ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA (COMPUTACIÓN)

Flujo de trabajo para el rastreo de bloques computacionales de sintenia a través de distintas especies

A workflow-based algorithm for tracing Computational Synteny Blocks along different species

Realizado por

Ricardo Holthausen Bermejo

Tutorizado por

Oswaldo Trelles Salazar

Cotutorizado por

Esteban Pérez Wohlfeil

Departamento

Departamento de Arquitectura de Computadores

UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio 2019

Fecha defensa:

El Secretario del Tribunal

Resumen

La comparación de secuencias de textos es un área de notable relevancia dentro de las Ciencias de la Computación. Existen varios problemas clásicos representativos del área, como el problema de la Subsecuencia Común de mayor Longitud, consistente en encontrar información compartida por diferentes cadenas de texto. Sus aplicaciones van desde la Lingüística Computacional hasta la Bioinformática, entre otras. Al respecto de esta última, el área de la comparación de secuencias de genomas ha recibido mucha atención durante los últimos años, lo que ha llevado a un crecimiento destacado. Esto, junto a las mejoras técnicas en el rendimiento computacional, está contribuyendo a ampliar el conocimiento sobre quiénes somos. En particular, la evolución de las especies, a pesar de haber sido extensamente estudiada, sigue necesitando ser analizada, debido a su complejidad, tanto analítica como computacional. Nuevas herramientas para el estudio de Eventos Evolutivos pueden proveernos de información relevante sobre rasgos y enfermedades, además de una mayor comprensión de los mecanismos que subyacen a la evolución (que tienen un impacto directo en la salud). El flujo de trabajo presentado "BlockTracer" es una herramienta que permite a los investigadores/as seleccionar una serie de cromosomas de distintas especies y buscar bloques de información directamente relacionados entre ellos. Para conseguir esto, varios programas independientes han sido creados y orquestados. Este flujo de trabajo se presenta en la plataforma Galaxy, que permite computación en la nube manteniendo una interfaz de uso amigable para el usuario.

Palabras clave: Bioinformática; Flujo de trabajo; Bloques de sintenia computacionales; Eventos Evolutivos; Comparación de secuencias

Abstract

Sequences and texts comparison is a rather relevant area within Computer Science. Several classic problems, such as the Longest Common Subsequence problem, which relates to finding shared information between different text sequences, are related to this area. Its applications range from Computational Linguistics to Bioinformatics. Regarding the latter, the area of pairwise genomic sequences comparison has received a huge amount of attention during the last years, and therefore it has suffered a relevant growth. This, altogether with the technical improvements in computational performance, is helping us to acquire a better understanding of the world we live in. Within this area we find the evolution of species, which, yet thoroughly studied, needs more grasp due to its vast nature. New tools for studying Evolutionary Events could provide us with relevant information about traits and diseases, besides a better comprehension of evolution underpinnings (which have a direct impact on health). The BlockTracer workflow is a way for the researchers to select a certain set of species chromosomes, and search for directly linked pieces of information between them. In order to achieve this, several loose-coupled tools have been created and arranged, obtaining an efficient computational pipeline capable of reporting those shared blocks between species. This workflow is presented on a usability-centered online platform such as Galaxy, which allows cloud computing while keeping a user-friendly interface.

Keywords: Bioinformatics; Workflow; Computational Synteny Blocks; Evolutionary Events; Sequence Comparison

Table of contents

Introduction **13**
- Motivation 13
- Project objectives 16

Background **19**
- Introduction 19
- The Longest Common Subsequence Problem 19
- A Bioinformatics primer 22
- Maximum subarray problem 23
- Workflows 23
  - Taverna 24
  - The Galaxy Project 25
  - Pegasus Workflow Management System 27
  - Selection of Galaxy for the project 28

Analysis and design **31**
- General considerations 31
- Workflow specifications 32

Implementation **35**
- BlockTracer 36
  - Tracing method 37
  - Implementation 38
- Obtaining CSBs from HSPs: getCSB 39
- Workflow creation in a Galaxy instance 42
  - Tool registration in Galaxy 43
  - Workflow creation 45
  - The Galaxy-based BlockTracer workflow 46

Results and use case validation **49**
- Case of use 1: Tracing of mammalian chromosomes 49
- Case of use 2: Studying BlockTracer computational performance 55

Conclusions **57**
- Conclusions of the developed work 57
- Conclusiones del trabajo desarrollado 58

Bibliography **61**

1. Introduction

Motivation

Matching subsequences between different sources of text, known as the “Longest Common Subsequence Problem” (LCS) is an important and well known problem in Computer Science [1], mainly due to its applications in Computational Linguistics [2] and also Bioinformatics [3]. As its name explains, this problem consists on finding the longest common subsequence given a set of sequences. Depending on the application domain, the definition of “common” sequence can be more or less stringent. For instance, in control version solutions [4], an exact approach is needed (the small differences between pieces of code are important), whereas in the case of searching pieces of text in databases, we are mostly interested in the big picture, and using a strict approach can be counterproductive (*i.e.:* useful information could remain hidden). When the number of input sequences is part of the problem (*i.e.:* when there is an arbitrary, non-fixed number of input sequences), the LCS problem belongs to the NP-Hard class [5]. Nonetheless, when the number of input sequences is constant, this problem is solvable in polynomial time by means of Dynamic Programming techniques [6, 7].

A field in which the study of this problem is particularly important is Bioinformatics and in particular in the field related to the evolution of species. The advancements made in this area can contribute both to broaden the vision in fields such as Life Sciences, as well as regarding the development of new techniques in biomedicine, and, ultimately, helping us to obtain a better grasp of the world we live in [8].

DNA sequences (genomes, particularly) comparison and analysis provide relevant information regarding the evolution of species. Throughout the years, the genetic endowment of the organisms has replicated, integrating changes of a variety of sizes known as mutations, which are themselves heritable, and are part of the theory Darwin started during the 19th century [9]. Most of these mutations are only one base-pair sized (one letter in the DNA repertoire, known as Single Nucleotide Polymorphism, SNP), while other mutations represent severe rearrangements of the specie genetic code (thus, finding them is an instance of the LCS problem), possibly giving rise to new species. Among these "large scale" changes we find the "segments" relocations, whether through duplications, changes or translocation of a segment in the same chromosome, inversions, or even the relocation of segments between chromosomes.

The area of pairwise sequence has received a lot of attention during the last years [10]. This growth has taken place in two main ways, which have also provided feedback to one another. Firstly, the ability to produce large quantities of data, by means of improved sequencing techniques and new approaches for well-known issues (Next Generation Sequencing, NGS) [11] so that the DNA is sequenced in text strings, data subject be studied from the LCS problem viewpoint. Secondly, computer performance has been steadily increasing throughout the years [12]. Both elements have opened the door to the development of new, innovative tools for genetic sequences comparison in linear time and with a controlled memory consumption [13, 14], thus making feasible ideas earlier unattainable.

In the present project we address the development of a method for tracing both non-exact and exact DNA blocks and segments along different species' genomes. This can be seen as a specific instance of the LCS problem, where sequences are formed of DNA letters and we are searching for substrings (DNA blocks). When tackling this kind of problems, where several species are involved, the respective comparisons between their genomes are needed. This means a quadratic number of comparisons (given n sequences, the number of possible comparisons between them is $\frac{n*(n-1)}{2}$). This quadratic complexity, inherent to the study domain can be a challenge regarding the scalability of the proposed solutions.

Besides, when tracing blocks along different species, the complexity found is exponential in the number of comparisons involved (worst case scenario), given that, if a block in a chromosome of a species A appears in l chromosomes of another species B, then it would have to be studied if these blocks in l' chromosomes ($l' \leq l$) are also present in another m chromosomes of yet another species C, and so forth, resulting therefore in a non-balanced tree structure, which is an additional difficulty for the process.

Additionally, several steps are necessary prior to the block tracing, such as obtaining the actual blocks from smaller-sized fragments [15]. This not only requires a clustering of small fragments in larger blocks, but also requires maximising their score. This can be seen as a modified version of the “Maximum Subarray Problem” [16]. The base task of this problem consists on finding, given a two-dimensional array, its subarray whose element-wise sum is maximum. Thus, a naive approach requires an $O(n^3)$ solution. As part of this project, a heuristic alternative for this problem will be also studied.

Finally, the organisation of different tools created towards the same target has been done in the last years by means of computational workflows. This is due to the characteristics of Bioinformatics research, which requires structuring and organising the different phases, processes and pieces of software that solutions and experiments are composed of. In this sense, a workflow can be defined as an orchestrated and repeatable pattern of an activity enabled by the systematic organization of resources into operations that process information [17]. Several approaches regarding organising and making workflows have been developed online and are available for the scientific community (Galaxy [18], Taverna 2 [19], Pegasus [20], ...). One of the most popular tools in the last years is Galaxy, a platform that allows researchers to create and share workflows for them to be utilised by other people interested in it. Nonetheless, prior to creating this workflow, the development and registering of the different tools that compose them in a Galaxy instance is needed.

The current project has been developed within the BitLab research group, which is part of the Departamento de Arquitectura de Computadores at the Universidad de Málaga. High Performance Computing, Cloud Computing and workflow management systems are the main lines of research in BitLab. This work benefits from the work done by several group members during the last years in the field of advanced computing applied to Life Sciences. Thus, this work is part of the software suit produced not only in the group but with external collaborations, such as the international platform Elixir [21].

Project objectives

The main goal pursued in this work is to develop a workflow capable of tracing Computational Synteny Blocks (CSBs) throughout a set of species. The workflow employs heuristic approaches to generalized versions of the LCS and the MSP problems, while achieving significant performance and a combination of coarse and fine-grained methodologies. In order to achieve this, our work has to have several features, which are described below.

- Documentation
    - Documenting not only the code developed, but also the whole development process allows for a project this size to be studied, replicated and still be useful in the future. The creation of a thorough and exhaustive documentation takes part in the set of targets taken into account in the present project.
- Agile development
    - The methodology approach chosen for this project is based on the Scrum agile framework. This decision was made mainly because of the advantages an iterative approach provides for a project that is subject to changes (*n.b.:* originally the executive summary of this proposal included the waterfall model as the design approach for carrying out the project, but it was re-evaluated and changed accordingly).
- Individual modules linked by datafiles:
    - The different pieces that form a workflow must be interchangeable and have to be able to be replaced with upcoming versions, while keeping file formats, allowing the whole process to keep its original characteristics.
- Efficient, readable, high-level programming for processing data:

    - As it is suggested by the Moore’s law [12], computational power rises up every two years. Besides, there is an increasing trend in the programming community towards more readable languages, sometimes taking advantage of the performance that characterises lower-level languages too [22, 23]. For these reasons, we find it recommendable to make use of higher level programming languages, which stimulate a more readable way of coding, while preserving an always desirable efficiency.
- Open workflow to allow easier extensions and/or modifications
    - When the approach followed to develop an application is a monolithic one, tasks such as feature adding or bug fixing can become difficult to the extent that, sometimes the whole project is abandoned. One alternative for avoiding this kind of problems is to organise all the process in a workflow in which the applications members of it can be fixed or even modified in an easier manner.
- Parameter customization in order to provide flexible results:
    - There is a wide variety of users for a single application. Normally not all users pursue the same objectives, and they need to do some parameters setting and tweaking in order to obtain the desired results. This will be taken into account in order to provide a broad range of customizable parameters.
- Both coarse and fine-grained approaches covered:
    - Normally, having a detailed point of view of a certain topic is beneficial. However, this means, in the field of comparative genomics, to run long and demanding processes whose results are eventually useless. Combining this fine-grained approach with a coarse-grained one that allows the users to grasp the big picture, or even preview information of interest is important in these fields.

2. Background

Introduction

In the following chapter we will discuss several topics related to this project's development. We will analyse them and review the bibliography related to them, providing a primer on those subjects that are more theoretical, and discussing the different approaches available to tackle them in the case of the problems.

We will first visit the Longest Common Subsequence problem, its history, applications, algorithms created to solve it and the current state of this question. After that, a brief primer on Bioinformatics will be provided, defining some basic concepts that are needed to understand both the application of the current project and the next section, the one devoted to the Maximum Subarray Problem, which, analogously to the first one, will review different elements regarding this problem and its applications. Finally, a review on the main tools available for workflows management, publishing and sharing will be described. In this section it will be also explained which tool was chosen in order to publish the workflow related to this project and the main reasons for it.

The Longest Common Subsequence Problem

Prior to reviewing the Longest Common Subsequence (LCS) problem, two basic concepts have to be introduced. Firstly, a string is a chain of symbols or characters over an alphabet Σ. Secondly, a subsequence of a string is the string resulting from deleting zero or more symbols from it.

Given n strings $\{S_1, S_2, ..., S_n\}$, finding the longest subsequence that is present in each of them is called the LCS problem. As mentioned previously, the relevance of this problem relies on its applications, primarily in the areas of Computational Linguistics [2] and Bioinformatics [3]. In the latter, and specifically in the field of comparative genomics, finding a subsequence present in the genetic code of different species means finding a relationship between them, which can ultimately provide us with information regarding how the species became what they are (*i.e.:* what changes did occur) and what are the underpinnings of the mechanisms that made those changes possible.

Regarding the complexity of the LCS problem, it is NP-hard when the number of sequences is part of the problem (*i.e.:* for an arbitrary n) [5]. Nonetheless, when n is fixed, the problem becomes solvable in polynomial time by using dynamic programming techniques [24].

Given the fact that the LCS problem is interesting regarding both its applications and its complexity, it has been thoroughly studied throughout the time. This situation originated a set of algorithms that tackle the problem from different perspectives, always trying to obtain a more efficient way of solving the LCS, and studying the conditions under which the tractability of the problem changes. Hereafter, an overview of the different ways this problem has been tackled is provided, ranging from the first dynamic programming algorithms to the current metaheuristics approaches [25].

In order to solve the LCS problem for two sequences, the standard method is the usage of dynamic programming. Thus, the solution is obtained by creating a matrix in which each cell will be determined by the function in Figure 1.

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) \frown x_i & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \text{longest}\{LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

Figure 1: Definition of the function that fills the matrix to obtain the LCS of two sequences.

As can be seen, the cell value will have no value if it is in the first row or in the first column. For the rest of cells, when the two characters compared are the same, it is appended to the resulting LCS. When they are not, the LCS is the longest between the previous element in the same row or in the same column.

This approach, while optimal, has a time complexity of $O(l^n)$, where l is the length of the sequence and n is the number of sequences in which to search the LCS. Thus, it has a poor scalability.

Several alternative approaches have been used in order to decrease the time and space complexity of the problem, while still providing good quality solutions. The Hunt and Szymanski's algorithm [26], for example, obtains the LCS of two sequences in $O(n\ log(n))$ time. However, the worst case time complexity is $O(n^2\ log(n))$.

Other approaches rely on heuristic approximations which try to find near optimal solutions in reasonable time, being thus a rather practical point. Regarding this problem, we find three main heuristic approaches:

- The Large Neighborhood Search algorithms, also known as Local Search Algorithm, which is an improvement algorithm (*i.e.:* given a feasible solution for the problem, iteratively searches for a better solution) that relies in a large neighborhood structure to find better solutions [27]. The heuristic algorithm proposed by Easton and Singireddy [28], also called Time Horizon Specialized Branching (THSB) heuristic, achieves a theoretical runtime both linear in the length of the sequences and also regarding the number of sequences of the LCS problem. However, this approach is exponential in the horizon size chosen.
- Deposition and extension approach, presented by Ning in 2010 [29], provides an alternative whose performance is especially interesting in the case of having many sequences in which to search for the LCS. As its name suggests, this heuristic is based on two steps. The deposition step consists on finding a common subsequence based on fine tuning of search range, and then in a second step the common subsequence is extended. The time complexity of this approach is $O(l^2 n^2 |\Sigma|)$, where l is the length of the sequence and n is the number of sequences.
- Beam Search approaches are an incomplete derivative of Branch and Bound techniques. The central idea of this set of approaches is to allow the extension of partial solutions in several possible manners, calculate an upper bound value for

each chosen extension, and, by means of a greedy function, select the best found complete solution. Regarding the performance of this heuristics, it was proven to be better than the THSB and the extension algorithm [30].

In this project we will work with sequence comparison algorithms, and therefore we will need to select the programs based on their algorithm implementation and scalability in order to enable tracing of blocks from multiple species. The fact that the number of species is a variable in our project involves facing a NP-hard problem. Therefore, a proper approach has to be selected in order to obtain a balance between performance and high quality results.

A Bioinformatics primer

As the domain of application of the present project turned out to be Bioinformatics, in order to introduce the next section (the one regarding the Maximum Subarray Problem), it is needed to provide a brief primer on Bioinformatics, where the fundamental concepts that are involved in our work are explained:

- Nucleotides: They are one of the basic structural elements of DNA and RNA [31]. A nucleotide is composed of a base, which can be represented by one of four letters (A, C, G or T). Thus, when sequencing genetic code, it is represented by a string formed by these letters.
- SNP: They are the most common type of genetic variation among people [32]. It consists on a difference between DNA sequences of just one nucleotide.
- High-scoring Segment Pair (HSP) or fragment: This is a term related to local alignment tools such as BLAST or Gecko, and represents a portion of genetic code that is shared by two sequences [33].
- Computational Synteny Block (CSB) [15]: This concept refers to a similar element to frags, but with larger dimensions. Thus, a CSB is a set of fragments that conserves both strand and collinearity. Studying shared DNA between species can be done in several levels of detail. Using CSB allows a better

performance compared with HSP (less number of elements to compare), while keeping a rather fine-grained approach.

Therefore, A CSB is a piece of information composed of HSPs, which in essence are similar strings of nucleotides that allow SNPs and other forms of DNA mutation. One of the underlying problems for our current project is to find a proper technique to obtain CSBs from HSPs, which is addressed in the following section.

Maximum subarray problem

Starting from the Bioinformatics primer from the previous section, and knowing the necessity of a method to obtain CSBs from HSPs, we arrive at the Maximum Subarray Problem, whose study provides a broad vision regarding the obtaining of CSBs.

The Maximum Subarray Problem (MSP) consists on, given a $M \times N$ matrix of real numbers, finding a rectangular submatrix such that the sum of the numbers present on it is maximized [34]. This problem was first published in 1984 and, the main applications it has are related to image processing, pattern recognition, data mining and biological sequences analysis [35].

Different solutions have been provided for this problem throughout the years. Firstly, altogether with the publication in which the problem was firstly explained, a cubic solution was included. Until 1998, no approach managed to provide a lower complexity bound. That year, Tamaki and Tokuyama [36] obtained a subcubic algorithm by means of matrix multiplication. After this, some more improvements were obtained in similar and constrained cases (small K for the K-MSP, one dimensional arrays, ...) [37].

Workflows

When it comes to workflow management tools, several alternatives have been created thus far [38]. In the following section we will review several of them, their characteristics and the elements that differentiate each one from the others, as well as the main reasons for the selection of Galaxy for registering and managing the current project workflow.

Since the advent of the internet, and especially since its establishment as a general purpose tool, several initiatives have appeared in order to make easier the research work. One field in which these initiatives have received a special attention is the publishing and sharing of workflows [38, 39]. Moreover, as the usability became a characteristic to be taken into account when developing a digital service, more features aimed to improve the users' experience were added, not only for their interaction with the interface, but also regarding the actual management and edition of workflows. After a profound and comprehensive analysis and study of the available alternatives for this task, we highlight three main online tools in which researchers can share their experiments and processes in order for them to be used by other users, and even be modified. Before reviewing each one of the tools that are being currently used for this purpose, it is noteworthy that they are not really "rivals", in the sense that during the last years they have been working on their interoperability, rather than competing.

Taverna

The first of the tools to be reviewed in this section is Taverna Workflow System. This project was born in 2004, with the publication of the Taverna software as a program for composing and enacting bioinformatics workflows [19]. After several versions, it eventually evolved to "an open source and domain-independent Workflow Management System – a suite of tools used to design and execute *scientific workflows* and aid *in silico* experimentation.", obtaining a major relevance and, in the last years, even coupling with the Galaxy project, which will be later discussed. Taverna allows its users to compose workflows formed by both local tools and distributed Web-Services. These "complex analysis pipelines" workflows can then be executed locally or in larger-scale infrastructures (e.g.: supercomputers, Grids or clouds).

Taverna understands workflows as "reusable informatic analysis protocols". Thus, this tool aims to be scalable, and able to interact with as many other tools as possible. The

Taverna Suite is currently compatible with RESTful web services, Grid services, cloud, R scripts and also command line interface (CLI) scripts.

Usability is another desirable feature for a tool that will be used by a large number of users with different levels of skill regarding computer usage. In this sense, Taverna developers created the “Taverna Player”, an interface available for executing workflows both through web-browsers and third-party clients.

In order for Taverna to be “social”, in the sense of the possibility for the created workflows to be published, shared, reused, etc. by other users, its developers rely on the myExperiment repository. Another interesting feature is the possibility for the users to keep track of their executions. In this sense, Taverna included the “Taverna Provenance Suite” as part of the whole tool, allowing users to record workflow invocations, intermediate and final results.

Finally, as any other tool that aims to be collaborative, the community behind it is a key factor to take into account when comparing alternatives. In the case of Taverna, it has been widely used in most fields of Bioinformatics, with the myExperiment [39] repository as the main site for the people to share their workflows. Regarding Taverna development, it is an Open Source project, characteristic that has provided a robust support and a wide range of plugins available until now.

The Galaxy Project

According to the primary Galaxy publication [18], it is a “web-based scientific analysis platform used by tens of thousands of scientists (...)”. The project started in 2005, aiming to achieve three main goals: accessibility, reproducibility and an effective communication between users. As an open-source software, both the Galaxy team and the open-source community have carried out large advancements in several areas, namely its core framework, the available tools, tutorials and training material, or the user interface.

The whole Galaxy Project is composed of four main elements that complement each other: (1) The main public Galaxy Server, which has been online since 2007, and features a large set of tools for large-scale genomics analyses, besides a huge amount of data from its usage throughout the years (analysis histories, publication supplements, complex pipelines, …). (2) The Galaxy framework and software ecosystem, which is an open-source repository that can be used by everyone in order to run a Galaxy Server. (3) The Galaxy toolshed; a resource for publishing workflows, tools and visualizations. In this site, Galaxy administrators can find and share tools to be added to their instances. (4) The Galaxy Community, created with the goal of addressing issues that can exist in any field (for users, administrators, developers, educators, ...).

Regarding the last update of Galaxy, the features that have received more attention are:

- The scalability (in two senses: the web-based interface and the server backend to provide reliable performance in a multiuser environment).
- The user experience and interface enhancements, with elements such as an interactive workflow editor
- The development of tools, with the Galaxy Toolshed as an important factor for this, providing the users with a kind of 'App-store' for Galaxy instances
- The Interactivity, throughout analyses and visualization. This is achieved with the Galaxy Interactive Environments, which is an integration of Galaxy with Jupyter.
- Enhancements for the infrastructure: In order to obtain robustness in a production environment, uWSGI was adopted as the Galaxy's default web application server. This decision was made according to the necessity to make possible concurrent task execution and load-balancing.

Finally, and as it was mentioned at the beginning of this section, the Galaxy community is an important factor for the success of this project as a tool for Bioinformatics professionals. The Galaxy main server has more than 120000 users, carrying out up to 245000 analyses per month. Besides, there are more than 7000 scientific publications referencing Galaxy [40], and there are also in-person events such as the Galaxy Community Conference (GCC), or the Bioinformatics Open Source Conference (BOSC), which take place yearly.

Pegasus Workflow Management System

The last workflow management tool we will discuss thoroughly is the Pegasus Workflow Management System. It was first launched in 2001 and it provides ways of composing and sharing workflows, as well as running, monitoring and debugging them [20]. The highlighted feature in Pegasus' system is the portability. By keeping the workflow description and its execution environment separated, portability across different infrastructures is achieved, as well as the possibility of carrying out optimizations at both compile time and runtime.

In order to provide a description of the workflows, Pegasus relies on Direct Acyclic Graph in XML format (DAX) [41]. This format provides a way of specifying jobs, input parameters and files, and also dependencies.

Scalability is also a Pegasus concern. In order to achieve this characteristic, not only the resources used for the execution, but also the size of the workflow can be easily upgraded. In this sense, it is possible to run workflows that have up to one million computational tasks. This is feasible thanks to a great extent to the compile time workflow restructuring (which involves reduction of workflows and data reuse), and also to the runtime optimizations.

When executing a complex and large pipeline, a number of problems (*v.g.:* job execution errors, data transfer failures) can appear. In order to tackle this issue, Pegasus has several features that follow providing reliability. Among them we find job retries, parallel transfers for data movement, recovery for failed workflows and workflows replanning.

Finally, usability is also a Pegasus matter of interest. In this sense, tools for workflow composition and monitoring and debugging (*v.g.:* Pegasus mapper log, job logs) are available. Besides, there are also means for the users to provide feedback regarding the different components of Pegasus, thus making possible fixing bugs or developing upgrades for this system.

Selection of Galaxy for the project

After a thorough study of the main workflow management utilities available online, we decided to use Galaxy for the workflow developed in the present project. This is due mainly to the field of study the pipeline relates to: Bioinformatics. Galaxy is a platform that is capital in data-driven biomedical and bioinformatic science. Therefore, the availability of useful tools for developing the current workflow will be broader in the case of Galaxy, besides the future applications the workflow can have, in the sense that the possibilities for it to be used by the Life Sciences community is larger if it is developed for this platform. Besides these reasons, it is noteworthy that Galaxy is backed by the ELIXIR platform [21] through its publicly available portal [42].

Regarding workflow management tools, in this section it was shown the common features and achievements they pursue, being the scalability a capital characteristic, as well as a trend towards the enhancement of usability. During the revision of the bibliography and the study of these tools, we have not found a great difference between them. The actual disparity appears to be the community behind each one. Galaxy has the larger community when it comes to Life Sciences.

Finally, another element to take into account is the activity of the BitLab group. This research team has a Galaxy instance where most of their tools are available. As this work will use some of these tools, the logical step is to develop the current tool for this Galaxy instance, as it can become part of the research group ecosystem.

In summary, the tracking of CSBs along species requires the tackling of several problems whose complexity imposes a computational challenge, particularly (1) to find a proper technique to obtain sequence comparisons (*i.e.:* an accurate, efficient way of solving the LCS problem). (2) Once the HSPs are obtained, which are rather fine-grained for our purposes, we need to find a way to subsume them in CSBs. (3) Regarding the usage of the tool to be developed, a user-friendly, efficient and extended workflow management tool has to be used. For these purposes we have selected an instance of the Galaxy Project. In order to overcome these barriers, we have selected different algorithms for particular instances of the problems, which will be combined, extended and connected with other new software pieces in order to generate a fully functional workflow that makes block tracing feasible.

3. Analysis and design

In the following chapter, the main requirements for the proposed method will be described. This implies studying the inputs and the way they will be processed. Besides other elements to decide what information is available, how should our method handle it, the way the whole workflow should be executed and the use of external platforms for it, and the definition of a format for the output files. In general, the requirements studied will be high-level related, as the low-level ones will be handled in the implementation section of this project.

General considerations

- Programming Language: When dealing with the processing of large amounts of data, a good performance is needed. Besides, as the different elements that compose a workflow should be able to be modified or maintained, a readable programming language is advisable. One of the most popular programming languages for doing data science related activities is Python. The main desirable characteristic of Python is the code readability it allows, making thus easier both code development and maintenance. Regarding the performance that can be obtained by using this language, it would be expected to be somewhat poor, given the high-level nature of Python. Nonetheless, there are a number of libraries for data science, which were developed in C, thus providing a rather adequate performance. Thus, Python programming language has been selected to develop the main scripts for the current project.
- Platform: When working with Bioinformatics software, most of it is developed to be used in a UNIX-like environment (Ubuntu/Debian). The main reasons for this are: (1) the ease of use in the case of system administration, (2) the desirable characteristics of the command line interface, which is a powerful tool and (3) the cost of UNIX-like software (it is cheaper than Microsoft or Apple alternatives, or

even free). Based on these reasons, for the development of our project, Ubuntu and Debian were selected for development tasks.

- Workflow Management System: As it was mentioned in the background section, the Workflow Management tool chosen for the current project was the Galaxy Platform. After a thorough study of the main alternatives for workflow-related environments, we decided to use Galaxy due to its relation with Bioinformatics, as well as the fact that several tools that will be part of the final workflow are already available in this platform. Besides, the scalability pursued by the Galaxy developers opens the door to future contributions regarding the visualization of our workflow results.
- Results format: In order to provide our results with a certain flexibility, the results obtained in the different parts of our workflow must follow a standard. Standardisation is capital when dealing with the files associated to a workflow composed by several programs. If this is not possible, then ad-hoc tools for pre-process files are needed (*i.e.:* parsers).

Workflow specifications

Our software will be organised as a workflow. A simplified representation can be seen in Figure 2. In the current section, the different modules that compose the pipeline will be analysed. Regarding the low-level details and definitions, they will be given in the next section (*i.e.:* Implementation).

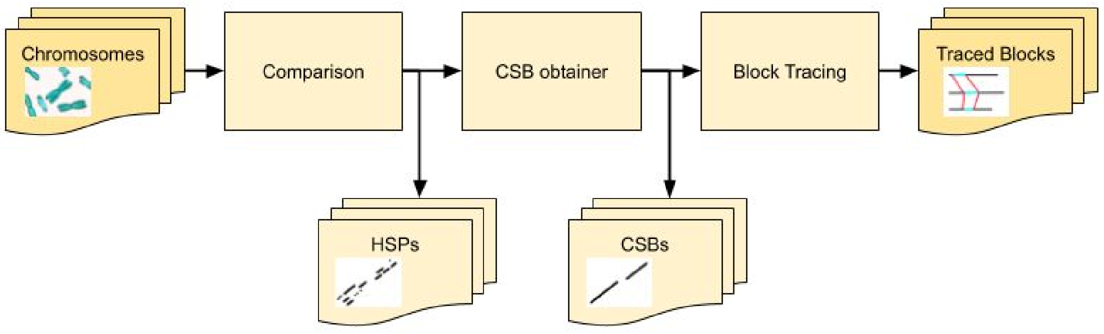


Figure 2: Simplified representation of the workflow.

The workflow is composed of mainly three kinds of data-processing tools. Firstly, the selection of a proper software in order to perform sequence comparisons between species chromosomes is needed. A number of sequence comparison tools are available. Among them we can mention BLAST [14], which is one of the most widely used software for this purpose; and others such as BOWTIE [43], SOAP [44] or LASTZ [45]. However, given the software ecosystem in which this project will be developed, as well as several desirable features of the BitLab software[1] (Gecko and Chromeister [46]) such as their compatibility in order to reduce the search space, or the possibility of avoiding computational starvation with the performance improvement that Chromeister provides (*i.e.:* with other solutions each comparison can have a quadratic runtime), the sequence comparison tools selected for the current project will be Chromeister (for coarse-grained comparison) and Gecko (for fine-grained comparison, using the Chromeister output to reduce the search space, and therefore the workflow runtime).

The second main data processing task is to obtain CSBs from HSPs. In order to do this, it is required a software that, given an input CSV file containing HSP information (coordinates in both sequences, score, degree of similarity, length, etc.), merging them together in larger blocks, while keeping an acceptable score.

Finally, the third task is related to the proper block tracing. Once the CSBs are available for each comparison, it is needed a tool that traces them and checks if they are conserved throughout the different comparisons. Therefore, a proper way of assessing this "conservation" has to be devised. Besides, there are some desirable features for this part of the workflow, such as the possibility for the users to set the traced block "depth" (*i.e.:* how many species have to share that block in order to be taken into account).

[1] https://chirimoyo.ac.uma.es/bitlab/portfolio/

4. Implementation

In this section we describe the implementation details regarding the developed workflow. The two developed tools are analyzed and their code is thoroughly depicted. After this we describe the process needed in order to register the developed workflow in a Galaxy instance.

According to Figure 2, Figure 3 depicts a specific version of the developed workflow. In this case the pipeline works with the Chromeister, Guided-Gecko (or Gecko), getCSB and BlockTracer tools; however, any other tool that performs a similar task can be used.

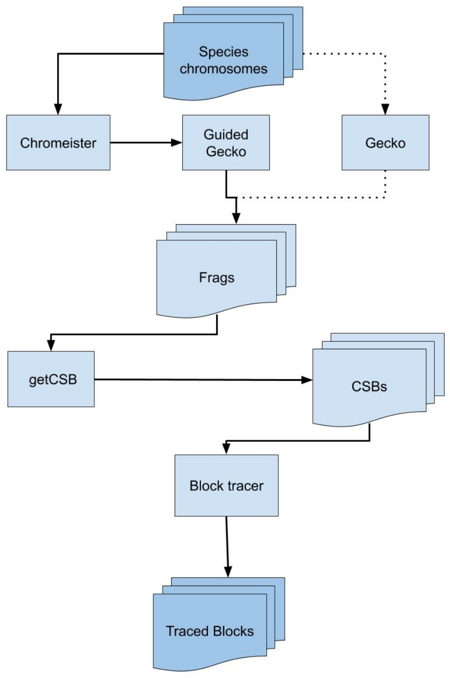


Figure 3: BlockTracer workflow depiction.

The workflow receives as input n species chromosomes. The first step is to obtain the pairwise comparison between the sequences in the following manner: given n sequences, $\{s_1, s_2, ..., s_n\}$, we need the $n-1$ comparisons between them ($s_1\ vs\ s_2$,

$s_2\ vs\ s_3$, …, $s_{n-1}\ vs\ s_n$). In order to perform these comparisons, two different tools are used. The first one, which follows a coarse-grain approach, is Chromeister. Chromeister is a software that provides a heuristic approach for detecting potential shared segments between DNA sequences. The interesting point regarding the use of this tool is, besides its execution speed, its interoperability with the second tool used, Gecko [13]. Gecko is a software for comparative genomics which focuses on detecting HSPs. In contrast with Chromeister, the approach is rather fine-grained (one of its main features is its high accuracy results). Nonetheless, Gecko is prepared to use the output of Chromeister in order to shrink the search space, thus significantly decreasing the execution time of the comparisons.

Once the different comparisons have been done we are provided with $n-1$ files containing HSPs. These files are then processed by the second developed tool, getCSB. This software will obtain CSBs from the HSPs provided by Gecko, making thus easier tracing them between species.

Finally, the BlockTracer tool will be executed in order to trace the blocks obtained by getCSB. The resulting file will consist of a .csv file with each traced block information (species and chromosomes related and the block's coordinates in the species genomes).

The presence of the Gecko regular version in Figure 3 means there is a fine-grained way of running our workflow. In this case the results of the comparisons are more accurate than with the Guide-Gecko version, as the search space is larger.

BlockTracer

This section describes the kernel of our work. We will discuss the main characteristics of the BlockTracer tool and how was it implemented. Firstly, the general considerations regarding the tracing method used will be studied. After this, a thorough description of the BlockTracer script and functionalities will be provided.

Tracing method

Regarding the method used in order to trace the blocks between species, we have to discuss two issues. The first one is, given two blocks, measuring the degree to which they are related. In order to answer this question, we will study the overlap coefficient concept. The second one is how to do the actual trace of a block, given several species. In order to assess overlapping, the overlap coefficient or Szymkiewicz-Simpson coefficient is used [47]. This method provides a value between 0 and 1, and it is obtained by dividing the overlapping part length by the length of the smaller sequence:

$$overlap(X, Y) = \frac{|X \cap Y|}{min(|X|, |Y|)}$$

In our case, the overlap between two blocks is determined by the following funcion:

$$overlap(X, Y) = overlap(<x1, x2>, <y1, y2>) = \frac{min(x2, y2) - max(x1, y1)}{min(x2-x1, y2-y1)}$$

Being x_1 and y_1 the blocks' starting coordinates and x_2 and y_2 the blocks' ending coordinates.

Regarding the actual tracing of blocks, when tracing a block along n species $(s_1, s_2, ..., s_n)$, we have to check the overlapping coefficient between $n-1$ comparisons:

$$(c_1\ \{s_1\ vs.\ s_2\},\ c_2\ \{s_2\ vs.\ s_3\},\ ...,\ c_{n-1}\ \{s_{n-1}\ vs.\ s_n\})$$

In each comparison c_i we can find m blocks $(c_i = \{b_{i1}, b_{i2}, ..., b_{im}\})$. When in two consecutive comparisons c_i and c_j (where $i = j - 1$) a block is traced (*i.e.:* the overlap coefficient is larger than a given threshold), two situations can occur:

a) b_{ik} length is equal or greater than b_{jl}: then no action has to be done, as in the next iteration just the coordinates of b_{jl} will be taken in order to trace blocks in c_k (with $k = j + 1$)

b) b_{ik} length is less than b_{jl}: Then new coordinates for b_{jl} have to be calculated, so that in the next iteration just the shared part of b_{jl} is taken into account when tracing blocks in c_k.

Implementation

The BlockTracer is the last tool that is run in the developed workflow. It receives a CSB list as input, which was generated from large genetic sequences (such as genomes) comparisons. The first task that is done is to extract all blocks available in the input file, and store them in a different data structure (a Python list) per comparison. This step would yield n lists of blocks (where n is the number of comparisons performed). The code developed for this part of the project is available at its GitHub repository[2].

The next step would be to iterate over this list, doing $n-1$ comparisons between each pair of lists of blocks, looking for overlapping blocks. All overlapping blocks will be stored in new lists, thus yielding $n-1$ lists. This step is repeated until only one list is obtained, which will contain the blocks shared between all the species.

Finally, a linked list for each traced block will be created. All the lists obtained in the previous step will be iterated, looking for the relations discovered between blocks and adding the blocks to a linked list accordingly. Figure 4 shows the pseudo-code for the BlockTracer programme.

[2] https://github.com/eseuteo/BlockTracer

```
procedure BLOCK_TRACER(List $comparison\_list$) // Load a list for each comparison blocks
    list_A ← ∅
    for $comparison$ in $comparison\_list$ do
        list_B ← comparison.get_blocks()
        list_A ← list_A.append(list_B)
    end for
     // Study overlapping between blocks:
    history ← ∅
    history ← history.append(list_A)
    while list_A.length > 1 do
        list_B ← ∅
        for $blocks\_1$ $blocks\_2$ in $list\_A$ do // i.e.: foreach pair of consecutive lists of blocks
            list_C ← check_overlap(blocks_1, blocks_2)
            list_B ← list_B.append(list_C)
        end for
        history ← history.append(list_B)
        list_A ← list_B
    end while
end procedure
```

Figure 4: Pseudocode for the BlockTracer programme.

Obtaining CSBs from HSPs: getCSB

Our workflow includes getCSB, a program that receives a .csv file containing information regarding HSPs, such as their location in both species (starting and ending coordinates), score, length, strand, etc., and returns a similar .csv file, where the suitable HSPs are merged and turned into CSBs. The code developed for this part of the project is available at its GitHub repository[3]. The data obtained with the getCSB software can, once the workflow execution has finished, be accessed in the Galaxy interface.

For the implementation of this tool we have relied on the following Python libraries:

- Pandas [48], for all the data-processing tasks related to dataframes, as it allows writing efficient, readable code for this purpose.
- Numpy [23], for all the auxiliary processes related to mathematical operations.

[3] https://github.com/eseuteo/getCSB

The getCSB programme works by carrying out the following steps (the corresponding pseudocode can be found in Figure 5):

1) It loads the input HSP file into a dataframe
2) Afterwards, it obtains the diagonal value of each HSP by subtracting the starting coordinates of the HSP on each axis ($diagonal = yStart - xStart$).
3) Then, the HSPs are sorted regarding two values: the diagonal value obtained in step 2 and the starting coordinate on the X axis (the reference species sequence). Sorting the HSPs by their diagonal provides a straightforward way of pre-clustering them. Most of the time all the HSPs that compose a CSB are located in a similar diagonal. Regarding the usage of the starting coordinate on the X axis, it was selected in order to not to provide an excessive weight to the diagonal value, as otherwise, two HSPs in the same diagonal but far from each other in the X axis could be considered as part of the same CSB.
4) Finally, the current dataframe containing the sorted HSPs is iterated and the CSBs are created. In order to assess if a HSP can be put into a CSB, it is checked whether their combined score, minus a penalty based on the distance between them (*i.e.:* it grows linearly regarding the gap between HSPs), is greater than zero. In the case that 5 consecutive HSPs cannot be part of the current CSB, no more HSPs will be taken into consideration, starting thus the search of a new CSB.

```
procedure GET_CSB(List $hsp\_df$)
    output_CSBs ← ∅
    hsp_df['diagonal'] ← hsp_df['xStart'] - hsp_df['yStart']
    sorted_df ← hsp_df.sort_by('diagonal', 'xStart')
    iter ← sorted_df.iterator()
    current_csb ← iter.next()
    while iter.has_next() do
        iter_candidates ← copy(iter)
        candidate ← iter_candidates.next()
        continue_value ← 1
        while continue_value > 0.1 do
            if is_suitable(candidate) then
                current_csb ← current_csb.update(candidate)
                iter ← iter.remove(candidate)
                continue_value ← min(continue_value * 2, 1)
            else
                continue_value ← continue_value / 2
            end if
            candidate ← iter_candidates.next()
        end while
        output ← output.append(current_csb)
        current_csb ← iter.next()
    end while
end procedure
```

Figure 5: Pseudocode for the get_CSB programme.

Figure 6 shows a simple example of the getCSB programme functioning. The input comparison (left), containing a huge number of HSPs is processed and a file containing considerably fewer CSBs is generated (right).

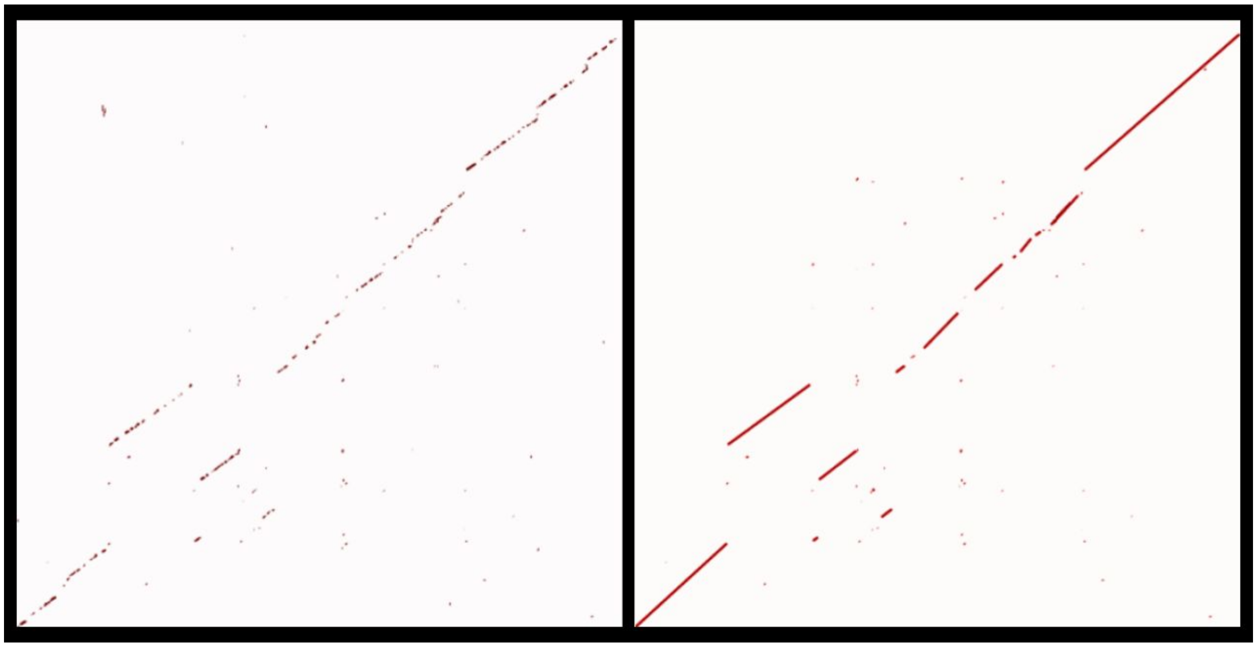


Figure 6: Example of the result provided by getCSB (right), given a HSP input file (left).

Workflow creation in a Galaxy instance

In the following section, the main steps that have to be done in order to create and register the workflow in a Galaxy instance will be described. This step is crucial regarding the future usage of our workflow in two main ways. Firstly, the creation of the workflow in a Galaxy instance will allow people interested in using the workflow to do it without tedious procedures such as run each part of the workflow, setting each tool's inputs, etc. Secondly, a thorough description of the steps needed to create and register this workflow in Galaxy will allow administrators to make our workflow available in their Galaxy instances, thus making the tool accessible in any Galaxy server.

The first requirement in order to register this workflow in a Galaxy instance is to ensure the availability of all the tools that compose it. Hereafter, the tools needed in order to run our workflow, as well as a brief description of them is available.

- Chromeister: A coarse-grained pairwise comparison approach
- Guided-Gecko: A fine-grained pairwise comparison approach that relies on Chromeister results.

- getCSB: A software for obtaining CSBs from HSPs.
- BlockTracer: A tool for tracing related CSBs between several species.

Besides these tools, a number of auxiliary scripts for file-management are also needed:

- File adapter: A tool for adapting Gecko's output files to BlockTracer's input files.
- File concatenator: A simple tool for concatenating files.

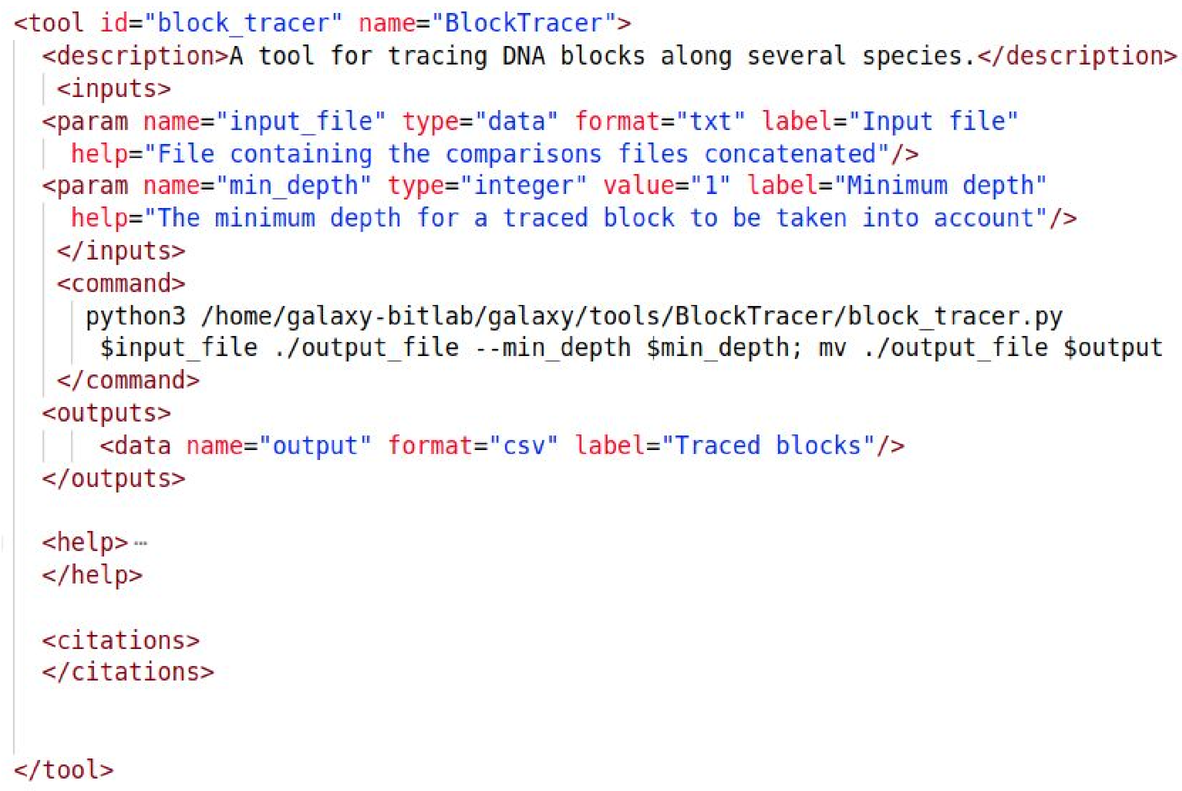
Tool registration in Galaxy

The basic process to perform a custom tool registration in Galaxy is available at the official tutorial[4]. Nonetheless, some special steps have to be taken into account and therefore, in the following section, they will be thoroughly described.

In order to register a tool in a Galaxy instance, the first step is to install the tool. In order to do this, the standard process is to clone the tool repository under the $GALAXYPATH/galaxy/tools/ directory and follow the corresponding instructions of each tool installation.

Once this is done, the tool definition file has to be created. This is an XML file containing the details regarding the execution of the tool, and has to be located in the same folder as the actual tool. Figure 7 shows the regular structure of this XML file.

[4] https://galaxyproject.org/admin/tools/add-tool-tutorial/



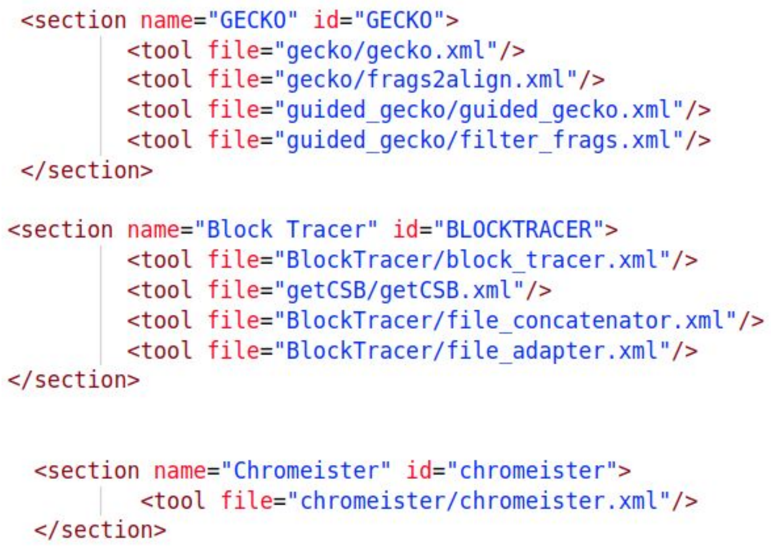
```
<tool id="block_tracer" name="BlockTracer">
  <description>A tool for tracing DNA blocks along several species.</description>
   <inputs>
  <param name="input_file" type="data" format="txt" label="Input file"
    help="File containing the comparisons files concatenated"/>
  <param name="min_depth" type="integer" value="1" label="Minimum depth"
    help="The minimum depth for a traced block to be taken into account"/>
   </inputs>
   <command>
     python3 /home/galaxy-bitlab/galaxy/tools/BlockTracer/block_tracer.py
      $input_file ./output_file --min_depth $min_depth; mv ./output_file $output
   </command>
  <outputs>
      <data name="output" format="csv" label="Traced blocks"/>
  </outputs>

  <help>…
  </help>

  <citations>
  </citations>


</tool>
```

Figure 7: Tool descriptor for the BlockTracer tool.

As can be seen, the elements that have to be available are:

- Description: A brief description regarding the tool, which will be shown in the Galaxy interface.
- Zero or more parameters: Indicated by the <param> label, they will contain information regarding the tool input parameters, such as their name, type, etc.
- Command: This will be the actual command that will run the tool.
- Outputs: Analogously to the parameters section, this provide information regarding the tool outputs, if any (name, format, etc.).
- Other fields: Some other fields can be set, such as a help section for making easier the use of the tool, or citations.

Finally, we have to make our Galaxy instance aware of the existence of the new tool. In order to do this, we have to modify the tool_conf.xml file under the /galaxy/config directory. Figure 8 depicts the structure of this file, which contains sections in which the different tool descriptors are listed,



```xml
<section name="GECKO" id="GECKO">
        <tool file="gecko/gecko.xml"/>
        <tool file="gecko/frags2align.xml"/>
        <tool file="guided_gecko/guided_gecko.xml"/>
        <tool file="guided_gecko/filter_frags.xml"/>
</section>

<section name="Block Tracer" id="BLOCKTRACER">
        <tool file="BlockTracer/block_tracer.xml"/>
        <tool file="getCSB/getCSB.xml"/>
        <tool file="BlockTracer/file_concatenator.xml"/>
        <tool file="BlockTracer/file_adapter.xml"/>
</section>


<section name="Chromeister" id="chromeister">
        <tool file="chromeister/chromeister.xml"/>
</section>
```

Figure 8: Part of the Galaxy configuration file.

Workflow creation

Once all the needed tools are available in our Galaxy instance, the actual workflow can be created. In order to do this, we can make use of the Galaxy workflow editor and use the drag-and-drop interface available for it. We are also providing the workflow as a standalone tool at the BitLab group repository[5].

[5] http://mango.ac.uma.es/compartir

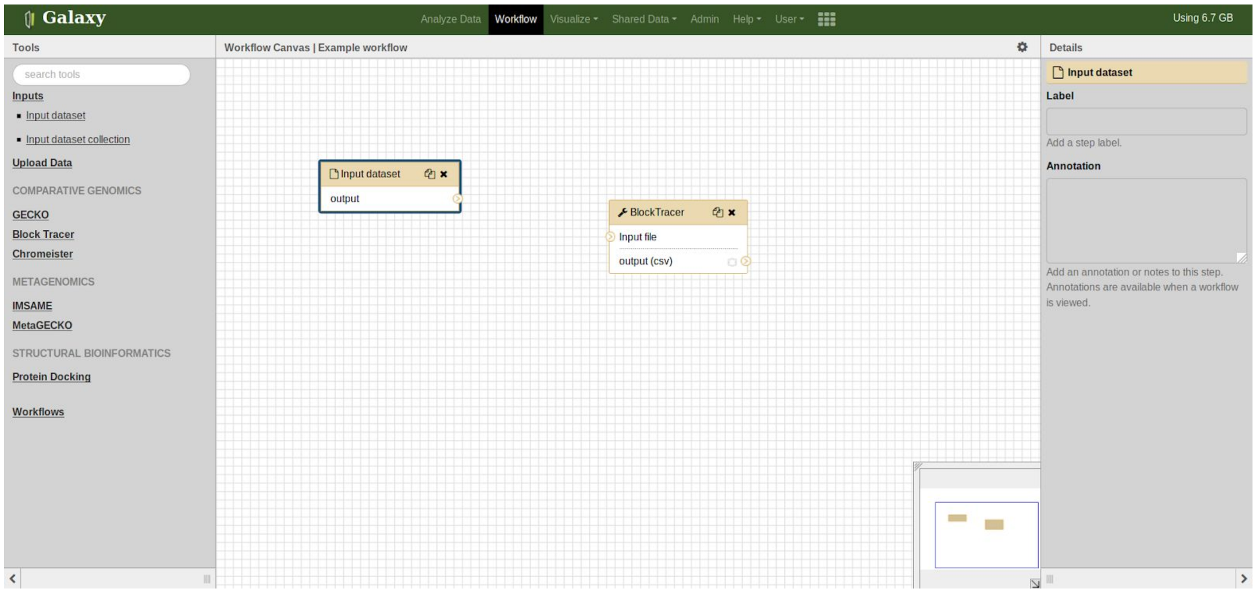


Figure 9: Galaxy workflow editor interface.

As can be seen in Figure 9, instance tools are available in the left column, and can be dragged to the editor. Once there, the different processes can be linked throughout their inputs and outputs.

The Galaxy-based BlockTracer workflow

In this final section of the Implementation and results chapter, the final result of our work will be described (*i.e.:* the workflow available in the galaxy instance after registering the different tools and creating the pipeline described at the beginning of this chapter). Two versions of the workflow will be shown; the first one is a 5-species particular exercise created with the Galaxy interface available in order to illustrate the characteristics of this platform, while the second one is a user-transparent version, which allows the execution of the workflow with 3 or more species.

Figures 10 and 11 show the details of the Galaxy Blocktracer workflow for the tracing of CSBs among 5 species chromosomes. This example intends to be didactic, in the sense that for the case of n species chromosomes, the tool used would be the user-transparent version. As it is shown, firstly the 5 input chromosomes are given to both Chromeister and Guided-Gecko processes. The Chromeister hits output file is used as input for Guided-Gecko (Figure 11).

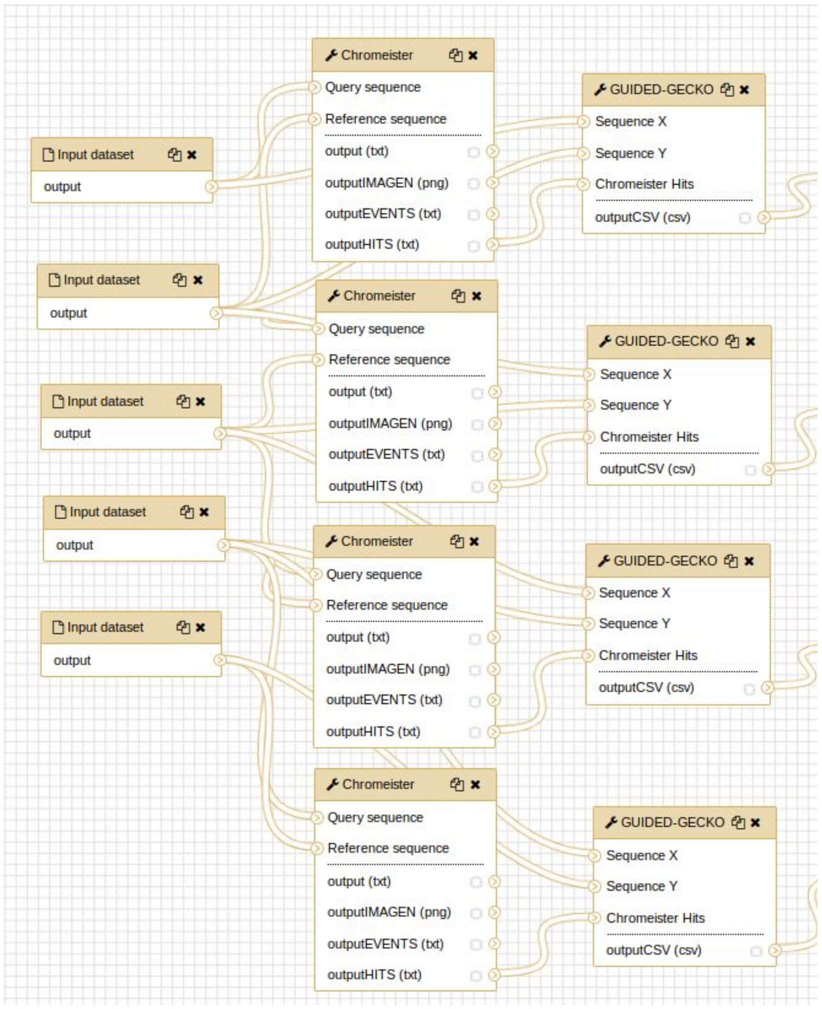


Figure 10: Galaxy Blocktracer Workflow (1/2).

After this (Figure 11), the Guided-Gecko output (*i.e.:* HSPs) is processed by getCSB, thus obtaining the CSBs. Then, as the BlockTracer program needs the input files to be in a different format than the Guided-Gecko one, the format is adapted. Following this, the four files are concatenated in one final file, which will be used by the BlockTracer program as the input. The workflow output, thus, will be a CSV containing information regarding the traced blocks (species and chromosomes that share the block, and coordinates of the block in their respective chromosomes).

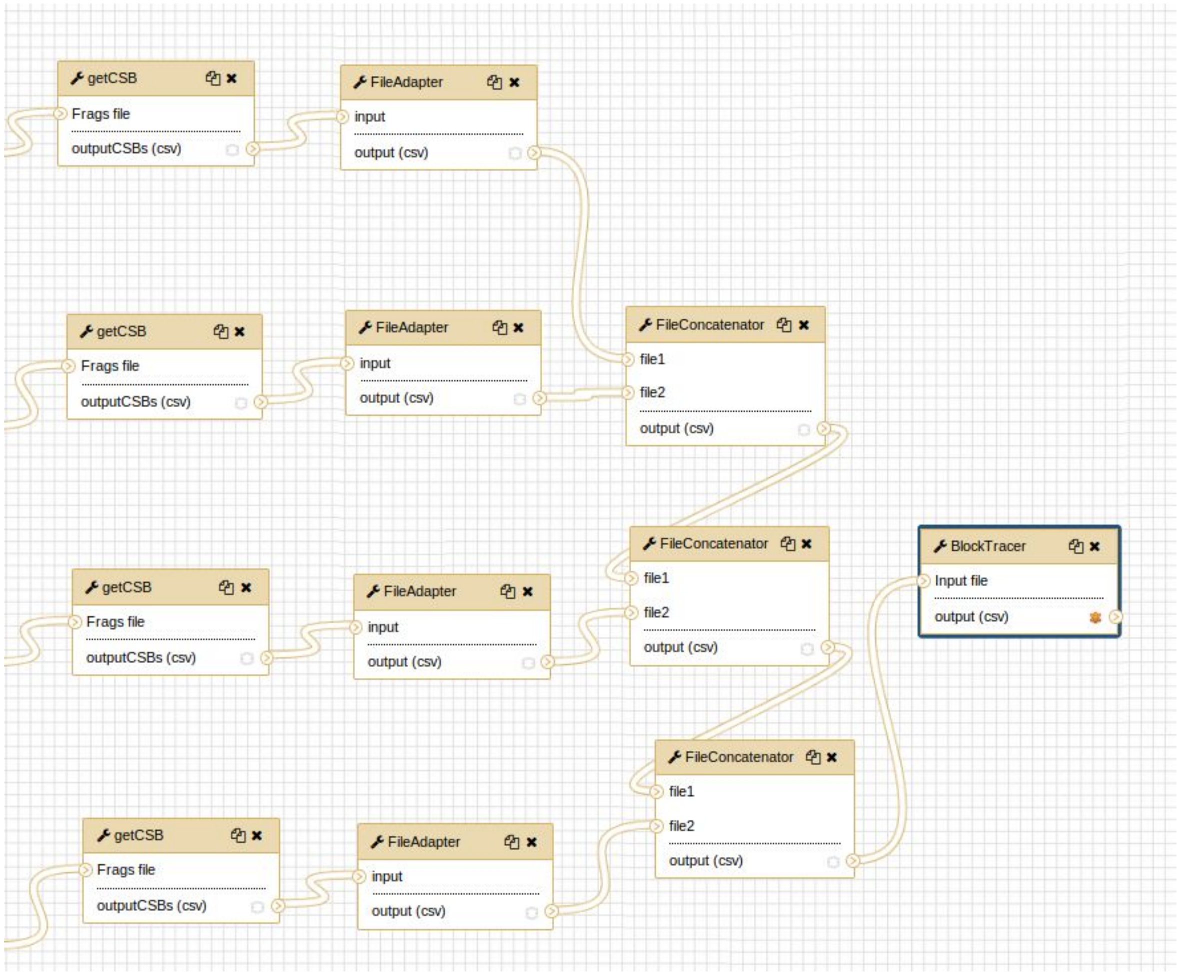
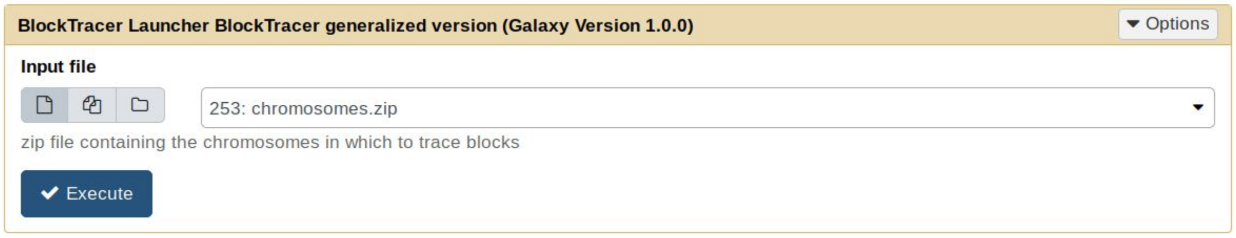


Figure 11: Galaxy Blocktracer Workflow (2/2).

The interface of the general, transparent version of the workflow is depicted in Figure 12. In this case, the input for the workflow is just a .zip file containing a folder with the sequences in which to trace the blocks and a set of optional parameters.



BlockTracer Launcher BlockTracer generalized version (Galaxy Version 1.0.0)

Input file

253: chromosomes.zip

zip file containing the chromosomes in which to trace blocks

Figure 12: BlockTracer general version interface.

5. Results and use case validation

In order to assess the suitability of the presented workflow, it is necessary to describe a use case in which the functionality of our tool is evaluated. In the current section we will analyse the characteristics of the developed workflow, as well as provide a description of two use cases. The first one aims to validate the pipeline functionality, whereas the second one is an analysis on its performance.

Case of use 1: Tracing of mammalian chromosomes

In order to carry out the experiment, two different executions were made. In the first one the coarse-grained version of our workflow was used, while in the second one the fine-grained approach was employed. Our goals with this experiment were: (1) compare the performance and results quality of the fine-grained version against the coarse grained one, and (2) assess the correctness of both approaches by contrasting their results with information from other sources.

The current use case consists on the usage of the BlockTracer workflow in order to find shared blocks between five species. The datasets used for our case of use are five chromosomes of five different mammal species, which are available at the Ensembl repository [49], and can be downloaded and used. The chromosomes selected were the following ones:

- *Homo sapiens* (human), chromosome 6
- *Pongo abelii* (orangutan), chromosome 6
- *Mus musculus* (mouse), chromosome 13
- *Equus caballus* (horse), chromosome 20
- *Bos taurus* (cow), chromosome 23

For the sake of clarity, from now onwards we will use the “HOMSA”, “PONAB”, “MUSMU”, “EQUCA” and “BOSTA” abbreviations to refer to each of the species listed, respectively.

These five mammals were used mainly due to the broad vision it provides with respect to the mammalians phylogeny, as they are not all closely related nor excessively unrelated. The chosen ordering of species is of relevance since more CSBs can be obtained and studied regarding the following species in the process. Therefore, the species were sorted regarding their phylogeny [50]. Thus, we took into account their similarity with the reference (*i.e.:* the HOMSA), in order to find DNA blocks that can be traced throughout all of them. Regarding the selection of the chromosomes, we used the Chromeister software as a screening tool, which provided insights with respect to the similarity of the chromosomes, as it informs us about which pairs of sequences are prone to have shared blocks. Therefore, we performed all-vs-all comparisons between HOMSA, PONAB, MUSMU, EQUCA and BOSTA chromosomes in order to obtain a set of sequences that were suitable for our experiment.

The results obtained were, at first glance, rather different (as we can see if the number of blocks detected are compared in Figure 14 (coarse-grained) and Figures 15, 16, and 17 (fine-grained)). Nonetheless, this behaviour is expected due to the contrast between both workflows level of detail. The usage of Chromeister in the first one provided a smaller number of blocks, but a clearer depiction of the big picture, while using Gecko in the second one yielded a larger number of more specific and accurate blocks.

Besides this fact, we find same-depth blocks traced with both methods. This is, we find related blocks between the five species. In order to analyse the correspondence of both results, we need to rely on external information. Thus, in order to assess the suitability of the solutions found, as well as in order to describe the shared information between them, we compare our results to a reference study [51]. Figure 13 shows four dotplots of pairwise chromosome comparisons between the five species of the current use case. Besides, there are some parts highlighted in red, which will come in hand next in this section. The blue elements in the images represent coincidences in the sequences (so, for instance, if both sequences were the same, a perfect blue diagonal would appear in the dotplot image).

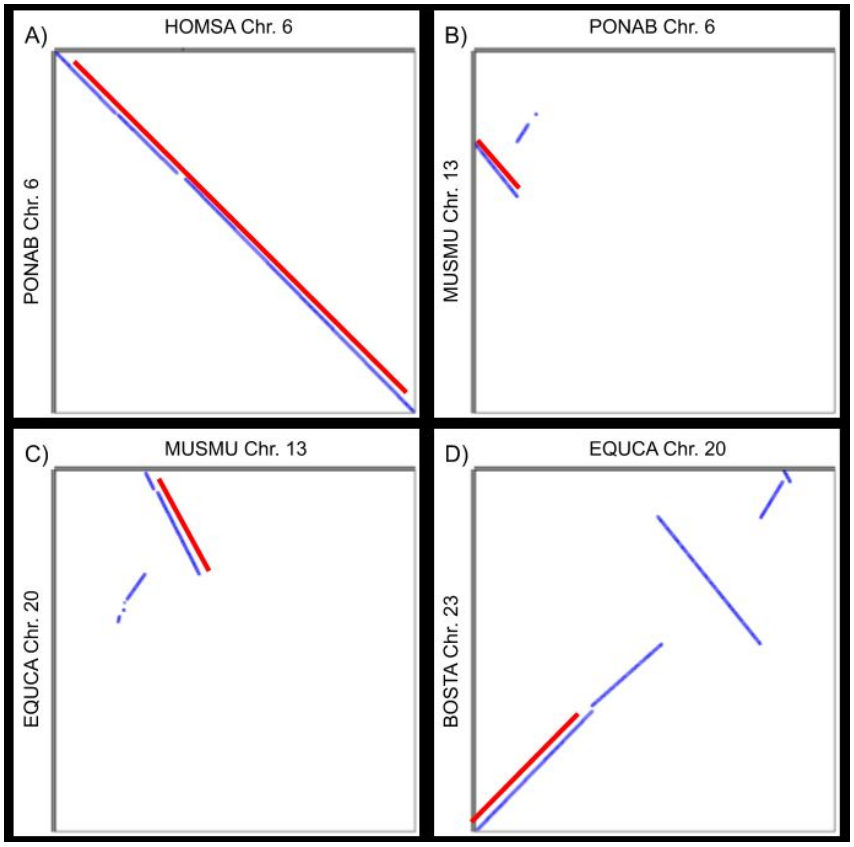


Figure 13: representation of pairwise comparisons between each pair of chromosomes selected for the experiment (A: HOMSA vs. PONAB; B: PONAB vs. MUSMU; C: MUSMU vs. EQUCA; D: EQUCA vs. BOSTA). The blue lines represent synteny blocks, whereas red lines represent traced blocks throughout the experiment described in the main body of the manuscript.

If we compare Figure 13 with the results obtained by the coarse-grained approach (Figure 14), we can see clear correspondence between them. HOMSA and PONAB share most of their 6th chromosomes (*i.e.:* a large block between them would be found), PONAB and MUSMU share two small blocks at the beginning of the 6th chromosome of the former and around the first ⅓ of the 13th chromosome of the latter, which is then related to a block at the beginning of the 20th EQUCA chromosome, which is finally found to be related to the ending part of the BOSTA 23rd chromosome.

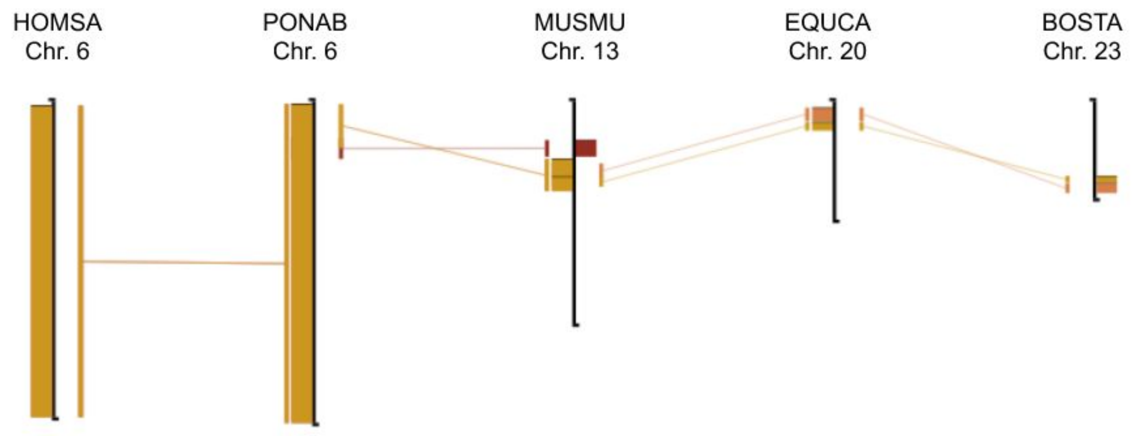
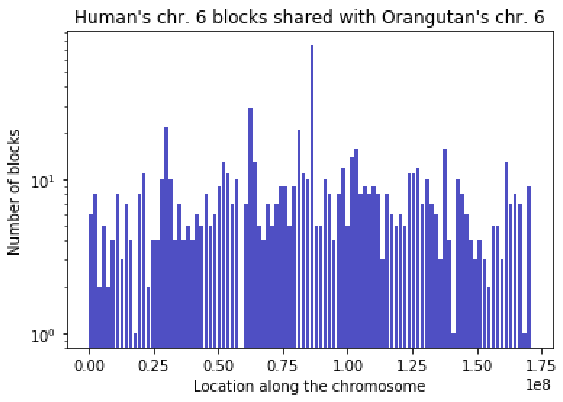


Figure 14: Depiction of the results obtained in the coarse-grained version of the BlockTracer workflow. Same colour lines represent shared regions.

When trying to find correspondence between chromosomes comparisons and the results of the fine-grained version of the BlockTracer workflow, it can be harder, as large blocks such as the one found between HOMSA and PONAB are rather "broken down into pieces", and trying to represent them analogously to Figure 14 can be impractical. Therefore, a new way of assessing the location of the blocks found by the fine-grained version have to be found. One option is to represent the blocks' starting coordinates in a histogram. In this way, we can find more blocks' starting points along the same places where the blocks are placed in the coarse-grained version. Figure 15 gives an account of this phenomenon.

Figure 15 (left) shows that the blocks can be found all along the human's 6th chromosome. This coincides with Figure 13A, as both chromosomes are closely related and almost all of it is shared between both species' chromosomes.



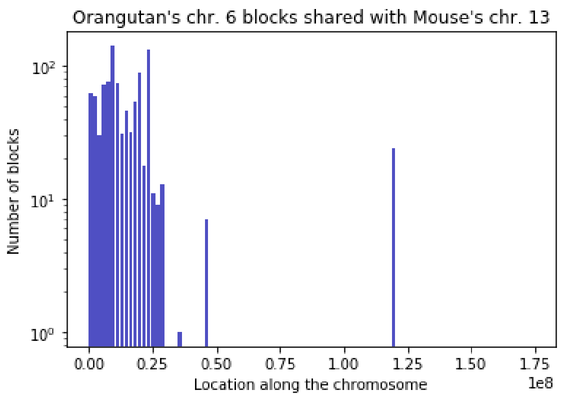


Figure 15: Histogram showing the 6th human chromosome blocks that are shared with the 6th orangutan chromosome (left). Histogram showing the 6th orangutan chromosome blocks that are shared with the 13th mouse chromosome (right).

Regarding Figure 15 (right), we find a different result from the previous figure. Nonetheless, the meaning is rather similar. In this case, the blocks shared between orangutan's 6th chromosome and mouse's 13th chromosome are mainly located at the beginning of the orangutan's chromosome (Figure 13B). Hence the histogram; we can see how the vast majority of the blocks starting coordinates are located in the mentioned region.

The next sequence in the experiment is the mouse 13th chromosome. In this case we find that all blocks that were found in the orangutan sequence are related to several blocks near the center of the chromosome. Figure 16 depicts this phenomenon. The blocks found correspond to the one highlighted in red in Figure 13C.

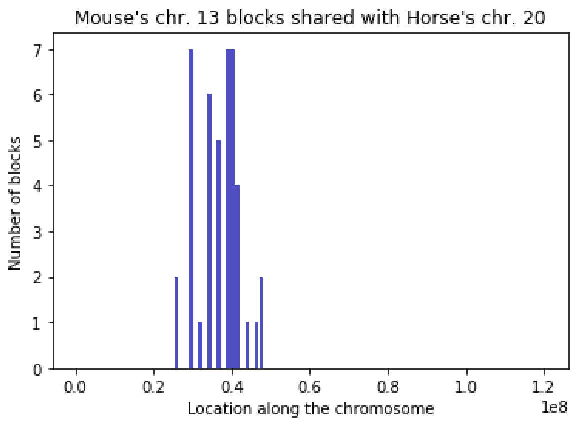


Figure 16: Histogram showing the 13th mouse chromosome blocks that are shared with the 20th horse chromosome.

Our experiment finally brings us to Figure 17. It shows two histograms, each one corresponding to one of the chromosomes involved (horse's 20th chromosome and cow's 23rd chromosome). An alternative depiction for this phenomenon is available in Figure 13 D.

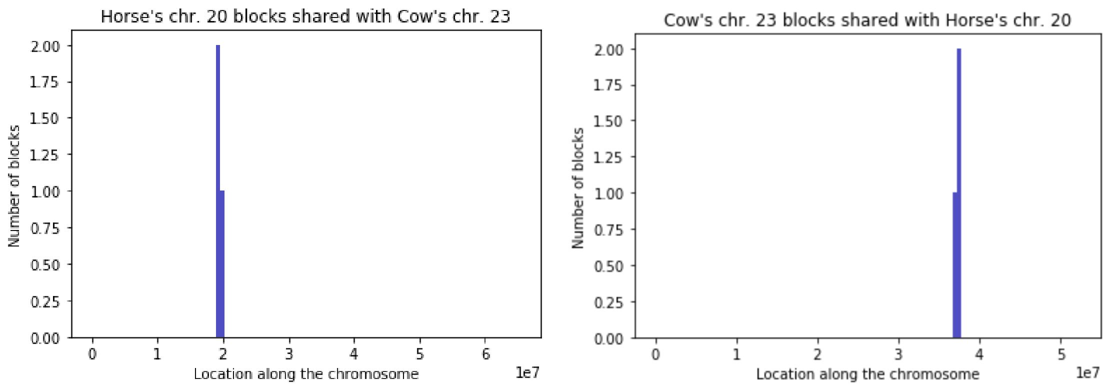


Figure 17: Histograms showing the 13th mouse chromosome blocks that are shared with the 20th horse chromosome.

Case of use 2: Studying BlockTracer computational performance

Our second case of use is aimed at assessing the capabilities of our workflow when facing different kinds of inputs and search spaces. The scalability of the proposed software is tested as well. In this sense, we carried out two experiments. The first experiment performed consisted on selecting 16 *Mycoplasma pneumoniae* bacteria sequences in order to obtain execution statistics. Afterwards, a similar experiment was carried out using the X chromosome from 16 species, in order to analyze the behaviour of the workflow in a demanding scenario. Besides measuring time and memory consumption results, both experiments were analyzed in order to study which input parameter had the largest impact on the performance.

The species selected were the following 16 mammalians, and in the following order: (1) *Homo sapiens*, (2) *Pan troglodytes*, (3) *Pan paniscus*, (4) *Gorilla gorilla*, (5) *Pongo abelii*, (6) *Papio anubis*, (7) *Macaca mulatta*, (8) *Macaca fascicularis*, (9) *Callithrix jacchus*, (10) *Rattus norvegicus*, (11) *Mus musculus*, (12) *Oryctolagus cuniculus*, (13) *Canis familiaris*, (14) *Felis catus*, (15) *Equus caballus*, (16) *Bos taurus*. All of them are available at the Ensembl repository [49]. These were selected due to (1) their evolutionary distance and (2) their X chromosomes were fully assembled.

Regarding the *Mycoplasma pneumoniae* sequences used, they were obtained from the NCBI repository[6], namely: *M129, FH, 309, PO1, PI 1428, M129-B7, 19294, M29, 39443, 51494, 54089, 54524, 85084, 85138, FH* and *M1139*.

The parameters selected in the previous steps to the execution of the BlockTracer software were changed in order for the number of blocks traced to be balanced between comparisons (the number of blocks between *Homo sapiens* and *Pan troglodytes* is quite larger than that between *Mus musculus* and *Oryctolagus cuniculus*). Therefore, the Gecko parameters referring to the minimum similarity and size of a HSP to be reported had to be set accordingly, increasing them when the species were closely related and relaxing them in the opposite case.

[6] https://www.ncbi.nlm.nih.gov/

Number of sequences	Execution time for bacteria (seconds)	Execution time for mammalians (seconds)	Memory usage for bacteria (megabytes)	Memory usage for mammalians (megabytes)
4	0.670	1.081	12.990	37.603
8	2.415	18.774	15.974	77.185
16	3.771	113.313	19.615	481.486

Table 1: BlockTracer average time and memory consumption for bacteria and mammalians experiments.

Table 1 shows the time and memory consumption for both experiments. It is noteworthy the difference regarding the sequences lengths; while the bacteria strains sizes were less than 1Mbp, the mammalians X chromosomes ranged between 80 and 160 Mbp. There are clear differences between both experiments, as the bacteria time and memory consumption grows linearly regarding the number of sequences in which the blocks are traced. On the other side, the time and memory needed for the chromosomes experiment seem to grow in faster pace. Given the differences in the size of the sequences in both experiments, we can speculate our approach provides a block tracing method that scales linearly regarding the number of sequences in which we want to trace blocks, whereas the scalability with respect to the length of the sequences involved is different.

6. Conclusions

Conclusions of the developed work

In the present manuscript, we have described a methodology for tracing both DNA blocks and CSBs along different species. Additionally, an implementation of a workflow that is time and space-efficient has been developed. Several low-coupled modules were arranged together in a complex pipeline which is able to obtain the location of DNA blocks that are shared by several species, in a transparent-to-the-user fashion. The developed workflow is available in a cloud Galaxy instance, and can be executed by any user, anywhere. The functionality of our workflow is aimed at helping Life Science researchers to enable genetic blocks tracing, effectively enlarging their grasp regarding the evolution underpinnings by studying the chromosomes structure and ultimately their relevance with respect to diseases and other health issues. Our work aims to be a step further regarding unveiling the conundrum that is the origin of life, and understanding the development of phenotypic traits (*v.g.:* behavioral traits, diseases, etc.), which can be ultimately utilised for the benefit of humankind. Among others, the developed workflow enables researchers to perform several studies, such as:

- Quantifying the rates of evolutionary events in species.
- Incorporating information regarding evolutionary events into other methods for generating species phylogenies.
- Analysing the underlying distributions that govern sequence mutation in order to predict future evolutionary events.
- Studying the consequences of evolutionary events in the species divergence.
- Validating evolutionary models.

The development of this work has thus provided a space and time-efficient method for tracing related blocks throughout genetic sequences, which could be also generalizable to large text files. In this sense, the potential applications of our work cover issues such

as (1) obtain information regarding the quantity of shared blocks between texts, (2) provide, with an adjustable accuracy, the location coordinates of those blocks and (3) assess the quality (*i.e.:* the rate of similarity between related blocks) of the blocks.
Besides, a heuristic method for enlarging HSPs found in pairwise genomic comparisons into larger pieces of information (CSBs) has also been provided, allowing the users to have a broader vision regarding relations between sequences, as larger-sized blocks mean the possibility of a coarse-grained approach.
Regarding the future work, our project opens the door to different studies of evolution underpinnings. It can be used in order to study the border regions of Evolutionary Events, where some key factors of genetic rearrangements could remain hidden. Besides, our work can enhance the research of other kinds of shared blocks, such as Segmental Duplications or Long Interspersed Nuclear Elements. Regarding the actual workflow, the design decisions made during the development process allows the modification and extension of our pipeline. Our work is thus open to enhancements, not only regarding the actual functionalities, but also with respect to its integration with alternative platforms and interfaces (apart from the current one, Galaxy). Besides, our workflow was developed within the BitLab research group environment, which takes part of the ELIXIR organisation, a european consortium devoted to Life Sciences research throughout collaboration between different countries. Thus, ongoing and future work in this area is guaranteed.

Conclusiones del trabajo desarrollado

En el presente trabajo se ha descrito una metodología para rastrear tanto bloques de ADN como bloques de sintenia computacionales a través de distintas especies. Adicionalmente, se ha llevado a cabo una implementación eficiente en tiempo y espacio de un flujo de trabajo. Varios módulos independientes han sido organizados en un proceso complejo que es capaz de obtener bloques de ADN compartidos por varias especies de forma transparente para el usuario. El flujo de trabajo desarrollado está disponible en una instancia de la plataforma Galaxy y puede ser ejecutado por cualquier usuario independientemente de dónde se encuentre. La funcionalidad del flujo de trabajo desarrollado pretende ayudar a los investigadores de las ciencias de la vida en la tarea del rastreo de bloques, consiguiendo así aumentar los conocimientos disponibles acerca de los mecanismos subyacentes a la evolución por medio del estudio de la estructura de los cromosomas, información que puede contribuir a la obtención de descubrimientos en cuanto a enfermedades y cuestiones que atañen a la salud humana. El trabajo desarrollado pretende ser un paso más en el esclarecimiento del origen de la vida y en la comprensión de los mecanismos de rasgos fenotípicos (desde rasgos comportamentales hasta enfermedades), lo que puede, ulteriormente, ser empleado en beneficio de la humanidad. Entre otros elementos, el presente trabajo permite a investigadores/as a realizar estudios en áreas como:

- Cuantificación de los ratios de eventos evolutivos en las especies.
- Incorporación de la información sobre eventos evolutivos en otros métodos para llevar a cabo filogenias.
- Analizar el funcionamiento de las mutaciones para predecir futuros eventos evolutivos.
- Estudiar las consecuencias de los eventos evolutivos en la divergencias entre especies.
- Validar modelos evolutivos.

El desarrollo de este trabajo, pues, proporciona un método eficiente en tiempo y espacio para rastrear bloques relacionados a lo largo de secuencias genéticas. Esto puede ser generalizable a ficheros de texto de gran envergadura. En este sentido, las potenciales aplicaciones que cubre el trabajo desarrollado son (1) la obtención de información acerca de la cantidad de bloques compartidos entre textos, (2) proveer, con una precisión ajustable, la localización de estos bloques y (3) evaluar la calidad (*i.e.:* el ratio de similaridad entre bloques relacionados) de los bloques.

Además se ha desarrollado un método heurístico para la ampliación de los HSPs hallados en comparaciones de genomas en bloques de información de mayor tamaño (CSBs), lo que permite a los usuarios a obtener una visión más amplia en cuanto a las relaciones entre secuencias, debido a que bloques de mayor tamaño significan la posibilidad de utilizar un enfoque de grano grueso.

Con respecto al trabajo futuro, el proyecto abre la puerta a diferentes estudios sobre los mecanismos en que se basa la evolución. Puede ser empleado para estudiar las regiones localizadas en los bordes de los eventos evolutivos, donde, al parecer, algunos factores claves de las reorganizaciones y mutaciones genéticas podrían encontrarse. Por otro lado, el trabajo desarrollado puede potenciar la investigación de otros tipos de bloques compartidos entre especies, como son las Segmental Duplications o las Long Interspersed Nuclear Elements. En cuanto al flujo de trabajo en sí, las decisiones de diseño tomadas durante el proceso de desarrollo permite la modificación y extensión de nuestro trabajo. Se trata de una herramienta abierta a mejoras, no sólo al respecto de las funcionalidades actuales, sino también en cuanto a la integración con otras plataformas e interfaces (aparte de la actual, Galaxy). Además, el flujo de trabajo ha sido desarrollado dentro del grupo de investigación BitLab, que forma parte de la organización Elixir, un consorcio europeo dedicado a la investigación en ciencias de la vida a través de la colaboración entre distintos países. Por lo tanto, el trabajo futuro en este área está garantizado.

7. Bibliography

1. Hirschberg, Daniel S. "Algorithms for the longest common subsequence problem." *Journal of the ACM (JACM)* 24.4 (1977): 664-675.

2. Heckel, Paul. "A technique for isolating differences between files." *Communications of the ACM* 21.4 (1978): 264-268.

3. Ullman, J. D., A. V. Aho, and D. S. Hirschberg. "Bounds on the complexity of the longest common subsequence problem." *Journal of the ACM (JACM)* 23.1 (1976): 1-12.

4. Rönnau, Sebastian, Geraint Philipp, and Uwe M. Borghoff. "Efficient change control of XML documents." *Proceedings of the 9th ACM symposium on Document engineering*. ACM, 2009.

5. Maier, David. "The complexity of some problems on subsequences and supersequences." *Journal of the ACM (JACM)* 25.2 (1978): 322-336.

6. Tsai, Yin-Te. "The constrained longest common subsequence problem." *Information Processing Letters* 88.4 (2003): 173-176.

7. Bergroth, Lasse, Harri Hakonen, and Timo Raita. "A survey of longest common subsequence algorithms." *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*. IEEE, 2000.

8. Quinn, Alexander J., and Benjamin B. Bederson. "Human computation: a survey and taxonomy of a growing field." *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2011.

9. Darwin, Charles (1872). “The Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life” (6th ed.). London: John Murray. OCLC 1185571.

10. Wooley, John C., Adam Godzik, and Iddo Friedberg. "A primer on metagenomics." *PLoS computational biology* 6.2 (2010): e1000667.

11. Schuster, Stephan C. "Next-generation sequencing transforms today's biology." *Nature methods* 5.1 (2007): 16

12. Schaller, Robert R. "Moore's law: past, present and future." *IEEE spectrum* 34.6 (1997): 52-59.

13. Torreno, Oscar, and Oswaldo Trelles. "Breaking the computational barriers of pairwise genome comparison." BMC bioinformatics 16, no. 1 (2015): 250.

14. Altschul, Stephen F., et al. "Basic local alignment search tool." *Journal of molecular biology* 215.3 (1990): 403-410.

15. Arjona-Medina, Jose A., and Oswaldo Trelles. "Computational Synteny Block: A framework to identify evolutionary events." *IEEE transactions on nanobioscience* 15.4 (2016): 343-353.

16. Takaoka, Tadao. "Efficient algorithms for the maximum subarray problem by distance matrix multiplication." *Electronic Notes in Theoretical Computer Science* 61 (2002): 191-200

17. https://www.businessprocessglossary.com/7002/workflow (Last accessed April, 2019)

18. Afgan, Enis, et al. "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update." *Nucleic acids research* 44.W1 (2016): W3-W10.

19. Wolstencroft, Katherine, et al. "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud." *Nucleic acids research* 41.W1 (2013): W557-W561.

20. Deelman, Ewa, et al. "Pegasus, a workflow management system for science automation." *Future Generation Computer Systems* 46 (2015): 17-35.

21. https://elixir-europe.org/ (Last accessed June, 2019)

22. McKinney, Wes. "Data structures for statistical computing in python." *Proceedings of the 9th Python in Science Conference*. Vol. 445. 2010.

23. Oliphant, Travis E. *A guide to NumPy*. Vol. 1. USA: Trelgol Publishing, 2006.

24. Gusfield, Dan. Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge university press, 1997.

25. Blum, Christian, and Maria J. Blesa. "A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem." *Journal of Heuristics* (2018): 1-29.
26. Hunt, James W., and Thomas G. Szymanski. "A fast algorithm for computing longest common subsequences." *Communications of the ACM* 20.5 (1977): 350-353.
27. Ahuja, Ravindra K., et al. "A survey of very large-scale neighborhood search techniques." *Discrete Applied Mathematics* 123.1-3 (2002): 75-102.
28. Easton, Todd, and Abhilash Singireddy. "A large neighborhood search heuristic for the longest common subsequence problem." *Journal of Heuristics* 14.3 (2008): 271-283.
29. Ning, Kang. "Deposition and extension approach to find longest common subsequence for thousands of long sequences." *Computational biology and chemistry* 34.3 (2010): 149-157.
30. Blum, Christian, Maria J. Blesa, and Manuel López-Ibáñez. "Beam search for the longest common subsequence problem." *Computers & Operations Research* 36.12 (2009): 3178-3186.
31. https://www.ncbi.nlm.nih.gov/Class/MLACourse/Original8Hour/Genetics/nucleotide.html (last accessed May, 2019)
32. https://ghr.nlm.nih.gov/primer/genomicresearch/snp (last accessed May, 2019)
33. https://www.ncbi.nlm.nih.gov/books/NBK62051/ (last accessed May, 2019)
34. Bentley, Jon. "Programming pearls: perspective on performance." *Communications of the ACM* 27.11 (1984): 1087-1092.
35. Bae, Sung Eun, and Tadao Takaoka. "Algorithms for the problem of k maximum sums and a VLSI algorithm for the k maximum subarrays problem." *7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004. Proceedings.*. IEEE, 2004.
36. Tamaki, Hisao, and Takeshi Tokuyama. "Algorithms for the Maxium Subarray Problem Based on Matrix Multiplication." *SODA*. Vol. 1998. 1998.

37. Bae, Sung E., and Tadao Takaoka. "Improved algorithms for the k-maximum subarray problem for small k." *International Computing and Combinatorics Conference*. Springer, Berlin, Heidelberg, 2005.

37. Piccolo, Stephen R., and Michael B. Frampton. "Tools and techniques for computational reproducibility." *GigaScience* 5.1 (2016): 30.

38. Liu, Ji, et al. "A survey of data-intensive scientific workflow management." *Journal of Grid Computing* 13.4 (2015): 457-493.

39. De Roure, David, et al. "Towards open science: the myExperiment approach." *Concurrency and Computation: Practice and Experience* 22.17 (2010): 2335-2353.

40. https://galaxyproject.org/galaxy-project/statistics/ (Last accessed April, 2019)

41. https://pegasus.isi.edu/documentation/tutorial_wf_generation.php (Last accessed April, 2019)

42. https://usegalaxy.eu/ (Last accessed June, 2019)

43. Langmead, Ben, et al. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome." *Genome biology* 10.3 (2009): R25.

44. Li, Ruiqiang, et al. "SOAP: short oligonucleotide alignment program." *Bioinformatics* 24.5 (2008): 713-714.

45. Harris, Robert S. "Improved Pairwise Alignmnet of Genomic DNA." (2007).

46. Perez-Wohlfeil, E., Diaz-del-pino, S., Trelles, O. "Ultra-fast genome comparison for large-scale genomic experiments." Scientific reports (2019) (accepted).

47. Vijaymeena, M. K., and K. Kavitha. "A survey on similarity measures in text mining." *Machine Learning and Applications: An International Journal* 3.2 (2016): 19-28.

48. McKinney, Wes. "pandas: a Python data analysis library." *see http://pandas. pydata. org* (2015).

49. Zerbino, Daniel R., et al. "Ensembl 2018." *Nucleic acids research* 46.D1 (2017): D754-D761.

50. Miller, Webb, et al. "28-way vertebrate alignment and conservation track in the UCSC Genome Browser." *Genome research* 17.12 (2007): 1797-1808.

51. http://narcisse.toulouse.inra.fr/animals/cgi-bin/narcisse.cgi (Last accessed June, 2019)