008. URL: https://doi.org/10.1016/j.pt.2017.04.008.

[7] Bruno Arcà and Josè MC Ribeiro. "Saliva of hematophagous insects: a multifaceted toolkit". In: *Current Opinion in Insect Science* 29 (Oct. 2018), pp. 102–109. DOI: 10.1016/j.cois.2018.07.012. URL: https://doi.org/10.1016/j.cois.2018.07.012.

[8] Kethi Reddy. *embl-api-validator*. Version 1.1.265. Feb. 20, 2018. URL: https://mvnrepository.com/artifact/uk.ac.ebi.ena.sequence/embl-api-validator/.

[9] "The Gene Ontology Resource: 20 years and still GOing strong". In: *Nucleic Acids Research* 47.D1 (Nov. 2018), pp. D330–D338. DOI: 10.1093/nar/gky1055. URL: https://doi.org/10.1093/nar/gky1055.

[10] Petri Törönen, Alan Medlar, and Liisa Holm. "PANNZER2: a rapid functional annotation web server". In: *Nucleic Acids Research* 46.W1 (May 2018), W84–W88. DOI: 10.1093/nar/gky350. URL: https://doi.org/10.1093/nar/gky350.

[11] Matthias Blum et al. "The InterPro protein families and domains database: 20 years on". In: *Nucleic Acids Research* 49.D1 (Nov. 2020), pp. D344–D354. DOI: 10.1093/nar/gkaa977. URL: https://doi.org/10.1093/nar/gkaa977.

[12] Catherine M. Flanley et al. "Phlebotomus papatasi sand fly predicted salivary protein diversity and immune response potential based on in silico prediction in Egypt and Jordan populations". In: *PLOS Neglected Tropical Diseases* 14.7 (July 2020). Ed. by Daniel K. Masiga, e0007489. DOI: 10.1371/journal.pntd.0007489. URL: https://doi.org/10.1371/journal.pntd.0007489.

[13] *Annotating Genomes with GFF3 or GTF files*. URL: https://www.ncbi.nlm.nih.gov/genbank/genomes_gff/. (accessed: 04.02.2021).

[14] *Bio.SeqFeature module*. URL: https://biopython.org/docs/1.75/api/Bio.SeqFeature.html. (accessed: 02.02.2021).

[15] *Bio.SeqIO package*. URL: https://biopython.org/docs/1.75/api/Bio.SeqIO.html?highlight=seqio#. (accessed: 02.02.2021).

[16] *Bio.SeqRecord module*. URL: https://biopython.org/docs/1.75/api/Bio.SeqRecord.html. (accessed: 02.02.2021).

[17] *Biopython README file*. URL: https://github.com/biopython/biopython/blob/master/README.rst. (accessed: 02.02.2021).

[18] *BLAST topics*. URL: https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp. (accessed: 04.02.2021).

[19] *BLAST topics*. URL: https://blast.ncbi.nlm.nih.gov/Blast.cgi. (accessed: 06.02.2021).

[20] *EMBLmyGFF3*. URL: https://github.com/NBISweden/EMBLmyGFF3. (accessed: 02.02.2021).

[21] *Index of /pub/databases/embl/release/doc/*. URL: http://ftp.ebi.ac.uk/pub/databases/embl/release/doc/. (accessed: 24.02.2021).

[22] *Protein Similarity Search*. URL: https://www.ebi.ac.uk/Tools/sss/psisearch/. (accessed: 06.02.2021).

[23] Amjith Ramanujam. *Python Profiling*. Youtube. URL: https://www.youtube.com/watch?v=QJwVYlDzAXs.

[24] *subprocess — Subprocess management*. URL: https://docs.python.org/3/library/subprocess.html. (accessed: 02.02.2021).