**Ivan Cardona Ferrer**

# ACTUALITZACIÓ, MANTENIMENT I AMPLIACIÓ D'UN SOFTWARE PER AL PROCESSAMENT D'ARBRES FILOGENÈTICS

**TREBALL DE FI DE GRAU**

**dirigit per Aïda Valls Mateu**

**Doble titulació de grau en Enginyeria Informàtica i en Biotecnologia**

UNIVERSITAT ROVIRA I VIRGILI

**Tarragona**

**2024**

**Resum.**

Les tecnologies de tipus Next Generation Sequencing han experimentat un augment exponencial en els darrers anys. Això ha creat una creixent necessitat en el desenvolupament de programes informàtics que siguin capaços de processar i interpretar els milers de milions de dades biològiques que generen aquestes tecnologies. Les aplicacions elaborades per a tal propòsit requereixen d'una actualització i manteniment constant, donats els ràpids avanços i continus descobriments tant en el món informàtic com en el biològic. Aquest treball pretén ser un clar exemple de la feina tan necessària que s'ha de dur a terme per mantenir actualitzat i en constant evolució el programari especialitzat en el processament i interpretació de dades òmiques. Concretament, l'aplicació informàtica tractada en aquest treball, anomenada Phylograph, consisteix en un editor d'arbres filogenètics que va deixar de ser funcional fa diversos anys. S'han dut a terme diverses tasques d'actualització i manteniment per tal d'aconseguir una nova versió executable actualitzada i funcional. A banda, s'ha dut a terme el disseny i implementació de noves funcionalitats per poder mostrar i tractar dos tipus de representacions d'arbre filogenètic de gran interès avui dia, com són la representació polar amb distàncies i la representació polar sense distàncies. També s'ha implementat la possibilitat d'emmagatzemar els documents que contenen les filogènies en multitud de formats i s'ha habilitat la interoperabilitat entre ells. La nova versió de Phylograph s'ha validat amb diversos conjunts de dades corresponents a seqüències víriques. El resultat de la validació ha estat exitós per totes les funcionalitats de l'aplicació. S'han fet proves satisfactòries també amb aquelles filogènies amb un nombre molt gran de dades a tractar, que són les que suposen un repte més important per a aquest tipus d'eines informàtiques.

**Resumen**.

Las tecnologías de tipo Next Generation Sequencing han experimentado un aumento exponencial en los últimos años. Esto ha creado una creciente necesidad en el desarrollo de programas informáticos que sean capaces de procesar e interpretar los miles de millones de datos biológicos que generan estas tecnologías. Las aplicaciones elaboradas para tal propósito requieren de una constante actualización y mantenimiento, dados los rápidos avances y continuos descubrimientos tanto en el mundo informático como en el biológico. Este trabajo pretende ser un claro ejemplo del trabajo tan necesario que debe llevarse a cabo para mantener actualizado y en constante evolución el software especializado en el procesamiento e interpretación de datos ómicos. Concretamente, la aplicación informática tratada en este trabajo, llamada Phylograph, consiste en un editor de árboles filogenéticos que dejó de ser funcional hace varios años. Se han llevado a cabo diversas tareas de actualización y mantenimiento para conseguir una nueva versión ejecutable actualizada y funcional. Además, se ha llevado a cabo el diseño e implementación de nuevas funcionalidades para poder mostrar y tratar dos tipos de representaciones de árbol filogenético de gran interés hoy en día, como son la representación polar con distancias y la representación polar sin distancias. También se ha implementado la posibilidad de almacenar los documentos que contienen las filogenias en multitud de formatos y se ha habilitado la interoperabilidad entre ellos. La nueva versión de Phylograph se ha validado con varios conjuntos de datos correspondientes a secuencias víricas. El resultado de la validación ha sido exitoso para todas las funcionalidades de la aplicación. Se han realizado pruebas satisfactorias también con aquellas filogenias con un número muy grande de datos a tratar, que son las que suponen un reto más importante para este tipo de herramientas informáticas.

**Abstract**.

Technologies such as Next Generation Sequencing have experienced an exponential increase in recent years. This has created a growing need in the development of computer programs that are capable of processing and interpreting the billions of biological data generated by these technologies. The applications developed for this purpose require constant updating and maintenance, given the rapid advances and discoveries in both the IT and biological worlds. This work aims to be a clear example of the much-needed work that must be carried out to keep software specialized in processing and interpretation of omic data up-to-date and constantly evolving. Specifically, the computer application discussed in this work, called Phylograph, consists of a phylogenetic tree editor that ceased to be functional many years ago. Several update and maintenance tasks have been conducted in order to achieve a new, updated and functional executable version. In addition, the design and implementation of new functionalities has been carried out to be able to show and treat two types of phylogenetic tree representations of great interest today, such as the polar representation with distances and the polar representation without distances. The possibility of storing the documents containing the phylogenies in a multitude of formats has also been implemented and so the interoperability between them. The new version of Phylograph has been validated with several datasets corresponding to viral sequences. The result of the validation has been successful for all the functionalities of the application. Satisfactory tests have also been conducted with those phylogenies with a very large number of data to deal with, which are the ones that represent the most important challenge for these types of computer tools.

## Agraïments

La realització d'aquest treball no hauria sigut possible sense el suport que en tot moment m'ha proporcionat l'empresa. Des del primer dia em vaig sentir acollit com un membre més d'aquella xicoteta família que representa Biotech Vana SL. Voldria mostrar el meu agraïment per haver-me confiat aquest projecte que guardaven amb tanta estima i tanta cura, i per haver-me brindat l'oportunitat de ser la persona encarregada de dotar-lo de nou de vitalitat.

Particularment, voldria donar les gràcies a Carlos, el cap de l'empresa, qui va confiar-me aquesta tasca i em va encoratjar per arribar fins ací, obrint-me al seu torn la porta a un món laboral ple d'oportunitats. També vull donar les gràcies a Genís, qui sempre ha estat per ajudar-me en la mesura del possible amb tots els problemes que m'anaven sorgint; a Ahmed, per haver-me guiat i orientat amb els seus coneixements sobretot al principi quan encara no estava familiaritzat amb l'aplicació; a Bea, per donar-me ànims i haver-me ajudat amb aquells dubtes d'índole més biològica; i finalment a Ana, Raquel i Aya per mostrar-me el seu suport tant de manera directa com indirecta.

A banda, no voldria deixar sense agrair la meua família, amics i éssers volguts, els quals han contribuït a fer de mi qui soc avui dia i m'han acompanyat en tota aquesta etapa tan important per a mi.

Finalment, donar les gràcies a Aïda Valls, qui m'ha tutoritzat aquest projecte i sempre s'ha mostrat atenta al desenvolupament del mateix, proporcionant-me de manera constant suggeriments i crítiques constructives per tal d'acabar fent una millor feina.

# Índex

# Índex de taules

# Índex de figures

# 1 Introducció

La **bioinformàtica** té un paper fonamental en la investigació biològica i biomèdica avui dia. A través d'aquesta branca científica-tècnica es facilita una ràpida interconnexió entre els diferents avanços en les tecnologies de seqüenciació de tercera generació, conegudes com a NGS[1]. Aquestes tecnologies van enfocades a l'anàlisi de **dades òmiques** [1], és a dir, aquelles dades que es generen fruit d'algun tipus de mesurament en molècules biològiques. Aquest tipus d'informació és utilitzada a sovint per estudiar les relacions entre les diferents molècules i com la seva interacció afecta al funcionament general de les cèl·lules, teixits i organismes. El terme "òmica" es refereix a diversos camps d'investigació biològics, que poden classificar-se segons el tipus de molècula objecte d'estudi. Aquests poden ser metabolòmica (estudi dels metabòlits -molècules implicades en el metabolisme-), genòmica (estudi dels gens -ADN-), transcriptòmica (estudi dels transcriptomes -molècules de RNA-), proteòmica (estudi de les proteïnes) o metagenòmica (estudi de la diversitat genètica als ecosistemes).

Els anàlisis de dades òmiques permeten estudiar i entendre millor sistemes biològics complexos, com per exemple els humans, facilitant així la comprensió de l'organisme i ajudant al desenvolupament de la medicina personalitzada. Al seu torn, faciliten la identificació de marcadors de malaltia, molt importants per a la síntesi de nous fàrmacs. L'aplicació efectiva d'aquestes tècniques incideix de manera directa en tota la cadena analítica de valor biològic i sanitari. Això és degut a que la decodificació i gestió de dades biològiques requereix enfrontar-se al processament de milers de milions de lectures de seqüenciació, conegudes com a *reads*, solament abordables a partir de l'automatització de les diverses tasques per anàlisi.

Aquest procés d'automatització de les tasques analítiques requereix del desenvolupament de programes informàtics i software especialitzat per poder cobrir aquestes necessitats tan sofisticades. Aquest Treball Fi de Grau aborda el problema presentat a partir de les necessitats de l'empresa on he treballat, "Biotech Vana SL". Per tal de poder explicar millor els objectius del projecte, primer s'exposen uns quants conceptes clau de diferents disciplines científiques. A continuació, es presenta el *software* propietat intel·lectual de l'empresa anomenat *Phylograph*, en el qual s'ha centrat el projecte, i introdueixo informació sobre la mateixa empresa on s'ha realitzat.

## 1.1 Conceptes Clau

Abans de passar a explicar el programari objecte d'estudi, veurem una sèrie de conceptes importants per poder comprendre d'una manera clara i entenedora l'abast de l'aplicació bioinformàtica en la qual he treballat.

### *1.1.1 Bioinformàtica*

La bioinformàtica és una subdisciplina científica que involucra l'ús de ciències informàtiques per obtenir, emmagatzemar, analitzar i disseminar dades i informació biològiques, tals com seqüències d'ADN i aminoàcids, o anotacions sobre aquelles seqüències. Els científics i el personal clínic utilitzen bases de dades que organitzen i cataloguen aquesta informació biològica per augmentar l'enteniment de la salut i la malaltia i, en determinats casos, s'utilitzen per proveir millor atenció mèdica [2].

[1] Next Generation Sequencing

Dit d'una altra manera, la bioinformàtica és un camp de les ciències computacionals que duu a terme l'anàlisi de seqüències de molècules biològiques. Normalment s'aplica als gens, a l'ADN, a l'ARN, o a les proteïnes, i resulta especialment útil per comparar seqüències de gens i proteïnes entre diferents organismes, podent veure les relacions evolutives entre els organismes, i intentant esbrinar quina és la funció d'aquests gens i proteïnes. Realitzant una comparativa, podríem dir que la bioinformàtica s'encarrega de la part lingüística de la genètica. De la mateixa manera que els lingüistes estudien els patrons en el llenguatge, els bioinformàtics estudien els patrons a les seqüències d'ADN o de proteïnes.

#### 1.1.1.1 Next Generation Sequencing

**Next-generation sequencing (NGS)** és una tecnologia de seqüenciació massivament paral·lela utilitzada per a seqüenciar ADN i ARN, així com per a la detecció de variants/mutants [3]. Aquesta tecnologia té la capacitat de seqüenciar centenars i milers de gens o inclús genomes complets en un període de temps curt. Les mutacions o variacions de seqüència detectades per NGS permeten el diagnòstic de malalties i la presa de decisions terapèutiques especialitzades així com d'un millor seguiment i tractament del pacient. Tot això oferint noves oportunitats per al desenvolupament de la medicina personalitzada.

### *1.1.2 Filogenètica*

La filogenètica és l'estudi de les relacions entre organismes, basades en el seu material genètic revelat a través de la seqüenciació de DNA i RNA.

#### 1.1.2.1 Arbre Filogenètic

Un **arbre filogenètic**, també anomenat **filogènia**, és una manera de representar visualment relacions evolutives. Podríem dir que és la millor conjectura per a un científic sobre com ha evolucionat un organisme o grup d'organismes. Els biòlegs evolutius utilitzen els arbres filogenètics per a descriure com individus o grups estan emparentats entre ells a través d'un ancestre comú [4].

Els arbres filogenètics estan composats per "**fulles**" i "**branques**". Les fulles es troben als extrems de l'arbre i representen els organismes l'evolució dels qual estem descrivint. Aquesta representació pot ser una lletra, una imatge o el nom de la espècie/element genètic. Una fulla pot representar un organisme individual, una espècie sencera o inclús un únic gen, i es referida de manera tècnica amb el nom de **taxa** o **OTU**[2]. Cada fulla està connectada per una sèrie de branques, les quals representen les relacions de les fulles entre sí. A la *Figura 1* podem observar l'estructura bàsica amb els diferents components.

[2] OTU: Operational Taxonomical Unit

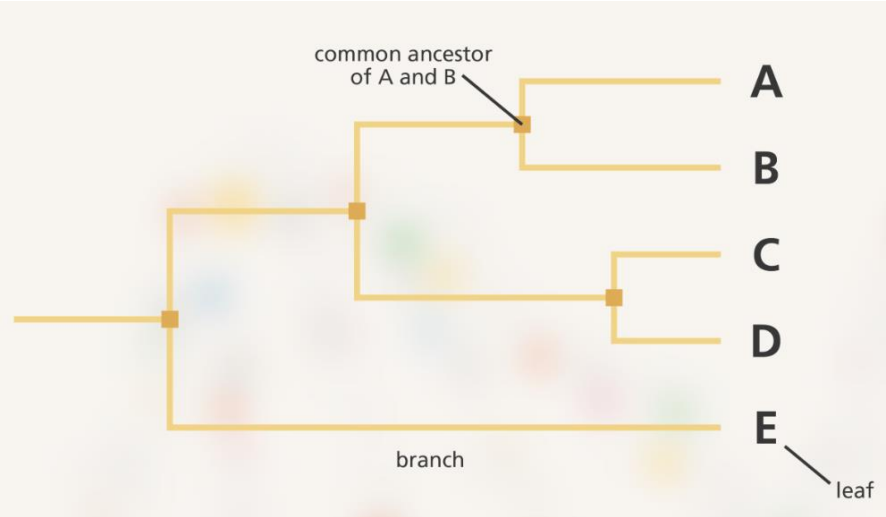



**Figura 1.** Diagrama bàsic d'un arbre filogenètic.
Font: https://www.yourgenome.org/theme/what-is-phylogenetics/

El procés constructiu de tot arbre filogenètic consta de diversos passos [5]. Es parteix d'un ancestre comú conegut, el qual representa l'**arrel**, i partir d'aquest surten diferents branques cada vegada que ocorre un canvi evolutiu. Posem per cas que el canvi evolutiu es correspon amb un nou tret que fa a l'organisme que el desenvolupa diferent respecte als altres individus (p.e. l'habilitat de volar o un bec més llarg). En el moment que ocorre aquest canvi es forma una nova branca a l'arbre, i totes aquelles branques que en el futur en derivin d'aquesta posseiran aquest tret com a comú (aquest agrupament s'anomena **clade**). Per exemple, l'ancestre comú de totes les aus és un dinosaure que va desenvolupar ales farà uns 150 milions d'anys. Per tant, totes les noves branques que van sorgir a partir d'aquest individu tenen algun tret nou però totes corresponen a organismes amb ales (podríem dir que tots els successors de l'ancestre comú de les aus formen un clade).

Un altre concepte important és el de **node**. Les branques que difereixen en els diferents nivells de l'arbre tenen un ancestre comú en algun punt, el qual es pot classificar estructuralment com a **node intern**. Treballant amb l'exemple anterior, en algun punt evolutiu a partir del desenvolupament d'ales per part de l'ancestre comú de les aus, es va produir un canvi pel qual un d'aquests organismes va adquirir a més la capacitat de volar. Per tant, a la nova branca de l'arbre ara tindríem una altra bifurcació. L'ancestre comú de les aus seria un node intern, i a partir d'aquest sortiria una nova branca corresponent a les aus amb capacitat de volar i una altra amb aquelles que no tenen la capacitat adquirida. Si no hi tinguéssim més nous canvis a les noves branques comentades, aquestes representarien el nivell final de l'arbre fent la funció de fulla o **node extern**.

A partir d'aquesta informació podem deduir que les fulles amb branques compartides més recents estan més emparentades entre sí (comparteixen més trets en comú), mentre que com més nodes interns (també anomenats *branching points* en anglès) tinguem separant dues fulles més llunyanament emparentades entre sí estaran (menys trets comuns). No obstant això, algunes vegades els organismes perden trets amb els quals van evolucionar fruit d'una adaptació a un ambient canviant (p.e. en el nostre cas tractat, un individu que va desenvolupar en el passat la capacitat de volar la perd en algun moment). Per aquest motiu, la realització i l'estudi de filogènies és molt important. A la *Figura 2* podem veure un exemple relacionat.

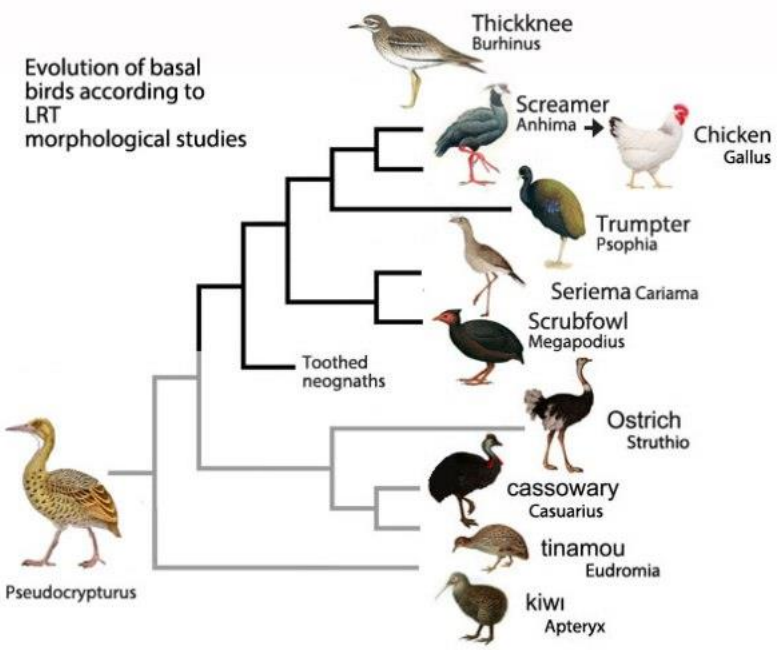



**Figura 2.** Exemple d'arbre filogenètic mostrant les relacions evolutives entre diferents espècies d'aus.
**Font:** https://pterosaurheresies.wordpress.com/2018/03/30/bird-phylogeny-molecules-vs-morphology/

Fins ara hem posat per cas la construcció clàssica d'arbres filogenètics basada en l'observació de canvis en trets físics i/o comportaments. Però, recentment també existeix la possibilitat de generar filogènies a partir de la seqüenciació dels genomes dels organismes, centrant-nos en canvis en la seqüència de DNA com a punt d'embrancament. Això és d'especial utilitat quan tractem amb virus i bactèries, ja que l'observació clàssica en aquests casos no és ni molt menys determinant.

Per últim, cal remarcar que tots els arbres filogenètics són hipòtesis, és a dir, representen la nostra millor estimació sobre com agrupar organismes basant-nos en característiques que siguem capaços de mesurar. Amb les noves tècniques de seqüenciació, podem reconstruir una filogènia examinant les seqüències de nucleòtids i/o proteïnes i combinant aquestes amb el nostre enteniment i diferents models evolutius. D'aquesta manera podem esbrinar amb més precisió esdeveniments evolutius que van ocórrer en el passat, al mateix torn que obtenim més informació sobre els processos evolutius que es donen a les seqüències. I tot això proporcionant un major enteniment sobre com funciona l'evolució, possibilitant també el desenvolupament de millors models matemàtics informàtics evolutius.

### *1.1.3 Tipus de Representacions d'Arbres Filogenètics*

Els arbres filogenètics es poden representar visualment de diferents maneres [6], [7]. L'elecció d'un tipus o un altre de representació dependrà dels criteris a escollir per part de la persona investigadora.

#### 1.1.3.1 Sense Arrel o Radial

Els arbres **sense arrel** il·lustren la relació entre els nodes que actuen com a fulles sense realitzar suposicions sobre la seva ascendència. No tenen una arrel especificada i només mostren el patró de ramificació de les relacions evolutives entre taxons o OTUs, sense cap informació sobre el seu avantpassat comú. Els taxons es poden disposar formant una representació **radial**, amb els nodes al voltant d'un punt central de manera que els més propers apareixen agrupats. Podem veure un exemple a la *Figura 3*.

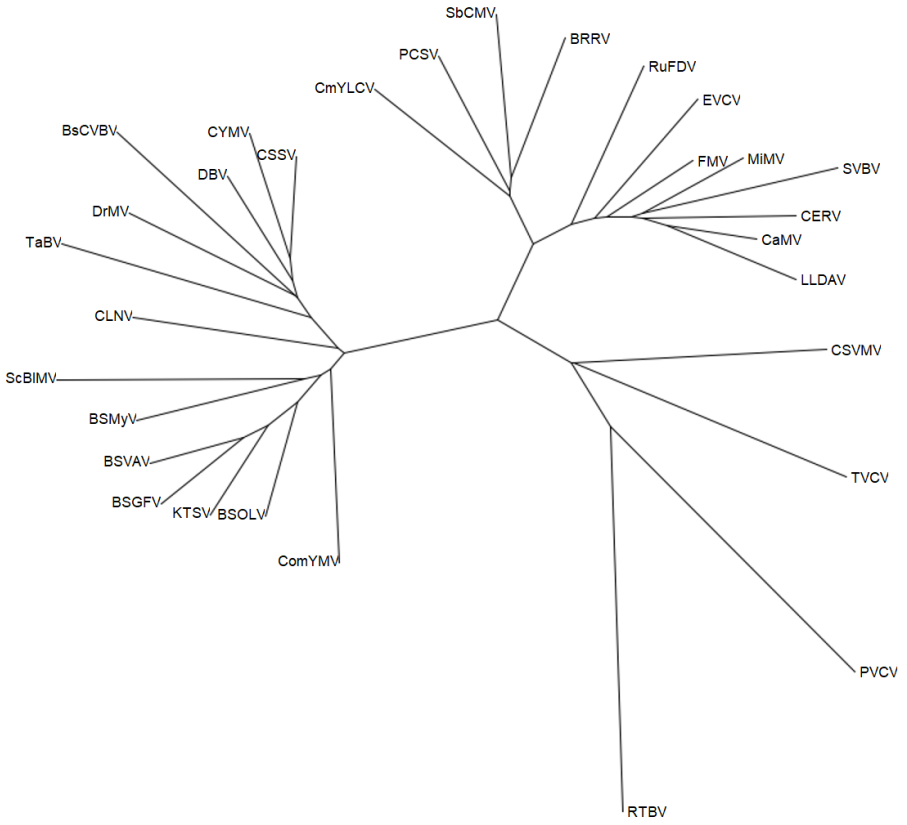



**Figura 3.** Representació radial de la filogènia de *AP_caulimovirus*.

Font: material de treball.

### 1.1.3.2 Fenograma

Un **fenograma** és un gràfic en arbre que utilitza informació basada en diferències o similituds fenotípiques (trets característics) per a la seva construcció, sense tenir en compte l'historial evolutiu. Difereix respecte als cladogrames i filogrames, què sí empren dades sobre les relacions evolutives per a construir la representació. Podem veure un exemple d'aquesta representació a la *Figura 4.*

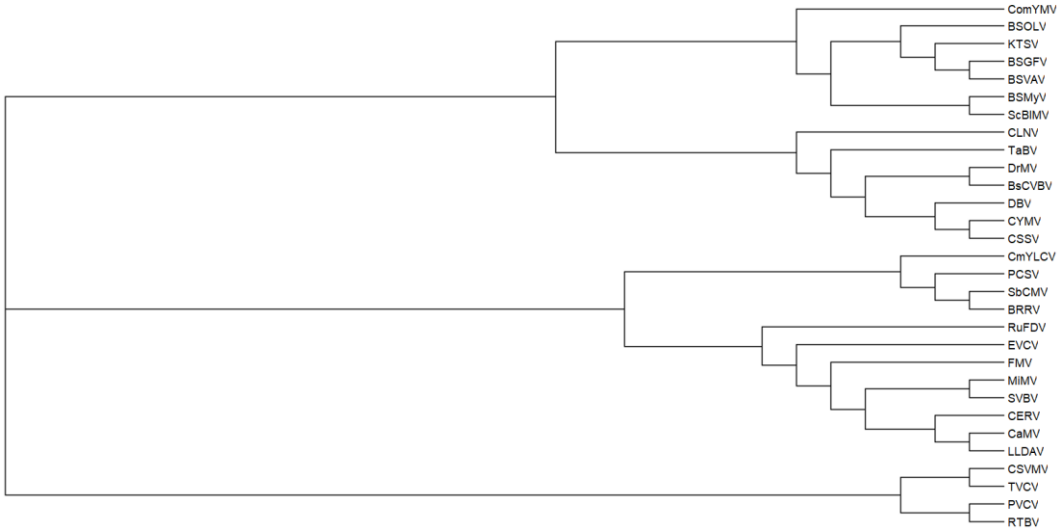



**Figura 4.** Representació en forma de fenograma de la filogènia de *AP_caulimovirus.*

Font: material de treball.

### 1.1.3.3 Cladograma

Un **cladograma** és un arbre jeràrquic ramificat que mostra únicament les relacions entre els diferents clades a través dels patrons de ramificació evolutius. Hi ha dos subtipus principals: el **rectangular** (utilitza branques formant clades rectangulars; *Figura 5*) i l'**inclinat** (utilitza branques formant clades en forma de V; *Figura 6*). Cal remarcar que, a diferència dels filogrames, els cladogrames no estan escalats; és a dir, les longituds de les branques no reflecteixen la càrrega de divergència evolutiva entre taxons o OTUs.

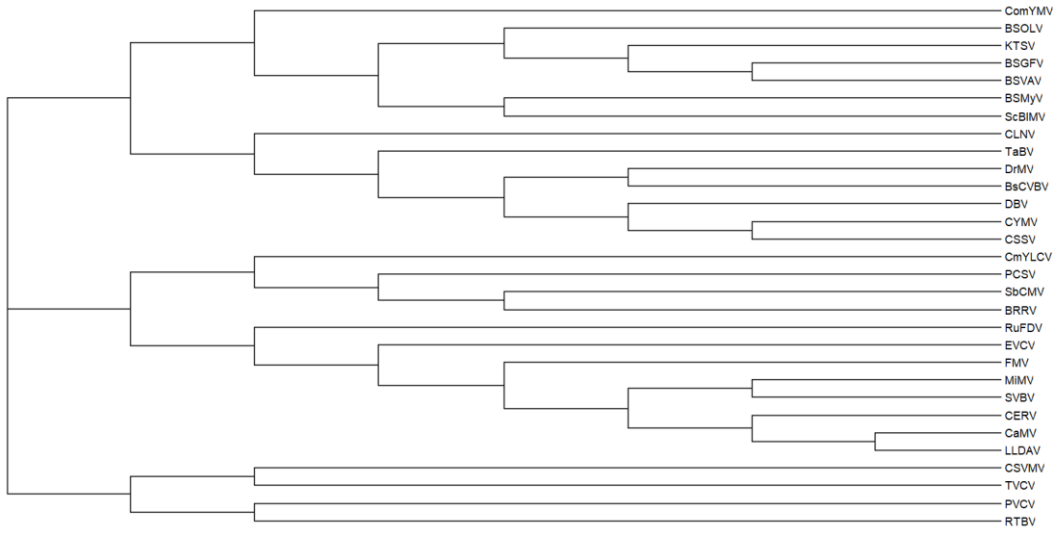



**Figura 5.** Representació en forma de cladograma rectangular de la filogènia de *AP_caulimovirus*.

Font: material de treball.

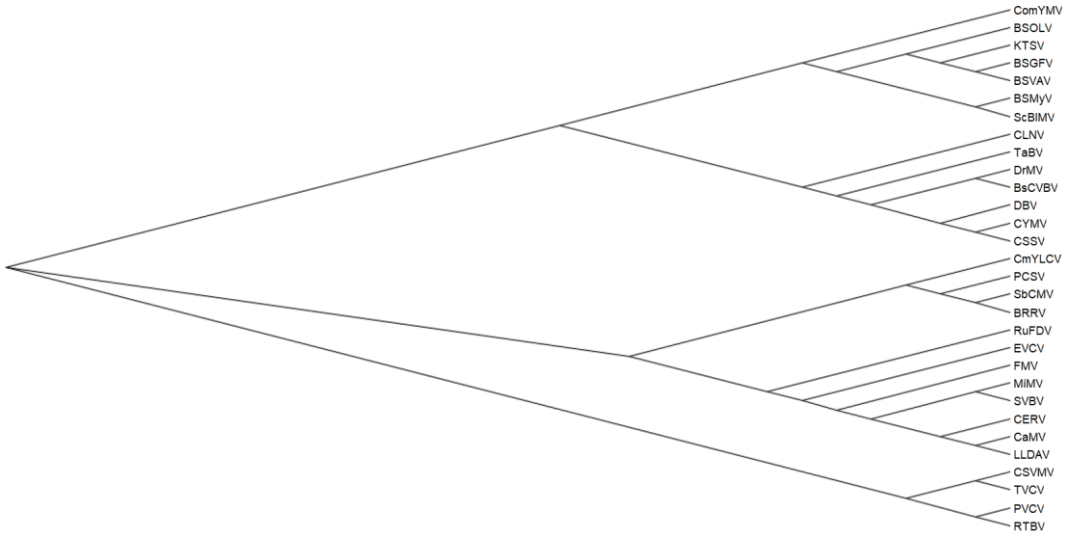



**Figura 6.** Representació en forma de cladograma inclinat de la filogènia de *AP_caulimovirus.*

Font: material de treball.

### 1.1.3.4 Filograma

Un **filograma** és un tipus d'arbre que representa les relacions evolutives entre organismes tenint en compte tant els patrons de ramificació com la quantitat de càrrega de divergència evolutiva. Aquesta representació sí està escalada, ja que les longituds de les branques són proporcionals a la quantitat de divergència evolutiva. Podem veure'n un exemple a la *Figura 7.*

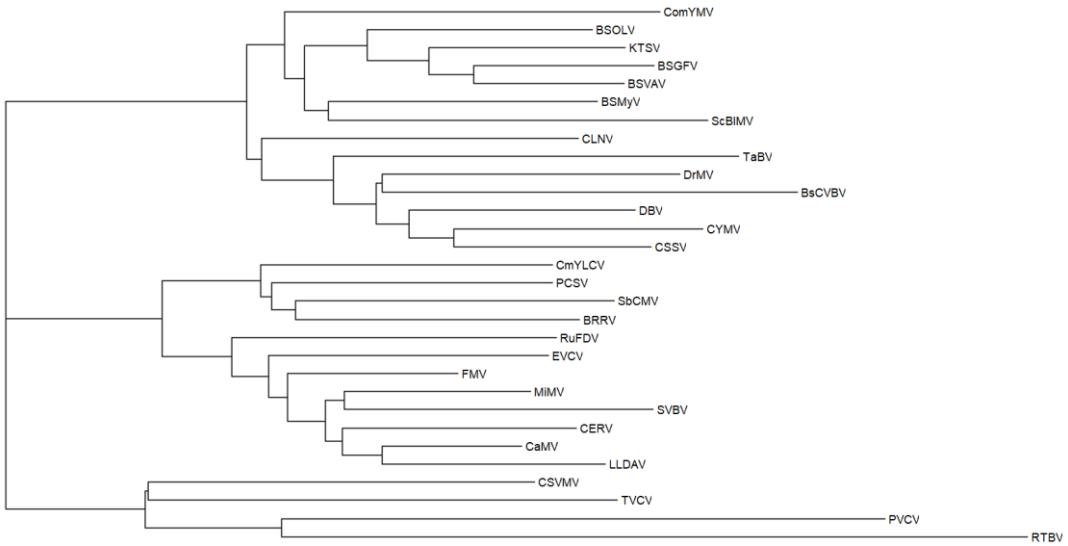



**Figura 7.** Representació en forma de filograma de la filogènia de *AP_caulimovirus.*

Font: material de treball.

### 1.1.3.5 Polar

Els arbres **polars** representen els taxons o OTUs en coordenades polars, amb les distàncies i direccions mesurades des d'un punt central. Aquest punt central es correspon amb l'arrel, i els diferents nodes s'agrupen formant anells concèntrics al voltant d'aquest punt. S'obtenen diferents capes de cercles en base als diversos nivells temporals de canvis evolutius. Hi ha dos subtipus principals: **amb distàncies** (a escala; la longitud de les branques mostra la càrrega de divergència evolutiva) i **sense distàncies** (no estan escalats; totes les fulles estan a la mateixa distància respecte del punt central). La *Figura 8* i la *Figura 9* són exemples d'aquests dos subtipus, respectivament.

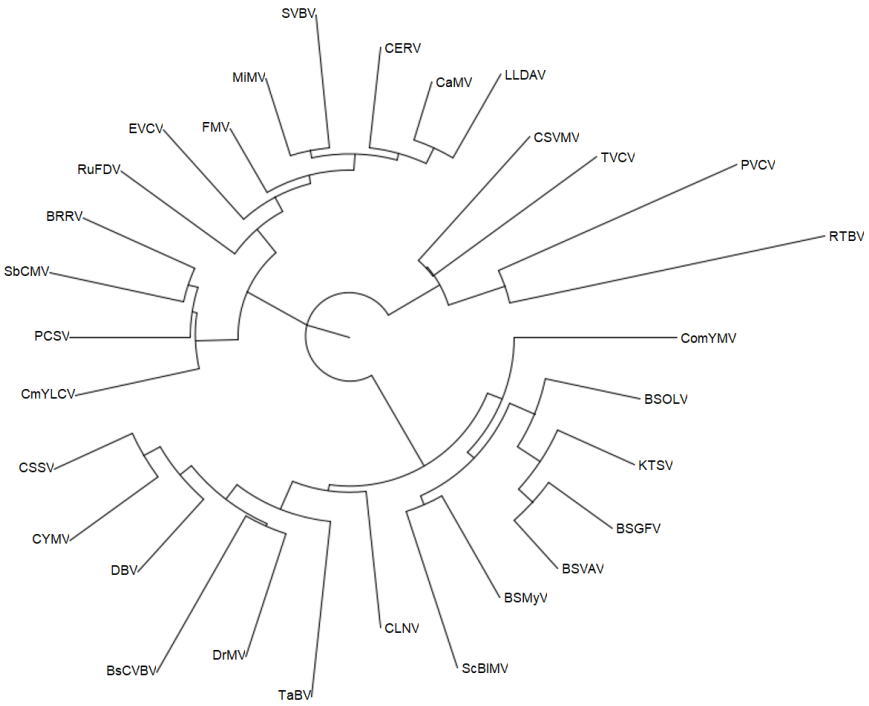



**Figura 8.** Representació polar amb distàncies de la filogènia de *AP_caulimovirus.*

Font: material de treball.

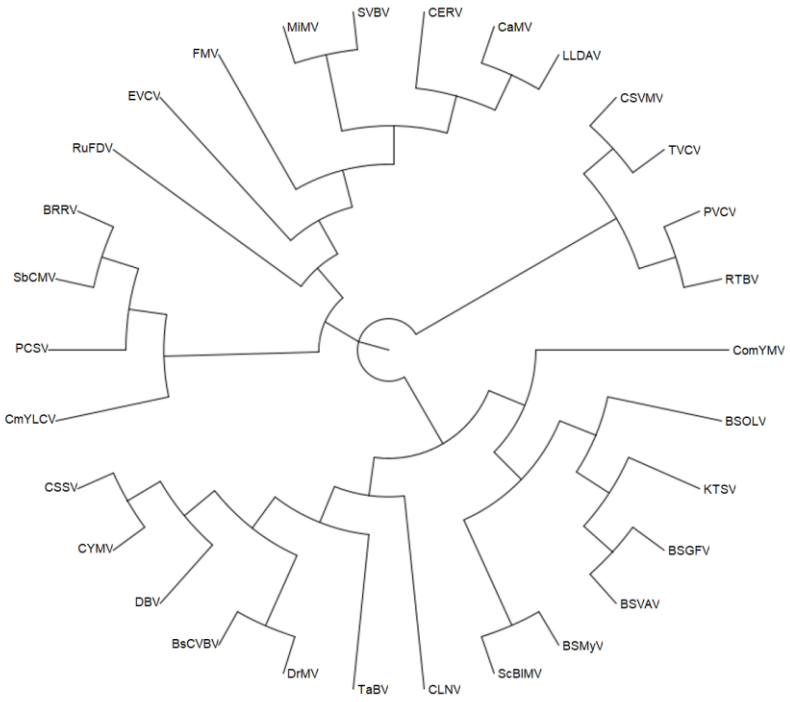



**Figura 9.** Representació polar sense distàncies de la filogènia de *AP_caulimovirus.*

Font: material de treball.

## 1.2 Phylograph

L'objecte d'estudi d'aquest treball és un *software* propietat intel·lectual de l'empresa Biotech Vana SL anomenat *Phylograph* [8]. Aquest programa està enfocat al processament i tractament d'arbres filogenètics.

Phylograph és un editor d'arbres multifuncional especialment indicat per a arbres grans. L'aplicació és capaç de llegir arbres de fins a 2000 fulles i construeix i edita dibuixos de gràfics en diferents dissenys. L'eina incorpora un ampli conjunt de funcions per expandir, comprimir, invertir i/o girar un arbre. A més a més, permet el tall de branques i la incorporació de decoracions com etiquetes, claudàtors, caixes i fletxes. L'eina també permet a l'usuari desar el dibuix de l'arbre com a projecte reeditable i ofereix l'opció d'exportables en diversos formats d'imatge, inclòs un format HTML adequat per a bases de dades.

Phylograph és una aplicació Java. Això significa que aquest instrument és capaç d'executar-se en ordinadors personals (PCs) i en estacions de treball com a programa autònom. El model MVC[3] és usat com a patró de programació amb l'objectiu de mantenir la independència i la visualització de les dades. Amb aquest model l'aplicació queda dividida en tres capes: *Model*, *View* i *Controller*. La capa *Model* conté la lògica del programa i les funcions executables. La capa *View* defineix la interfície gràfica de l'usuari i presenta tots els elements visuals (botons, llistes, camps de text, etc.) en un finestra principal. La capa *Controller* proporciona la connexió entre les altres dues capes.

## 1.3 L'Empresa: Biotechvana

Biotech Vana SL (o Biotechvana) és una empresa tecnològica basada principalment en la Bioinformàtica i la Biologia Computacional, si bé implementa altres models tipus e-business i serveis TIC. Va ser creada l'any 2006 pels investigadors genetistes de la Universitat de València Carlos Llorens i Andrés Moya, junt amb el programador Ricardo Futami i diversos col·laboradors i capital privat.

En base a la seva activitat principal l'entitat ofereix productes software, serveis d'anàlisi d'òmiques i consultoria de I+D+i a universitats, centres de recerca i hospitals, així com també a la indústria biotecnològica i farmacèutica. Els esforços de I+D+i de Biotechvana s'han dedicat a desenvolupar actius software propis (codi font i patents) i protocols de *know-how* tècnic per posicionar-se en la carrera bioinformàtica. Entre d'altres actius, han desenvolupat un paquet de software que treballa tant: a) enllaçat amb un servidor gestionant *pipelines* basades tant en software lliure com propietari o; b) com un *standalone software* d'escriptori. De la mateixa manera, es realitzen serveis computacionals; incloent anàlisis i creació d'infraestructures de tipus bases de dades.

En el que respecta a matèria organitzativa i econòmica, Biotech Vana SL és una microempresa (pyme) amb seu al Parc Científic de la Universitat de València que es composa actualment d'un equip format per 7 treballadors fixos, amb Carlos Llorens com a CEO. L'any 2023 va obtenir una facturació al voltant dels 190.000€ i ingressos de projectes d'I+D per valor de 250.000 euros, i presenta un capital social superior als 100.000€. Forma i ha format part de diversos programes de subvenció d'àmbit autonòmic, nacional i europeu, entre ells el Programa Operatiu Regional del Fons Europeu de Desenvolupament Regional (FEDER) de la Comunitat Valenciana o els fons NextGenerationEU. L'empresa també rep el suport del Gobierno de España a través del Ministerio de Ciencia e Innovación i la Secretaría de Estado de Digitalización e Inteligencia Artificial. A banda, ha rebut el guardó com a PYME Innovadora de "Innovative SME".

[3] MVC: Model View Controller

### *1.3.1 Posició personal de treball a l'empresa*

En relació a la feina feta amb Phylograph, he treballat en el projecte de manera individual com a desenvolupador de software. Inicialment vaig rebre especial ajuda d'un treballador enginyer informàtic que tenia assignat com a tutor; sobretot en el que respecta a la configuració de l'entorn de desenvolupament i la exportació i instal·lació del programari requerit al meu ordinador personal. Una vegada configurat l'entorn i donades les pautes inicials se'm va deixar treballar de manera autodidacta per tal de familiaritzar-me amb el programa i adquirir els coneixements específics sobre aquest. La planificació de les tasques i el disseny dels nous components del software, així com la seva implementació i la realització de les proves pertinents, han anat a càrrec meu. Tot això sempre seguint els diversos objectius marcats en tot moment pel cap de l'empresa, el qual junt amb el tutor em supervisaven la feina i em donaven suport quan ho necessitava.

## 1.4 Objectius

L'objectiu principal d'aquest Treball Fi de Grau és realitzar tasques d'actualització, manteniment i ampliació d'un programari antic propietat de Biotech Vana, dedicat a l'anàlisi de dades òmiques mitjançant arbres filogenètics.

Des del punt de vista de la enginyeria informàtica, hi ha hagut dos reptes. D'una banda, la realització de tasques de testeig, actualització i manteniment d'una eina software antiga, i d'altra banda, l'ampliació del programa amb noves funcionalitats que s'han hagut de dissenyar des de zero.

Podem dividir l'objectiu principal en diversos sub-objectius de la següent manera:

- Actualitzar codi antiquat a una versió més recent i garantir la seva funcionalitat.
- Explorar el funcionament general del programa amb una nova versió actualitzada i guanyar coneixements sobre aquest.
- Realitzar proves de test exhaustives sobre les diferents funcionalitats del programa per tal d'assegurar un comportament adequat i detectar errors.
- Corregir tots els errors detectats a les proves i optimitzar el codi quan sigui necessari.
- Ampliar el software amb noves funcionalitats sol·licitades (estudi de requisits, disseny i implementació).
- Generar documentació tècnica del treball realitzat per donar suport a futures revisions i ampliacions.

## 1.5 Estructura del document

La resta del document està organitzat en 4 capítols:

- Al Capítol 2 s'explica el funcionament general del programa amb l'objectiu de tenir més context i una major comprensió sobre l'aplicació. Es realitza un seguiment del flux bàsic de treball a través d'un exemple.
- Al Capítol 3 s'explica de manera detallada tots els procediments duts a terme sobre el Phylograph, des de l'estat inicial fins el final. Aquest capítol agrupa les fases d'actualització, manteniment o realització de proves, i ampliació.
- Al Capítol 4 s'analitzen els resultats obtinguts a les diferents tasques desenvolupades i es realitza una avaluació dels mateixos.
- Al Capítol 5 es sintetitza la feina feta amb unes breus conclusions junt amb una valoració personal del treball i una explicació de les perspectives de futur per a l'aplicació.

# 2 Funcionament General del Programa Phylograph

En aquest capítol s'explicarà de manera detallada el funcionament general de l'aplicació a través d'un exemple. Es realitzarà un seguiment complet de tot el procés, des de que entren les dades biològiques des d'un fitxer fins que surten en format de visualització d'arbre per poder ser observades i treballades per la persona investigadora.

A la *Figura 13* podem veure representat tot aquest flux de manera esquemàtica.

## 2.1 Pre-processament

Inicialment les dades filogenètiques es troben guardades en un arxiu de text de manera que puguin ser reconegudes per l'aplicació. Al món de la filogènia hi ha dos formats d'emmagatzemament molt estesos que són *Newick* i *Nexus*, els quals passarem a explicar més en detall en capítols posteriors. A la *Figura 10* podem veure una visió d'un arxiu d'aquest tipus amb l'exemple sobre el qual treballarem.

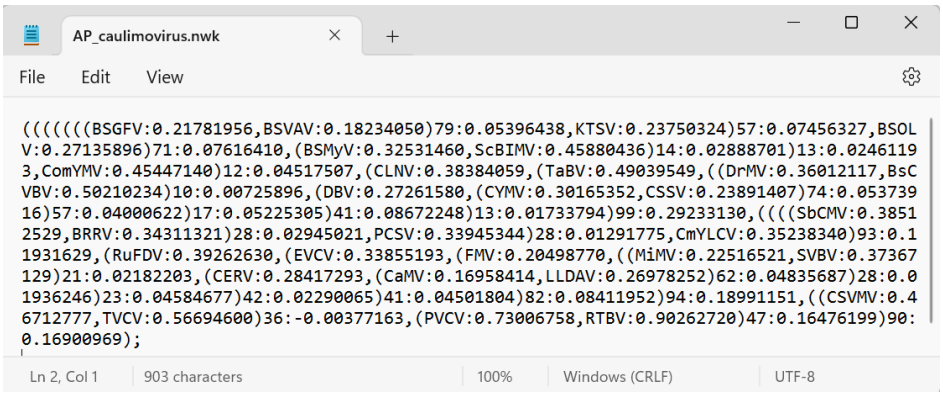



**Figura 10.** Visió d'un arxiu formatat amb informació filogenètica (*AP_caulimovirus*).

Una vegada activem la funció per obrir l'arxiu a través de l'aplicació (*Figura 11*), el primer que es fa internament és comprovar si el fitxer que volem obrir està en un format reconegut i, si és el cas, esbrinar a quin dels formats acceptats es correspon.

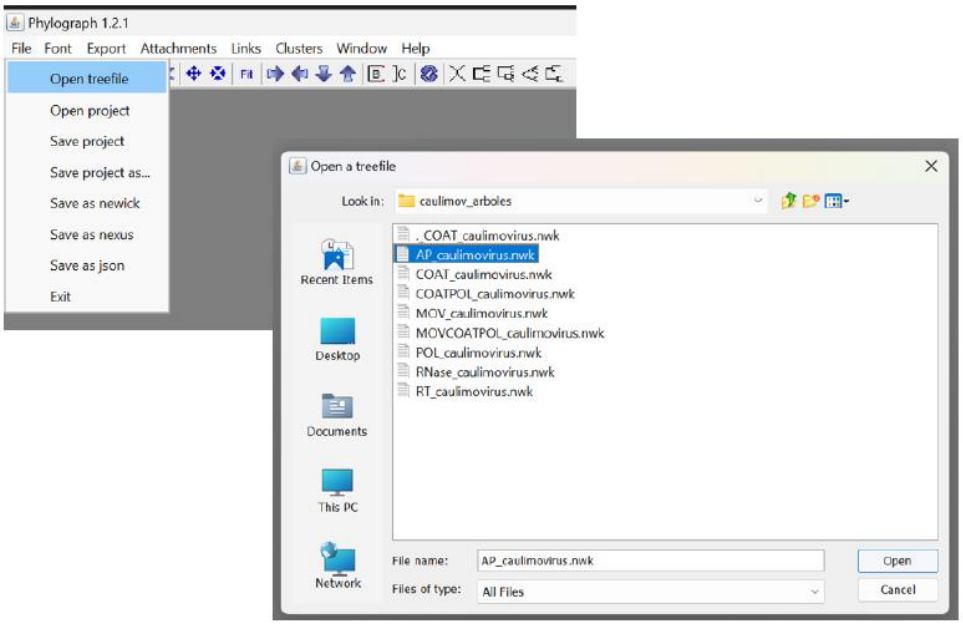



**Figura 11.** Captures dels menús del Phylograph per obrir de manera natural un arxiu d'arbre.

A una classe del projecte Java anomenada *Controller* és on trobarem la funcionalitat comentada al paràgraf anterior, junt amb altres relacionades amb l'obertura i el salvat de fitxers de diferents característiques. A més a més, aquesta classe és la responsable de dur a terme un control sobre el flux general del programa, ja que crida instàncies d'altres classes que són clau en tot el procés. Dins seu hi trobem la funció "openTreeFile", que és l'encarregada de realitzar les comprovacions inicials, i acabarà transformant qualsevol fitxer vàlid en el tipus *newick* (el més bàsic i simple de processar).

```
public void openTreeFile(String treeString, String fileName, String
treeAbsolutePath) {
            ParseTreeFile ptf = new ParseTreeFile(treeString);
            NewickToList ntl = new NewickToList(ptf.getNewick());

    ...

            TreeElement treeList = ntl.getTreeList();
```

Dalt es mostra un fragment de codi de la funció, que crea instàncies dels objectes de les classes "ParseTreeFile", encarregada de transformar qualsevol format acceptat a *newick*, i "NewickToList". Aquesta darrera classe agafarà el document newick i el convertirà en una llista d'elements d'arbre, els quals serviran com a dades d'entrada per a l'etapa de processament.

Un element d'arbre o **TreeElement** és un tipus d'objecte creat al programa per representar els diferents nodes i fulles que componen un arbre filogenètic. Està compost per 5 atributs:

- id → identificador de l'element de tipus int
- node → booleà que ens indica si l'element és un node (true) o una fulla (false)
- text → String que contindrà el nom de l'element a tractar (només si és fulla)
- x → double que indica la posició X de l'element
- y → double que indica la posició Y de l'element

Els atributs "x" i "y" tindran un valor de 0.0 fins que no es calculin les coordenades segons el tipus de representació al panell de visualització durant l'etapa de processament.

La classe anomenada *NewickToList* és l'encarregada de crear els diferents elements d'arbre i els agrupa finalment a una llista que contindrà tots aquests elements. Aquesta llista és una ArrayList d'objectes anomenada **treeList** què conté l'estructura amb tot els elements d'arbre que conformen la filogènia. Presenta diferents nivells de profunditat que es corresponen amb les branques i subbranques de l'arbre amb els nodes pare i els fills *(Figura 12)*. Aquesta informació és la que serà utilitzada posteriorment a l'etapa de processament per realitzar el càlcul de les coordenades i, en última instància, la representació gràfica segons el tipus.

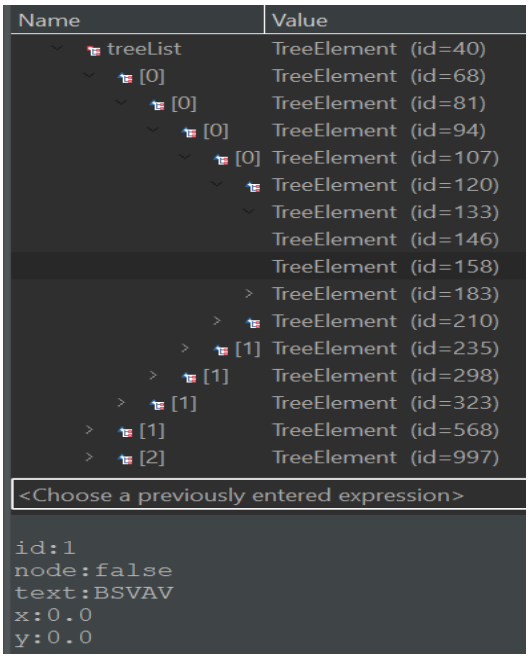



**Figura 12.** Captura on es mostra la manera en què el programa estructura els diferents elements d'una filogènia, junt amb els atributs d'un element d'arbre (fulla) seleccionat a mode d'exemple.

## 2.2 Processament

Després del pre-processament, la informació continguda inicialment a l'arxiu d'entrada ha sigut transformada de manera interna a una estructura jeràrquica d'arbre interpretable pel programa. Ara es passarà a realitzar un processament sobre els elements d'aquesta estructura per a poder donar finalment una representació gràfica verídica de la filogènia tractada.

La classe que presenta una paper dominant en aquesta etapa es diu *InternalF*; la més extensa i important del programa. És l'encarregada d'inicialitzar els objectes de la gran majoria de classes que componen el Phylograph, és a dir, pràcticament crea i gestiona tots els atributs relacionats amb els arbres i les diferents funcionalitats del programa. Qualsevol modificació que l'usuari faci utilitzant les eines proporcionades per la interfície gràfica, en algun moment de l'execució aquesta modificació passarà a ser gestionada per alguna funció o subclasse de *InternalF*.

En el cas exemplificat i en qualsevol altre, el constructor de *InternalF* crea totes les variables necessàries per al tractament de l'arbre:

```
/**
       * Internal frame for tree layout.
       *
       * @param controller       controller layer of MVC
       * @param view             view layer of MVC
       * @param treeL            treeList of TreeElements
       * @param fileName         name of opened file
       * @param treeAbsolutePath absolute path of opened file
       * @param treeType         type of tree: parsimonia, nj...
       * @param frameWidth       width
       * @param frameHeight      height
       * @param scaleValue       scale value
       */
```

```
      public InternalF(Controller controller, View view, TreeElement
treeL, String fileName, String treeAbsolutePath,
                  int treeType, int frameWidth, int frameHeight, double
scaleValue) {

            super(fileName, true, true, true, true);
            this.winTitle = fileName;
            this.controller = controller;
            this.view = view;
            this.treeAbsolutePath = treeAbsolutePath;
            this.treeList = treeL;
            this.treeType = treeType;
            this.scaleValue = scaleValue;

            setPreferredSize(new Dimension(frameWidth, frameHeight));
            setResizable(false);

            setVisible(true);
            pack();
            treeUtils = new TreeUtils();
            treegram = new TreeGram();
            treeList.setRoot(true);
            treeUtils.setTotalLength(treeList); // set accumulative
lengths for each element
            treeUtils.setDescendantCount(treeList); // set accumulative
descendant count for each element
            treeUtils.orderTree(treeList); // order node children, first
otus last nodes
            maxLength = treeUtils.getMaxLength(treeList);

     ...
     treeList = treegram.getCoords(treeList, treeView, maxLength,
negativeType);
     ...
```

Cal destacar la funció **getCoords**, la qual s'encarrega de realitzar el càlcul de les coordenades per a la posterior representació gràfica dels diferents nodes i fulles de l'arbre. Aquest càlcul variarà depenent del tipus d'arbre que es vulgui representar. Però, a l'obrir un arxiu, inicialment es tracta el cas per defecte que es correspon a una representació en forma de **filograma**.

Més concretament, a getCoords() de la classe *TreeGram* és on es realitza una comprovació per saber el tipus d'arbre que es vol representar. Posteriorment, es crida a una altra funció amb el mateix nom pertanyent a una classe corresponent al tipus de representació objectiu. En aquest cas es cridarà a la funció getCoords() de la classe *Phylogram* (si s'hagués establert per defecte una representació en forma de fenograma s'hagués cridat a getCoords() de la classe *Phenogram*, per exemple). Una vegada dins la classe corresponent, s'executaran funcions que calculen les posicions X i Y dels diferents nodes de l'arbre. La implementació d'aquestes funcions de càlcul de coordenades canviarà segons el tipus de representació que es vulgui dur a terme. Finalment, a través de *setters* es modificaran els atributs X, Y de cada TreeElement de la treeList, i aquestes dades ja estaran llestes per a ser passades a les rutines encarregades de la representació gràfica.

## 2.3 Representació

Continuant amb el codi vist a la secció anterior, el següent pas és el de la representació de l'arbre, que està implementada a la classe *PanelTree*:

```
    // get coordinates depending on treeView
    panelTree = new PanelTree(treeList, this, treeView);

    ...

double width = Toolkit.getDefaultToolkit().getScreenSize().getWidth();
double height = Toolkit.getDefaultToolkit().getScreenSize().getHeight();
double marginX = (width - fontDialog2.getWidth()) / 2;
double marginY = (height - fontDialog2.getHeight()) / 2;

    ...

getMaxXPosition(treeList, 0.0);
moveElements(treeList);
updatePanel();
```

La classe *PanelTree* rep la llista amb els elements de la filogènia (ara ja amb les coordenades correctament calculades), una instància de la classe *InternalF*, i el tipus de visualització que va a representar-se. Aquesta classe conté una funció anomenada paintComponent(), la qual detectarà el tipus d'arbre a representar i escollirà la rutina adient per dur a terme la representació. Així, junt amb les coordenades dels nodes i diferents atributs provinents de InternalF, aquesta funció s'encarregarà de crear i dibuixar els diferents elements gràfics sobre un panell que serà visible per l'usuari.

Finalment, s'haurà obtingut una representació gràfica en forma de filograma de l'arxiu que contenia la informació filogenètica inicial (*Figura 7*). Ara l'usuari ja podrà passar a realitzar les valoracions i modificacions que cregui pertinents, fent-se servir de la multitud de funcionalitats ofertes pel programa.

## 2.4 Diagrama de flux general de l'aplicació

La *Figura 13* mostra el diagrama de flux de l'aplicació per al cas genèric (explicat fins ara amb un exemple concret), és a dir, quan l'usuari entra a l'aplicació i vol obrir un arxiu d'arbre qualsevol. Hi podem veure les classes explicades a les seccions anteriors. Les fletxes contínues indiquen la relació de seqüencialitat entre les diferents classes junt amb la informació que passen unes a altres a sobre. Les fletxes discontinues indiquen les relacions de control d'unes classes sobre altres mitjançant la creació d'instàncies (p.e. la classe *Controller*, al llarg del flux d'execució, crea instàncies de les classes *ParseTreeFile, NewickToList, TreeElement* i *InternalF*; i crida funcions de les mateixes). Dintre de cada quadre hi trobem la classe involucrada en cursiva i, si n'hi ha, a sota es troba la seva funció principal pertanyent al flux.

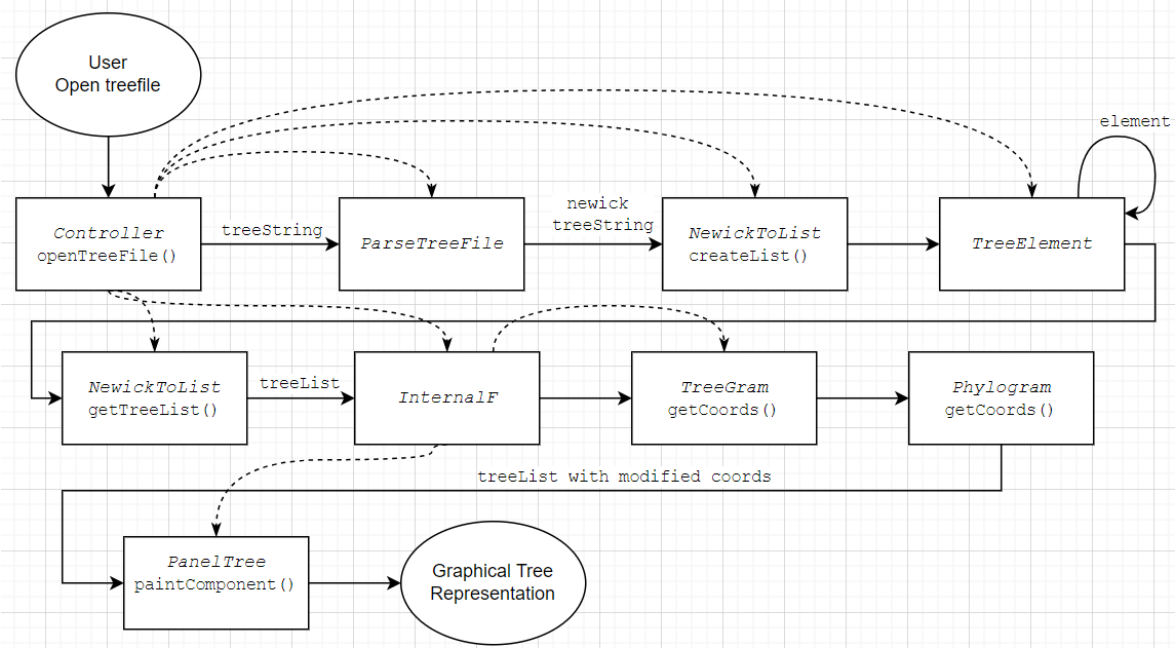



**Figura 13.** Diagrama de flux general per a un cas estàndard mostrant les relacions entre classes.

# 3 Desenvolupament del Projecte

En aquest capítol s'explicaran de manera detallada les diferents fases que s'han dut a terme per tal d'actualitzar, mantenir i ampliar el software estudiat a aquest treball, convertint-lo així en una aplicació perfectament funcional, al dia i utilitzable per a l'edició d'arbres filogenètics.

## 3.1 Entorn de Desenvolupament

*Phylograph* està implementat íntegrament en llenguatge Java. L'entorn d'execució necessari per a l'aplicació en qüestió ve proporcionat per diverses dependències Maven i la JRE[4] System Library per a Java Standard Edition 11.

S'empren llibreries i diversos *java packets* per cobrir les necessitats del programa. Cal destacar les llibreries seleccionades per a la creació de la interfície gràfica d'usuari (GUI); l'aspecte més rellevant donada la naturalesa de l'aplicació. A la llibreria Java AWT[5] trobem les classes *Graphics* i *Graphics2D* i el paquet *geom*, utilitzats per a representar gràficament els diferents elements visuals que componen els arbres filogenètics. Al seu torn, la llibreria *javax.swing* és clau per implementar el menú d'usuari.

L'eina utilitzada per dur a terme les diferents proves i modificacions sobre el codi ha sigut l'entorn de desenvolupament integrat Eclipse IDE 2022-06. La portabilitat del programa i la capacitat de funcionament com a entitat autònoma ha permès en tot moment l'execució sobre un ordinador personal amb sistema operatiu Windows 11. El programa també és completament operatiu amb sistemes Linux i/o amb altres versions de Windows.

### *3.1.1 Dependències Maven*

"Maven Dependencies" [9] fa referència a les biblioteques externes que són gestionades pel sistema de gestió de dependències Maven en un projecte Java. Maven és una eina de gestió de projectes que permet automatitzar la construcció, la gestió de dependències i el desplegament d'aplicacions Java. Quan afegim dependències, Maven descarrega automàticament les biblioteques necessàries des de repositoris remots i les inclou al nostre projecte Maven. Les dependències en qüestió són totes les biblioteques externes que el nostre projecte Java necessita per a funcionar correctament, i Maven s'encarrega de gestionar-les i incloure-les automàticament durant el procés de construcció. Això facilita la gestió de dependències i assegura que les versions de les biblioteques siguin coherents entre tots els desenvolupadors del projecte.

## 3.2 Fase d'Actualització

La primera tasca a realitzar en el procés de manteniment del software va ser l'actualització del codi de Phylograph de la versió en què es trobava inicialment a Java 11.

L'aplicació tenia una primera versió datada en l'any 2008 amb Java 1.6.2, la qual va ser pujada posteriorment a un repositori git personal de l'empresa l'any 2011. Durant els següents anys va sofrir lleugeres modificacions fins el 2015, quan es va realitzar el darrer commit. La versió Java no va ser actualitzada en cap moment.

[4] JRE: Java Runtime Environment

[5] AWT: Abstract Window Toolkit

A l'inici del nostre estudi es va agafar com a punt de partida aquest darrer commit. Una vegada configurat l'entorn de desenvolupament amb Eclipse i amb el projecte importat correctament, primer que tot es va realitzar la reinstal·lació de la JRE System Library i es va substituir l'entorn d'execució Java SE-1.6 per Java SE-11. Després es va instal·lar el sistema de gestió de dependències Maven i s'hi van afegir diverses dependències entre les quals destaquem les que incorporen les biblioteques *javafx-graphics* i *javafx-swing* per la rellevància que tindran posteriorment. La configuració de les dependències es va realitzar de manera natural amb la creació d'un fitxer *pom.xml*. Aquest nou fitxer actua com un element important de cara al futur, ja que descriu la configuració del projecte Maven, incloent al seu torn les dependències, la informació sobre el mateix i els plugins que s'han d'utilitzar durant el cicle de construcció del projecte.

### *3.2.1 Problemes i Errors*

Com a conseqüència d'aquest canvi de versió van aparèixer nombrosos problemes tals com *warnings* o *deprecated code* i errors. L'actualització duta a terme i el gran salt entre versions va comportar la necessitat de realitzar una sèrie de modificacions fruit de noves funcionalitats o nous paradigmes de programació publicats en versions més recents de Java respecte a la inicial. Ara passarem a exposar un parell dels canvis més rellevants per al nostre programa a mode exemplificador.

1. Implementació del concepte de "genèrics".

A partir de Java 5.0 es va introduir la característica de *generics*, que permet especificar el tipus d'objectes que una col·lecció pot contenir o manipular, augmentant així la seguretat dels tipus i la claredat del codi. No obstant això, a l'any 2007 i ja amb Java 6.0 aquesta nova pràctica encara no estava massa estesa i l'ús de genèrics era opcional. És per això que al codi es van trobar línies tals com les següents:

```
o  for (Iterator iter = parent.iterator(); iter.hasNext();) {
o  java.util.List textLabelList = internalF.getLabelList();
o  public class TreeElement extends ArrayList implements Cloneable {
o  private java.util.List listOtuLength = new ArrayList();
```

2. Modificacions associades al tractament d'esdeveniments.

Amb el pas dels anys i la publicació de noves versions, sovint es produeixen canvis en les anomenades "constants" pròpies de Java. Aquestes actualitzacions són especialment freqüents en el que respecta al tractament d'esdeveniments, amb l'objectiu de proporcionar una representació més precisa de les interaccions i entenedora pel programador. Al codi es van trobar diverses línies relacionades amb açò, per exemple:

```
o  private int RIGHT_SHIFT_MASK = MouseEvent.BUTTON1_MASK +
   MouseEvent.SHIFT_MASK;
o  if((me.getModifiers() & RIGHT_SHIFT_MASK) == RIGHT_SHIFT_MASK){
```

### *3.2.2 Solucions Implementades*

El codi obsolet va ser substituït per les noves alternatives recomanades i s'hi van realitzar les adaptacions necessàries per reajustar codi dependent d'aquest canvis. També s'hi va treballar per eliminar els warnings.

Respecte als exemples comentats a la subsecció anterior, passarem a mostrar les solucions proposades.

Abans de la implementació del concepte de *generics* no es podia especificar el tipus d'elements que contenia una col·lecció. Això vol dir que l'iterador podia recórrer qualsevol tipus d'objectes, i el tipus concret d'elements retornats per l'iterador s'havia de determinar manualment amb l'ajuda d'una conversió de tipus o *casting*. A l'actualitzar a Java 11, aquesta nova sintaxi introduïda pels genèrics fa el codi més segur i llegible, ja que el compilador pot detectar errors relacionats amb el tipus durant el temps de compilació i proporcionar missatges d'error més precisos en cas de problemes.

- `for (Iterator<?> iter = parent.iterator(); iter.hasNext();) {`

Amb **Iterator<?>**, s'especifica que l'iterador pot recórrer qualsevol tipus d'objectes sense necessitat de casting.

- `List<TextLabel> textLabelList = internalF.getLabelList();`

Ací s'especifica que la instància de List només pot contenir objectes de tipus TextLabel. Aquest canvi millora la seguretat del codi, ja que impedeix la inserció d'objectes de tipus incorrectes en la llista.

- `public class TreeElement extends ArrayList<Object> implements Cloneable {`

En aquesta línia s'utilitzen genèrics per especificar que TreeElement contindrà elements de tipus **Object**. Això significa que qualsevol tipus d'objecte pot ser afegit a la llista **TreeElement**, ja que Object és la classe arrel de totes les classes en Java. Aquest canvi garanteix que només s'afegeixin objectes de tipus Object (o subclasses d'aquest) a la llista TreeElement. Com en el cas anterior, també es millora la llegibilitat del codi, ja que és més clar quin tipus d'elements esperem que contingui la llista en qüestió.

- `private ArrayList<String> listOtuLength = new ArrayList<String>();`

A la sentència mostrada, en comptes de declarar la variable listOtuLength com una instància de java.util.List, utilitzem genèrics per especificar que la variable contindrà únicament elements de tipus String, millorant així la fiabilitat i la especificitat del codi.

Respecte als exemples exposats pels canvis en el tractament d'esdeveniments:

- `private int RIGHT_SHIFT_MASK = MouseEvent.BUTTON1_DOWN_MASK + MouseEvent.SHIFT_DOWN_MASK;`

En aquesta línia de codi, s'utilitzen les constants BUTTON1_DOWN_MASK i SHIFT_DOWN_MASK per representar el botó esquerre del ratolí premut i la tecla Shift premuda, respectivament. Aquest canvi reflecteix la substitució de les constants antigues (BUTTON1_MASK i SHIFT_MASK) per les noves en Java, que proporcionen una representació més precisa de les interaccions amb el ratolí i el teclat.

- `if ((me.getModifiersEx() & RIGHT_SHIFT_MASK) == RIGHT_SHIFT_MASK)`

Aquest canvi reflecteix l'ús del mètode getModifiersEx() en comptes de getModifiers() per obtenir una representació més precisa dels modificadors de teclat en versions més recents de Java. Utilitzar getModifiersEx() permet detectar modificadors de teclat estesos, com ara Ctrl, Alt i Meta, a més dels modificadors de teclat bàsics com Shift. Això millora la precisió del codi en les interaccions amb el teclat i el ratolí.

Totes aquestes modificacions proposades al canviar de versió són detectades i assenyalades pel compilador i/o l'eina de detecció d'errors integrada en l'entorn de desenvolupament. Algunes apareixen com a simples suggeriments marcats amb un warning i d'altres són directament classificades com a *deprecated code* o errors.

En el que respecta al propòsit de l'actualització del codi, veiem com els canvis introduïts ajuden a millorar aspectes clau com la seguretat i la llegibilitat del programa. Podríem concloure aquest apartat afirmant la importància de mantenir sempre el codi actualitzat a una versió recent, per tal d'evitar errors o confusions i així garantir un correcte funcionament de l'aplicació maximitzant al seu torn el rendiment i la eficiència de les operacions.

## 3.3 Fase de Testeig Exhaustiu

Una vegada actualitzada l'aplicació a Java 11, es va passar a realitzar una sèrie de proves test de manera intensiva sobre la totalitat del programa per tal d'assegurar una funcionalitat correcta i adequada del mateix.

### *3.3.1 Testeig General*

Primer, es va dur a terme un estudi del codi sobre les diferents classes i les seves relacions i dependències, per tal de comprendre el patró de comportament general del Phylograph. L'objectiu era poder conèixer l'estructura del programa orientat a objectes i de la seva relació amb els postulats filogenètics en el que respecta al processament i representació d'informació d'aquest tipus.

La combinació de coneixements biològics i informàtics, junt amb un maneig detallat del *debugger* i de les eines de suport de l'entorn de desenvolupament, em van permetre la comprensió del comportament bàsic del programa, explicat al Capítol 2 d'aquest treball. Es va realitzar un estudi a nivell de cas genèric, és a dir, es va agafar un arxiu amb informació filogenètica i es va descriure el flux principal del programa. Aquest comença en el moment en què l'usuari prem el botó per obrir l'arxiu i finalitza quan li és mostrada aquesta informació per pantalla en forma d'arbre filogenètic. Tots els arxius utilitzats per realitzar les proves van ser proporcionats per l'empresa i consistien en seqüències de diverses proteïnes víriques.

Una vegada estudiat el flux principal, es va passar a fer tests de la resta de fluxos associats a les diferents opcions disponibles per a l'usuari. Això s'explicarà a la subsecció que ve a continuació.

### *3.3.2 Testeig Funcional*

Arribats a aquest punt, es van començar a dur a terme proves test de manera concreta sobre totes i cadascuna de les funcionalitats del programa. L'objectiu era localitzar possibles errors durant l'execució que no haguessin sigut detectats prèviament, o trobar funcions amb un comportament que no fos l'esperat.

Per completar aquesta laboriosa tasca, es va decidir seguir un ordre estructurat i les proves es van realitzar en diferents tandes. Primer, les proves corresponents al menú principal (*Figura 14*), les quals al seu torn es van agrupar en les diferents seccions que el componen. Després, les proves corresponents a la barra d'eines (*Figura 14*).

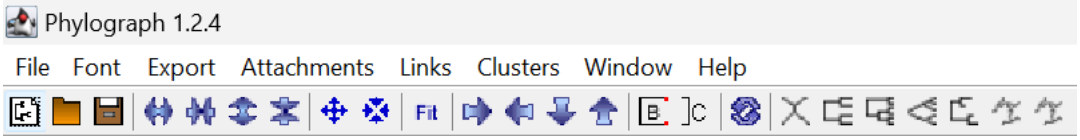



**Figura 14.** Captura de la interfície gràfica de Phylograph amb el menú principal a la part superior i la barra d'eines a la part inferior.

Per realitzar els tests es va utilitzar en tot moment el material de treball proporcionat per l'empresa. Els passos emprats eren:

1. Estudiar el flux de programa que es produïa a l'executar la funcionalitat.
2. Establir quin seria el resultat desitjat (amb ajuda de coneixements filogenètics especialitzats).
3. Comparar el resultat generat amb l'esperat.

En cas de funcionament correcte, es finalitza el test d'aquella funcionalitat.

En cas de la detecció d'un error o de l'obtenció d'un resultat que no es corresponia amb el predit:

4. Utilitzar l'eina *debugger* de manera intensiva per tal de trobar el punt o punts de codi malfuncional.
5. Reparar la secció de codi disruptiva, i tornar al pas 3.

### *3.3.3 Resultats de les proves*

| **Problema Detectat** | **Reparació Realitzada** | Èxit |
|---|---|---|
| Eliminació de clústers incorrecta si hi havia més d'un clúster -veure Annex A- present al fitxer *cluster_default* (s'eliminaven tots encara que volguessis eliminar només un) | Modificació de les condicions del bucle que itera sobre els diferents clústers presents al fitxer per tal d'eliminar només els clústers seleccionats | Sí |
| Color i font per defecte inexistents en clústers presents en arxius salvats com a projecte -veure Annex B-(.phylo) | Atributs de font i color per a clústers afegits a les classes *ProjectOpener* i *ProjectSaver* | Sí |
| Botó de "Exit" de l'aplicació no funcional (mancança de listener) | Addició del *listener* corresponent a la classe de control i creació d'una función exit() associada a *InternalF* | Sí |
| Links i clústers mostrats en pantalla no guardats després de desar un arxiu com a projecte | Addició de *Getters* i *Setters* per als clústers i links, implementació de comprovacions per avaluar si hi ha links i/o clústers mostrant-se en pantalla abans de guardar un projecte, creació de funcions per habilitar o no la visibilitat de clústers i links presents segons les comprovacions prèvies | Sí |
| Posició i mida dels clústers mostrats en pantalla fixa i independent dels canvis de dimensió o moviment en l'arbre | Implementació de la funció AdjustClusters() per tal de modificar la posició i la dimensió dels clústers d'acord amb els canvis realitzats per part de l'usuari en aquests mateixos paràmetres sobre l'arbre | Sí |

**Taula 1.** Principals problemes detectats i reparacions realitzades al Phylograph fruit de diverses proves test.

En general, tal i com s'esperava després d'haver dut a terme la fase d'actualització de manera exitosa, no s'hi van trobar gaire errors en el funcionament del programa. Cal destacar però, tot el relacionat amb la gestió dels clústers ja que la majoria de funcionalitats en les que estava present aquest element no responien amb un comportament desitjat.

Aquesta fase de testeig va servir per acabar de polir el codi, eliminant tots aquells resultats no esperats. Però, sobretot, em va permetre guanyar una enorme experiència en la comprensió del programa i en els diferents fluxos de treball que gestiona. Aquest factor serà determinant per poder implementar amb més agilitat i eficiència les noves funcionalitats sol·licitades; tractades en la secció vinent.

## 3.4 Fase d'Ampliació

Un cop actualitzat el programa, i després d'haver realitzat tasques de prova tant de funcionament general com de testeig específic de cadascuna de les funcionalitats disponibles, ara s'explicarà el procés d'incorporació de dues noves opcions: (1) suport de diversos formats de dades i (2) representació d'arbres polars.

Per cadascuna, s'introduirà si cal un poc de context per explicar la nova funcionalitat sol·licitada i a continuació s'explicarà el procés de disseny i implementació de la mateixa junt amb els reptes que va presentar.

### *3.4.1 Formats d'Emmagatzematge de Dades*

Perquè una aplicació bioinformàtica pugui realitzar un processament, construcció i representació adequades d'un arbre filogenètic, tota la informació necessària ha de ser emmagatzemada en un format que sigui interpretable per la aplicació en qüestió.

Hi ha diversos formats que segueixen diferents criteris i guarden aquesta informació filogenètica imprescindible de manera distinta. Com a puntualització important, cal remarcar que un arxiu d'arbre no conté informació de seqüència, sinó que simplement és el resultat de les distàncies genètiques (parentesc o semblança) entre les diferents seqüències. Per tant, no podem extraure dades de seqüenciació d'un arxiu filogenètic.

En aquest punt tractarem alguns dels formats d'emmagatzemament més representatius i estesos, i que utilitza el software objecte d'estudi en aquest treball. Inicialment el programa només permetia guardar els arxius en un únic format reconegut (newick), malgrat tenir la capacitat de llegir diversos tipus diferents.

**Newick** és el format estàndard per a desar la representació d'arbres. Un arbre sencer pot ser expressat com un *string*, llistant tots els nodes que el formen i les relacions (distància de branca) entre cadascun d'ells. El text de l'arbre acaba amb un punt i coma, i els nodes interns es representen amb un parell de parèntesis coincidents. Entre els parèntesis poden haver-hi al seu torn els nodes descendents d'aquest node. Els nodes germans estan separats per una coma i les fulles es representen amb els seus noms. La longitud d'una branca (des del node pare al node fill) es representa amb un nombre real després del node fill i va precedida de dos punts. Les dades singulars (p.e. valors *bootstrap*, que indiquen el grau de confiança de les dades calculades [10]) associades a nodes o branques internes es poden codificar com a etiquetes de nodes i representar-se mitjançant text/números simples abans dels dos punts. A la *Figura 15* podem veure un exemple de la notació comentada.

```
(BSOLV:27.135896,(KTSV:23.750324,(BSGFV:21.781956,BSVAV:18.23405)79.0:5.396438)57.0:7.456327)0.0:0.0;
```

**Figura 15.** Captura del contingut d'un fitxer d'arbre en format newick d'un clade extret de *AP_caulimovirus*. Font: material de treball.

#### 3.4.1.1 Anàlisi de Requisits

Es va sol·licitar la addició de noves funcions que permetessin guardar els arxius d'arbre també en format *nexus*, emprat a sovint per al desament d'informació filogenètica, i en format *json*, molt útil per a la visibilitat i comprensió de les dades per part del l'usuari.

Per a poder guardar efectivament les dades filogenètiques en els dos nous formats, s'han de complir els següents punts:

01. El programa ha de ser capaç d'identificar els formats *nexus* i *json*.
02. El programa ha de poder llegir arxius formatats de tipus *nexus* i/o *json*.
03. El programa ha de tenir la capacitat de transformar tots els formats d'entrada vàlids en el format de processament bàsic *newick*, en cas que sigui necessari.
04. El programa ha de tenir la capacitat d'interpretar una llista d'elements d'arbre i transformar-la amb la notació corresponent a un arxiu en format *nexus* o *json*,
    i. Aquest arxiu s'ha de poder emmagatzemar a un directori del computador de l'usuari de manera permanent.
    ii. L'arxiu desat i la seva posterior utilització no ha de comprometre cap dels punts anteriors.

D'acord amb els punts exposats i el flux seqüencial del codi, construïm el diagrama d'activitats per a l'obertura d'un fitxer (*Figura 16*) i un altre per al desament amb els nous formats (*Figura 17*). En roig apareixen ressaltades les parts del flux que han de ser modificades per poder adaptar-se a les noves necessitats.

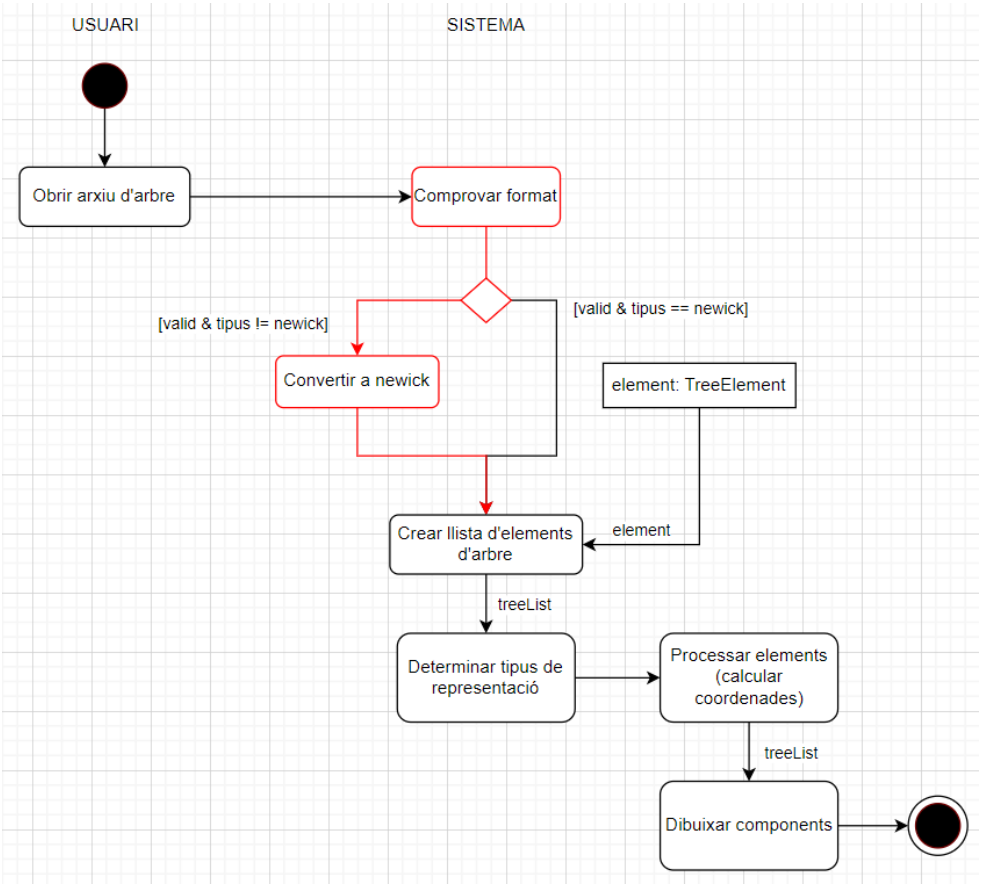



**Figura 16.** Diagrama d'activitats per a l'obertura d'un fitxer d'arbre.

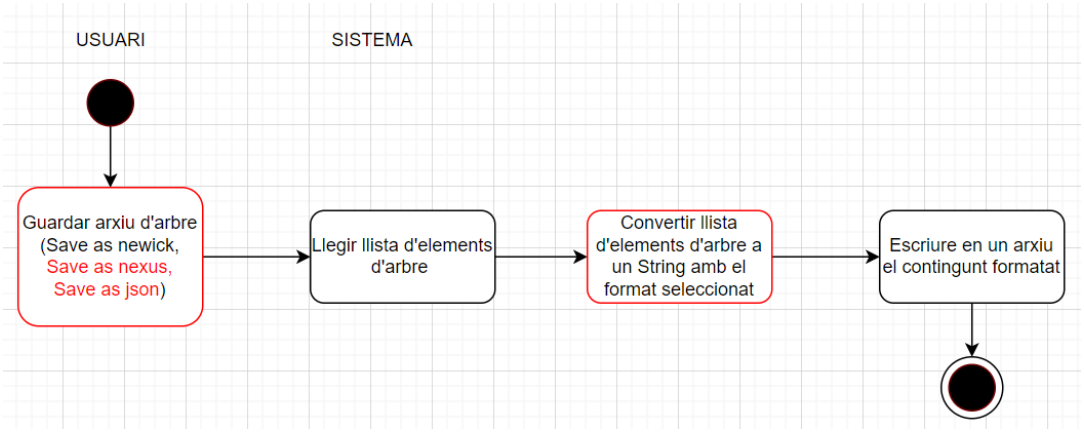



**Figura 17.** Diagrama d'activitats per al guardat d'un fitxer d'arbre.

### 3.4.1.2 Disseny i Implementació

Analitzant els requisits i els diagrames d'activitats, veiem com hem d'habilitar la intercanviabilitat entre formats. És a dir, no només hem de permetre desar arxius d'arbre en els nous formats *nexus* i *json*, sinó que també hem de fer possible la seva lectura perquè el programa sigui complet i coherent.

Per dur a terme això, primer que tot, haurem d'implementar algun tipus de lògica que sigui capaç d'identificar el format en què està desada la informació filogenètica a l'arxiu. Abans d'això però, cal explicar l'estructura de cadascun d'aquests dos nous formats.

**Nexus** és un format d'arxiu dissenyat per a contenir dades sistemàtiques que puguin ser utilitzades per programes informàtics. Els objectius d'aquest format són permetre la possibilitat d'una futura expansió, la capacitat d'incloure tipus d'informació diferents, garantir la independència respecte a un sistema operatiu en particular i un fàcil processament per part dels programes. Amb aquest propòsit, el format és modular, amb un arxiu consistent de blocs separats cadascun dels quals conté un tipus d'informació en particular. Els blocs poden contenir informació relativa a taxons, caràcters morfològics i moleculars, distàncies, codis genètics, conjunts, arbres, etc.[11]. A la *Figura 18* podem veure'n un exemple representant el cas més bàsic (només s'hi troba el bloc referent al contingut de l'arbre), què es correspondrà amb l'implementat al programa.

```
#NEXUS
begin trees;
tree tree1 =
(BSOLV:27.135896,(KTSV:23.750324,(BSGFV:21.781956,BSVAV:18.23405)79.0:5.396438)57.0:7.456327)0.0:0.0;
end;
```

**Figura 18.** Captura del contingut d'un fitxer d'arbre en format nexus d'un clade extret de *AP_caulimovirus*. Font: material de treball.

**JSON**[6] és emprat per representar dades estructurades seguint la sintaxi dels *JavaScript objects*. És sovint utilitzat per transmetre dades en aplicacions web (p.e. per enviar dades des d'un servidor a un client de manera que puguin ser mostrades a un web, o viceversa) [12]. En el cas de l'ús d'aquest tipus de format en arbres filogenètics, ens permet construir la jerarquia d'una manera simple i visual, utilitzant els mateixos tipus de dades que es poden utilitzar en un objecte JavaScript estàndard. Es genera una estructura formada per un array de nodes on cada node té un identificador, una etiqueta i un conjunt de nodes fills (subarrays).

6 JSON: JavaScript Object Notation

L'identificador es correspon al nom del node i la etiqueta fa referència a la longitud de la branca. Els corresponents nodes fills apareixen identificats sota el mot "children". Cada node es separa amb una coma i es troba tancat entre dos claus. Cada clade es troba entre claudàtors i, a continuació del claudàtor que simbolitza el tancament d'un clade, s'hi afegeix una coma seguida del valor *bootstrap* i la longitud del node "pare" corresponent. A la *Figura 19* podem veure un exemple de l'estructura de text comentada, que es correspondrà amb la implementada al Phylograph.


```
{
"root": {
         "children": [
                {
                 "name": BSOLV,
                 "length": 27.135896
                },
                {
                 "children": [
                        {
                         "name": KTSV,
                         "length": 23.750324
                        },
                        {
                         "children": [
                                {
                                 "name": BSGFV,
                                 "length": 21.781956
                                },
                                {
                                 "name": BSVAV,
                                 "length": 18.23405
                                }
                         ],
                         "bootstrap":79.0,
                         "length":5.396438
                        }
                 ],
                 "bootstrap":57.0,
                 "length":7.456327
                }
         ],
         "bootstrap":0.0,
         "length":0.0
        }
}
```


**Figura 19.** Captura del contingut d'un fitxer d'arbre en format json d'un clade extret de *AP_caulimovirus*. Font: material de treball.

#### *3.4.1.2.1 Interoperabilitat de Formats*

Una vegada explicat quina notació presenta cada format per guardar la informació, ara ja podem definir unes característiques que permetin al programa identificar-ne el tipus al començar la lectura d'un arxiu. Per fer això crearem una nova classe "TreeFormat", on realitzarem les comprovacions:

```
public class TreeFormat {
 private String type = "newick";

 TreeFormat(String file) {
       // check if tree file has NEXUS format
       if (file.indexOf("#NEXUS") != -1) {
             type = "nexus";
       }
       // check if tree file has JSON format
       if (file.indexOf("{") != -1) {
             type = "json";
       }
 }
```

D'aquesta manera, el format per defecte continuarà sent *newick*, mentre que quan es detecti un començament característic d'un fitxer amb format *nexus* o *json* s'hi canviarà al corresponent tipus.

La classe anomenada "ParseTreeFile" serà l'encarregada d'obtenir aquesta informació i transmetre-la al flux de treball principal:

```
public ParseTreeFile(File file) {
          String input = fts.getString(file.getAbsoluteFile()); // gets
string of file
          TreeFormat ttype = new TreeFormat(input); // determines type
          String tipo = ttype.getType();

          if (tipo.equalsIgnoreCase("nexus")) {
                NexusToNewick ntn = new NexusToNewick(input);
                output = ntn.getNewick();
          } else if (tipo.equalsIgnoreCase("json")) {
                JsonToNewick jtn = new JsonToNewick(input);
                output = jtn.getNewick();
          } else // newick
          {
                output = input;
          }
 }
```

Veiem com haurem hagut de crear les classes "NexusToNewick" i "JsonToNewick" per tal que el programa sigui capaç de transformar el contingut dels fitxers que tinguin algun d'aquests dos formats a format newick. Això és un pas indispensable, ja que la posterior etapa de processament de la informació sempre es fa partint de la lectura de dades del tipus estàndard (newick).

En el cas d'arxius amb format *nexus* només caldrà realitzar una implementació que extregui la part corresponent al text on es troba la informació de l'arbre (a la *Figura 18*, quarta línia començant per dalt). En cas que més avant decidírem implementar nous tipus de blocs, hauríem d'eliminar tots aquests i quedar-nos-en únicament amb la part indicada del bloc "tree". Pels arxius amb format *json* la implementació adient consistirà en la identificació dels diferents tipus d'elements al document (root, children, name, length...) i substituir-los per la notació que es fa servir en el format newick per identificar els mateixos elements, eliminant al seu torn caràcters redundants que no s'empren a la nova notació.

*3.4.1.2.2 Noves Opcions d'Emmagatzemament*

Per implementar les dues noves opcions d'emmagatzemament sol·licitades, primer que tot haurem d'habilitar al menú de l'aplicació aquestes noves funcionalitats. Afegirem al llistat de la secció "File" del menú les opcions "Save as nexus" i "Save as json", i habilitarem els listeners corresponents per quan l'usuari les seleccioni:

```
final class SaveNexusListener implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
                saveNexus();
        }
 }
 final class SaveJsonListener implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
                saveJson();
        }
 }
```

Els listeners "SaveNexusListener" i "SaveJsonListener" cridaran a noves funcions anomenades *saveNexus( )* i *saveJson( )*, les quals crearan instàncies de dues classes noves anomenades "NexusSaver" i "JsonSaver". Aquestes classes al seu torn delegaran la part central de la lògica requerida a les classes "ListToNexus" i "ListToJson", què haurem hagut d'implementar per tal de convertir el llistat d'elements d'arbre ja processat a un arxiu de text amb la notació específica del nou format corresponent.

```
public class JsonSaver {
      /**
       * @param internalF parent internal frame
       * @param treeList  target node
       */
      public void saveJson(InternalF internalF) {
            File output = null;
            boolean hasBootstrap = false;
            int treeType = 1;
            ListToJson ltn = new ListToJson();
     ...
// create Json
String Json = ltn.getJsonString(internalF.getTreeList(), hasBootstrap);

// write file
writeJson(output, Json);
}
```

Dalt es mostra un fragment de codi corresponent a la classe "JsonSaver" i a la seva funció associada *saveJson( )*. Les dues noves implementacions s'han realitzat respectant el flux de treball i el model d'implementació ja existent per al desament d'arxius en format newick. Això ens permet garantir un correcta connexió entre les classes *Controller* i *InternalF*, claus en la gestió dels esdeveniments activats per l'usuari i en el posterior tractament del flux d'informació.

Les classes "ListToNexus" i "ListToJson" contindran funcions per a recórrer els diferents nodes de l'arbre, identificar-ne el seu paper (arrel, node intern, fulla, etc.) i les seues característiques (longitud, nivell, etc), i transformar tot això a notació específica. Una vegada convertida la llista d'elements d'arbre en un String formatat, aquestes dades passaran a ser escrites a un fitxer de manera permanent, gràcies a les funcions *writeJson( )* i *writeNexus( )* que fan servir operacions de tipus "FileWriter" i "BufferedWriter".

### *3.4.2 Arbre Polar (amb Distàncies i Sense Distàncies)*

Un arbre polar és un tipus de representació filogenètica emprada per representar les relacions evolutives entre diferents espècies, individus o gens. La diferència respecte a altres tipus comuns de representació (filograma, cladograma, etc) radica en la disposició radial dels diferents nodes, al voltant d'un punt central conegut com a arrel. Això aporta un altre tipus de visió a la persona investigadora, que li pot resultar útil en determinades situacions a l'hora d'interpretar els resultats.

#### 3.4.2.1 Anàlisi de Requisits

Com que les dues funcionalitats sol·licitades estan altament relacionades, es realitzarà un anàlisi conjunt de les mateixes. A nivell de requisits, s'han de garantir les següents condicions:

01. El programa ha de ser capaç d'identificar quan l'usuari vol canviar la representació d'arbre actual a tipus "polar amb distàncies" o "polar sense distàncies".
02. El programa ha de poder modificar les variables que assignen internament el tipus d'arbre per assignar-li el corresponent a la representació polar.
03. El programa ha d'escollir el flux de treball corresponent al tipus seleccionat.
04. El programa ha de ser capaç de realitzar els càlculs de les coordenades polars de manera correcta i respectant les distàncies en cas que sigui necessari.
    i. S'ha de tenir la capacitat de passar de coordenades rectangulars/cartesianes a coordenades polars, i a l'inrevés.
    ii. Si s'han calculat les coordenades polars prèviament sense tenir en compte les distàncies, aquestes s'han de poder afegir a totes les posicions.
05. El programa ha de tenir la capacitat de dibuixar els elements geomètrics requerits, així com de calcular la seva mida, orientació i posició, per tal de mostrar a l'usuari una representació satisfactòria d'un arbre filogenètic de tipus polar (amb distàncies o sense).

Seguint el protocol definit, ara passarem a elaborar el diagrama d'activitats corresponent (*Figura 20*) per tal que es garanteixin els requisits establerts.

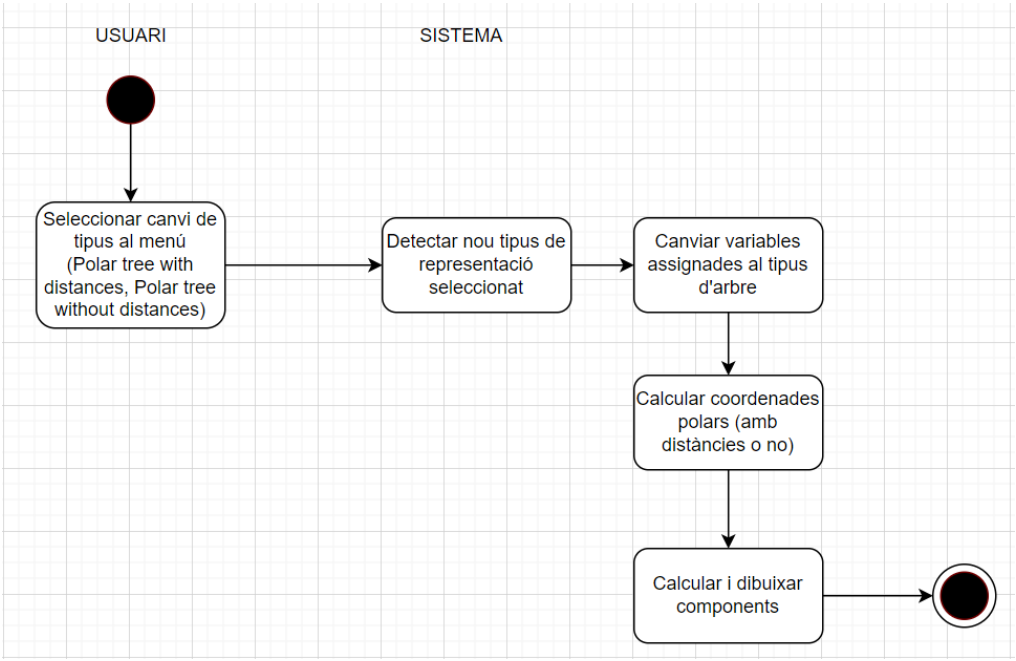



**Figura 20.** Diagrama d'activitats per al canvi de representació d'arbre a algun dels dos tipus polars.

### 3.4.2.2 Disseny i Implementació

Una vegada analitzats els requisits i el diagrama d'activitats, veiem com haurem d'implementar una sèrie de noves classes per als nous arbres "polar amb distàncies" i "polar sense distàncies". A més a més, haurem d'afegir noves funcions a aquelles classes ja existents que s'encarreguin del processament de la informació necessària per canviar el tipus de representació quan l'usuari en selecciona un de nou.

Primer que tot, a la classe *View*, que és l'encarregada d'acollir els diferents elements que conformen la interfície gràfica d'usuari (botons, menús, barra d'eines), caldrà afegir dos botons a la barra d'eines per a les noves opcions (a la *Figura 14*, els dos botons de més a la dreta). A aquests dos nous botons, que anomenarem "PolarDist" per a la representació amb distàncies i "Polar" per a la representació sense distàncies, els associarem un *action command* amb aquests noms.

Quan l'usuari premi alguna d'aquestes opcions, s'activarà un *listener* que cridarà a una funció "changeTreeType" i li passarà com a paràmetre el tipus associat al nom del botó. Aquesta funció de la classe *Controller* li passa l'execució a la classe *InternalF* i aquesta al seu torn cridarà a una altra funció "changeTreeType" de la classe *TreeTypeChanger*. És en aquest punt on, segons el tipus, es canviaran una sèrie de característiques i variables de l'arbre. Acte seguit, es cridarà a la funció **getCoords()** per realitzar el càlcul de les coordenades segons el tipus assignat i finalment es realitzaran una sèrie d'operacions per ajustar la visió d'arbre al panell de visualització amb què treballarà l'usuari.

La implementació detallada dels càlculs de coordenades segons si es tracta d'una representació filogenètica de tipus "polar amb distàncies" o "polar sense distàncies" s'explicaran a continuació de manera separada. A una classe anomenada *TreeGram* és on trobarem implementada la funció **getCoords( )**, la qual presenta una sèrie de condicionals (un per cada tipus de representació d'arbre implementat). Segons el tipus, es cridarà a una funció local amb el mateix nom d'una classe corresponent a la representació seleccionada.

```
public TreeElement getCoords(TreeElement treeList, String type, double
maxLength, int negativeLength) {
            if (type.equalsIgnoreCase("phenogram")) {
                  return phenogram.getCoords(treeList, maxLength);
            } else if (type.equalsIgnoreCase("cladogram")) {
                  return cladogram.getCoords(treeList, maxLength);
            } else if (type.equalsIgnoreCase("phylogram")) {
                  return phylogram.getCoords(treeList, maxLength,
negativeLength);
            } else if (type.equalsIgnoreCase("slanted")) {
                  return slanted.getCoords(treeList, maxLength);
            } else if (type.equalsIgnoreCase("unrooted")) {
                  return unrooted.getCoords(treeList);
            } else if (type.equalsIgnoreCase("polarDist")) {
                  return polarDist.getCoords(treeList);
            } else if (type.equalsIgnoreCase("polar")) {
                  return polar.getCoords(treeList);
            } else {
                  return null;
            }
      }
```

En el codi mostrat a dalt, veiem seleccionat en roig les noves comprovacions que haurem d'afegir per habilitar els dos tipus nous de representació.

#### *3.4.2.2.1 Càlcul de les Coordenades Polars (Amb Distàncies)*

Caldrà crear una classe anomenada *PolarDist*, i dintre d'aquesta serà imprescindible implementar la funció *getCoords( )*, la qual rebrà per paràmetre la llista d'elements d'arbre i serà l'encarregada de calcular les seves coordenades.

A la implementació proposada, primer que tot cridarem a una instància de la classe *TreeUtils*. Aquesta classe ja implementada incorpora una sèrie de funcionalitats per processar informació de l'arbre que ens seran d'utilitat.

A continuació, crearem un element fonamental per a tota representació polar, això és l'arrel. L'arrel és l'element central de l'arbre al voltant de la qual es distribuiran els nodes i fulles. Per tal d'obtenir l'arrel de la filogènia, simplement haurem d'agafar el primer element de la llista i cridar el seu pare amb la funció *getFather( )*, implementada a la classe *TreeElement*. Ara ens caldrà determinar la proporció de la mida de l'arrel respecte la totalitat de la mida de l'arbre. Realment aquest paràmetre no és rellevant en el que respecta a l'anàlisi filogenètic, ja que la seua variació simplement permet a l'usuari fer l'anell central de l'arbre més gran o més petit, comprimint o expandint al seu torn la resta de branques i respectant sempre les proporcions entre aquestes. Per tant, ens hem decantat per establir una longitud estàndard amb un valor de 1. Per mantenir aquesta constant a 1 haurem d'avaluar la mida de l'arbre. Això ho podrem fer a través d'una funció que s'encarregarà de trobar la longitud màxima. A aquesta funció l'anomenarem "getMaxXPosition", la qual anirà recorrent la llista d'elements d'arbre i acumularà la longitud dels nodes fills a la posició X. Si la nova posició X calculada és superior a una prèvia, s'actualitzarà aquest valor màxim. Una vegada recorreguda la llista tindrem emmagatzemada a una variable la posició X màxima de l'arbre, i utilitzarem aquesta informació per mantenir la proporció de la longitud de l'arrel.

Després d'això, ens caldrà calcular un altre paràmetre molt important per a aquest tipus de representació: l'increment de la posició Y. Aquesta variable serà clau per definir una separació fixa entre els diferents nivells de nodes que componen l'arbre (nivell 0: arrel; nivell 1: fills del node arrel; nivell 2: fills dels fills del node arrel...). Això ho farem, primer que tot, definint el nombre de descendents que té l'arbre fent una crida a la funció *setDescendantCount( )* de la classe *TreeUtils*, comentada amb anterioritat, passant-li l'arrel com a paràmetre. Acte seguit, cridarem a la funció *getDescendantCount( )* de *TreeElement* per agafar aquest valor calculat, i l'emmagatzemarem a una variable. Perquè l'increment de Y sigui proporcional als diferents nivells, dividirem 1 entre el nombre de descendents obtinguts.

Una vegada tenim l'arrel, la seua longitud (que serà el punt de partida de les coordenades X) i l'increment de la posició Y calculades, ara ja podrem passar a explicar la lògica de la rutina que s'encarregarà de l'establiment de les noves coordenades als diferents elements de la filogènia per tal d'obtenir una representació polar. Aquesta funció, que anomenarem **assignPolarCoordinates( )**, serà de tipus recursiu i rebrà com a paràmetres l'element de la llista actual a tractar i la posició X. Explicarem el seu funcionament amb el següent codi en alt nivell:

```
Funcio assignPolarCoordinates(node_tractat, posicioX)
      Si (node_tractat.esNodeIntern()) llavors
            per (tots_els_fills) fer
              longitudFill = fill.obtenir_longitud()
              assignPolarCoordinates(fill, posicioX + longitudFill)
              posicioY = posicioX + fill.agafarPosicioY
            fiper
            posicioY = posicioY / node_tractat.agafarMida()
            /*actualitzar noves posicions del node tractat*/
            per (tots_els_fills) fer
                  polarConversion(fill)
            fiper
      sino ...
```

Primer que tot, es comprovarà el tipus d'element tractat. Si és un node intern (no és una fulla) s'agafarà la longitud dels seus fills i es cridarà de nou a la funció un cop per cada fill passant-li la longitud acumulada del pare més la seva pròpia. D'aquesta manera, s'obtindrà la coordenada X de cada fill tractat i, acte seguit, es calcularà la Y de la mateixa forma. A continuació, es realitzarà el càlcul per obtenir la posició Y del node pare processat (dividint el seu valor entre la mida del mateix) i s'actualitzaran les seves coordenades.

Després d'això es recorreran una vegada més els fills de cada node intern i es cridarà a la funció **polarConversion( )**, passant-li com a paràmetre el fill a processar. Aquesta funció és l'encarregada de convertir les noves coordenades calculades de cartesianes a polars. Per tant, podríem dir que fins ara només havíem calculat les coordenades basades en les longituds dels nodes i els diferents nivell de l'arbre (semblant al càlcul per a una representació de filograma). La funció de conversió de coordenades queda implementada de la següent manera:

```
/**
  * Convert rectangular coordinates into polar coordinates
  * @param parent: a tree element from the tree list
  */
 private void polarConversion(TreeElement parent) {
       double x = parent.getX();
       double y = parent.getY();

       double angle = (y * 360);
       double r = Math.toRadians(angle);
       double xP = (x * Math.cos(r));
       double yP = (x * Math.sin(r));

       parent.setX(xP);
       parent.setY(yP);
 }
```

Per realitzar aquest càlcul imprescindible per a una correcta representació polar ens hem hagut de basar en les següents fórmules matemàtiques de conversió de coordenades cartesianes i polars [13]:

$$x = r\cos\theta$$

$$y = r\sin\theta$$

$$r^2 = x^2 + y^2$$

$$\tan\theta = \frac{y}{x}$$

Veiem com a la implementació duta a terme la variable "x" de cada node fa referència al radi que tindrà la coordenada polar d'aquest node a la representació. D'altra banda, la variable "r" està associada al càlcul de l'angle theta, què indicarà la posició en l'espai de la coordenada polar al voltant del punt arrel central. Aquest valor s'obté a partir de la multiplicació del valor de la coordenada "y" de cada node per la totalitat de graus a una circumferència, passant posteriorment el resultat a radians.

En el cas dels nodes fulla, no serà necessari cridar a la funció **polarConversion( )** ja que aquests necessàriament són fills d'un node intern i per tant les seues coordenades ja hauran sigut calculades al ser tractats com a fills del seu pare amb anterioritat. És a dir, en aquesta situació només en caldrà ficar com a coordenada X la longitud de l'element i com a coordenada Y el resultat de l'increment calculat abans d'entrar en la funció d'assignació de coordenades.

Una vegada modificades les coordenades de la llista d'elements d'arbre per a obtenir la representació polar, només ens quedarà executar un parell de subrutines dins de la funció principal **getCoords( )** per tal d'assegurar més endavant una visualització completa:

1. getMinCoords( ). Rep com a paràmetre un node (inicialment l'arrel). Recorrerà tota la llista d'elements i obtindrà les coordenades polars X i Y mínimes de manera recursiva.
2. translateCoords( ). Rep com a paràmetre un node (inicialment l'arrel). Recorrerà tota la llista d'elements de manera recursiva i aplicarà un sumatori del valor absolut de les coordenades per tal que totes les coordenades negatives passin a ser positives i per tant puguin ser visibles al panell de visualització.

*3.4.2.2.2 Càlcul de les Coordenades Polars (Sense Distàncies)*

Serà necessari crear una nova classe que anomenarem "Polar", on implementar *getCoords( )*. El disseny i la implementació de les funcions seran semblants a la classe "PolarDist", amb les següents diferències:

- Addició de la funció **getMaxLevel( )**. Aquesta rutina recorrerà de manera recursiva tots els elements de la llista de nodes i calcularà el nivell màxim de l'arbre. Aquest valor fa referència al màxim nombre de descendents que pot tenir un node a la filogènia tractada. Aquest valor l'emmagatzemarem a una variable que utilitzarem posteriorment.
- Canvis en **assignPolarCoordinates( )**. Dintre d'aquesta funció, al processar per primer cop els fills del node tractat, haurem d'invocar a la nova rutina getMaxLevel( ) per tal d'obtenir el nombre màxim de descendents que té aquest node fill. Dit d'una altra manera, estarem guardant el "nivell" de l'arbre on es troba aquest element (recordem: nivell 0 arrel, nivell 1 fills de l'arrel, etc.). Una vegada sabem el nivell al que pertany el node tractat, cridarem de manera recursiva a la funció assignPolarCoordinates( ). A diferència de la implementació "polar amb distàncies" en què li passàvem el node fill a tractar i la seva longitud sumada a la posició X del pare, ara li continuarem passant l'element a tractar però hi haurà un important canvi al segon paràmetre. A l'eliminar les distàncies, ara tots els elements que pertanyen a un mateix nivell es trobaran exactament a la mateixa distància del centre. És a dir, continuem tenint en compte les variacions genètiques sorgides al llarg del temps (branques) però no realitzem la comparació del nivell de divergència evolutiva entre dues noves (longitud).

Per tant, on abans teníem:

```
assignPolarCoordinates(children, xPosition + length);
```

Ara caldrà ficar:

```
assignPolarCoordinates(children, (maxX + rootLengthProportion -
((maxX/(maxTreeLevel)) * (maxLevel - level))));
```

Veiem ací la necessitat presentada de dividir l'arbre en nivells que tinguin la mateixa distància els uns respecte els altres, començant des de l'arrel (nivell 0), per poder realitzar el càlcul corresponent i col·locar cada node al nivell (distància respecte a l'arrel) que li correspon. Una vegada fet això ja es pot fer la conversió de coordenades de manera normal, obtenint finalment la representació polar sense distàncies desitjada.

Al igual que amb el tipus "polar amb distàncies", una vegada substituïdes les coordenades cartesianes per les polars corresponents, la llista d'elements modificada li serà delegada a la classe control *InternalF*. Aquesta al seu torn li passarà la informació a *PanelTree*, que serà la classe encarregada de representar tots els components de manera gràfica sobre un panell de visualització.

*3.4.2.2.3 Càlcul i Dibuix dels Components d'un Arbre Polar*

La classe *PanelTree* és una extensió de *JPanel*, i s'encarregarà de representar sobre aquest panell tots els components gràfics de l'aplicació referents a la representació d'arbre. El constructor d'aquesta classe rep per paràmetre la llista d'elements d'arbre, junt amb una instància de *InternalF* que contindrà informació d'utilitat i el tipus de representació a construir. Dintre d'aquesta classe *PanelTree* hi trobem la funció **paintComponent( )**, la qual rep com a paràmetre una instància de la classe *java.awt.Graphics* que serà la llibreria principal que s'utilitzarà per a la creació dels diferents components gràfics. Aquesta funció inicialitza les variables necessàries per al dibuix del panell gràfic principal i finalment realitza una comprovació on es determina el tipus d'arbre a representar. Una vegada identificat el tipus de representació, es farà una crida a la funció específica corresponent. Aquesta rutina serà l'encarregada de determinar els components gràfics a generar, així com de calcular de manera efectiva la posició, la mida i l'orientació del mateixos, per obtenir finalment la filogènia correctament representada de manera gràfica segons el tipus establert.

Per als nous casos de representacions polars, ens caldrà afegir una nova opció de comprovació a la funció paintComponent( ):

```
....
if(treeView.equalsIgnoreCase("cladogram")||treeView.equalsIgnoreCase("phy
logram")) {
      paintTreeRectangular(gc, treeList);
     } else if (treeView.equalsIgnoreCase("slanted")) {
                paintTreeSlanted(gc, treeList);
           } else  if (treeView.equalsIgnoreCase("unrooted")) {
                      paintTreeUnrooted(gc, treeList);
           } else
                if(treeView.equalsIgnoreCase("polarDist")||
                  (treeView.equalsIgnoreCase("polar"))) {
                       paintTreePolar(gc, treeList, treeView);
               } else {
                       paintTreeRectangular(gc, treeList);
                      }
               g2.drawImage(bimage, null, 0, 0);
```

Arribats a aquest punt, passarem a explicar el disseny i implementació de la nova funcionalitat **paintTreePolar( )**.

A aquesta funció li passarem una instància de la llibreria gràfica per poder utilitzar les seves funcions junt a la llista d'elements d'arbre, els quals anirà tractant un per un. Primer que tot, necessitarem emmagatzemar en diferents variables les coordenades X,Y del node tractat. També ens serà d'utilitat desar les coordenades dels seus fills directes, que seran "germans" entre ells. Una vegada tenim aquestes coordenades emmagatzemades, haurem de tenir en compte el factor d'escalada de l'arbre, ja que aquest influirà en el valor de les variables. És per això que haurem de multiplicar la X i la Y pels factors d'escalada de l'eix horitzontal i vertical, i sumar-li uns marges establerts prèviament.

A més a més, s'hauran de tenir presents les coordenades del node arrel, també amb el seu corresponent factor d'escalada, ja que com vam explicar anteriorment aquests valors es calculen d'una manera especial en comparació amb la resta de nodes. A banda, ens convindrà tenir guardades les coordenades X cartesianes del node tractat (recordem que en aquest moment la llista que rep la funció ja té els seus elements amb les coordenades transformades a polars), ja que les necessitarem pel càlcul d'algunes figures geomètriques. Si fem el disseny en pseudocodi d'aquesta part d'inicialitzacions:

```
xPare = node.agafar_coordenada_X()
yPare = node.agafar_coordenada_Y()
xGermans[] = nova_llista[node.mida]
yGermans[] = nova_llista[node.mida]

pareXCartesia = transformar_a_coordenades_cartesianes(xPare, yPare)

xPareEscalat = (xPare * factorEscaladaX) + margeX
yPareEscalat = (yPare * factorEscaladaY) + margeY

xArrelEscalat = (0.0 + menorCoordenadaX * factorEscaladaX) + margeX
yArrelEscalat = (0.0 + menorCoordenadaY * factorEscaladaY) + margeY
```

Una vegada realitzades les inicialitzacions, passem a tractar el node. Si aquest és un node intern (no és una fulla) haurem de calcular-li un arc, d'on posteriorment de cada extrem es connectarà amb els nodes fills (d'acord amb els criteris establerts per una correcta representació polar). D'aquest arc haurem de calcular la seua mida, o millor dit el seu radi, i l'angle que haurà de tenir. Per calcular el radi simplement haurem d'agafar la posició X del node en coordenades cartesianes, ja que aquest valor ens indica la posició i distància real respecte la resta de nodes, i el multiplicarem pel factor d'escalada perquè canviï de mida adequadament. Per obtenir el valor de l'angle, el càlcul serà més complex ja que per veure on comença i on acaba l'angle haurem de tenir en compte les coordenades dels nodes fills. Per fer això necessitarem recórrer aquests fills i calcular els valors X,Y entre els germans i, posteriorment, passar la coordenada Y del fill tractat a cartesiana per determinar l'altura real a la que s'haurà de col·locar a la representació. Acte seguit, realitzarem el càlcul efectiu de l'angle; si estem tractant el primer fill calcularem l'angle de començament de l'arc i si és el segon el d'acabament.

Un cop tenim aquests angles, podrem passar a calcular el punt mig de l'arc, que necessitarem per determinar la posició on dibuixar-lo. Això ho farem primer obtenint l'angle mig i després passant-lo a radians (fins ara estàvem treballant en graus) i, acte seguit, restarem la mida del radi de la respectiva coordenada polar per després multiplicar-lo pel cosinus dels radians calculats en el cas de la coordenada X o pel sinus en el cas de la Y. El pseudocodi de la part de representació dels arcs quedaria així:

```
si (node.es_node_intern()) llavors
      angleComençament = 0
      angleAcabament = 0
      radiArc = pareXCartesia * factorEscaladaX

      per (tots els fills del node) fer
            fillActual = agafar_fill(i)
            xFill = fillActual.agafar_coordenada_X()
            yFill = fillActual.agafar_coordenada_Y()
            xGermans[i] = (xFill * factorEscaladaX) + margeX
            yGermans[i] = (yFill * factorEscaladaY) + margeY
            fillYCartesia  =  transformar_a_coordenades_cartesianes(xFill,
yFill)
            si (i == 0) //primer fill
                  angleComençament = calcular_angle(fillYCartesia)
            sinó // segon fill
                  angleAcabament = calcular_angle(fillYCartesia)

      angleMig = (angleComençament + angleAcabament) / 2
      radians = convertir_a_radians(angleMig)
      centreArcX = xPareEscalat – radiArc * cosinus(radians)
      centreArcY = yPareEscalat – radiArc * sinus(radians)
```

La funció **calcular_angle( )** rebrà la coordenada Y, que indicarà l'altura en sistema cartesià, i transformarà aquest valor en un angle mitjançant un càlcul amb el rang angular d'una circumferència (360 graus). Així doncs, també estarem fent una mena de conversió de cartesià a polar per obtenir l'angle definitiu desitjat:

```
Funció calcular_angle(y)
      angle = y * rangAngular
      angle = angle mod 360
      retorna angle
fiFunció
```

El fet de calcular el modular es deu a que alguns resultats de les multiplicacions poden donar un valor superior a 360 graus, per tant, estaríem donant una altra volta a la circumferència i ens interessa el valor de l'angle dins del rang normal (de 0 a 360 graus).

Una vegada calculades les posicions dels arcs junt amb el seu radi i els seus respectius angles de començament i d'acabament, ara ens caldrà connectar els extrems de cada arc (que tindrà com a punt central el node tractat) amb les coordenades dels nodes fills mitjançant el traçat de línies. Com hem fet fins ara, haurem de diferenciar les que hauran de sortir de l'arc central (el corresponent a l'arrel de l'arbre) de la resta. Si ens trobem tractant l'arrel, haurem d'obtenir la posició X,Y central de l'arc del node (de manera semblant a com hem fet anteriorment) i, posteriorment, les posicions X,Y de l'angle de començament i acabament d'aquest mateix arc.

El càlcul d'aquestes últimes posicions consisteix a sumar la posició de l'arrel més el radi de l'arc central i multiplicar-ho pels sinus/cosinus dels angles de començament i acabament. Acte seguit, traçarem tres línies:

- De l'arrel original a l'arrel escalada i distanciada (recordem com anteriorment havíem establert el valor de la proporció de la distància de l'arrel a 1).
- Del punt de començament de l'arc fins la posició del primer fill.
- Del punt d'acabament de l'arc fins la posició del segon fill.

Cal recordar que els arbres filogenètics són binaris, és a dir, de cada canvi evolutiu representat per un node intern en sortiran dues branques més. No obstant això, l'arrel és un cas especial i es podria donar la situació que aquesta tingués tres fills. Per tant, aquest cas s'haurà de tenir en compte i quan es doni la situació s'haurà de traçar una línia addicional que anirà del punt central de l'arc arrel fins la posició del tercer fill. D'aquesta manera, la part explicada quedaria així:

```
Si (node.es_node_arrel()) llavors
      arrelX = xArrelEscalat + radiArc * cosinus(radians)
      arrelY = yArrelEscalat + radiArc * sinus(radians)
      començamentArcX = xArrelEscalat + radiArc * cos(angleComençament)
      començamentArcY = yArrelEscalat + radiArc * sin(angleComençament)
      acabamentArcX = xArrelEscalat + radiArc * cos(angleAcabament)
      acabamentArcY = yArrelEscalat + radiArc * sin(angleAcabament)

      dibuixar_linia(arrelX, arrelY, xArrelEscalat, yArrelEscalat)
      dibuixar_linia(començamentArcX,    començamentArcY,    xGermans[0],
yGermans[0])
      dibuixar_linia(començamentArcX,    començamentArcY,    xGermans[1],
yGermans[1])

      si (tercer_fill)
            dibuixar_linia(arrelX, arrelY, xGermans[2], yGermans[2])
      fsi
```

Finalment, haurem de dibuixar també l'arc central amb el calculat prèviament:

```
      dibuixar_arc(xArrelEscalat,         yArrelEscalat,         radiArc,
angleComençament, angleAcabament)
```

En cas que el node tractat no sigui l'arrel, els càlculs seran igual però ara haurem de sumar al radi el punt central de l'arc del node en qüestió, en comptes del punt del node arrel. A banda, només traçarem les línies que van dels extrems de l'arc a les posicions dels fills (ara el node pare solament pot tenir-ne 2):

```
sinó // no és el node arrel
      començamentArcX = centreArcX + radiArc * cosinus(angleComençament)
      començamentArcY = centreArcY + radiArc * sinus(angleComençament)
      acabamentArcX = centreArcX + radiArc * cosinus(angleAcabament)
      acabamentArcY = centreArcY + radiArc * sinus(angleAcabament)

      dibuixar_arc(centreArcX, centreArcY, radiArc, angleComençament,
angleAcabament)
fsi
```

Per a una major comprensió de tot l'explicat a aquesta part, veiem a la *Figura 21* una representació esquemàtica del patró de dibuix dels diferents elements geomètrics per a un node qualsevol.

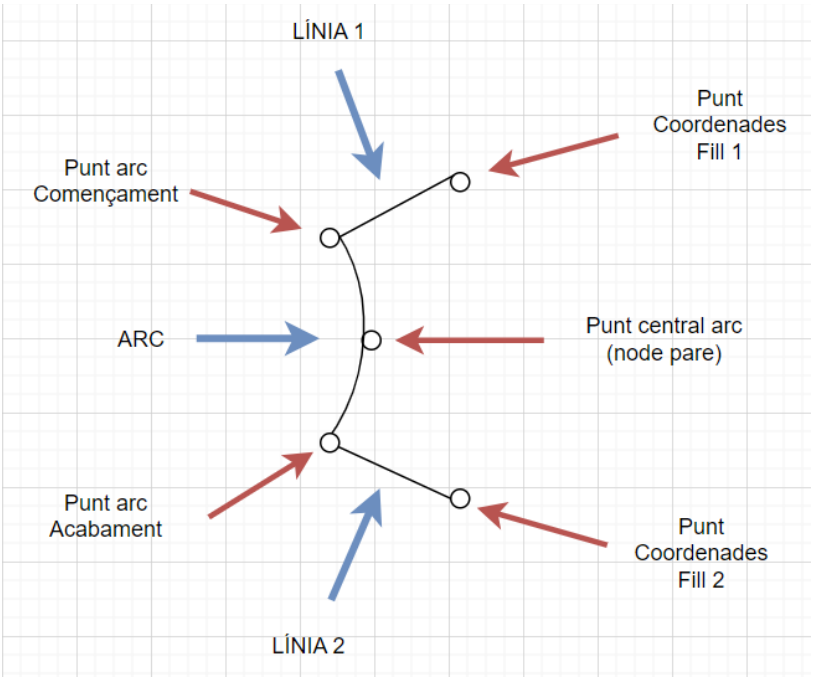



**Figura 21.** Esquema explicatiu dels elements que conformen l'estructura geomètrica bàsica de les representacions filogenètiques de tipus polar.

Ara ja tindríem les funcionalitats requerides completament definides. En el que respecta a la implementació, cal destacar la utilització de la llibreria Graphics 2D per representar les figures geomètriques:

- **Arc2D.** Aquesta classe l'emprarem per generar els arcs mitjançant la creació d'una instància de la mateixa i després cridant a la funció predefinida **setArcByCenter( )**, la qual rep com a paràmetres:
    - Coordenada X del centre de l'arc (Double).
    - Coordenada Y del centre de l'arc (Double).
    - Radi de l'arc (Double).
    - Angle de començament de l'arc en graus (Double).
    - Extensió angular de l'arc en graus (Double).
    - Tipus d'arc (OPEN, CHORD o PIE). En el nostre cas ens interessarà un arc de tipus "obert", què representaria un arc tradicional.
- **g2.draw( ).** Aquesta rutina rep com a paràmetre un tipus d'element geomètric, el qual s'encarregarà de dibuixar. En la nostra implementació, rebrà l'arc a pintar del node tractat a cada iteració recursiva de la funció *paintTreePolar( )* descrita a dalt.
- **g2.drawLine( ).** Com bé indica el seu nom, aquesta funcionalitat de la llibreria serà la responsable de realitzar el traçat de les diferents línies. Els paràmetres a rebre són:
    - Coordenada X del primer punt.
    - Coordenada Y del primer punt.
    - Coordenada X del segon punt.
    - Coordenada Y del segon punt.

Una vegada tots els components han sigut generats, la funcionalitat **g2.drawImage( )** a la funció principal **paintComponent( )** serà l'encarregada de dibuixar aquests components a la imatge del panell que visualitzarà l'usuari.

# 4 Avaluació i Resultats

En aquest capítol es presentaran els resultats que s'han obtingut mitjançant la realització de diferents tipus de proves sobre les noves funcionalitats, i es farà una avaluació dels mateixos.

## 4.1 Proves de lectura/escriptura dels nous formats de fitxers

Pel cas dels formats d'arxiu, a la *Taula 2* es mostren els resultats de funcionament de les diferents opcions implementades, junt a una referència visual en aquells casos on és possible obtenir-ne una.

| **Funcionalitat** | **Èxit** | **Referència visual (si escau)** |
|---|---|---|
| Lectura de fitxer en format *nexus* | Sí | |
| Lectura de fitxer en format *json* | Sí | |
| Guardat de fitxer en format *nexus* | Sí | Figura 18. Captura del contingut d'un fitxer d'arbre en format nexus d'un clade extret de *AP_caulimovirus*. Font: material de treball. |
| Guardat de fitxer en format *json* | Sí | Figura 19. Captura del contingut d'un fitxer d'arbre en format json d'un clade extret de *AP_caulimovirus*. Font: material de treball. |

**Taula 2.** Resultats obtinguts en la comprovació de correcte funcionament de les opcions relacionades amb els formats de fitxer d'arbre.

## 4.2 Proves de validació dels nous tipus d'arbres

Pel cas dels arbres polars, es van realitzar proves amb els dos tipus implementats utilitzant arxius d'arbres de diferents mides.

- Arbres petits (<50 fulles): *Figura 8* i *Figura 9* (30 fulles).
- Arbre mitjans (entre 50 i 300 fulles): *Figura 22* i *Figura 23* (291 fulles).
- Arbres grans (>300 fulles): *Figura 24* i *Figura 25* (1822 fulles).

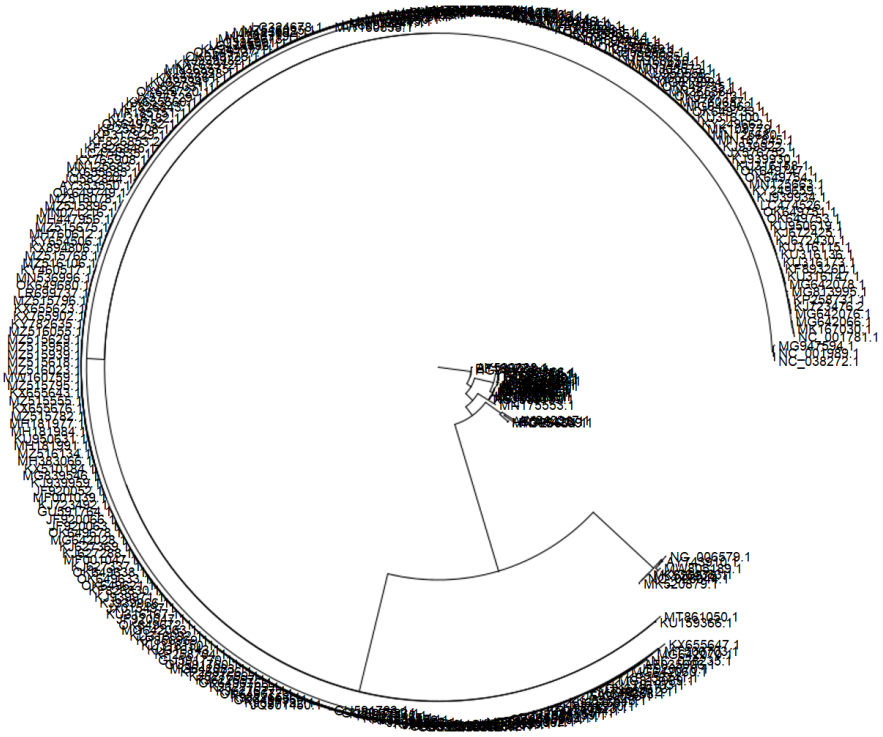


**Figura 22.** Representació polar amb distàncies d'un arbre de mida mitjana.

Font: material de treball.

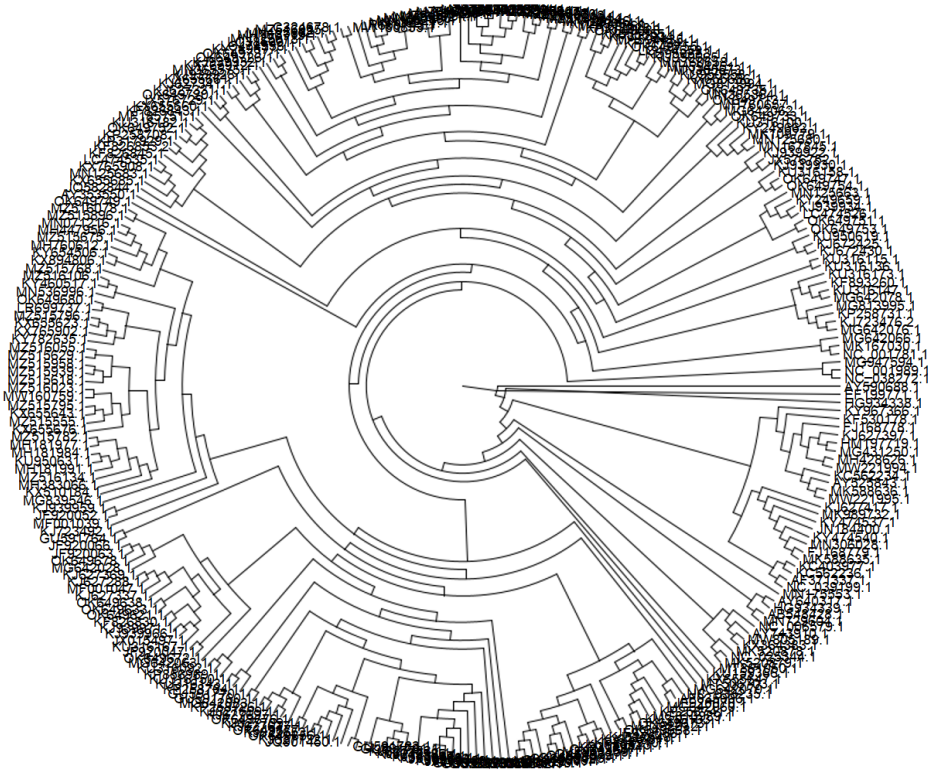


**Figura 23.** Representació polar sense distàncies d'un arbre de mida mitjana.

Font: material de treball.

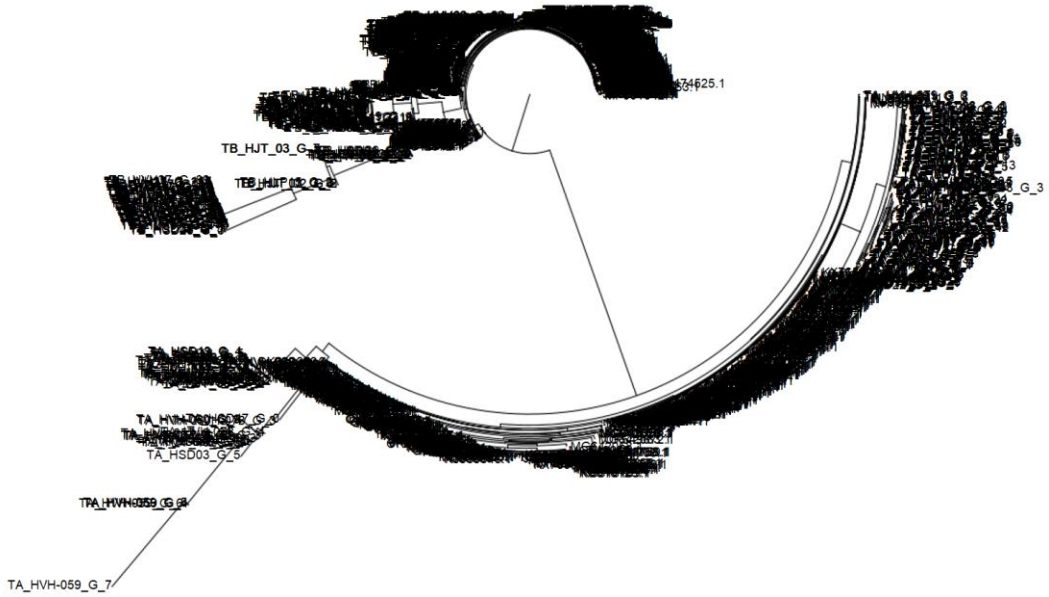



**Figura 24.** Representació polar amb distàncies d'un arbre de mida gran.

Font: material de treball.

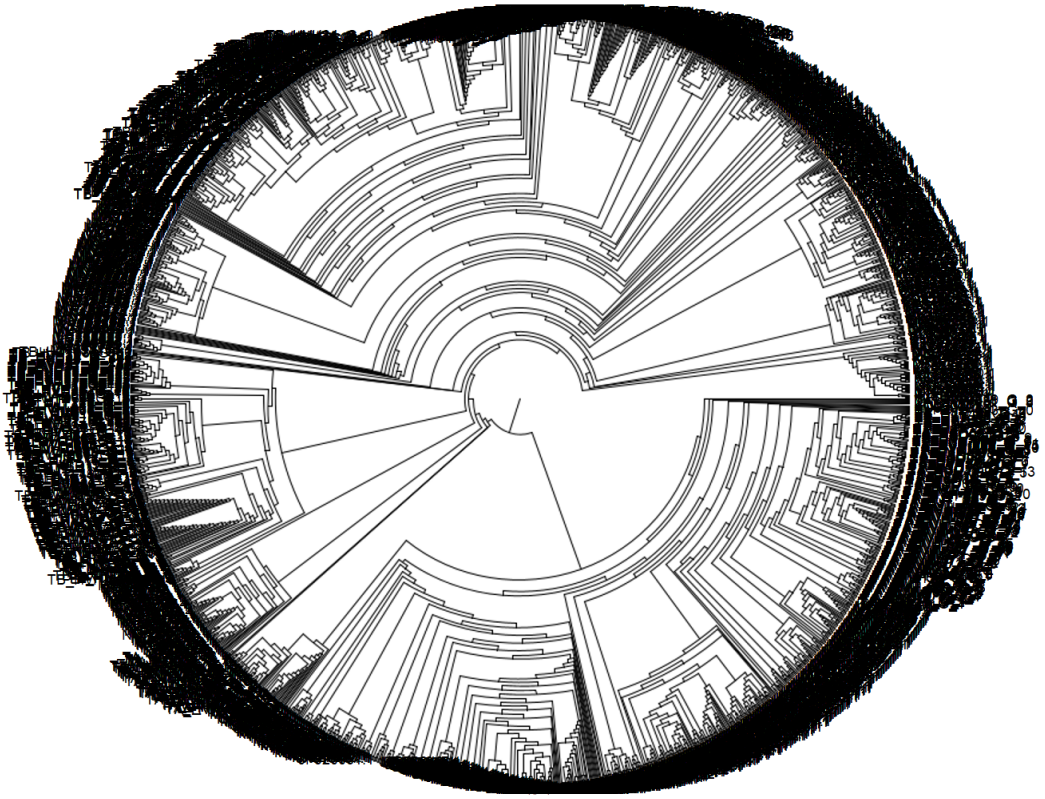


**Figura 25.** Representació polar sense distàncies d'un arbre de mida gran.

Font: material de treball.

Amb els resultats obtinguts, i realitzant una valoració fonamentada en els coneixements filogenètics adquirits, es va determinar que el Phylograph respon de manera correcta a arxius amb informació filogenètica de mida i característiques variables, sense veure's afectat el resultat final per l'increment de dades a processar (sempre dins d'unes limitacions reals).

Cal comentar que als arbres de mida mitjana i gran la informació a les fulles apareix condensada, dificultant la llegibilitat i l'estudi de l'arbre. No obstant això, aquest és un problema que tenen també altres programes d'una índole similar, ja que la quantitat d'informació a representar és molt superior a la mida de la pantalla on és mostrada. Aquest inconvenient es soluciona perfectament a mesura que utilitzem l'eina de zoom proporcionada pel Phylograph, la qual ens permet visualitzar l'arbre d'una manera totalment interpretable i correcta fins a l'últim detall.

Un altre aspecte a remarcar és que la resolució obtinguda inicialment amb la llibreria base de gràfics per a la representació dels elements als diferents tipus d'arbres va ser d'una qualitat baixa, tal i com veiem a la *Figura 26.* És per això que més endavant es va decidir utilitzar la funcionalitat **setRenderingHint** proporcionada per la pròpia llibreria i que, amb l'ajustament de diferents paràmetres, va permetre aconseguir resolucions de major qualitat per les figures (veure *Figura 3*).

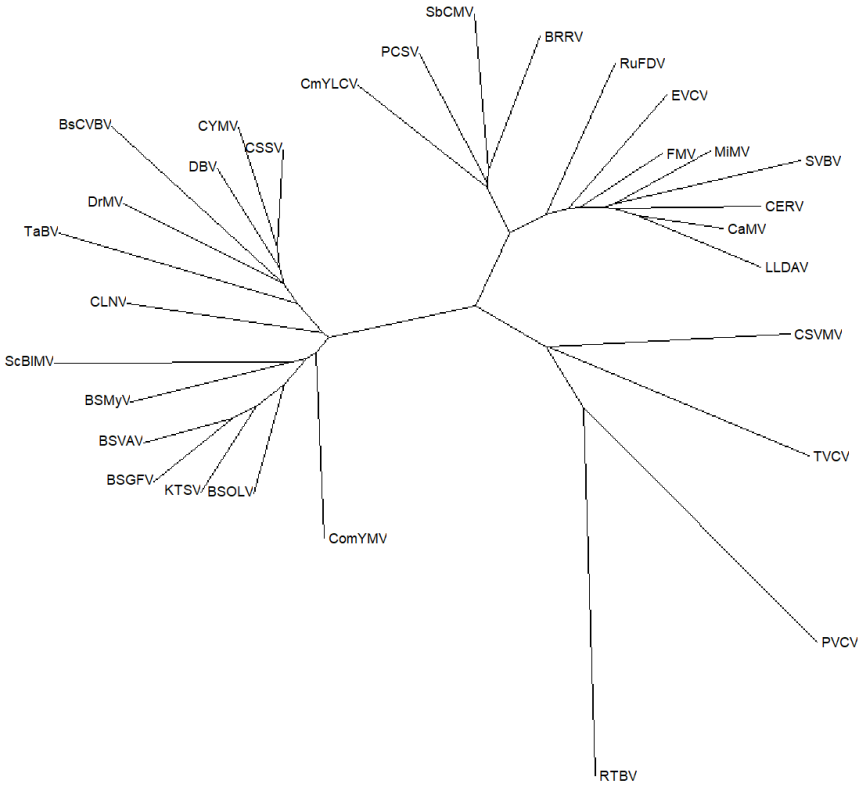



**Figura 26.** Representació radial de la filogènia de *AP_caulimovirus*, abans de la millora en el renderitzat dels gràfics.

Font: material de treball.

## 4.3 Comparativa amb altres eines

Per assegurar-nos de que les representacions resultants eren totalment fiables, es van realitzar diverses comparacions amb altres eines de propòsit semblant al Phylograph àmpliament reconegudes internacionalment.

D'una banda, es van comparar els resultats obtinguts amb l'utilitat *treeviewer* del NCBI[7]. Aquest aplicatiu online pertanyent al Centre Nacional de Medicina dels Estats Units també permet pujar arxius per a la visualització i posterior edició d'arbres filogenètics [14]. Realitzant una comparativa de la representació polar al Phylograph utilitzant l'exemple per defecte tractat en aquest treball (*Figura 8* i *Figura 9*) amb la representació polar al *treeviewer* del NCBI (*Figura 27*), podem confirmar que ambdues són equivalents. No obstant això, el *treeviewer* construeix la representació polar fent una fusió dels dos subtipus amb distàncies (línies grosses) i sense distàncies (línies primes), la qual cosa pot resultar confusa per l'usuari i a més no genera una representació completament fidel respecte a quan aquests subtipus es tracten per separat, tal i com fem al Phylograph. Així mateix, l'aplicatiu del NCBI presenta una interfície gràfica més pobra en el que respecta a l'oferta de funcionalitats per editar els arbres, disposa de menys tipus de representacions i tarda més temps en processar les dades.

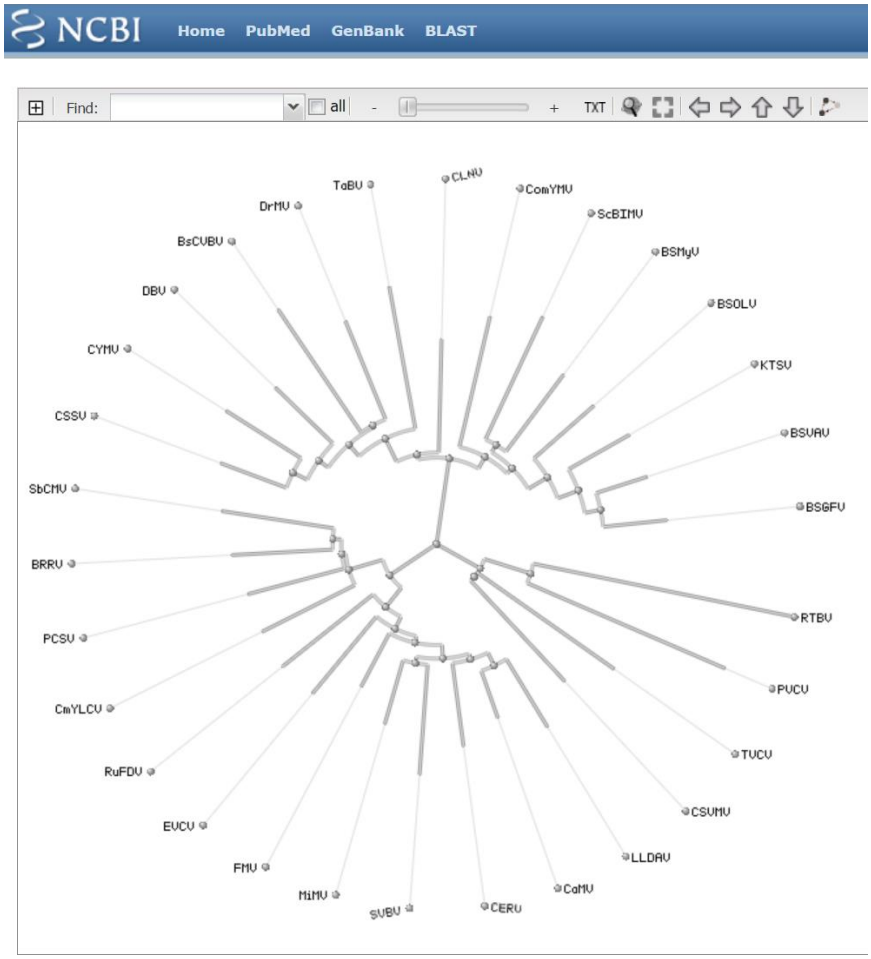



**Figura 27.** Representació polar de la filogènia de *AP_caulimovirus*, utilitzant l'eina *treeviewer* del NCBI.

[7] National Center for Biotechnology Information

D'altra banda, es va utilitzar l'eina online "Simple Phylogeny" del EMBL[8] - EBI[9] per realitzar més comparacions. Aquest software de l'Institut Europeu de Bioinformàtica permet representar arbres filogenètics a partir d'un arxiu de seqüències d'alineaments [15]. És a dir, aquesta utilitat necessita com a *input* les seqüències d'ADN o de proteïnes que conformen els diferents nodes externs que volem estudiar. Això pot suposar un inconvenient, ja que l'usuari necessita disposar de més informació perquè el programa sigui capaç de processar les dades d'interès (recordem quan es va explicar anteriorment que un arxiu d'arbre normal no conté informació de seqüència, sinó simplement les relacions entre els diferents nodes). A més a més, només disposa d'opcions de representació en forma de filograma i arbre polar, i no conté gaire funcionalitats d'edició. Comparant els resultats del Phylograph (*Figura 8*) amb els de "Simple Phylogeny" (*Figura 28*) veiem de nou com són equivalents.

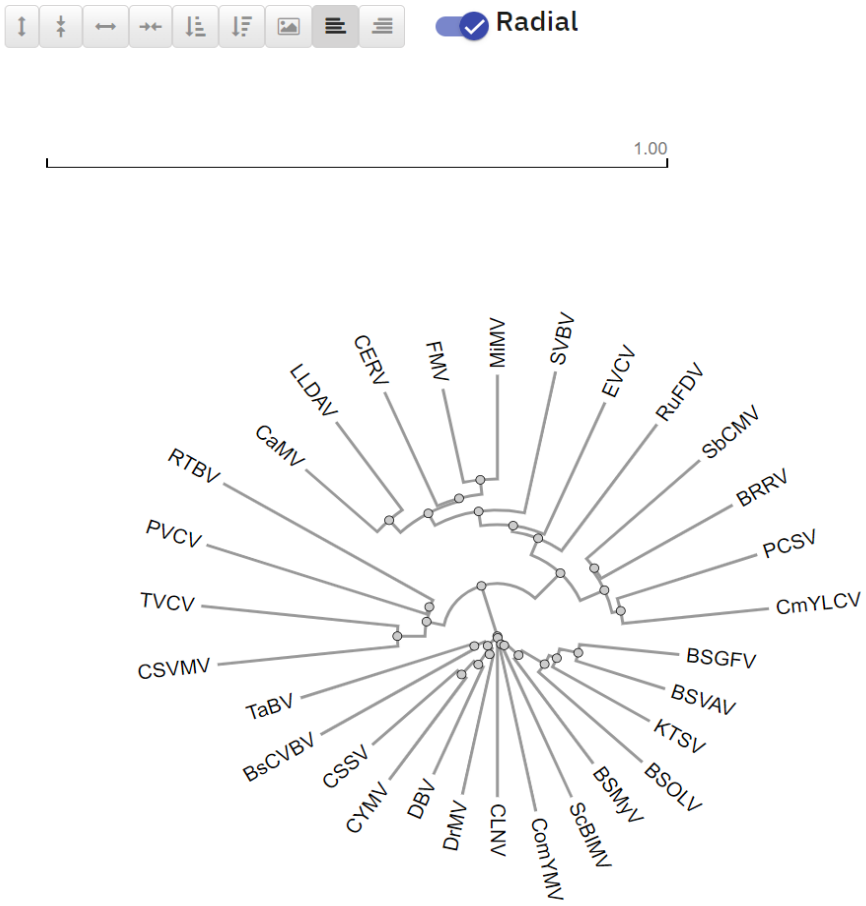



**Figura 28.** Representació polar amb distàncies de la filogènia de *AP_caulimovirus*, utilitzant l'eina "Simple Phylogeny" del EMBL-EBI.

## 4.4 Documentació

Com a resultat d'aquest treball fi de grau, a banda de proporcionar la nova versió de Phylograph i de la seva validació, s'ha elaborat documentació científico-tècnica en llengua anglesa per deixar constància de les noves implementacions i els canvis duts a terme -Annex C-. A més a més, s'ha desenvolupat una guia d'usuari on queden recollides explicacions de totes i cadascuna de les funcionalitats ofertes pel programa -Annex D-.

[8] European Molecular Biology Laboratory

[9] European Bioinformatics Institute

# 5 Conclusions i Perspectives de Futur

Una vegada finalitzat el treball i explicat de manera detallada el seu propòsit i el desenvolupament del mateix, podríem dir que la feina realitzada ha estat exitosa. Tots els resultats finals han sigut els esperats i, per tant, la justificació dels procediments duts a terme durant les diferents etapes ha quedat totalment validada.

Hem vist com al llarg de la memòria s'han anat complint els diferents objectius proposats. Partint d'una aplicació amb software desactualitzat, hem sigut capaços d'actualitzar-lo a una nova versió i hem dut a terme una sèrie de proves test exhaustives per assegurar la correcta funcionalitat de tot el programa. Posteriorment, hem pogut ampliar l'aplicació amb noves opcions, seguint pas per pas els procediments establerts en l'àmbit de l'enginyeria del software tant en el que respecta al disseny com a la implementació. Tot això ens ha permès obtenir finalment una nova versió de Phylograph actualitzada, ampliada i completament funcional.

Podem observar que des del moment de l'actualització primera hem creat un codi més modularitzat i optimitzat, mitjançant l'ús de tècniques tals com la implementació de genèrics o una millor gestió dels esdeveniments. Al seu torn, la realització dels múltiples tests ha propiciat la redacció de la guia d'usuari, la qual serà d'extrema utilitat per a aquells futurs usuaris que utilitzin aquest software. Així mateix, la implementació dels nous formats de lectura/escriptura fa que el programa sigui molt més ric i variat en el que respecta a la interpretabilitat, processament i exportació de fitxers filogenètics. D'una altra banda, la implementació de la construcció d'arbres de tipus polar suposa una gran millora en la part de representació gràfica, ja que aquestes representacions estan molt sol·licitades avui dia per la seva utilitat en els estudis filogenètics. A més a més, la documentació tècnica redactada a posteriori on s'expliquen tots els canvis duts a terme i les implementacions realitzades facilitarà la futura manutenció del programa.

Aquest treball ens serveix com un clar exemple de la gran importància que té per a les empreses i entitats la realització de tasques de manteniment i actualització sobre programari propi, així com també de la necessitat de tenir sempre una planificació programada de les mateixes. A més a més, s'exposa la capacitat d'evolució que deu tenir un codi de propòsit particular per tal de poder dissenyar i afegir noves funcionalitats, adaptant-se sempre a les necessitats del moment i sense quedar-se enrere respecte a possibles competidors.

Així doncs, Phylograph es presenta com una potent eina software per a la representació i edició d'arbres filogenètics. La seva àmplia oferta de funcionalitats junt a la seva gran capacitat de processament i operativitat fa que no tingui res a envejar respecte a altres aplicacions creades per al mateix propòsit. En definitiva, la completesa i alta fiabilitat proporcionada pel Phylograph el converteix en una utilitat que qualsevol persona realitzant un estudi filogenètic no hauria de dubtar en fer servir.

## 5.1 Opinió Personal

Personalment, crec que aquest conjunt de tasques tan laborioses en les quals s'emmarca el Treball Fi de Grau m'han ajudat a desenvolupar-me tant en l'àmbit professional com en l'acadèmic. He vist com he sigut capaç de participar en tot el procés productiu de manteniment i desenvolupament d'un actiu software, combinant al mateix temps coneixements de les ciències biològiques que han sigut claus per a la comprensió del programa. A banda, he tingut la capacitat d'assolir tots els objectius presentats en base a una planificació degudament estipulada. Alguns d'aquests punts a complir van presentar un important repte en el seu moment, sobretot en el que respecta al disseny i la implementació de les noves opcions, perquè requerien d'uns coneixements molt tècnics del camp de treball i de l'entorn de l'aplicació. No obstant això, vaig haver d'assumir la responsabilitat de ser l'únic membre de l'empresa treballant de manera activa en el projecte i que posseïa una comprensió específica del mateix, fruit d'un estudi intensiu del codi. Aquesta assumpció de responsabilitats penso que és la que va possibilitar finalment l'assoliment i el desenvolupament correcte de totes les tasques requerides.

Aquest ha sigut un projecte que inicialment era aliè a la meua persona, però que des del primer dia vaig agafar amb il·lusió com un repte personal i que finalment i després de molt de treball i esforç he sigut capaç d'interioritzar-lo i fer-lo meu. En resum, estic content d'haver-li donat una nova vida i un nou futur al Phylograph perquè penso que aquesta aplicació pot ser de gran utilitat en el seu àmbit d'acord al propòsit per la qual va ser creada.

## 5.2 Phylograph al futur

El desenvolupament de Phylograph té una llarga trajectòria la qual encara no ha acabat. D'ençà fa més de deu anys fins a l'actualitat, ha experimentat alguns canvis amb el transcurs del temps, sent especialment destacable la feina presentada en aquest treball. A banda de l'actualització i les noves funcionalitats implementades, es té prevista la realització de més fases de refinament junt amb l'addició de noves opcions per fer el programa encara més complet. Una vegada quedi completat el procés, està planificada la redacció d'un article científic i la seua publicació en una revista d'alt impacte per, finalment, llançar al mercat el producte i posar de manera definitiva aquesta eina al servei de la ciència.

M'agradaria donar les gràcies una vegada més a l'empresa per convidar-me també a participar de manera activa en aquesta última etapa. En base al meu coneixement adquirit sobre l'aplicació, penso que ja que es poden realitzar multitud d'opcions d'edició sobre els nodes interns i les branques que els componen (arrelament com a subgrup, rotació de la branca, mostrar/amagar node, establiment de color, edició separada com a subarbre, desament com a clúster...), també seria positiu implementar noves funcionalitats que proporcionin tractaments addicionals sobre les fulles, tals com la implementació d'etiquetes personalitzades. A banda, seria de gran utilitat explotar la capacitat i modularitat del format *nexus* per afegir nous blocs d'informació filogenètica als arxius, en comptes d'únicament tenir el bloc bàsic corresponent a l'estructura de l'arbre, i així poder representar posteriorment aquesta nova informació de manera gràfica al panell. En el que respecta a la interfície d'usuari, potser caldria reestructurar els menús i renovar la barra d'eines per tal d'obtenir una visualització més estètica i moderna.

Totes aquestes propostes junt amb altres establertes per l'empresa que queden fora dels objectius d'aquest treball seran contemplades a la nova fase d'ampliació del programa, per tal d'obtenir un Phylograph encara més competent i científicament rigorós.

# Referències


[1] C. M. Micheel *et al.*, "Omics-Based Clinical Discovery: Science, Technology, and Applications," Mar. 2012, Accessed: May 06, 2024. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK202165/

[2] "Bioinformatics." Accessed: Apr. 20, 2024. [Online]. Available: https://www.genome.gov/genetics-glossary/Bioinformatics

[3] D. Qin, "Next-generation sequencing and its clinical application," *Cancer Biol Med*, vol. 16, no. 1, p. 4, 2019, doi: 10.20892/J.ISSN.2095-3941.2018.0055.

[4] "What is phylogenetics?" Accessed: Apr. 20, 2024. [Online]. Available: https://www.yourgenome.org/theme/what-is-phylogenetics/

[5] "Phylogenetic Tree or Evolutionary Tree | Construction and Overview." Accessed: Jun. 04, 2024. [Online]. Available: https://www.geeksforgeeks.org/phylogenetic-tree/

[6] "Nina Kranke - Phenograms, Cladograms, Phylogenetic Trees." Accessed: Jun. 04, 2024. [Online]. Available: https://nina-kranke.com/how-phenograms-and-cladograms-became-molecular-phylogenetic-trees/

[7] "Phylogenetic Tree- Definition, Types, Steps, Methods, Uses." Accessed: Jun. 04, 2024. [Online]. Available: https://microbenotes.com/phylogenetic-tree/

[8] "Biotechvana." Accessed: Apr. 27, 2024. [Online]. Available: https://bioinformatics.biotechvana.com/index.php/article/23

[9] "Maven – Introduction to the Dependency Mechanism." Accessed: May 03, 2024. [Online]. Available: https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

[10] B. Efron, E. Halloran, and S. Holmes, "Bootstrap confidence levels for phylogenetic trees," *Proc Natl Acad Sci U S A*, vol. 93, no. 23, pp. 13429–13434, Nov. 1996, doi: 10.1073/PNAS.93.23.13429/ASSET/9196EE72-60A3-4CB1-99F1-489B67DAA752/ASSETS/GRAPHIC/PQ0060195006.JPEG.

[11] D. R. Maddison, D. L. Swofford, and W. P. Maddison, "NEXUS: an extensible file format for systematic information," *Syst Biol*, vol. 46, no. 4, pp. 590–621, Dec. 1997, doi: 10.1093/SYSBIO/46.4.590.

[12] "Working with JSON - Learn web development | MDN." Accessed: Apr. 26, 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

[13] "Polar co-ordinates." [Online]. Available: www.mathcentre.ac.uk

[14] "NCBI Tree Viewer." Accessed: May 31, 2024. [Online]. Available: https://www.ncbi.nlm.nih.gov/tools/treeviewer/

[15] "Simple Phylogeny < EMBL-EBI." Accessed: May 31, 2024. [Online]. Available: https://www.ebi.ac.uk/jdispatcher/phylogeny/simple_phylogeny

[16] M. Balaban, N. Moshiri, U. Mai, X. Jia, and S. Mirarab, "TreeCluster: Clustering biological sequences using phylogenetic trees," *PLoS One*, vol. 14, no. 8, Aug. 2019, doi: 10.1371/JOURNAL.PONE.0221068.

## Annex A: Clústers filogenètics al Phylograph

En un arbre filogenètic, un clúster és tota aquella agrupació de seqüències diferents que comparteixen un mateix ancestre comú [16].

L'aplicació Phylograph disposa d'una funcionalitat la qual permet realitzar aquest tipus d'agrupacions. Punxant el botó dret a sobre de qualsevol node intern, hi ha una opció anomenada "Save Cluster". Aquesta opció reconeixerà les fulles que tenen com a ancestre comú el node intern seleccionat, i desarà aquesta informació de clúster a un arxiu anomenat *cluster_default*. Acte seguit, quan l'usuari seleccioni les funcionalitats de la secció "Clusters" del menú, el programa s'encarregarà de llegir aquest arxiu i representarà de manera gràfica els clúster desats (*Figura 29*).

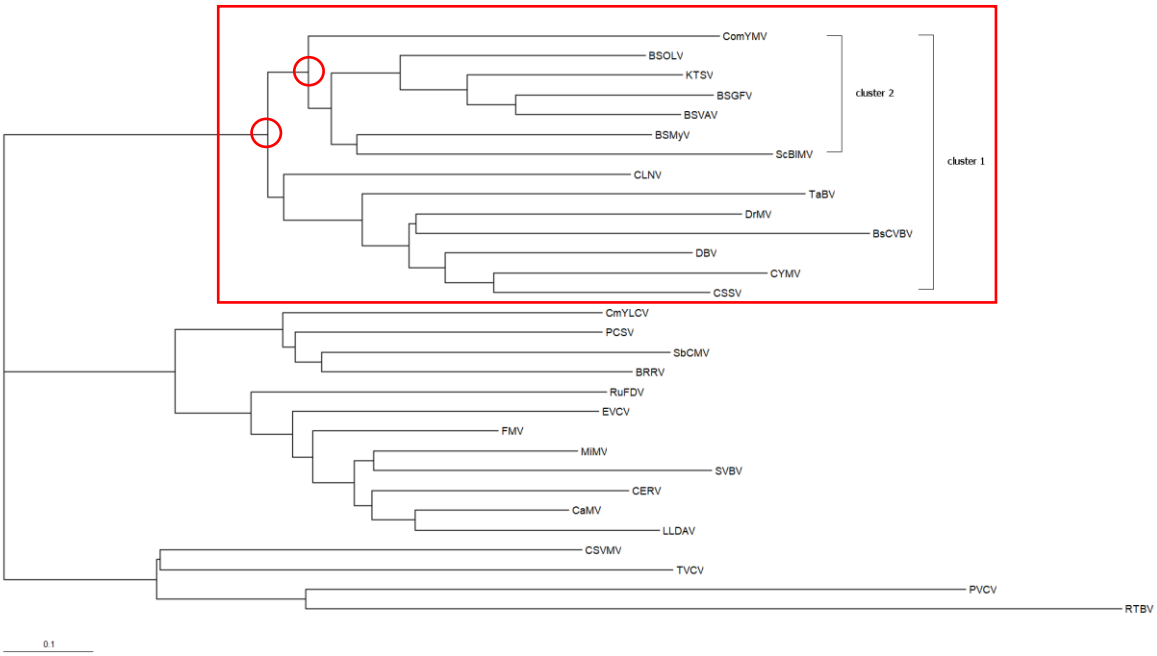



**Figura 29.** Captura d'una filogènia on s'han creat dos clústers a partir dels nodes marcats amb un cercle roig.

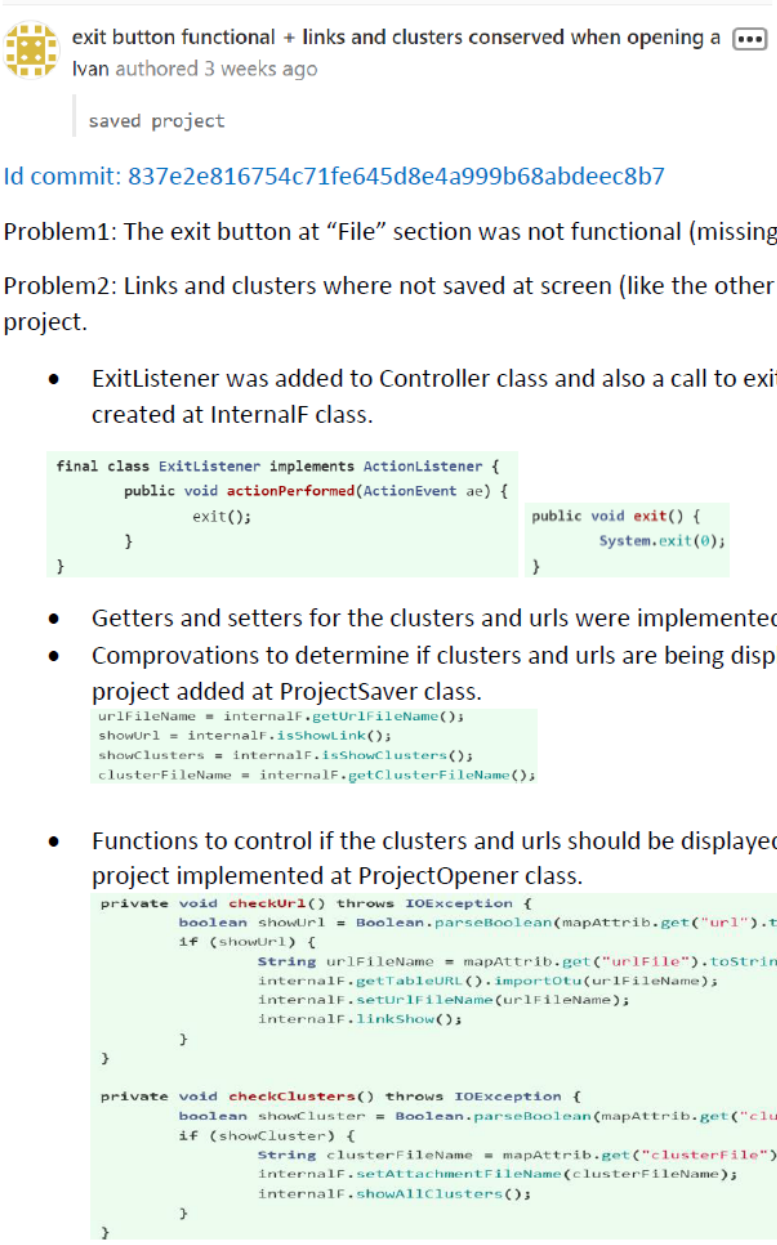
## Annex B: Salvat d'un arxiu com a projecte

A l'aplicació, una vegada obert l'arxiu d'arbre desitjat, hi ha una opció a la secció "File" del menú anomenada "Save Project", la qual permet desar la filogènia estudiada com a un projecte. D'aquesta manera, la persona investigadora pot conservar les modificacions realitzades amb el programa (creació de clústers, coloració i etiquetatge de branques...) i seguir treballant més endavant.

Quan es desa una filogènia com a projecte es generen tres nou arxius:

- tree_project.phylo → Conté informació bàsica pel projecte i representa el seu últim estat (localització de l'arxiu a l'arbre de directoris, tipus de representació, dimensions, clústers, fonts i mètriques utilitzades, etc).
- tree_project_components.xml → Conté informació sobre les característiques per defecte dels diferents components (mida, tipus de font, etc).
- tree_project_tree.xml → Desa totes les dades necessàries dels nodes i fulles de l'abre perquè després pugui ser interpretat i processat de nou correctament.

# Annex C: Mostra de la redacció de documentació científico-tècnica



exit button functional + links and clusters conserved when opening a
Ivan authored 3 weeks ago

saved project

Id commit: 837e2e816754c71fe645d8e4a999b68abdeec8b7

Problem1: The exit button at "File" section was not functional (missing listener).

Problem2: Links and clusters where not saved at screen (like the other elements) after saving a project.

- ExitListener was added to Controller class and also a call to exit() function, which was created at InternalF class.

```
final class ExitListener implements ActionListener {
        public void actionPerformed(ActionEvent ae) {
                exit();
        }
}
```

```
public void exit() {
        System.exit(0);
}
```

- Getters and setters for the clusters and urls were implemented at InternalF class.
- Comprovations to determine if clusters and urls are being displayed before saving a project added at ProjectSaver class.

```
urlFileName = internalF.getUrlFileName();
showUrl = internalF.isShowLink();
showClusters = internalF.isShowClusters();
clusterFileName = internalF.getClusterFileName();
```

- Functions to control if the clusters and urls should be displayed when opening a project implemented at ProjectOpener class.

```
private void checkUrl() throws IOException {
        boolean showUrl = Boolean.parseBoolean(mapAttrib.get("url").toString());
        if (showUrl) {
                String urlFileName = mapAttrib.get("urlFile").toString();
                internalF.getTableURL().importOtu(urlFileName);
                internalF.setUrlFileName(urlFileName);
                internalF.linkShow();
        }
}

private void checkClusters() throws IOException {
        boolean showCluster = Boolean.parseBoolean(mapAttrib.get("cluster").toString());
        if (showCluster) {
                String clusterFileName = mapAttrib.get("clusterFile").toString();
                internalF.setAttachmentFileName(clusterFileName);
                internalF.showAllClusters();
        }
}
```

**Figura 30.** Captura d'un fragment de la documentació tècnica elaborada en base als canvis realitzats al programa.

# Annex D: Mostra de la redacció de la guia d'usuari del programa

## 2.2.- Tool Bar

The TOOL BAR provides access to many functionalities related to graphic processing of the tree, for better manageability and editing. It allows tree analysis in a more visual way adapted to each user.

The Tool Bar is organized as follows:


A1 A2 A3 B1 B2 B3 B4 C1 C2 D E1 E2 E3 E4 F1 F2 G H1 H2 H3 H4 H5

**A1)** [Open treefile] → Opens a treefile.
Same functionality as [File → Open treefile] from the Menu Bar.

**A2)** [Open project] → Opens a previously saved project.
Same functionality as [File → Open project] from the Menu Bar.

**A3)** [Save project] → Saves a project. If no project exists, creates a new project.
Same functionality as [File → Save project] from the Menu Bar.

**B1)** [Expand width] → Expands the width of the tree.
**B2)** [Contract width] → Contracts the width of the tree.
**B3)** [Expand height] → Expands the height of the tree.
**B4)** [Contract height] → Contracts the height of the tree.

**C1)** [Expands tree] → Increases tree width and tree height at the same time.
**C2)** [Compress tree] → Decreases tree width and tree height at the same time.

**D)** [Fit tree to window] → If tree is not visualized completely at screen, fits it to window size.

**E1)** [Move right] → Moves tree to the right.
**E2)** [Move left] → Moves tree to the left.
**E3)** [Move down] → Moves the tree down.
**E4)** [Move up] → Moves the tree up.

**Figura 31.** Captura d'un fragment de la guia d'usuari on s'expliquen totes les funcionalitats del Phylograph.