Micromeda: a genome property prediction pipeline and web visualization tool

by

Lee Bergstrand

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Biology

Waterloo, Ontario, Canada, 2020

Author's Declaration

Abstract

Understanding the distribution of biochemical pathways across microorganisms is critical to understanding these organism's evolution, ecology, and industrial applicability. Advances in genome sequencing and pathway databases have made genomically predicting what pathways an organism possesses a common technique. Researchers are moving on to scaling such analyses towards comparing the presence and absence of pathways across multiple microbes from the same environment or lineage. However, performing such analyses at scale is currently bottlenecked by the sheer number of pathways per organism and the lack of powerful tools to facilitate such comparisons.

This thesis presents a new set of tools, called Micromeda, that will assist users in making comparative genomic analyses. Micromeda consists of three core components. These components are Micromeda-Client, which generates interactive heat maps that allow users to perform visual pathway comparisons; Micromeda-Server, which provides data to Micromeda-Client; and Pygenprop, which allows users to perform programmatic comparisons of multiple organism pathways. Micromeda uses the Genome Properties database as its pathway information source. This database is unique from other pathway databases because it maps directly between protein domains and pathway steps. The domains that the database uses are those from the InterPro consortium of protein databases.

With Micromeda, the process of discovering an organism's pathways begins with the domain annotation of an organism's proteins by InterProScan. Afterwards, Pygenprop is used to combine these annotations with information from the Genome Properties database to predict biochemical pathways. This prediction of pathways from domain data results in the creation of a Micromeda file. This novel file type carries both the pathway annotations for multiple organisms and the sequences of proteins that support these annotations. In the context of the Genome Properties database, such pathways are referred to as genome properties, and pathway annotations are referred to as property assignments. The newly created Micromeda file can later be uploaded to Micromeda-Client and Server for heat map-based visualization.

Pygenprop uses object orient programming techniques to represent the Genome Properties database as a series of in-memory objects. These objects are used extensively within Pygenprop's property assignment process and Micromeda as a whole. Pygenprop is written in Python. The library's tight integration with the Python data science ecosystem, which results in it being compatible with many emerging data science and machine learning tools, lays the foundation for the library becoming the backbone of a new generation of automated pathway analysis tools.

Micromeda-Server is a Python web server application that provides data from uploaded Micromeda files to Micromeda-Client. Micromeda-Server makes data accessible via a web application programming interface (API). The API provides clients, such as Micromeda-Client, with access to property assignments and protein sequences found within uploaded Micromeda files. The API can also provide information about individual pathways and the overall structure of the Genome Properties database.

Micromeda-Client is a web client application whose purpose is to provide interactive pathway analysis heat maps to users. These heat maps are used to compare pathways across organisms within a dataset. The interactivity of these heat maps allows for pathway annotations to be aggregated into summaries of multiple pathways or be disaggregated down to a pathway step level. At a step level, users can see differences in the presence of pathways steps. Individual pathways of interest can also be looked up via text search. The heat map interface also allows users to download protein sequences that support individual pathway steps across multiple organisms.

Rather than having to spend time reviewing spreadsheets of pathway annotations or using existing ineffectual pathway annotation visualization software, researchers can now perform their analyses using Micromeda's streamlined and efficient heat maps. For large datasets, Pygenprop can be used to compare the predicted pathways of multiple organisms programmatically. Micromeda has the potential for shaping the way that future researchers perform pathway analysis.

Acknowledgements

I want to thank my supervisors, Dr. Josh Neufeld and Dr. Andrew Doxey, for their constant support and mentorship throughout the project. Feedback from my thesis committee, Dr. Laura Hug and Dr. Jim Wallace, was also much appreciated. I would also like to thank Jackson Tsuji and Dr. Sofie Thijs for their advice and feedback on user interface design. Concerning funding, I want to thank the Province of Ontario and the University of Waterloo for awarding me with a Queen Elizabeth II Graduate Scholarship in Science and Technology (QEII-GSST). I would also like to thank the Nuclear Waste Management Organization (NWMO) and Natural Sciences and Engineering Research Council (NSERC) for providing funding via a Collaborative Research and Development (CRD) grant awarded to my supervisors. Finally, I would like to thank Donya Saghattchi and John Zubak, from my startup company Amplytica Inc., for their consistent patience and support throughout the thesis process.

Figures 1.2, 1.4, 1.5, and 1.6 contain icons made by Freepik (flaticon.com/authors/freepik), Smashicons (flaticon.com/authors/smashicons), icongeek26 (flaticon.com/authors/icongeek26), and Gregor Cresnar (flaticon.com/authors/gregor-cresnar). The icons were acquired from flaticon.com under Flaticon Basic License (see file000.flaticon.com/downloads/license/license.pdf).

Table of Contents

List of Figures

List of Tables

Abbreviations

3NF 3rd normal form 62, 63

AJAX Asynchronous JavaScript and XML 107, 109, 110

antiSMASH antibiotics & Secondary Metabolite Analysis Shell 119

API application programming interface 72, 74–76, 80, 82, 84, 93–96, 113, 116, 125

BacDive the Bacterial Diversity Metadatabase 129

BLAST basic local alignment search tool 20

CDD Conserved Domains Database 16, 18

CDN content delivery network 113, 114

CLI command-line interface 11, 87, 127

CPU central processing unit 14, 18, 25, 78, 79

CSS Cascading Style Sheets version 3 105, 106

CSV comma-separated value 6, 36

DAG directed acyclic graph 19, 25, 26, 35, 36, 38, 39, 43, 77, 84, 86, 100, 102, 103, 105, 106, 118

DBMS database management system 60, 61

DNA Deoxyribonucleic acid 1, 3

DSL domain-specific language 60

E-value expected value 16–18, 51, 52, 54, 55, 65, 82, 84, 85, 94, 105, 116, 117

EBI European Bioinformatics Institute 7, 8, 15, 16, 19, 72, 94, 103, 118, 119

FASTA fast-all 8, 18, 23, 51–56, 58, 59, 73, 79, 82, 84, 85, 87, 94, 105, 111, 112, 121, 126

FMM From Metabolite to Metabolite 94

GB gigabytes 14, 25, 79

GHz gigahertz 14, 25

GO gene ontology 34, 115

HAMAP High-quality Automated and Manual Annotation of Proteins 16

HDF5 Hierarchical Data Format version 5 59

HMM hidden Markov model 16, 17

HPC high-performance computing 68

HTML Hypertext Markup Language version 5 105, 106, 109

HTTP Hypertext Transfer Protocol 75, 79, 80, 82, 85–89, 109, 111, 117

JavaScript European Computer Manufacturers Association (EMCA) script version 6 105–107, 109–111, 115, 118

JSON JavaScript Object Notation 28, 36, 46, 58, 59, 68, 69, 71–73, 82–87, 92, 93, 106, 109, 110, 116, 117

KAAS Kyoto Encyclopedia of Genes and Genomes Automatic Annotation Server 5–7, 94

KEGG Kyoto Encyclopedia of Genes and Genomes 3, 6, 8, 20, 21, 29, 30, 83, 94, 103, 115, 119, 120, 122, 123, 128

KO KEGG Orthology 121–123, 128

MAGs metagenome assembled genomes 1, 88, 112

MAPLE metabolic and physiological potential evaluator 6, 94, 128

MB megabytes 14, 25, 37, 45, 50, 58, 69, 79

MCR module completion ratio 128

N number of replicates 25, 38, 45, 50, 58, 67, 69, 87

NCBI Taxa ID National Center for Biotechnology Information Taxonomy identifier 14, 45, 46, 50, 58, 67, 69, 71, 87

ORM object-relational mapping 64

PaaS Platform as a Service 90

PANTHER protein analysis through evolutionary relationships 16

PCIe peripheral component interconnect express 25

Pfam Protein family 6, 15, 18, 24, 32, 34, 36, 38, 72

PSSM position-specific scoring matrices 17

RAM random access memory 12, 14, 23, 25, 37, 45, 50, 79, 80

RAST rapid annotation using subsystem technology 5–7

REST representational state transfer 82

Rfam non-coding RNA families 24

RPS-BLAST reverse position-specific basic local alignment search tool 17, 20

Scikit scientific Python tool kit 7, 51, 52, 54, 55, 116

SQL structured query language 60, 64

SSD solid-state drive 25

TIGRFAM The Institute for Genomic Research protein family 16, 18, 32, 38

TSV tab-separated value 11, 18, 40, 45, 51–53, 58, 59, 119, 120

TTL	time-to-live 76
UI	user interface 13, 15, 74, 92, 96, 100, 103, 106, 111–113, 118, 119, 125, 126
URI	uniform resource identifier 66
URL	uniform resource locator 75, 82, 85, 109, 111
USD	United States dollar 21
UUID	universally unique identifier 84
uWSGI	web server gateway interface server 89
YAML	YAML Ain't Markup Language 59

Chapter 1

Introduction

Deoxyribonucleic acid (DNA) sequencing allows us to read an organism's genome and, through software, learn more about organism's inherent capabilities without having to study it *in vivo* or *in vitro* [44]. In an environmental microbiology context, such sequenced genomes are used to learn more about microorganisms' metabolic capabilities and possibly shed light on these organism's ecological roles [44]. In an applied context, predictions of such metabolic capabilities are also useful for the selection of what microbes to use in a bioprocess. In a genetic engineering context, genomically derived information can be used to select what metabolic traits to remove from, or move between, organisms [144, 135].

Advances in DNA sequencing technology over the past decade have revolutionized our ability to acquire bacterial and archaeal genomes. For example, with newer sequencing technologies, such as Oxford Nanopore [76], a bacterial genome can be acquired in a matter of hours [98, 29]. Researchers have now moved on to extracting the genomes of unculturable microorganisms from environmental samples using culture-free techniques, such as metagenomic [127] and single-cell [61] sequencing. Over the past few years, the tree of life has been significantly expanded by the metagenome assembled genomes (MAGs) [24] of these unculturable organisms [71, 119]. With microbiologists' inability to gather new genomes rectified, the problem now shifts to interpreting the resulting new wealth of genomic data.

Due to their vast size and information density, the interpretation of genomes is often assisted by software. The tool that was developed as part of the thesis work, Micromeda, allows users to generate data visualizations that help them identify patterns in the presence and absence of biochemical pathways across organisms. Pygenprop, a library built to assist in the development of Micromeda, enables users to perform such comparisons programmatically. A key feature of both Micromeda and Pygenprop is their ability to not only compare the predicted metabolic features of organisms, in terms of biochemical pathways present, but also allow users to access the underlying protein sequences that support these predictions. Details about the information presented by Micromeda and its expected use cases are given within the sections below. Towards the end of this chapter, the data sources used by Micromeda are also discussed. The following chapters will discuss Pygenprop and Micromeda's overall implementation.

1.1 Enzymes and Biochemical Pathways

For many systems, both environmental and industrial processes can be carried out biochemically. From a biological context, such processes are carried out via a series of chemical reactions catalyzed by proteinaceous biological enzymes. Enzymes that facilitate similar chemical reactions often have similar sequences of amino acid residues, structures, and genes that encode them [60, 166].

A protein domain is a conserved subset of a protein's sequence that carries out a specific function and is evolutionarily conserved [131]. These domains are associated with distinct sequence motifs, which are patterns in the protein's amino acid sequence [131]. All enzymes have an active site (en.wikipedia.org/wiki/Active_site), and this active site often has a unique sequence motif. This motif can be used to identify specific enzymes or enzyme families uniquely [131].

A biochemical pathway represents a series of chemical reactions, that when chained together, are beneficial to a cell [108]. Examples of such reactions are the breaking down of a nutrient macromolecule into pieces that cells can use or the synthesis of components of cellular structure [159]. Each reaction step in a pathway is often [87, 148] catalyzed by a specific enzyme whose amino acid sequence, and thus structure and activity, is optimized for the reaction [108, 166, 51]. Thus, there is a mapping between specific enzymes (and the genes encoding them) and chemical reaction steps in biochemical pathways [151]. As a result, by reading the genome, researchers can predict what biochemical pathways an organism may possess [2, 151]. Also, the output from one pathway (*e.g.*, the monomers from the break down of a macro-nutrient) may be the input for a second biochemical pathway that builds cellular structures [159, 142]. Thus, all pathways in a cell are somehow connected and form a network of reactions [159, 142]. This network forms the cell's metabolism and is called its metabolic network [159].

1.2 The State of Pathway Analysis

For many decades scientists have been designing and executing studies to figure out what individual enzymes do and what substrates they can catalyze. The results of such studies are stored in pathway databases. Specifically, what genes encode for what enzymes, what enzymes catalyze what reactions, and what reactions belong to what biochemical pathways. These databases also map how pathways are connected within cells' metabolic networks. Examples of such databases include Kyoto Encyclopedia of Genes and Genomes (KEGG) [80], MetaCyc [85], Genome Properties [132], SEED subsystems [117], Reactome [40], and many others.

As the breadth and depth of the information within pathway databases increases, the information contained within is increasingly being used by automated tools that perform pathway analysis. Such tools help make rapid insights into the capabilities and roles of organisms in a variety of environments. Often this software is released in the form of a toolchain (*i.e.*, a pipeline) where separate bioinformatics software applications are run in series to generate a final output. Such pipelines take an organism's DNA genome sequence, perform *in-silico* transcription and translation (Fig. 1.1), identify enzymes, and identify the pathways that these enzymes support via information contained within pathway databases (Fig. 1.2). These tools perform some or all of the following key steps.

1. Prediction of what genes are present in an organism's genome.
2. Translation of these genes' sequences to protein for reduced redundancy (Fig. 1.1).
3. Taking known enzymatic protein sequences from pathway databases and using these sequences to search the predicted proteins to find those with high sequence similarity. Predicted proteins with high sequence similarity to known enzymes are likely to carry out the same enzymatic function (see Section 1.1). This process is called protein annotation (Fig. 1.1).
4. Using these newly found enzymes to figure out what chemical reactions could be carried out by an organism.
5. Chaining these reactions together to figure out what biochemical pathways are likely to be possessed by the organism (Fig. 1.1). This process is called pathway annotation.
6. Presentation of information about the pathways present and enzymes found in a way that is comprehensible by users.

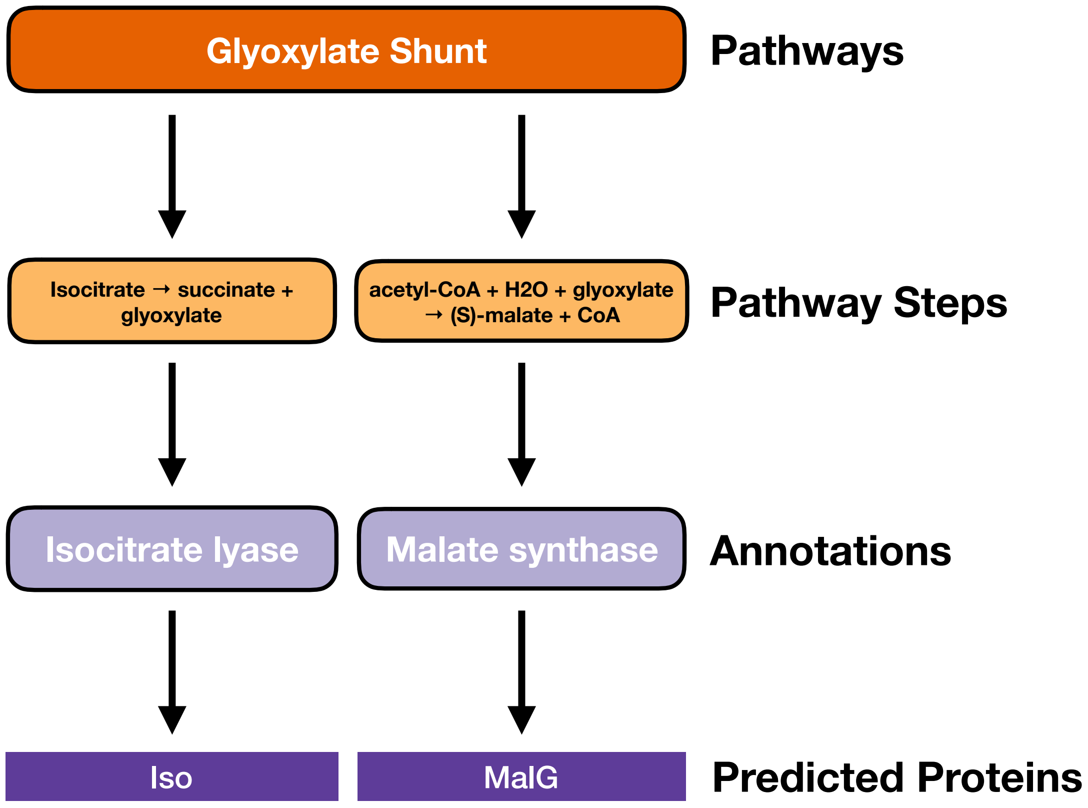


Figure 1.1: **How the glyoxylate shunt can be predicted from the presence of its supporting enzymes.** If a microorganism is to be classified as possessing a glyoxylate shunt, then it should have highly similar proteins to those previously known to carry out the pathway, such as Iso and MalG. Several steps are required to go from an organism's genome sequence to a prediction of its metabolic capabilities. The enzymes that carry out the pathway steps must be identified (*e.g.*, protein annotation). If found, these enzymes would indicate the presence of pathway steps. Finally, if all or many steps are present, then the biochemical pathway can be said to be present.

Users can deploy pathway prediction bioinformatics pipelines in two ways. The pipelines can either be installed on to a user's computer, where genomes can be processed directly, or be deployed on to a web server, where users can upload their genomes for remote processing. Some pipelines only work with pathway data from a specific database. For example, Pathway Tools [84] can only present information about pathways found within the MetaCyc [85] database. Often pipelines are optimized for generating data from the

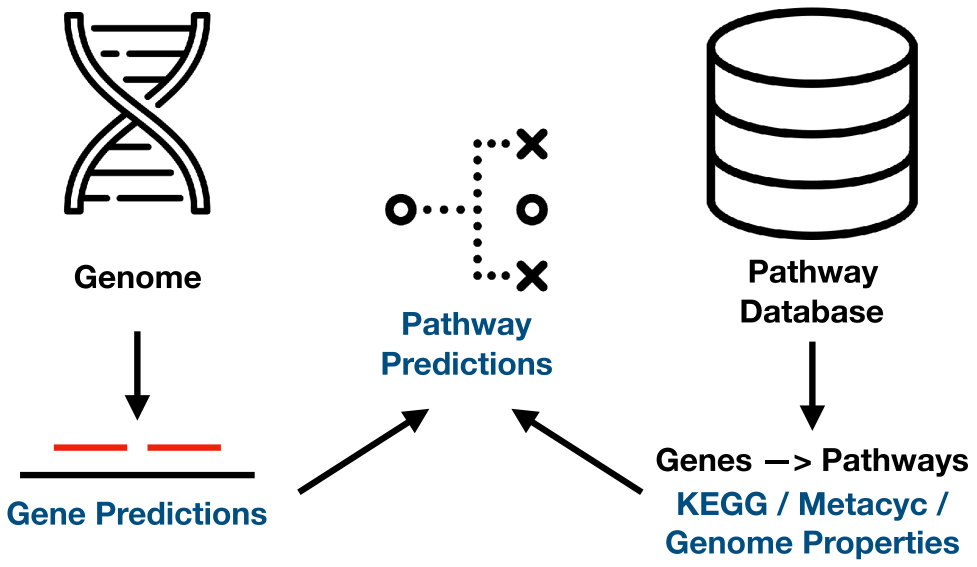


Figure 1.2: **How an organism's biochemical pathways are genomically predicted using information from pathway databases.** Predicting an organism's biochemical pathways involves joining together two distinct datasets. One is a prediction of what genes are possessed by an organism. The other is a database containing the knowledge of what genes are involved in previously known biochemical pathways. When predicted proteins with sufficient homology to the previously cataloged genes are found within an organism's genome, pathway prediction can be made.

genomes of a specific clade on the tree of life. For example, Prokka [137], a pipeline that predicts genes and annotates protein sequences, is only designed to work with the genomes of prokaryotic microbes. Prokka only carries out the first two steps of the key pathway analysis steps listed at the top of this section [137]. Once these proteins are predicted, they can be uploaded to servers such as Kyoto Encyclopedia of Genes and Genomes Automatic Annotation Server (KAAS) [110] for pathway annotation. Several tools can perform all of the pathway analysis steps outlined in the key pathway analysis steps list. For example, rapid annotation using subsystem technology (RAST) [9] can take the upload of whole or partial bacterial genomes, predict the genes of these genomes, and provide users with a report displaying found pathways.

1.3 The Current Bottlenecks of High Throughput Pathway Analysis

Due to the current plethora of tools for genome annotation and pathway determination, identifying pathways for single organisms is becoming a solved problem. Researchers have progressed to comparing the presence and absence of pathways across organisms. Such comparisons are applied in order to, for example, find information about individual organism's ecological roles, evolution, or suitability towards different industrial tasks. For example, the genomes of organisms that are closely related phylogenetically could be compared to determine those that may have lost or gained a pathway or pathway step over time. Alternatively, the pathways possessed by multiple organisms from within the same environment could be compared to shed light on their potential ecological niches. Pathway comparisons could also be used industrially to select organisms to add to bioprocess co-cultures.

Although assigning pathway presence and absence to individual organisms can be done quite rapidly, the comparison of these results across multiple organisms is currently a considerable bottleneck in the area of pathway analysis. Often pathway annotation tools that can process multiple genomes simultaneously present their results in the form of computer spreadsheets (*e.g.*, Microsoft Excel or comma-separated value (CSV) files [138]). For example, metabolic and physiological potential evaluator (MAPLE) [146] and KAAS [110], are KEGG-based pathway annotation platforms that allow users to download CSV files with pathway annotation results. RAST also allows for the download of pathway annotation CSVs. The web interfaces for these annotation systems also present annotation results in a similar table-based format. Such pathway annotation servers sometimes require multiple CSVs to be downloaded, one per annotated organism, which have to be joined to compare pathway annotations across organisms. After the generation of these joined annotation spreadsheets, users must manually scan through the thousands of pathway rows and organism columns to find pathway differences across organisms. Researchers with data science and coding skills may be able to generate custom R [128] or Python [156] scripts that assist them in this scanning task by filtering down these annotation spreadsheets to show only pathways that are different. Additionally, users could write custom scripts for generating data visualizations that accelerate pattern detection.

Libraries have been written to accelerate script development by helping scriptwriters interact with pathway data [167, 125, 165, 34, 75]. The majority of these libraries are written to support either the Protein family (Pfam) [167, 125, 165, 34] and Metacyc [75] databases. There is currently only one such library that supports the Genome Properties database, the Genome Properties library created by European Bioinformatics Institute (EBI) (see github.com/ebi-pf-team/genome-properties/tree/master/code). The majority of pathway bioinformatics libraries are written in Perl [160] or R. In contrast, very few are written in Python [167, 125, 165, 34, 75]. This lack of Python support makes the majority of these libraries unusable for developing software written Python. The Python programming language is increasingly being used in bioinformatics and is widely used by both academia and industry for writing machine learning software [5, 112]. The machine learning algorithms contained in Python libraries, such as scientific Python tool kit (Scikit)-learn [123], have great potential for predicting microbial metabolite production based on gene expression data and biochemical pathway networks [39, 41]. In addition to existing deficiencies in database compatibility and Python compatibility, the majority of pathway bioinformatics libraries also only focus on helping users download data from existing pathway databases [150, 34] or visualize biochemical pathways as graph networks [125], rather than helping users with making pathway comparisons across organisms. This focus on data download and network visualization make these libraries less useful for comparative genomics. Only clusterProfiler [165], an R library, provides cross-organism pathway comparison capability. Thus, there is also a gap for a pathway library that assists coders in making pathway presence and absence comparisons programmatically.

Pygenprop is a new Python pathway analysis library that was built around the Genome Properties database and was generated as part of the thesis work. The library attempts to address capability gaps found in existing pathway bioinformatics libraries by providing users with expanded capability to compare the presence and absence of pathways across organisms. Also, Pygenprop is tightly integrated with emerging Python-based data science and machine learning tools found within the Scipy [78] ecosystem, including Scikit-learn. Pygenprop is discussed at length in Chapter 2. Further details on the differences between it and existing pathway bioinformatics libraries can be found in Section 2.10.

Due to a lack of coding skills among biologists, there is a need for dedicated bioinformatics tools that simplify pathway comparisons across organisms. Software that visualizes the presence and absence of pathways would be of great use for performing such comparisons. Several emerging tools help users visualize pathway annotations from multiple organisms. Both KAAS and RAST present pathway data visualizations on their websites, such as supported pathway diagrams or supported pathway pie charts. However, the figures created by these sites are rather rudimentary and cannot easily be used to compare pathway annotations across organisms. Other annotation systems such as Microscope [154] and EBI's Genome Properties websites allow for comparisons of presence and absence of pathways using heat maps. These heat maps display pathway presence and absence according to rows of pathways by columns of organisms. However, these software programs' implementation, in terms of overall visual idioms used and supporting data presented, is currently lacking. Because of the thousands of pathways that could potentially be possessed by an organism are nested (*i.e.*, some pathways require the presence of other pathways to function), Microscope [154] shows pathway annotation results across a series of heat maps spread across many web pages. For similar reasons, EBI's Genome Properties website limits what pathways can be presented at the same time. As a result, the heat maps generated by Microscope and the EBI's Genome Properties website do not allow one to easily navigate between the presence and absence results of pathways that require the presence of other pathways to function. Also, both sites require users to browse through multiple web pages to learn more details about the pathways visualized in their heat maps, which is both time-consuming and challenging to do when using computers with small screens. FuncTree2 [42], a pathway visualization software designed to present KEGG annotation data, does not use heat maps but instead uses radial trees. Radial trees have diminished screen space utilization when compared to heat maps. There is currently a gap for tools that effectively presents pathway presence and absence data and allows for rapid comparisons in a visual manner.

Having rapid access to the protein sequences of enzymes that catalyze the same reaction steps across multiple organisms would be useful to researchers who want to perform phylogenetic analyses. Such analyses would help determine these enzyme's evolutionary history. Current pathway annotation visualization systems such as FuncTree2 or the Genome Properties website do not allow for the download of protein sequences that support the existence of a pathway step. Microscope does allow for the download of protein sequences. However, the proteins that support a pathway step across multiple organisms must each be downloaded individually from separate web pages. There is a gap for a tool that allows users to identify and rapidly acquire, in batch, the protein sequences that support a pathway step across multiple organisms.

Micromeda, the software presented within this thesis, more effectively conveys pathway presence and absence information than either FuncTree2, Microscope, or the Genome Properties website due to its use of interactive heat maps, which can dynamically show and hide pathway information from multiple organisms. Interactive heat maps allow only a single heat map page to be used and possess superior space utilization and user experience to radial trees due to the heat map's rectangular layout [114]. Micromeda also provides for the download of protein sequences that support pathway steps. Users can download protein sequences identified as supporting the presence of a pathway step, in fast-all (FASTA) format, directly from Micromeda's single-page visualization interface. The resulting downloaded FASTA file contains sequences from multiple organisms within a dataset that are known to possess a pathway step. The ability to download such sequences directly from the visualization page and having all of these sequences with the same file allows for users of Micromeda to have faster access to protein sequence information and reduced workload as compared to Microscope. A more comprehensive and detailed comparison of the visualization component of Micromeda to FuncTree2, Microscope, and the Genome Properties website can be found in Section 4.8.

1.4 The Micromeda Platform

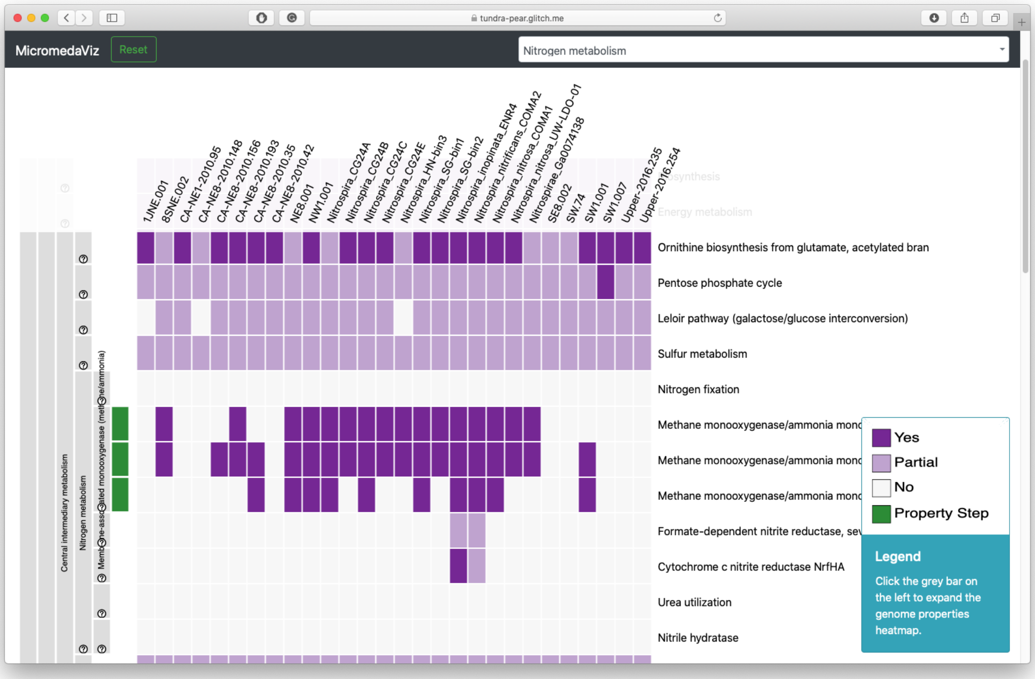

tundra-pear.glitch.me
MicromedaViz
Reset
Nitrogen metabolism
Ornithine biosynthesis from glutamate, acetylated bran
Pentose phosphate cycle
Leloir pathway (galactose/glucose interconversion)
Sulfur metabolism
Nitrogen fixation
Methane monooxygenase/ammonia mono
Methane monooxygenase/ammonia mono
Methane monooxygenase/ammonia mono
Formate-dependent nitrite reductase, sev
Cytochrome c nitrite reductase NrfHA
Urea utilization
Nitrile hydratase
Yes
Partial
No
Property Step
Legend
Click the grey bar on the left to expand the genome properties heatmap.
Central intermediary metabolism
Nitrogen metabolism

Figure 1.3: **Web-browser window containing Micromeda's heat map visualization and interface.** This heat map consists of pathway rows by organism columns and allows the comparison of pathway presence and absence across organisms. All other components of Micromeda were built to support this interface by providing it with data. Further explanation of the interface's design can be found in Chapter 4.

The bioinformatics system presented within this thesis, called Micromeda, is designed to address current gaps in the researcher's ability to compare pathway presence and absence across organisms. The platform generates these comparisons without losing information about the protein sequences that support the pathways' existence. The output of the platform is an interactive heat map that displays rows of pathways by columns of organisms (Fig. 1.3). Heat map cells are coloured by the level of support for a pathway's existence in each genome (Fig. 1.3). This data visualization is displayed within a user's web browser. As discussed in Chapter 4, this heat map is interactive, and users can tailor it to only display presence and absence for specific pathways or pathway steps. A software stack (see en.wikipedia.org/wiki/Solution_stack) consisting of several components, some of which were developed as part of the thesis project, is used to generate data for the visualizations that Micromeda presents. This stack is outlined in the list below.

- A program that generates protein sequences from predicted genes found within an organism's genome. For example, in the case of prokaryotic genomes, an existing tool such as **Prodigal** [74] would be used.
- A pre-existing sequence search program for scanning for identifying markers within the sequences of an organism's predicted proteins. These markers are used to identify enzymes that support the existence of a pathway. The search program chosen was **InterProScan5**, whose output data are used by the Genome Properties Database. An overview of InterProScan5 [79] and its methodology can be found in Subsection 1.4.2.
- A pathway database that maps between predicted protein sequences derived from an organism's genome and biochemical pathway steps. The database chosen was the **Genome Properties** database [132]. A short review of this database and the reason for its selection can be found in Subsection 1.4.2. This database is pre-existing and was not made as part of the thesis work.
- A software library that supports the generation of pathway annotations, rapid programmatic comparisons between organism pathway annotations and the generation of Micromeda files. The library is compatible with many emerging machine learning tools and opens up new avenues to their application to pathway analysis. This library is called **Pygenprop** and is discussed in Chapter 2.
- A file format that allows users to easily transfer an assessment of what pathways are possessed by multiple organisms and the protein sequences used to support this assessment. These files, called **Micromeda files**, are the files uploaded to Micromeda-Server, which is discussed in the bullet point below, and use a custom format that

is discussed in Section 2.6. The format allows for the storage of the pathway and sequence information in the most compact way possible.

- A server web application that runs on a remote computer system and in support of a client application. This server application accepts **Micromeda file** uploads from the client and provides the client with easy access to the data held within the file. This component is called **Micromeda-Server** and is discussed in Chapter 3.
- A client web application that runs in the user's browser. This web application allows users to upload **Micromeda files** to **Micromeda-Server**. This application draws pathway heat maps based on this uploaded data (Fig. 1.3). These heat maps allow users to make comparisons across pathways and organisms. This component is called **Micromeda-Client** and is detailed in Chapter 4. Links to a demonstration of the client interface can be found in Section 4.5.

The Micromeda platform can be subdivided into two core components: a toolchain for generating Micromeda files (labelled Micromeda Annotator in Fig. 1.5) and a web application for visualizing the data these files contain (labelled Micromeda Visualizer in Fig. 1.5). The individual steps for generating Micromeda files can be done manually using only three command-line interface (CLI) tools (see Fig. 1.4, Fig. 1.6, and en.wikipedia.org/wiki/Command-line_interface). For example, with prokaryotic genomes, Prodigal could be used to predict protein sequences from an organism's genome sequence, and InterProScan5 could be used to scan these proteins to identify enzymes that support the existence of specific pathways (Fig. 1.6). Pygenprop contains a CLI (see github.com /Micromeda/pygenprop#commandlineinterfacecli) that is automatically installed when the library is installed on a user's computer. This CLI allows users to generate Micromeda files from the output domain annotation tab-separated value (TSV) files generated by InterProScan. A tutorial that covers how to generate Micromeda files can be found at github.com/Micromeda/micromeda-workflow.

1.4.1 Micromeda Software Architecture Overview

Micromeda follows a client-server web architecture [145] (see en.wikipedia.org/wiki/Client-server_model and Section 3.1). Users interact with Micromeda-Client via their web browser, and this client allows them to upload Micromeda files to Micromeda-Server. Micromeda files contain all the information that the client requires to generate a pathway heat map. These files store pathway annotations, InterProScan5 output data, and supporting protein

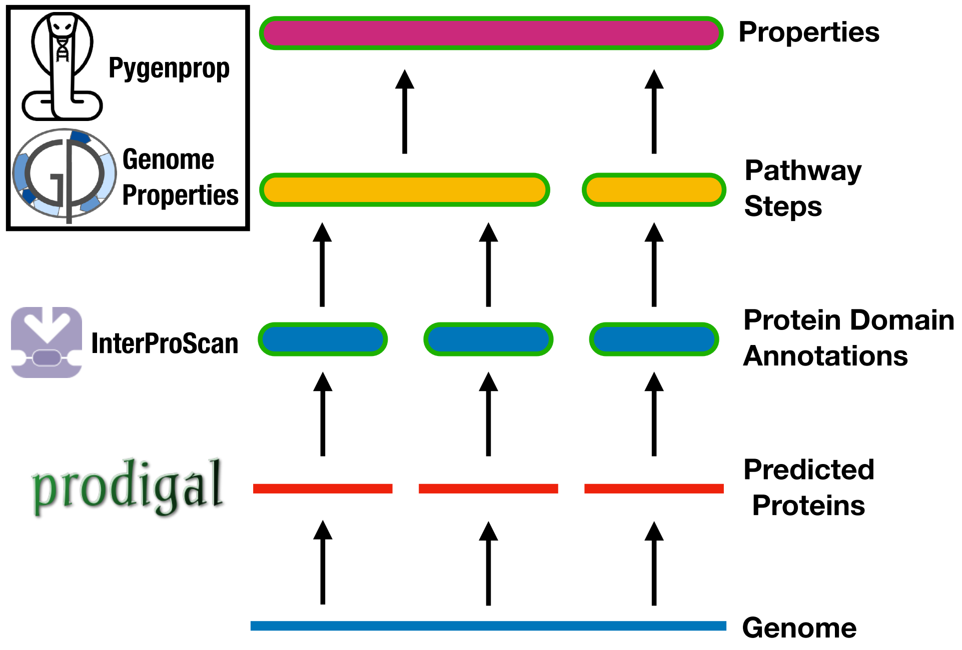


Figure 1.4: **Steps performed and software tools used by Micromeda to predict the genome properties of an organism.** For prokaryotes, proteins must first be predicted via Prodigal. These proteins are then scanned using InterProScan5. The results of InterProScan are then combined with the Genome Properties database to predict pathways steps. These predictions are carried out by Pygenprop, which also predicts the overall presence and absence of pathways.

sequences for multiple organisms (see Section 2.6). Having all these datasets in a single file simplifies the data upload process as only one file has to be uploaded by the user per heat map drawn. After upload, the contents of the uploaded file are stored temporarily in random access memory (RAM) on the computer used to host Micromeda-Server (see Section 3.2). Micromeda-Client will ask Micromeda-Server for data from this file as the client draws a heat map or responds to user activity (see Section 4.4). Multiple users can interact with Micromeda-Server and Client simultaneously[1].

[1]The number of simultaneous users that Micromeda-Server can support almost entirely depends on the server computer systems the software is run on and the deployment strategy used (see Section 3.6). Information about the scalability of Micromeda-Server, when running on a single computer, can be found

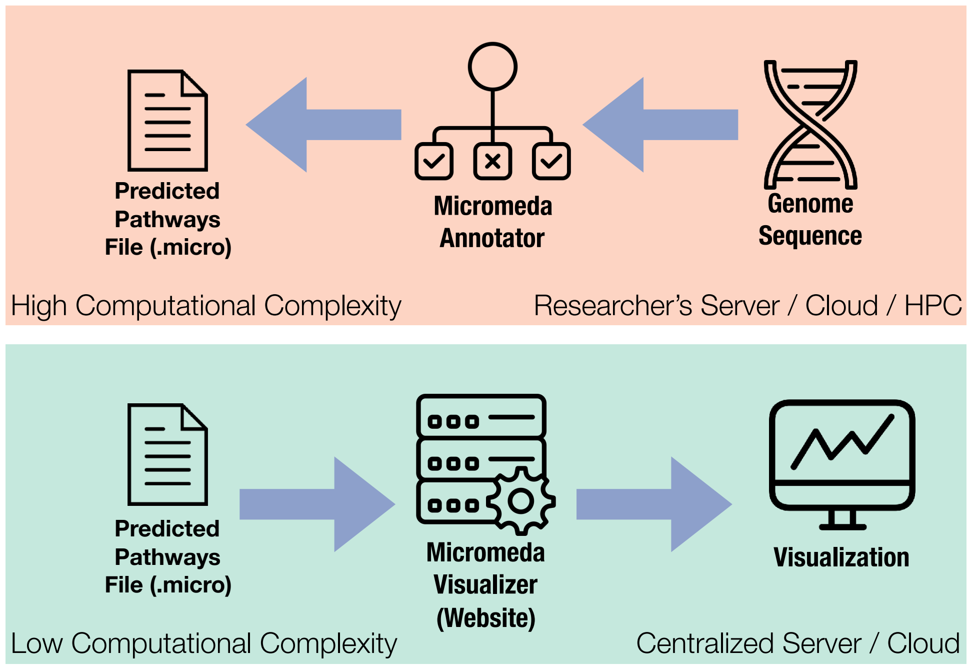

Predicted Pathways File (.micro)
Micromeda Annotator
Genome Sequence
High Computational Complexity
Researcher's Server / Cloud / HPC
Predicted Pathways File (.micro)
Micromeda Visualizer (Website)
Visualization
Low Computational Complexity
Centralized Server / Cloud

Figure 1.5: **Two core steps that users must perform to generate Genome Properties visualizations using Micromeda.** A local computer system must be used to execute a data generation step that creates a Micromeda file. Users can then upload this file to a second remote computer system that generates heat map visualizations. The most computationally complex step, Micromeda file generation, is not performed on the same server that generates the data visualizations.

Micromeda's user interface (UI) runs inside a user's web browser. The reasoning for using this approach, in contrast to building Micromeda as a native desktop application, is the approach's relative ease of deployment. End users only need to open the web address for the client to be loaded into their web browser and run. Because the application is web browser-based, it will work on any operating system with a modern web browser, including mobile devices such as tablet computers and cell phones.

The reason that Micromeda files exist is that they allow the rapid transfer of pathway annotation datasets that, in turn, allow for a separation of data generation and visualization. This separation is essential because there are vast computational complexity differences between generating pathway annotations and visualizing them. Micromeda's

toward the end of Section 3.2

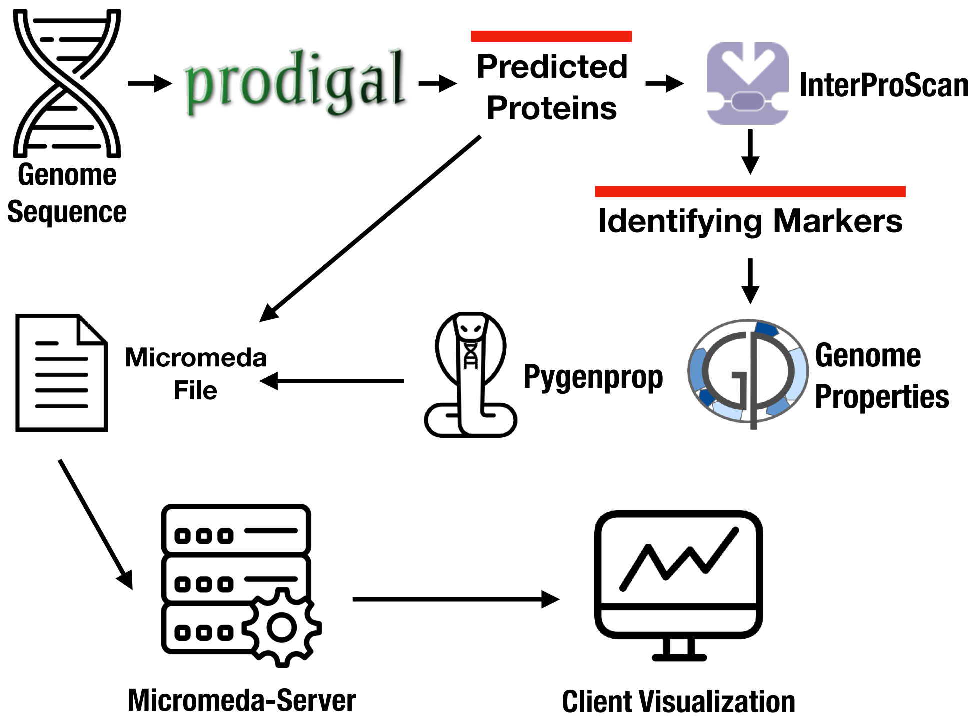


Figure 1.6: **How Micromeda files are built from InterProScan annotations of an organism's predicted proteins.** These files possess not only pathway annotations but also the protein sequences that support these annotations. Thus, Micromeda files allow for the transfer of complete pathway analysis datasets. Such files can be uploaded to a remote server for visualization.

pathway prediction method involves identifying specific enzymes by running InterProScan5 on the set of all predicted proteins of an organism. The algorithms used by InterProScan5 are very computationally complex. It takes on the order of two hours to scan through the 4313 proteins of *E. coli* K12 (National Center for Biotechnology Information Taxonomy identifier (NCBI Taxa ID): 1010810) using 100% of all the central processing unit (CPU) cores of a 16 core server[2].

In contrast, Micromeda can render a pathway heat map for over forty organisms in less than a second. Thus, if one wanted to have a web application that both computes and visualizes pathway annotations for uploaded genome sequences, then this application

[2]InterProScan5 was tested on a server with two Intel E5310 (4 CPU cores/4 threads/8 megabytes (MB) cache/1.60 gigahertz (GHz) clock speed) processors and 16 gigabytes (GB) of RAM.

would require the support of an extensive and well-maintained hardware infrastructure. Developing the code to build, maintain, and sustain such a system draws away from the core goal of the Micromeda platform, which was to design a tool that helps users visualize pathway differences across organisms. Hence, for Micromeda, I chose to have users generate pathway annotations locally, using InterProScan5 and other tools, and have them upload these files to a remote web server for visualization (Fig. 1.5). This design decision significantly reduces the hardware requirements for those who want to host Micromeda-Server and Client. The decision also reduces the overall design complexity of Micromeda-Client and Server and allows future development to focus on creating better and more feature-rich versions of Micromeda's UI and visualizations.

1.4.2 An Overview of the Data Sources Used by Micromeda: the Genome Properties Database, InterPro, and InterProScan5

The architecture and implementation of Micromeda's components, such as Pygenprop and Micromeda-Client, are tied closely to the structure of the Genome Properties database and the data presented by InterProScan and the InterPro consortium. Thus, it is pertinent to detail these tools before moving on to later chapters. As mentioned in Section 1.4, Micromeda uses Pygenprop to predict the biochemical pathways possessed by an organism. These predictions are made by providing Pygenprop with a copy of the Genome Properties database and the output of running InterProScan5 on the organism's predicted proteins. In addition to discussing the structure of the Genome Properties database, this subsection will also discuss how domain annotations produced by InterProScan5 are combined with mappings from the Genome Properties database to generate pathway predictions.

The InterPro Consortium of Protein Databases

Because the enzymes that carry out elements of metabolism in different organisms are highly similar and often evolutionarily related, it is useful to categorize proteins into groups that carry out a single function or share specific domains. Protein databases record the function of these groups of proteins and their domains and also store copies of these protein's sequences. In the past, many of these databases were maintained, operated, and funded independently. However, in the past two decades, many of the operators of these protein databases have banded together to form the InterPro consortium [6, 72, 73], which is headed by the EBI[37, 56]. As of 2019, the InterPro consortium of protein databases manages fourteen member databases [56, 73]. Examples of member databases include Pfam

[10], The Institute for Genomic Research protein family (TIGRFAM) [66], protein analysis through evolutionary relationships (PANTHER) [107], the Conserved Domains Database (CDD) [101], High-quality Automated and Manual Annotation of Proteins (HAMAP) [96], PRINTS [8] and many others. Hosting of many of these member databases has been moved to EBI maintained servers. In addition to the member databases, the consortium also provides the InterPro database [72, 56], which is a meta-database that allows for mapping between identical records for the same protein or domain groups across member databases. Each protein or domain catalogued is given a global InterPro identifier (*e.g.*, IPRXXXX) that is mapped to multiple identifiers for the same protein or domain within member databases (*e.g.*, PFXXXX or TFXXXX) [72, 56].

An import feature of protein databases is the ability to take the sequence of a novel protein (*i.e.*, a protein that is not currently in the database) and predict the placement and function of the protein's domains, which is a process called domain annotation. The search algorithms used by member databases of the consortium compare novel proteins to computational models (*i.e.*, a profile) that represents the sequence diversity (*i.e.*, not all domains from different proteins and organisms have the same sequence) of each domain in the database. If the novel protein possesses a region of high similarity to a model, then it is likely that the novel protein possesses the domain that the model represents. Because the member databases of the InterPro consortium were developed independently, the methods they use for sequence search also vary (Fig. 1.7). The majority of these databases use HMMER (Fig. 1.7) [48], which compares the sequence of a novel protein to a hidden Markov model (HMM) [45].

If a portion of an organism's protein and a model are highly similar in sequence, they form a match. The quality of this match can be quantified by metrics such as the expected value (E-value) score. This E-value score captures how likely it is that the match is real (*i.e.*, the organism's protein contains the domain) given the chance of finding an equivalent match randomly in other proteins. Another metric for match quality is the length of the region of high sequence similarity, the alignment, shared between the protein and the database domain. If it is determined that a match is of high quality, the aligned region of the organism's protein can be assigned the same name and function as the domain in the database. As discussed in Chapter 2, Pygenprop can generate Micromeda files that store such match information.

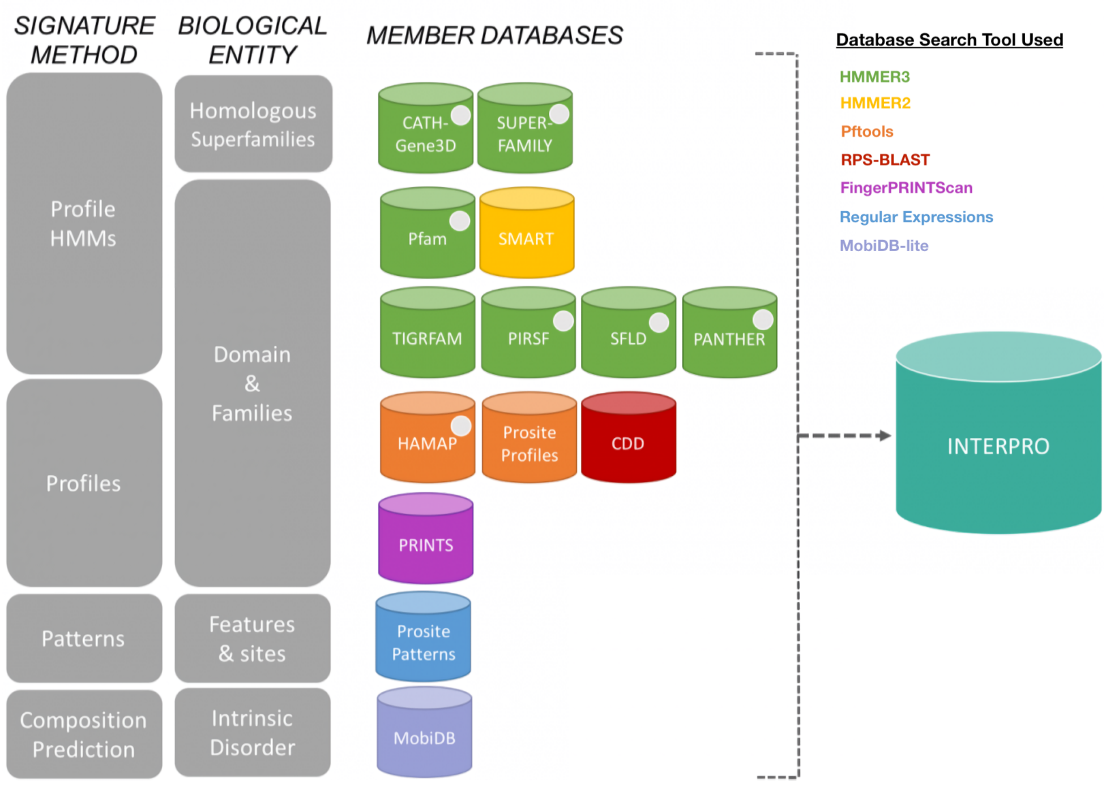


Figure 1.7: **Member databases of the InterPro consortium and the model-based sequence search techniques that these databases use to classify novel protein sequences.** These search techniques require that the databases store models that represent multiple proteins or proteins domains. Examples of such models include HMMs and position-specific scoring matrices (PSSM). Examples of software that use such models are HMMER [48] or reverse position-specific basic local alignment search tool (RPS-BLAST) [104]. This figure is modified from the one found at https://www.ebi.ac.uk/training/online/course/introduction-protein-classification-ebi/protein-classification-resources-ebi-interpro.

Each member database has algorithms for filtering out false positive matches, which are those that occur between a model and a region of a protein that does not carry out the same function as the domain that the model represents. The member databases perform this filtering by implementing unique cut-off values, such as minimum E-value scores or alignment lengths, that can be used to filter out matches that may be spurious. The cut-off values can be made unique to each model.

InterProScan

InterPro consortium created a tool, InterProScan, that allows users to compare a novel protein sequence to all domain models found within InterPro member databases. The tool is a software wrapper for and execution engine of the model-based sequence search techniques (*e.g.*, HMMER [48]) used by all member databases of the InterPro consortium. InterProScan also implements the false positive filtering techniques developed for each member database. The latest version of the software, InterProScan5, follows a Master/Worker architecture (see en.wikipedia.org/wiki/Master/slave_(technology)) where a master process schedules jobs for many worker processes. Depending on the number of CPU cores of the computer running InterProScan5, tens to hundreds of models can be run against a novel protein simultaneously. Due to its architecture, InterProScan5 can also run jobs across a compute cluster. Due to this scalability, InterProScan is capable of domain annotating every protein of a microorganism in only a few hours, depending on the computer the software is run on and the organism's genome size. InterProScan takes a FASTA file [121] containing an organism's predicted proteins as input and writes domain annotations and match data to TSV files. The match data includes supporting information such as E-value scores for matches and predicted domain start and stop points on the organism's annotated protein.

The Genome Properties Database

The backbone of Micromeda is the Genome Properties database [66]. The Genome Properties database takes advantage of the identifiability provided by protein domains to map from combinations of protein domains to enzymes that carry out pathway steps [132]. This mapping is in contrast to most other pathway databases, which map from whole proteins to biochemical pathways. If all the required domains for a pathway step are present in an organism's proteins, then the step is considered present. The domains used as markers by the Genome Properties database are those catalogued within the InterPro consortium of protein databases [6, 132].

The database goes beyond metabolism to include other organism capabilities such as cell motility (*e.g.*, the presence of flagella and chemotaxis) and even microbial viral immunity mechanisms such a CRISPR/Cas9 [70]. Within the database each capability is called a genome property. Multiple steps support each property, and each of these is supported by evidence that can be found from an organism's genome such as the presence of InterPro domains (*e.g.*, Pfam [10], TIGRFAM [65], or CDD [101] domains) in predicted proteins. Several genome properties are required as lines of evidence by others, and thus the database forms a rooted directed acyclic graph (DAG) (see en.wikipedia.org/wiki/Directed_acyclic_graph) of connected properties. There are five types of genome properties: pathways, metapathways, systems, guilds, and categories (see genome-properties.readthedocs.ioen/latest/flatfile.html#genome-property-types for details). The initial release of EBI Genome Properties, version 1.0 (January 9, 2018), had 584 properties and 3083 steps. The latest public release of the Genome Properties database, version 2.0 (August 30, 2018), contains 1296 properties and 6525 steps [132]. One of the core goals of the latest release was to expand the database beyond prokaryotic properties to include properties that are only possessed by eukaryotes or are shared between prokaryotes and eukaryotes. Version 2.0 has also incorporated pathways from MetaCyc [85]. The next version of the database, version 3.0, is still in active development and is planned to be released in spring 2020 (see github.com/ebi-pf-team/genome-properties/issues/30#issuecomment-557090961). Version 3.0 will contain fixes to existing properties, the addition of new properties[3] and code fixes to the Genome Properties Perl library (see github.com/ebi-pf-team/genome-properties/tree/master/code). Section 2.2 discusses how Pygenprop represents the structure of the Genome Properties database in memory.

If a specified number of required steps are found within the domain annotations of an organism's proteins, then the organism is understood to posses a specific genome property. The process of predicting what properties are possessed by an organism is called property assignment. To assign properties to an organism, InterProScan is first used to domain annotate all of its predicted proteins (*e.g.*, those produced by Prodigal), and these annotations are then combined with information from the Genome Properties database to assess what properties are supported. Each property in the database is assigned YES, PARTIAL, and NO support. With Micromeda, Pygenprop is used to carry out these assessments. The Genome Properties database is provided to Pygenprop in the form of a release file, whose contents are detailed in the Subsection 2.1.1. Subsection 2.3.2 details the exact algorithms used for generating assignments.

Though the Genome Properties database does catalogue properties that are not biochemical pathways (*e.g.*, complex cellular components such as flagella), the vocabulary used to describe the database is similar to the lexicon used to describe pathway components. For example, entities that support the existence of individual genome properties are called steps, not subcomponents or substructures. These naming conventions are still used even when a property represents a cellular structure or process rather than a pathway. For simplicity and consistency of language across the thesis, biochemical pathway-based

[3]There is no ceiling to the number of properties that could be added to the Genome Properties database. As long as domains catalogued by the InterPro consortium can identify the proteins involved in a pathway or structure, then a property representing the pathway or structure can be generated.

verbiage and examples are used throughout the thesis when providing example usages of Micromeda or describing the reasoning behind specific design decisions. However, it should be noted that Micromeda's analysis abilities are not strictly limited to pathway analysis. The tool can analyze any property in the Genome Properties database, including those describing non-metabolic capabilities.

Reasoning for Micromeda's Use of Genome Properties and InterProScan

During the development of Micromeda, there were four reasons for selecting the Genome Properties database over other databases and InterProScan over other search tools. These reasons are explained below.

One of the primary reasons for using the Genome Properties database and InterProScan is that they allow pathway annotations to be built from domains identified by model-based search tools such as HMMER [48] or RPS-BLAST [104]. When compared to non-position specific basic local alignment search tool (BLAST)-based [4] search methods, which are commonly used by other pathway annotation systems, these model-based tools are better at detecting enzymes whose sequences are phylogenetically divergent from those previously known [48]. This improved detection capability provides Micromeda with an advantage when it is applied to the genomes of previously unstudied organisms.

The Genome Properties database being domain-based also provides Micromeda with another advantage. Because the database is based on domains rather than whole proteins, Micromeda can detect the presence of enzymes that are split across multiple genes or have fused with other proteins. In a recent study that focused on confirming genome-predicted amino-acid auxotrophy across a variety of bacteria, the authors found that many predicted instances of auxotrophy were misannotations [126]. Many of these misannotations were the result of either gene fusions or the enzyme being split across multiple genes [126]. Traditional whole protein sequence-based detection methods such as BLAST [4], which are typically used by pathway annotation pipelines based on databases such as KEGG, were shown to miss these enzymes [126]. For example, if previous forms of a protein were all found to be encoded by a single gene, such whole sequence methods were shown to filter out versions of the enzyme that are split across genes due to inadequate alignment lengths for matches to individual subunits [126]. In contrast, InterProScan will detect all required domains, whether the proteins are on one gene or multiple.

Another advantage of Genome Properties is that it is freely available under an open-source licence and hosted on GitHub [132]. In contrast, two of the most prominent pathway databases, KEGG and BioCyc [82], have been commercialized. KEGG's website is free for academic use in terms of using the data held within for hypothesis testing (see kegg.jp/kegg/legal.html). However, bulk download of the entire database, as would be required for a pathway annotation system such as Micromeda, has a licensing fee of $2000 United States dollar (USD) per year (as of 2019 and see bioinformatics.jp/docs/subscription_fees.pdf). This fee increases to $5000 USD per year if a user "provide[s] any outside service using... KEGG data" (as of 2019 and see bioinformatics.jp/docs/subscription_fees.pdf). Thus, if Micromeda were built around the KEGG database, users creating Micromeda files would be required to pay $2000 USD per year, and any user hosting Micromeda server, as a public service, would be required to pay $5000 USD per year. BioCyc follows a similar paid scheme (metacyc.org/download.shtml).

The Genome Properties database has also been shown to have comparable coverage, in terms of organism proteins used to support the existence of pathways, to databases such a KEGG and Seed Subsystems [132]. This coverage was consistent across a variety of microbes from distant taxonomic clades [132]. Thus, there is no high-level database completeness disadvantage if Micromeda uses the Genome Properties database.

1.5 Summary

When Micromeda performs pathway annotation, it does so based on the presence of InterPro domains found within organisms' predicted proteomes. These domains are detected using model-based sequence search software that is orchestrated by InterProScan5. The contents of the Genome Properties database is used to map from the presence of domains to the presence of genome property steps. The presence of steps is used to infer YES, PARTIAL, or NO support for individual genome properties. The model-based search techniques used by InterProScan allow Micromeda to provide highly accurate and comprehensive results with no licensing fees.

Micromeda allows users to visualize the differences in pathway presence and absence across organisms. The tool generates visualizations based on the data contained within uploaded Micromeda files. Pygenprop is a software library that can not only produce Micromeda files but also make programmatic comparisons of pathway presence and absence across organisms. Potential improvements to individual components of Micromeda are highlighted in the summary section of each of their chapters. Potential improvements that would require modification of multiple components are highlighted in Chapter 5. As discussed in Chapter 5, Micromeda breaks new ground in both features and implementation and will increase both the speed and ease at which researchers perform pathway analysis.

Chapter 2

Development of a Python library for Programmatic Exploration and Comparison of Organism Genome Properties

During the development of Micromeda's server component, it was recognized that it would be useful to have a software library to assist with programmatic usage of the Genome Properties database. This library would be used to access the database's information, assign levels of support to individual properties and compare these assignments among organisms. A vital component of the thesis work was the development of this library, called Pygenprop [16]. Pygenprop is a Python library that provides an object-oriented framework [21] for representing the Genome Properties database and assessing the property assignments of multiple organisms. The library is deeply integrated with the Python data science software stack [78] through its representation of property assignments as pandas DataFrames [105]. Pygenprop is also interoperable with modern machine learning frameworks, opening up new use cases for pathway annotation data. Pygenprop additionally provides automation features for tracking the data used to generate property assignments and storing assignments for later use. This chapter will review the structure and function of Pygenprop's core components and the design decisions implemented during the library's creation. Pygenprop was recently published in Oxford Bioinformatics [16][1]. Source code and installation instructions for Pygenprop are located at github.com/Micromeda/pygenprop.

[1]Chapter 2 was modified from [16]. The description of the Pygenprop's internals was greatly expanded.

2.1 Parsing the Genome Properties Database

Before addressing the object-oriented programming aspects of Pygenprop, it is important to first discuss how the library imports data. In all use cases, Pygenprop requires the information found within the Genome Properties database and before the library can use this information, it must first be loaded into a computer's main memory (*i.e.*, RAM). This parsing of the Genome Properties database is the job of Pygenprop's Genome Properties database parser.

The Genome Properties database consists of a series of flat files (see Subsection 2.1.1). The database parser module loads these files from disk and encodes the information contained within them in a tree-like data structure. The layout of this data structure is detailed in the next section. A secondary goal of the parser is to build connections between individual properties as the database consists of a series of flat files, whose individual property records are not indexed nor connected.

2.1.1 Overview of the Genome Properties Flat File Database and Associated File Formats

The Genome Properties database (as of version 2.0) consists of a series of flat files that are hosted inside a GitHub repository (see github.com/ebi-pf-team/genome-properties). Information about individual properties is stored in the repository's **data** folder, and within this folder each property is assigned a second internal folder containing three files:

- A **DESC** file, that contains information about the property
- A **status** file that contains information on whether the property is public or has been manually curated
- A **FASTA** [121] file containing representative protein sequences that are known to carry out the steps of the property

The **data** folder contains information about both public and non-public genome properties.

In addition to the per-property folders, there is also a Genome Properties release file located in the **flatfiles** folder of the repository that also contains Genome Properties information. Specifically, this file, called **genomeProperties.txt**, is a concatenation of all the **DESC** files of all public properties. This file is created with each new release of the Genome Properties database on GitHub. Below is simplified view of the folder structure used by the Genome Properties GitHub repository.

```
├── code/ - # The Genome Properties Perl library
├── data/ - # Data about both public and private properties
│ ├── GenProp0001/
│ │ ├── DESC - # Detailed property information
│ │ ├── FASTA - # Sequences of proteins that carry out steps
│ │ └── status - # Public and manual curation status
│ └── GenProp0002/
│  ├── DESC
│  ├── FASTA
│  └── status
└── flatfiles/
 └── genomeProperties.txt
```

Pygenprop's Genome Properties database parser (see Section 2.1) is capable of parsing both single **DESC** files of individual properties and the concatenated **genomeProperties.txt** release file. The format of **DESC** files is very similar to the Stockholm sequence alignment format used by both the Pfam and non-coding RNA families (Rfam) databases [10, 62]. Like these file types, **DESC** files consist of a series of key-value pairs. Because these files use different keys than Stockholm, a custom parser had to be developed. An example **DESC** file can be found at raw.githubusercontent.com/ebi-pf-team/genome-properties/master/data/GenProp0145/DESC. A table of keys used in genome properties **DESC** files can be found at genome-properties.readthedocs.io/en/latest/flatfile .html#desc-file.

2.1.2 Parser Implementation and Performance

The database parser reads both **DESC** and **genomeProperties.txt** files one line at a time to decrease memory usage. While loading line by line, lines are split into key-value pairs and these pairs are loaded into a Python list. Once all keys for a single property are found, the key-value pairs are used to create a series of in-memory Python objects representing the property. In the case of **genomeProperties.txt** release files, Pygenprop repeats this process for all property records in the file, placing each in a Python list. Once parsing is completed, this list is used to create a **GenomePropertiesTree** object[2] that represents the database's rooted DAG structure. The parser then returns this final object.

Pygenprop's Genome Properties flat file parser processes a single **DESC** file in 415 µs ±6 µs (number of replicates (N) = 80) on average and the latest release of the entire Genome Properties database (the **genomeProperties.txt** of release 2.0) in 242.0 ms ±5.0 ms (N = 80)[3][4].

2.2 Development of an Object-Oriented Class Framework for the Representation of the Genome Properties Database

As discussed in the previous chapter, the Genome Properties database consists of a series of interdependent genome properties representing both metabolic and structural features of cells. Several properties are used as evidence of others, forming parent-child relationships between properties and an overall rooted DAG structure across the database. Pygenprop follows an object-oriented programming paradigm [21] (see en.wikipedia.org/wiki/Object-oriented_programming) and thus after parsing the Genome Properties database, Pygenprop represents the database as a series of in-memory objects (see Table 2.1 and Fig. 2.1) that contain information about individual properties (Fig. 2.2). These objects are connected in a linked list fashion [115] (see en.wikipedia.org/wiki/Linked_list), where objects point to each other. These connections are doubly linked, which enables climbing both up and down the DAG and between GenomeProperty, Step, FunctionalElement, and Evidence objects (Fig. 2.2 and Fig. 2.1). Methods and attributes of these objects can be used in software applications or explored interactively in Jupyter Notebooks [88]. The below subsections detail Pygenprop's Genome Properties database classes and how these classes can be used.

[2]Note that capital letters are used to differentiate objects and classes from the concepts they represent. For example **GenomeProperty** objects represent individual genome properties and **Step** objects represent individual property steps.

[3]Note that for the remainder of the thesis, unless otherwise noted, all performance tests were done on a Macbook Pro (model A1502), with an Intel Core i5-4258U 2.4 GHz processor (2 CPU cores, 4 threads, 3 MB L3 cache), 16 GB of RAM and a 256 GB peripheral component interconnect express (PCIe) solid-state drive (SSD).

[4]Note that for the remainder of the thesis, unless otherwise noted, Python function execution time was recorded with Python's built-in timeit package (docs.python.org/3.6/library/timeit.html). Python version 3.6 was used for all testing. Variances displayed are standard deviations.

Table 2.1: Summary of the object types used to represent the Genome Properties database.

Object Type	**What the Object Represents**
Tree	Genome Properties DAG
Genome Property	Single genome property
Literature Reference	Article discussing a genome property
Database Reference	Record in an external pathway database that is equivalent to a genome property
Step	Step supporting the existence of a genome property
Functional Element	Functional element supporting the existence of a step
Evidence	Evidence supporting the existence of a functional element

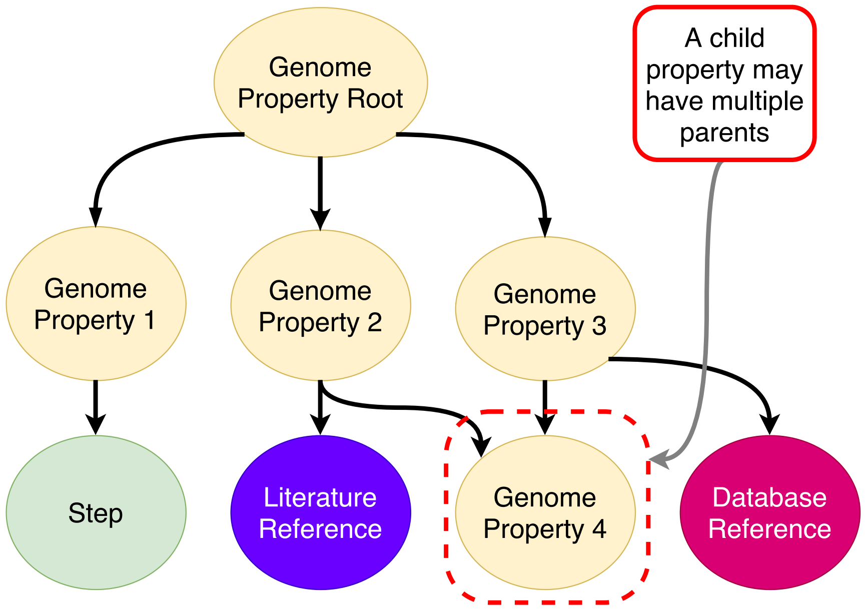


Figure 2.1: **In-memory objects that Pygenprop uses to represent the Genome Properties DAG.** These objects are interconnected. Parent property objects are connected to child property objects. Database references, literature references, and property steps are also present in the object model and are connected as children of individual property objects. The objects presented by Pygenprop can be used to build software that queries the Genome Properties database. Figure is from [16].

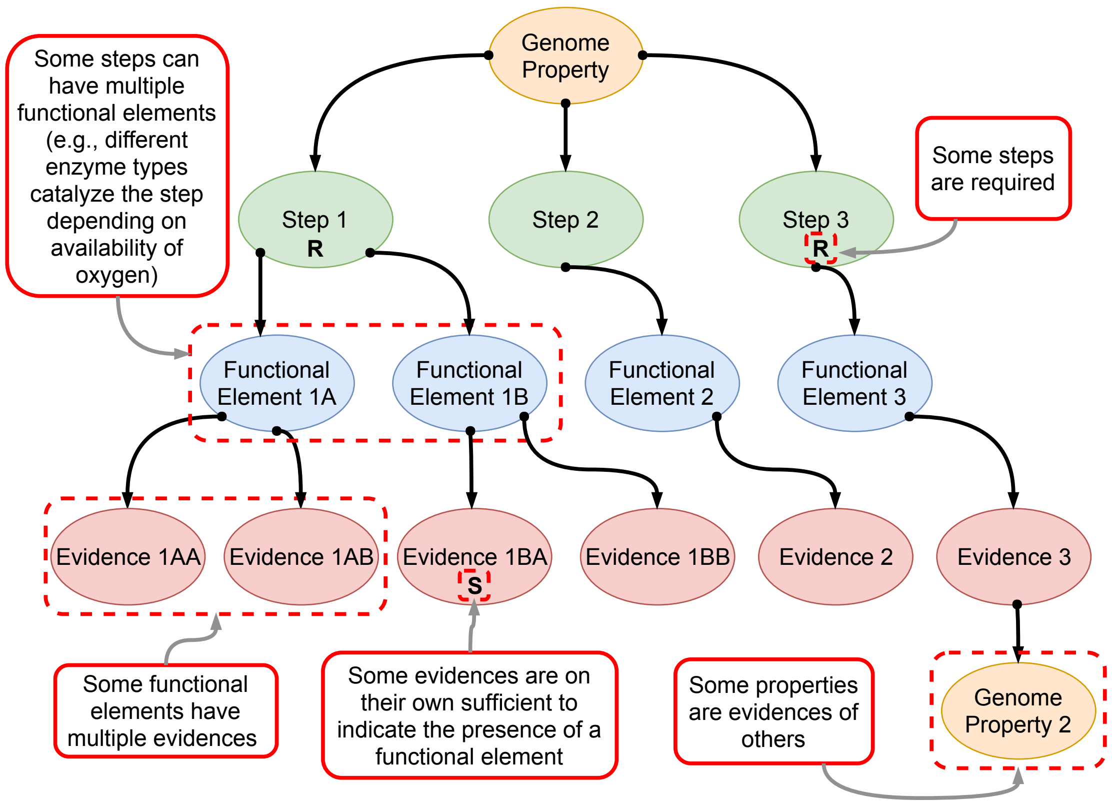

Genome Property
Some steps can have multiple functional elements (e.g., different enzyme types catalyze the step depending on availability of oxygen)
Step 1 R
Step 2
Step 3 R
Some steps are required
Functional Element 1A
Functional Element 1B
Functional Element 2
Functional Element 3
Evidence 1AA
Evidence 1AB
Evidence 1BA S
Evidence 1BB
Evidence 2
Evidence 3
Genome Property 2
Some functional elements have multiple evidences
Some evidences are on their own sufficient to indicate the presence of a functional element
Some properties are evidences of others

Figure 2.2: **In-memory objects that Pygenprop uses to support its property objects.** With Pygenprop, each property object is supported by a series of objects that represent a property's steps and lines of evidence. Functional element objects form links between steps and evidence. Property objects can be supported by multiple step, functional element, and evidence objects. Figure is from [16].

2.2.1 The GenomeProperty Class

The GenomeProperty class creates a blueprint for objects that represent individual genome properties. Once instantiated, these objects possess properties (*i.e.*, attributes whose return value is generated by a function) and attributes that represent data about the property. This information mirrors that provided in the genome property's **DESC** file before parsing. Information about property steps, database references, and literature references have been abstracted into separate classes. A summary of the methods, properties, and attributes of GenomeProperty objects can be seen in Table 2.2 and example code below.

Table 2.2: Methods, properties, and attributes of GenomeProperty objects.

Name	**Type**	**Description**
required_steps	Property	Return a list of step objects representing steps that are required to support the existence of the property
child_genome _property _identifiers	Property	Return a list of the genome property identifiers of child genome properties that are used as step evidences for the property
to_json	Method	Serialize the property to a JavaScript Object Notation (JSON) [25] string
databases	Attribute	List of database objects representing external database references to the property
references	Attribute	List of literature reference objects representing external articles discussing the property
private_notes	Attribute	Private internal notes about the property
tree	Attribute	GenomePropertyTree object (see Subsection 2.2.7) that the property belongs to
description	Attribute	Long-from description of the property
threshold	Attribute	Minimum number of required steps that must be assigned YES in order for the property to be assigned PARTIAL rather than NO support during property assignment
type	Attribute	Property type (*e.g.*, GUILD, CATEGORY, or PATHWAY)
steps	Attribute	List of step objects representing all steps that can support the existence of the property (including non-required)
public	Attribute	True if the property is publicly released
children	Attribute	List of child genome property objects representing properties the are used as step evidences by the property
name	Attribute	Name of the property
id	Attribute	Genome property identifier (*e.g.*, GenPropXXXX)
parents	Attribute	List of parent genome properties objects representing properties that use the property as step evidences

Example code for using GenomeProperty objects

```
property.id
Out: GenProp0144

property.name
Out: Chlorophyllide a biosynthesis from protoporphyrin IX

property.parents
Out: List of parent property objects

property.children
Out: List of child property objects

property.steps
Out: List of step objects

property.databases
Out: List of database reference objects

property.references
Out: List of literature reference objects
```

2.2.2 The DatabaseReference Class

The DatabaseReference class allows for the creation of objects that link a property to equivalent records in other pathway databases such as KEGG [80] and MetaCyc [85]. These objects are children of GenomeProperty objects (Fig. 2.1). For example, in the case of GenProp0145 (histidine degradation to glutamate), the GenomeProperty object would have two child DatabaseReference objects. One is for signifying the equivalent KEGG pathway (*e.g.*, map00340) and another for the equivalent MetaCyc pathway (*e.g.*, PWY-5028). DatabaseReference objects can be used to build software that links records for the same pathway across multiple databases. A summary of the attributes of DatabaseReference objects can be seen in Table 2.3 and example code below.

Table 2.3: Attributes of DatabaseReference objects.

Name	**Type**	**Description**
database_name	Attribute	Name of the external database (*e.g.*, KEGG)
record_title	Attribute	Name of the external database record that a property is equivalent to
record_ids	Attribute	Identifier of the external database record that a property is equivalent to (*e.g.*, a KEGG pathway map identifier)

Example code for using DatabaseReference objects

```
reference = property.databases[0]

reference.database_name
Out: MetaCyc

reference.record_title
Out: Pathway: 3,8-divinyl-chlorophyllide a biosynthesis III

# Returns a list to handle cases where
# there are multiple identifiers.
reference.record_ids[0]
Out: PWY-7159
```

2.2.3 The LiteratureReference Class

The LiteratureReference class lays out the foundation for objects that represent scientific articles that support the existence of a property, such as a review summarizing current knowledge about a metabolic pathway. Once instantiated, LiteratureReference objects are children of GenomeProperty objects (Fig. 2.1). A summary of the attributes of LiteratureReference objects can be seen in Table 2.4 and example code below.

Table 2.4: Attributes of LiteratureReference objects.

Name	**Type**	**Description**
number	Attribute	Number of the literature reference
pubmed_id	Attribute	PubMed [28] identifier of the literature reference

Table 2.4 continued from previous page

Name	Type	Description
title	Attribute	Title of the literature reference
authors	Attribute	Authors of the literature reference
citation	Attribute	Citation for the literature reference

Example code for using literature reference objects

```
reference = property.references[0]

reference.pubmed_id
Out: 17370354

reference.title
Out: Recent advances in chlorophyll biosynthesis.

reference.citation
Out: Photosynth Res. 2006;90(2):173-194.

reference.authors
Out: Bollivar DW
```

2.2.4 The Step Class

The Step class is used to generate objects representing individual genome property steps. These objects are children of parent GenomeProperty objects and have FunctionalElements objects as children (Fig. 2.2). A summary of the properties and attributes of Step objects can be seen in Table 2.5 and example code below.

Table 2.5: Properties and attributes of Step objects.

Name	Type	Description
name	Property	Return the name of the step
required	Property	Return true if the step is required for assignment of the parent genome property
property _identifiers	Property	Return a list of genome property identifiers of genome properties that are used as evidence for the step

Table 2.5 continued from previous page

Name	Type	Description
interpro _identifiers	Property	Return a list of InterPro identifiers [72] that are used as evidence for the step (*e.g.*, IPRXXXX)
consortium _identifiers	Property	Return a list of InterPro consortium member database (*e.g.*, Pfam or TIGRFAM [10]) signature accessions [72] that are used as evidence for the step (*e.g.*, PFXXXX)
genome _properties	Property	Return a list of child GenomeProperty objects that are used as evidence for the step
number	Attribute	Number of the step
parent	Attribute	Parent GenomeProperty object of the step
functional _elements	Attribute	List of FunctionalElement objects that are used to support the existence a step

Example code for using step objects

```
step = property.steps[0]

step.number
Out: 1

step.name
Out: Magnesium-chelatase subunit ChlD (EC 6.6.1.1)

step.required
Out: True

step.interpro_identifiers
Out: A list of InterPro identifiers (e.g., IPR011776)

step.consortium_identifiers
Out: A list of consortium signature identifiers (e.g., TIGR02031)

step.functional_elements
Out: A list of functional element objects
```

2.2.5 The FunctionalElement Class

The FunctionalElement class allows for the instantiation of objects that are placed between Step and Evidence objects during parsing (Fig. 2.2). Functional elements are not part of the original Genome Properties database schema and were added by Pygenprop to account for property steps that can be catalyzed by multiple enzyme families. This issue of having multiple types of enzymes capable of catalyzing a step is an open issue on the Genome Properties database GitHub repository (see github.com/ebi-pf-team/genome-properties/issues/29). The addition of FunctionalElements object addresses this issue. A summary of the attributes of FunctionalElement objects can be seen in Table 2.6 and example code below.

Table 2.6: Attributes of FunctionalElement objects.

Name	**Type**	**Description**
parent	Attribute	Step object that the FunctionalElement supports
evidence	Attribute	List of Evidence objects that support the existence of the functional element
name	Attribute	Name of the functional element
id	Attribute	Identifier of the functional element
required	Attribute	True if the functional element is required for assignment of the parent genome property

Example code for using FunctionalElement objects

```
element = step.functional_elements[0]

element.id
Out: element.id

element.name
Out: Magnesium-chelatase subunit ChlD (EC 6.6.1.1)

element.required
Out: True
```

```
element.evidence
Out: A list of evidence objects
```

2.2.6 The Evidence Class

The Evidence class allows for the generation of objects that represent individual pieces of evidence that support the existence of functional elements and, in turn, genome property steps. Pieces of evidence include the presence of InterPro consortium signatures [72] or support for the existence of other genome properties found in an organism's genome. A summary of the properties and attributes of Evidence objects can be seen in Table 2.7 and example code below.

Table 2.7: Properties and attributes of Evidence objects.

Name	**Type**	**Description**
has_genome _property	Property	Return true if the evidence is supported by the existence a genome property
property _identfiers	Property	Return a list of genome property identifiers of genome properties that are used by the evidence
interpro _identifiers	Property	Return a list InterPro identifiers of genome properties that are used by this evidence (*e.g.*, IPRXXXX)
consortium _identifiers	Property	Return a list of InterPro consortium member database (*e.g.*, Pfam) signature identifiers of genome properties that are used by this evidence (*e.g.*, PFXXXXX)
genome _properties	Property	Return a list of child genome property objects that are used by this evidence
parent	Attribute	Parent FunctionalElement object of this evidence
gene_ontology _terms	Attribute	List of gene ontology (GO) term identifiers [7] associated with the InterPro identifiers that are used by the evidence
evidence _identifiers	Attribute	List of both InterPro and signature identifiers used by the evidence
sufficient	Attribute	True if the evidence alone can prove the existence of a functional element

Example code for using Evidence objects

```
evidence = element.evidence[0]

evidence.has_genome_property
Out: false

evidence.sufficient
Out: true

evidence.interpro_identifiers
Out: A list of InterPro identifiers (e.g., IPR011776)

evidence.consortium_identifiers
Out: A list of consortium signature identifiers (e.g., TIGR02031)
```

2.2.7 The GenomePropertiesTree Class

GenomePropertiesTree[5] objects, as instantiated from the GenomePropertiesTree class, are used to represent the rooted DAG structure of the entire Genome Properties database. These objects contain a Python dictionary (*i.e.*, a key-value mapping) of GenomeProperty objects indexed by their property identifiers. Also, individual property objects point to each other using their parent/child attributes (Fig. 2.1 and Table 2.2), allowing for climbing up and down the DAG. These parent-child relationships between GenomeProperty objects are built during a GenomePropertiesTree object's instantiation. The GenomePropertiesTree class provides its objects with methods that allow users to search for specific GenomeProperty objects, and acquire lists of the root (*i.e.*, no parent property) and leaf (i.e., no child properties) GenomeProperty objects. A summary of the methods, properties, and attributes of GenomePropertiesTree objects can be seen in Table 2.8 and example code below.

[5]The word "tree", which refers to a different data structure from a DAG (branches of trees do not merge), is used in the name of the class that represents the Genome Properties database. The class's methods also use tree terminology. This naming was done as a means of convenience as most bioinformatics users are more familiar with trees.

Table 2.8: Methods, properties, and attributes of GenomePropertiesTree objects.

Name	**Type**	**Description**
build_genome_property_connections	Method	Iterate through every GenomeProperty that is a child of the tree; set these Properties' parent and child attributes (see Table 2.2) to point to matching child and parent GenomeProperty objects that are also children of the Tree. This method connects GenomeProperty objects to create a DAG structure.
to_json	Method	Serialize the property tree to a JSON string
create_metabolism_database_mapping_file	Method	Write a CSV file that maps from genome property identifiers to the identifiers of equivalent records found in Pfam and MetaCyc
root	Property	Return the top-level GenomeProperty that has no parent.
leafs	Property	Return a list of GenomeProperty objects whose steps are not supported by any other properties
genome_property_identifiers	Property	Return a list of the genome property identifiers (*e.g.*, GenPropXXXX) for all genome properties within the database
interpro_identifiers	Property	Return a list of InterPro identifiers that are used as evidence for steps (*e.g.*, IPRXXXX) within the database
consortium_identifiers	Property	Return a list of InterPro consortium member database (*e.g.*, Pfam) signature accessions that are used as evidence for steps (*e.g.*, PFXXXXX) within the database
consortium_identifiers_dataframe	Property	Return InterPro consortium signature accessions in the form of a pandas DataFrame [105]
genome_properties_dictionary	Attribute	Dictionary of genome property objects representing all genome properties within by the database; the dictionary is keyed by genome property identifier

Example code for using GenomePropertiesTree objects

```
tree = GenomePropertyTree(*property_object_list)
tree_two = parse_genome_properties_flat_file(prop_file_handle)

len(tree) # number of properties in the database
Out: false

tree.root
Out: The root FenomeProperty object

tree.leafs
Out: A list of leaf GenomeProperty objects
     (those with no child properties)

# Properties in the tree can be iterated.
for genome_property in tree:
        print(genome_property.id)
Out: Prints all genome property identifiers

# The tree can be rapidly searched
tree['GenProp1127']
Out: The GenomeProperty object representing GenProp1127.
```

2.2.8 Performance of Pygenprop's Genome Properties Database Representation

Pygenprop's representation of the Genome Properties database as a GenomePropertiesTree object and its children takes only up 11.2 MB[6] of RAM as of database version 2.0. This memory usage only takes up marginally more space than the database's original **genomeProperties.txt** file that takes up 1.8 MB on disk. The memory usage difference is due to the representation of the database as a series of objects and their associated data structures. However, because 11.2 MB takes up little RAM on a modern machine, a more

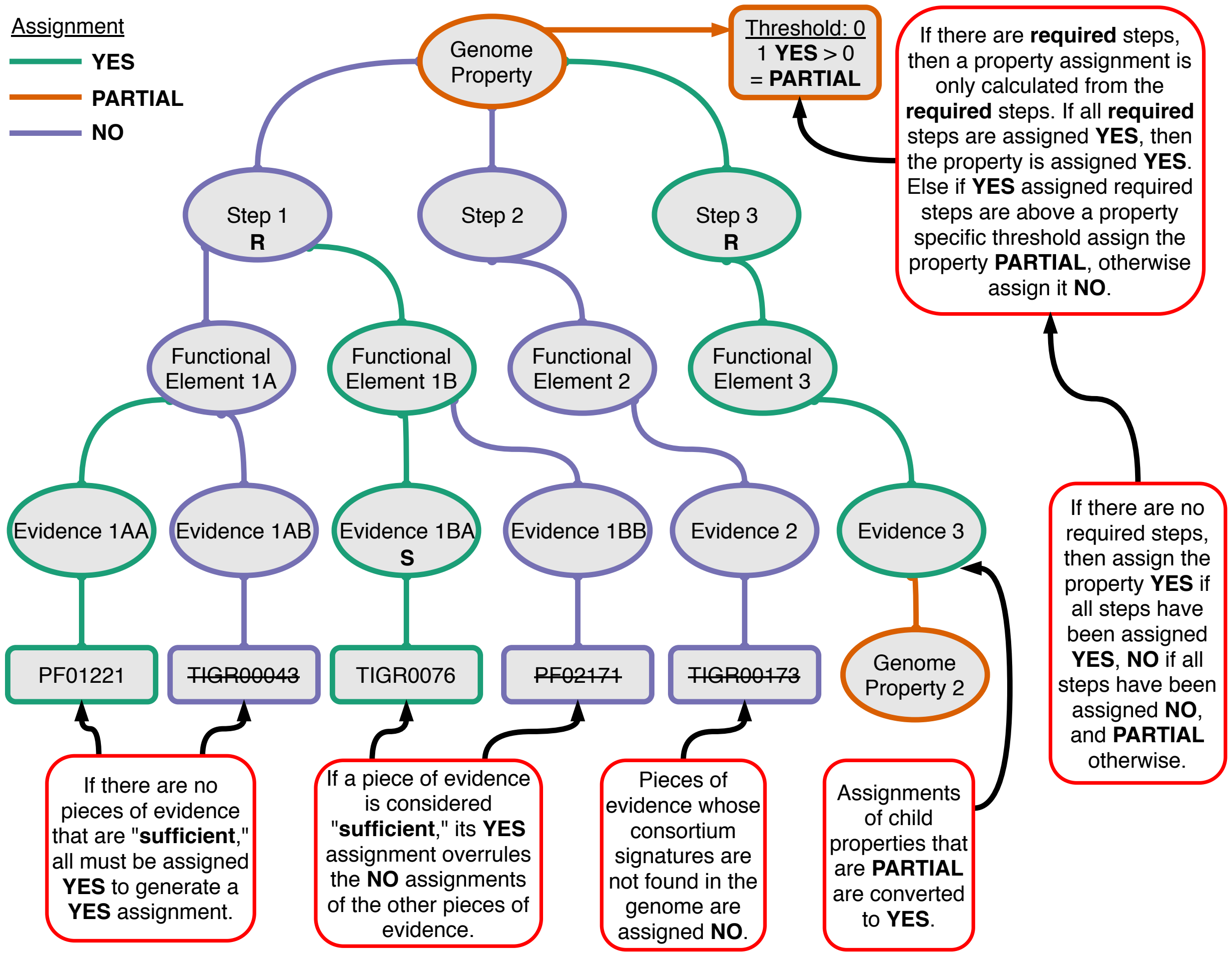
[6]Note that for the remainder of the thesis, unless otherwise noted, all memory usages for Python objects were recorded using Python's built-in getsizeof function (docs.python.org/3/library/sys.html#sys.getsizeof).

compact representation for the Genome Properties database was not pursued. Individual genome property objects can be looked up, by property identifier, from within a GenomePropertiesTree object within 277 ns ±8 ns (N = 80).

2.3 Assignment of Genome Properties to Organism Genomes

Information contained within the Genome Properties database can be used to assign YES, NO, or PARTIAL support for an organism possessing a genetically derived property, such as a biochemical pathway. These property assignments are generated from assignments of YES or NO support for these properties' underlying steps (Fig. 2.3). Steps cannot be assigned PARTIAL. Assignments of support for steps are calculated in two ways. One way of calculating step assignments is from the presence of InterPro consortium database signatures (*e.g.*, Pfams, TIGRFAMs, and others) in the domain annotations of an organism's proteins. These domain annotations are generated by running InterProScan [79] on an organism's predicted proteins. These InterPro annotations are used to calculate YES and NO assignments for steps. Because the Genome Properties database is a DAG, some properties' steps use the assignments of other properties as evidence (Fig. 2.1). The second way of calculating step assignments is from the presence of previously calculated YES, PARTIAL, or NO support for other genome properties in the organism's genome (Fig. 2.3). These child property assignments are used to calculate YES and NO assignments for steps (Fig. 2.3). Because steps can be only be assigned YES or NO, PARTIAL assignments of support for child properties cause the step they support to be assigned YES (Fig. 2.3). All steps of leaf properties only use InterPro domains as evidence; however, the steps of properties closer to Genome Properties DAG root may either use domains, other properties' assignments, or both as evidence. As a result, the assignments for all genome properties can be recursively calculated from DAG leaf to DAG root based solely on InterPro domain evidence.

Pygenprop's code for assigning genome properties is based on that of the Genome Properties Perl library (see github.com/ebi-pf-team/genome-properties/tree/master/code) that ships alongside the Genome Properties database. Pygenprop replicates the Perl library's support assignment functionality. The Python library evaluates properties' support, from DAG leaf to DAG root, using a recursive algorithm (Fig. 2.3 and Section 2.3.2). For each property, assignment starts by assigning each step evidence with YES or NO support and then recursively flowing this assignment up through functional elements, steps, and eventually back to the property itself (Fig. 2.3). The rules used for assigning support at different levels are detailed in the subsections below. For each genome that needs to have properties assigned, an AssignmentCache object is generated. This object contains all data required for property assignment and methods for assigning support using this data. A detailed description of this class is also found in the subsections below.



Assignment
YES
PARTIAL
NO
Genome Property
Threshold: 0
1 YES > 0
= PARTIAL
If there are required steps, then a property assignment is only calculated from the required steps. If all required steps are assigned YES, then the property is assigned YES. Else if YES assigned required steps are above a property specific threshold assign the property PARTIAL, otherwise assign it NO.
Step 1 R
Step 2
Step 3 R
Functional Element 1A
Functional Element 1B
Functional Element 2
Functional Element 3
Evidence 1AA
Evidence 1AB
Evidence 1BA S
Evidence 1BB
Evidence 2
Evidence 3
If there are no required steps, then assign the property YES if all steps have been assigned YES, NO if all steps have been assigned NO, and PARTIAL otherwise.
PF01221
~~TIGR00043~~
TIGR0076
~~PF02171~~
~~TIGR00173~~
Genome Property 2
If there are no pieces of evidence that are "sufficient," all must be assigned YES to generate a YES assignment.
If a piece of evidence is considered "sufficient," its YES assignment overrules the NO assignments of the other pieces of evidence.
Pieces of evidence whose consortium signatures are not found in the genome are assigned NO.
Assignments of child properties that are PARTIAL are converted to YES.

Figure 2.3: **Overview of the genome property assignment process used by Pygenprop.** The assignment algorithms use by Pygenprop recursively generate assignments for individual properties and steps from leaf to root along Genome Properties DAG. The assignments of child properties and steps are used to calculates assignments of YES, PARTIAL, and NO for parent properties.

2.3.1 The AssignmentCache Class

AssignmentCache objects, instantiated from the AssignmentCache class, are used to assign genome properties to an organism. These caching objects can be generated from three sources:

- InterProScan TSV files (protein domain annotation files)
- Python lists of InterPro member database signature accessions for an organism (as could be downloaded from precalculated InterProScan results of UniProt proteomes [35])
- Precalculated property assignment files generated by the Genome Properties Perl library

Pygenprop contains parsers for both InterProScan TSV files and Genome Properties precalculated property assignment files. In all cases, InterPro member database signature accessions are de-duplicated before their inclusion in an AssignmentCache object, as occurs in the Genome Properties Perl library.

AssignmentCache objects contain two Python dictionaries (*i.e.*, key-value mappings) for storing previously calculated property and step assignments, respectively. The object also contains a Python set that is designed to store all unique InterPro consortium signature identifiers found in an organism's protein domain annotations. The AssignmentCache has a method called **bootstrap_assignments** that uses a GenomePropertyTree (Subsection 2.2.7) object and data stored within the AssignmentCache itself to calculate levels of support for all properties. This function also calculates levels of support for steps. AssignmentCaches from multiple organisms can later be combined during the creation of GenomePropertiesResults objects (as discussed in Section 2.4 below) that allow comparison of property assignments across organisms. As mentioned at the top of this sectionthese caches can be from different sources. Unlike Pygenprop, the Genome Properties Perl library does not maintain the concept of an assignment cache and can only calculate property support for a single organism from a single InterProScan TSV file. A summary of the methods, properties, and attributes of AssignmentCache objects can be seen in Table 2.9 and example code below.

Table 2.9: Methods, properties, and attributes of AssignmentCache objects.

Name	**Type**	**Description**
cache_property_assignment	Method	Add a property assignment to the cache
get_property_assignment	Method	Return a property assignment from the cache
cache_step_assignment	Method	Add a step assignment to the cache
get_step_assignment	Method	Return a step assignment from the cache
flush_property_from_cache	Method	Remove a property assignment and its associated step assignments from the cache
synchronize_with_tree	Method	If a property whose assignment is cached is not found in the tree, remove its assignment and associated step assignments. This method allows for compatibility between different versions of the Genome Properties database and pre-calculated assignment files.
bootstrap_assignments	Method	Recursively assign support for properties from leaf to root using an internal set pre-calculated assignments and a InterPro consortium signature identifiers
bootstrap_missing_step_assignments	Method	Search through a genome property tree to find steps that are not in the cache. Assign these steps NO because they are missing. This method is used when pre-calculated step assignments that result in NO have been omitted to save disk space.
create_results_tables	Method	Return two pandas DataFrames representing property and step assignments for the organism
property_identifiers	Property	Return a list of genome property identifiers (*e.g.*, GenPropXXXX) for properties whose assignment are in the cache
property_assignments	Attribute	Python dictionary of YES, NO, and PARTIAL labelled property assignments keyed by genome property identifier
step_assignments	Attribute	Doubly nested Python dictionary of YES and NO labelled step assignments keyed by genome property identifier and step number

Table 2.9 continued from previous page

Name	**Type**	**Description**
interpro _signiture _accessions	Attribute	Set of InterPro consortium signature identifiers of domains found in the organism's protein domain annotations
sample_name	Attribute	Name of the organism or sample. When the AssignmentCache is created from a file, the sample name is set to the filename without file extension

Example code for using AssignmentCache objects

```
tree = parse_genome_properties_flat_file(prop_file_handle)

cache1 = parse_genome_property_longform_file(long_file_handle)
cache2 = parse_interproscan_file(interproscan_tsv_file_handle)
cache3 = AssignmentCache(sample_name='E_coli',
interpro_signature_accessions=identifier_list)

cache2.sample_name
Out: C_benthia_SPR155

cache2.get_property_assignment('GenProp1065')
Out: PARTIAL

cache2.get_step_assignment('GenProp1067', 2)
Out: YES

# Set GenProp2536 to YES
cache2.cache_property_assignment('GenProp2536', 'YES')

# Set GenProp2539 step two to YES
cache2.cache_step_assignment('GenProp2539', 2, 'YES')

# Remove GenProp2567 from the cache
cache2.flush_property_from_cache('GenProp2567')

# Bootstrap both step and property assignments
```

```
cache2.boostrap_assignments(properties_tree=tree)

# Create pandas DataFrames for per organism property
# and step assignments
tables = cache2.create_results_tables(properties_tree=tree)
property_table = tables[0]
step_table = tables[1]
```

2.3.2 Implemented Assignment Algorithms

As mentioned in Section 2.3, Pygenprop uses recursion, the process of program functions repeatedly calling themselves internally, to assign YES, NO, or PARTIAL support for individual properties found within the Genome Properties database. During assignment recursion, Pygenprop uses a GenomePropertiesTree object (Subsection 2.2.7) to provide itself with information about assignment requirements for each property and relevant connections between properties. In the context of AssignmentCache objects, the process of generating of assignments is referred to as bootstrapping. Bootstrapping is the term used to describe the assignment process because properties are assigned from pre-existing information stored within the cache, such as pre-calculated property and step assignments, and InterPro consortium signature accessions. This pre-calculated information grows as more properties are assigned. Pygenprop's recursive assignment algorithms, like those in the Genome Properties Perl library, assign support to both properties and property steps (Fig. 2.3). Assignments of support for steps are used to assign support for parent properties (Fig.2.3). The overall DAG-level assignment algorithm used by Pygenprop, which ultimately assigns support for the DAG's root property based on the assignments of the DAG's leaf properties, is an example of dynamic programming (see en.wikipedia.org/wiki/Dynamic_programming).

During the recursion process, the newly calculated step and property assignments are added continually to the AssignmentCache object's step and property assignment dictionaries (Table 2.9). During successive recursive assignment calculations, these dictionaries are checked first, using the AssignmentCache's **get_property_assignment** and **get_step_assignment** methods (Table 2.9), to find step and property assignments that have already been calculated in previous recursive cycles. During a recursive cycle, if a pre-calculated assignment result is found, recursion is stopped, and this cached assignment value is returned to the calling parent recursive function. This checking for and utilization of previously calculated results is called memoization (see en.wikipedia.org/wiki/ Memoization). Because the AssignmentCache object's assignment dictionaries are used as a cache, the rate of the assignment process will increase exponentially in speed as more properties are calculated, vastly reducing overall assignment time. Recursion also stops when step assignments are calculated for steps that are supported by InterPro domains 2.3).

Step assignments are calculated recursively from both functional element and evidence assignments (Fig. 2.3). Evidences are assigned YES or NO based on the presence an InterPro consortium signatures found in an AssignmentCache's **interpro_signature_accessions** attribute (Table 2.9) or a recursively calculated child property's assignment (Fig. 2.3). The signature identifier or child property to be used during calculations is specified inside each evidence's representative Evidence object (Table 2.7). Pieces of evidence are assigned NO in two situations:

- The evidence's InterPro consortium signature is not found in the AssignmentCache's **interpro_signature_accessions** attribute
- The evidence's child genome property has been assigned NO

Otherwise, each evidence is assigned YES (Fig. 2.3)[7]. Functional elements are assigned YES under two situations:

- If all underlying pieces of evidence have been assigned YES
- If a single piece of evidence that sufficient on its own to support the existence of a step is assigned YES[8].

Other than these two situations, the functional element is assigned NO (Fig. 2.3). Steps are assigned YES or NO based on the assignments of functional elements (Fig. 2.3). Steps are assigned YES only if all functional elements of that step have been assigned YES and are assigned NO otherwise. As noted in the second paragraph of Subsection 2.3.2, assignment results for already calculated steps are checked for before step assignment recursion and are added to the AssignmentCache after step assignment calculations. If a piece of evidence has a genome property as its child, then this property's assignment is calculated, creating another recursion cycle.

[7]If a step uses the assignment a child property as evidence and this child property's assignment is PARTIAL, then the step's evidence is assigned YES. This change from a PARTIAL to a YES assignment is done because the algorithms used for calculating property assignments expect YES or NO assignments as inputs, as would be the case if the property's steps used InterPro domains as evidence.

[8]As mentioned in Table 2.7, some pieces of evidence can be used as the sole piece of evidence for a step (Fig. 2.3).

Some properties have steps that are required to exist for the property to be assigned YES or PARTIAL. Also, each property possesses a **threshold** attribute (see Table 2.2) that specifies how many of these required steps must be present before an assignment of PARTIAL support can be applied to the property. These threshold values are included in the Genome Properties database and are predetermined for each property. They are stored in each property's **DESC** file (see Subsection 2.1.1). If there are required steps for a property, then it can only be assigned YES if all required steps are present (Fig. 2.3). The property is assigned PARTIAL only if the number of its required steps assigned YES is higher than the required steps threshold attribute (Fig. 2.3). If the number of required steps assigned YES is less than or equal to the required steps threshold, then the property is assigned NO support. It is important to note that the property assignment does not take into account steps that are optional, only those that are required. If a property's step's assignment value is not known, it is calculated using the recursive step assignment algorithm described earlier in Subsection 2.3.2.

Categorical properties, such as GenProp0011 (methanogenesis), do not have any required steps; all steps are optional. Thus a different assignment algorithm is required for these property assignments. Categorical properties are only assigned YES if all steps are assigned YES, NO if all steps are assigned NO, and PARTIAL otherwise (Fig. 2.3). Note that the generation of support assignments for categorical properties is unique to Pygenprop and is not performed by the Genome Properties Perl library. The recursion in the Perl library stops before it reaches categorical properties.

For a 2.9 MB InterProScan TSV file containing domain annotations for 4100 *Escherichia coli* K12 proteins (NCBI Taxa ID: 1010810), the resulting AssignmentCache object was found to take up 1.2 MB of RAM before bootstrapping assignments and 1.7 MB after. Assignment bootstrapping was found to take 76.7 ms $\pm$15.4 ms (N = 80) for K12.

2.4 Development of a Framework for Comparing Genome Property Assignments Across Multiple Organisms

One of the main goals of Pygenprop was to facilitate programmatic comparisons of the presence/absence of biochemical pathways across multiple organisms. Specifically, the library provides methods to filter out genome properties that are shared between organisms, thus highlighting differences in these organisms' metabolic or functional capabilities. Pygenprop's ability to assess these differences programmatically will allow future researchers to automate many aspects of pathway analysis, such as complex phenotype prediction and the discovery of correlations between pathway presence and patterns of niche partitioning [55]. To support programmatic exploration of genome properties assignments, Pygenprop includes the GenomePropertiesResults class.

2.4.1 The GenomePropertiesResults Class

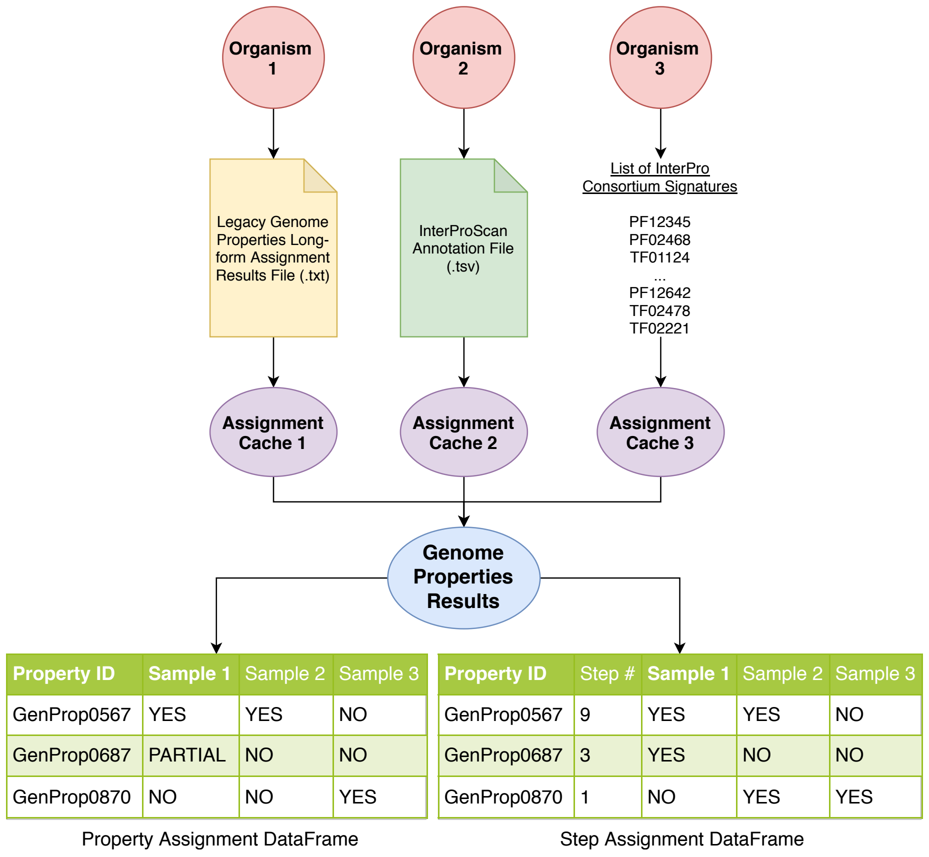
During their instantiation, objects of the GenomePropertiesResults class take a series of AssignmentCache objects (Subsection 2.3.1), potentially from disparate sources, as input (Fig. 2.4). During this process, the per-sample assignments found within these input caches are transformed into two indexed pandas DataFrames [105] that hold data for multiple samples, one for property assignments and another for step assignments (Fig. 2.6). The GenomePropertiesResults class also contains a series of methods that return versions of these DataFrames with filtered down step and property assignments. A summary of the methods, properties, and attributes of GenomePropertiesResults objects can be seen in Table 2.10 and example code below. GenomePropertiesResults objects become useful when used interactively in Jupyter Notebooks [88]. See github.com/Micromeda/pygenprop /blob/master/docs/source/_static/tutorial/tutorial.ipynb for an example notebook workflow using Pygenprop to compare virulence genome properties of *E. coli* K12 (NCBI Taxa ID: 1010810) and O157:H7 (NCBI Taxa ID: 83334).

Table 2.10: Methods, properties, and attributes of GenomePropertiesResults objects.

Name	**Type**	**Description**
get_results	Method	Return the assignment results as a pandas DataFrame for a series of genome properties at either a step or property level
get_results _summary	Method	Return a summary of assignment results as a pandas DataFrame for a series of genome properties at either a step or property level
get_property _results	Method	Return a list of assignments of support for all samples and for a given property
get_step_results	Method	Return a list of assignments for all samples and for a given property step
to_json	Method	Serialize the results object to a JSON property tree with assignment results for each sample annotating each property node

Table 2.10 continued from previous page

Name	**Type**	**Description**
to_assignment _database	Method	Serialize the results object to a Micromeda SQLite database file (.micro)
to_msgpack	Method	Serialize the results object to a MessagePack binary string
sample_names	Property	Return the names of all organisms used in the creation of the results object
differing _property _results	Property	Return a pandas DataFrame of property assignments with properties whose assignments are the same across all samples filtered out
differing_step _results	Property	Return a pandas DataFrame of step assignments with steps whose assignments are the same across all samples filtered out
supported _property _results	Property	Return a pandas DataFrame of property assignments with properties whose assignments are NO across all samples filtered out
supported_step _results	Property	Return a pandas DataFrame of step assignments with steps whose assignments are NO across all samples filtered out
property_results	Attribute	Pandas DataFrame of property assignments across all samples
step_results	Attribute	Pandas DataFrame of step assignments across all samples
tree	Attribute	GenomePropertiesTree object provided during instantiation of the GenomePropertiesResults object


Organism 1
Organism 2
Organism 3
Legacy Genome Properties Long-form Assignment Results File (.txt)
InterProScan Annotation File (.tsv)
List of InterPro Consortium Signatures
PF12345
PF02468
TF01124
...
PF12642
TF02478
TF02221
Assignment Cache 1
Assignment Cache 2
Assignment Cache 3
Genome Properties Results
Property Assignment DataFrame
Step Assignment DataFrame

Property ID	Sample 1	Sample 2	Sample 3
GenProp0567	YES	YES	NO
GenProp0687	PARTIAL	NO	NO
GenProp0870	NO	NO	YES

Property ID	Step #	Sample 1	Sample 2	Sample 3
GenProp0567	9	YES	YES	NO
GenProp0687	3	YES	NO	NO
GenProp0870	1	NO	YES	YES

Figure 2.4: **How GenomePropertiesResults objects are generated by combining the AssignmentCache objects generate for multiple organisms.** These caches can be generated from disparate sources, such as InterProScan results files or lists of InterPro signatures provided by a remote server. The resulting GenomePropertiesResults object possesses DataFrames containing either step and property assignments for multiple organisms. The comparative pathway analysis software can use these DataFrames.

Example code for using GenomePropertiesResults objects

```
tree = parse_genome_properties_flat_file(properties_file_handle)

cache_one = parse_interproscan_file(ipr_file_handle_one)
cache_two = parse_interproscan_file(ipr_file_handle_two)
results = GenomePropertiesResults(cache_one, cache_two,
                                  properties_tree=property_tree)
```

```
results.sample_names
Out: ['E_coli_K12', 'C_luteolum_DSM_273']

results.get_property_result('GenProp1065')
Out: ['NO', 'NO']

results.get_step_result('GenProp1067', 2)
Out: ['YES', 'NO']

# Get step assignments for GenProp1065 and GenProp1067 with
# property and step names.
results.get_results('GenProp1065', 'GenProp1067',
                    steps=True, names=True)
Out:
```

Property_Identifier	**Property_Name**	**Step_Number**	**Step_Name**	**E_coli_K12**	**C_luteolum_DSM_273**
GenProp1065	Radical SAM/SPASM TIGR04347/TIGR04031 system	1	RSAM-partnered protein, Htur_1727 family	NO	NO
GenProp1065	Radical SAM/SPASM TIGR04347/TIGR04031 system	2	Pseudo-rSAM protein/SPASM domain protein	NO	NO
GenProp1067	Defense systems	1	CRISPR systems	YES	YES
GenProp1067	Defense systems	2	Restriction enzyme system, type I	YES	NO
GenProp1067	Defense systems	3	DNA sulfur modification system dnd	NO	NO
GenProp1067	Defense systems	4	Abortive infection proteins	NO	NO
GenProp1067	Defense systems	5	Complement activation, common pathway 1	NO	NO

```
# Get property assignments for GenProp1065 and GenProp1067
results.get_results('GenProp1065', 'GenProp1067',
                    steps=False, names=False)
Out:
```

Property_Identifier	**E_coli_K12**	**C_luteolum_DSM_273**
GenProp1065	NO	NO
GenProp1067	PARTIAL	PARTIAL

```
# Get counts of YES and NO assignments for steps GenProp1065
# and GenProp1067
results.get_results_summary('GenProp1065', 'GenProp1067', steps=True)
Out:
```

Assignment	**E_coli_K12**	**C_luteolum_DSM_273**
NO	5	6
YES	2	1

```
# Get percentages of YES and NO assignments for steps GenProp1065
# and GenProp1067
results.get_results_summary('GenProp1065', 'GenProp1067',
                             steps=True, normalize=True)
Out:
```

Assignment	**E_coli_K12**	**C_luteolum_DSM_273**
NO	71.428571	85.714286
YES	28.571429	14.285714

When generated from two AssignmentCache objects containing assignments built from the proteomes of *Escherichia coli* K12 (NCBI Taxa ID: 1010810) and *Chlorobium chlorochromatii* CaD3 (NCBI Taxa ID: 340177), the resulting GenomePropertiesResults object took up 14.4 MB of RAM after instantiation. Creating the object took 180.0 ms ±10.0 ms (N = 80).

2.4.2 The use of Pandas for Compatibility With the Python Data Science and Machine Learning Software Stack

Pandas is a Python library for cleaning, filtering, and reshaping data. Pygenprop's GenomePropertiesResults object presents property and step assignments as pandas DataFrames, which are a two-dimensional data matrices with both column and row names. These DataFrames allow users to quickly query and filter assignments and join these assignments to pre-existing metadata. For example, gene expression data (microarray or transcriptomic), culture optimal growth conditions, or even host environmental conditions could be merged with genome property assignment results in only a few lines of pandas code.

These joined datasets provide great potential as a source of data for data mining or as training sets for machine learning algorithms. Pandas DataFrame objects are built on top of NumPy arrays [105], which are used extensively across the entire Python data science ecosystem [78]. The arrays allow for the transfer of data between algorithms; for example, to those found in machine learning libraries such as Scikit-learn [123], PyTorch [120] or Tensorflow [1]. When trained on pathway assignment data from Pygenprop's Dataframes, such algorithms could be used to build a new generation of bioinformatics classifiers that facilitate the prediction of high-level organism phenotypes or preferred environmental conditions.

2.5 Extension of the AssignmentCache and GenomePropertiesResults classes to Include Supporting Match Information

As discussed in Subsection 1.4.2, Genome properties are assigned based on the presence of InterPro domains that can be used, either singly or in combination, to uniquely identify enzymes or protein structures that act as evidence for property steps. It may be the case that domains used as step evidences will be found in more than one protein of an organism. It may also be the case that some of these proteins may be false positives that may posses the identifying domain, or a similar domain, but do not carry out a genome property step. To assist in filtering out these false positives, researchers often want direct access to match information, such as E-value scores, held within an InterProScan file. Alternatively, users may want access to the entire sequences of proteins containing predicted domains so these proteins can be analyzed more deeply. For example, the proteins that are predicted to possess a motif that supports the existence of a property step could be compared phylogenetically to reference proteins that are already known carry out this step in other organisms. Previously, with the Genome Properties Perl library, the information required to perform the such analyses was kept in four separate file types:

- GenomeProperties.txt files kept property information
- Per-organism long-form property assignment files kept property and step assignments
- Per-organism FASTA files [121] kept protein sequences
- Per-organism InterProScan TSV files kept domain annotations

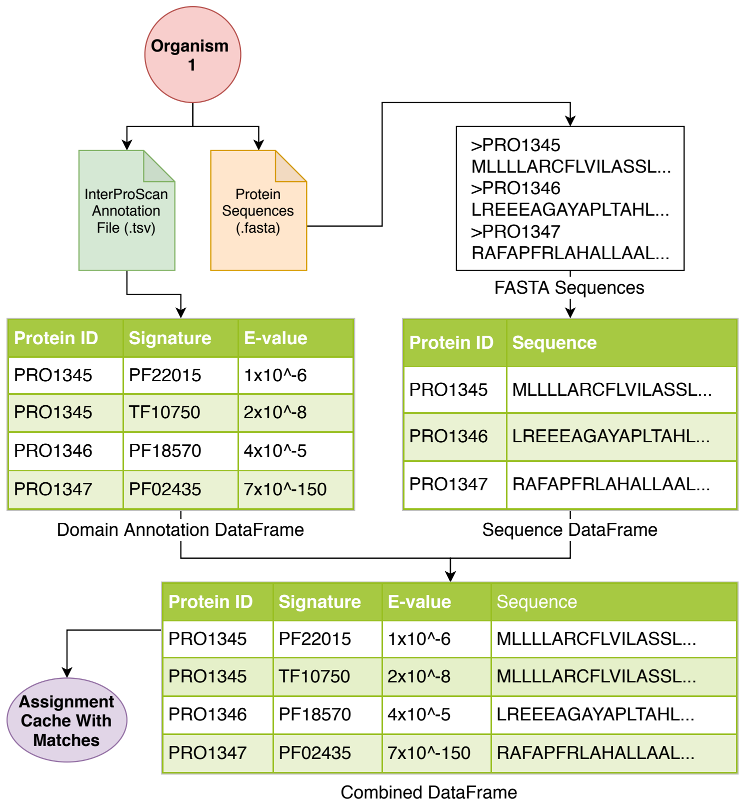
For a given organism, if a researcher wanted to find proteins that supports the existence of a genome property step, then they would have to write a script to parse all four of these file types, combine the data contained within each and perform searches on this joined data. These scripts would be difficult and time-consuming to write as the researcher would need to write much "boilerplate" code to carry out the joining of the datasets before analysis. Because the file types listed at the start of this section are created per-organism, if one wanted to apply such scripts to multiple organisms, then these scripts would have to be able to remember what file belongs to what organism. This tracking would further complicate script development.

Pygenprop already possesses much much of the aforementioned boilerplate code. For example, code for parsing the Genome Properties database and for comparing the presence/absence of genome properties across organisms. To make it easier for researchers to access domain and sequence information for proteins that support the existence of genome property steps, Pygenprop contains extended versions of both the AssignmentCache and GenomePropertiesResults classes. These classes possess attributes, properties, and methods related to accessing the supporting information that was initially provided only within InterProScan TSV and FASTA files. These classes are called AssignmentCacheWithMatches and GenomePropertiesResultsWithMatches, respectively.

2.5.1 The AssignmentCacheWithMatches Class

The AssignmentCacheWithMatches class extends the AssignmentCache class via class inheritance [140] (see en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)). In addition to the attributes, properties, and methods inherited from the AssignmentCache class, the AssignmentCacheWithMatches also possesses equivalents used for accessing supporting information such as domain annotation E-value scores and protein sequences that support the existence of property steps.

AssignmentCacheWithMatches objects are generated by parsing a FASTA file [121] of an organism's proteins and an associated InterProScan TSV of the domain annotations of these proteins (Fig. 2.5). Pandas is used to parse the TSV file's protein sequence identifier, InterPro consortium signature accession and E-value score columns (Fig. 2.5). The FASTA file parsed using Scikit-bio's FASTA file parser (Fig. 2.5)[136]. The results of these parsings are merged into a single DataFrame called **matches** that contains information that maps from InterProScan TSV data to a column of raw protein sequences. A summary of the attributes of AssignmentCacheWithMatches objects can be seen in Table 2.11.


Organism 1
InterProScan Annotation File (.tsv)
Protein Sequences (.fasta)
FASTA Sequences
Domain Annotation DataFrame
Sequence DataFrame
Assignment Cache With Matches
Combined DataFrame

```
>PRO1345
MLLLLARCFLVILASSL...
>PRO1346
LREEEAGAYAPLTAHL...
>PRO1347
RAFAPFRLAHALLAAL...
```

Protein ID	Signature	E-value
PRO1345	PF22015	1x10^-6
PRO1345	TF10750	2x10^-8
PRO1346	PF18570	4x10^-5
PRO1347	PF02435	7x10^-150

Protein ID	Sequence
PRO1345	MLLLLARCFLVILASSL...
PRO1346	LREEEAGAYAPLTAHL...
PRO1347	RAFAPFRLAHALLAAL...

Protein ID	Signature	E-value	Sequence
PRO1345	PF22015	1x10^-6	MLLLLARCFLVILASSL...
PRO1345	TF10750	2x10^-8	MLLLLARCFLVILASSL...
PRO1346	PF18570	4x10^-5	LREEEAGAYAPLTAHL...
PRO1347	PF02435	7x10^-150	RAFAPFRLAHALLAAL...

Figure 2.5: **How AssignmentCacheWithMatches are made from input InterProScan TSV and FASTA files.** AssignmentCacheWithMatches objects store property assignments, InterProScan annotations, and supporting protein sequences for a single organism. Such objects are generated from a FASTA file of an organism's proteins and InterProScan TSV file of domain annotations of these proteins. AssignmentCacheWithMatches objects possess a DataFrame that maps between InterProScan annotations and protein sequences.

Table 2.11: Attributes of AssignmentCacheWithMatches objects that are not possessed by AssignmentCache objects.

Name	**Type**	**Description**
matches	Attribute	Pandas DataFrame containing domain annotation information and protein sequences for an organism

2.5.2 The GenomePropertiesResultsWithMatches Class

The GenomePropertiesResultsWithMatches class extends the GenomePropertiesResults class via class inheritance [140]. In addition to the attributes, properties, and methods inherited from the GenomePropertiesResults class, GenomePropertiesResultsWithMatches class also possesses equivalents used for accessing supporting information. This supporting information includes domain annotation E-value scores and protein sequences that support the existence of property steps. A pandas DataFrame within each instantiated GenomePropertiesResultsWithMatches object stores this information. Unlike AssignmentCacheWithMatches objects, GenomePropertiesResultsWithMatches objects maintain supporting data for more than one organism.

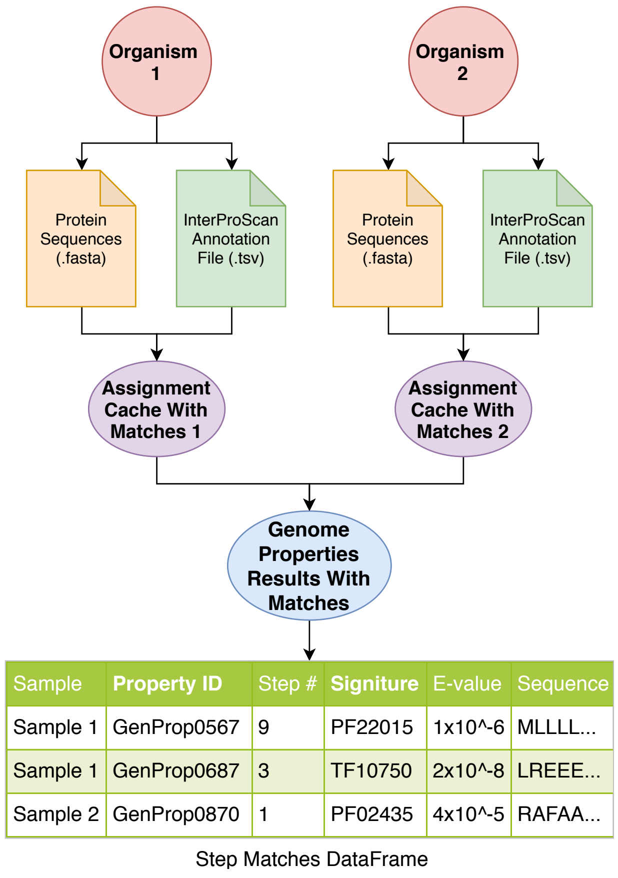
GenomePropertiesResultsWithMatches objects are generated by combining a series of AssignmentCacheWithMatches objects for different organisms (Fig. 2.6). During instantiation, the **matches** DataFrame (Table 2.11) of multiple input AssignmentCacheWithMatches objects are combined into a single more massive DataFrame. Sample names, genome property identifier, and step number columns are then used to index this DataFrame, allowing for fast lookups. Having a large combined DataFrame allows for E-value scores and sequences to be compared across organisms. During the creation of this DataFrame, domain annotations and proteins that do not support genome property steps are filtered out. The GenomePropertiesResultsWithMatches class provides a variety of convenience functions for accessing domain annotations for proteins that assist in the assignment of a property step. These functions are capable of providing filtered DataFrames that compare matches across organisms. Objects of the GenomePropertiesResultsWithMatches class can be also be used to generate FASTA files, using Scikit-bio, containing proteins that are predicted to carry out a property step in multiple organisms. A summary of the methods, properties, and attributes of GenomePropertiesResultsWithMatches objects can be seen in Table 2.12 and example code below.

Table 2.12: Methods, properties, and attributes of GenomePropertiesResultsWithMatches objects not possessed by GenomePropertiesResults objects.

Name	**Type**	**Description**
get_sample _matches	Method	Return the step matches DataFrame filtered to include matches from only one sample
get_property _matches	Method	Return the step matches DataFrame filtered to include matches from only one genome property
get_step _matches	Method	Return the step matches DataFrame filtered to include matches from only one genome property step

Table 2.12 continued from previous page

Name	**Type**	**Description**
get_supporting _proteins_for _step	Method	Return a list of Scikit-bio sequences objects for proteins that have domain annotations that support a specific property step to a FASTA file
write _supporting _proteins_for _step_fasta	Method	Write the protein sequences that have domain annotations that support a specific property step to a FASTA file
top_step _matches	Property	Return the matches DataFrame with only the matches with the lowest E-value for each sample and property step retained
step_matches	Attribute	Pandas DataFrame containing both domain annotations and sequences for proteins that support genome property steps in multiple samples


Organism 1
Organism 2
Protein Sequences (.fasta)
InterProScan Annotation File (.tsv)
Protein Sequences (.fasta)
InterProScan Annotation File (.tsv)
Assignment Cache With Matches 1
Assignment Cache With Matches 2
Genome Properties Results With Matches
Step Matches DataFrame

Sample	Property ID	Step #	Signiture	E-value	Sequence
Sample 1	GenProp0567	9	PF22015	1x10^-6	MLLLL...
Sample 1	GenProp0687	3	TF10750	2x10^-8	LREEE...
Sample 2	GenProp0870	1	PF02435	4x10^-5	RAFAA...

Figure 2.6: **How GenomePropertiesResultsWithMatches are built from AssignmentCacheWithMatches object derived from multiple organisms.** GenomePropertiesResultsWithMatches objects store property assignments, InterProScan annotations, and supporting protein sequences for multiple organisms. Such objects are generated by combining the AssignmentCacheWithMatches objects generated for multiple organisms. These caches are generated from per-organism FASTA files and InterProScan files. GenomePropertiesResultsWithMatches objects also retain the Property and step assignment DataFrames from Fig. 2.4 (not shown).

Example code for using GenomePropertiesResults objects

```
tree = parse_genome_properties_flat_file(properties_file_handle)

cache_one = parse_interproscan_file_and_fasta_file(ipr_file_one,
                                                    fasta_file_one)
cache_two = parse_interproscan_file_and_fasta_file(ipr_file_two,
                                                    fasta_file_two)
results = GenomePropertiesResultsWithMatches(cache_one, cache_two,
                                    properties_tree=property_tree)

results.get_step_matches('GenProp1764', 1)
Out:
```

Sample_Name	**Signature_Accession**	**Protein_Accession**	**E-value**	**Sequence**
C_chlorochromatii_CaD3	PF00994	NC_007514.1_1113	7.1e-19	MITV...
C_chlorochromatii_CaD3	PF00994	NC_007514.1_151	1.3e-31	MRAV...
C_chlorochromatii_CaD3	PF00994	NC_007514.1_1114	1.6e-29	MTFT...
C_luteolum_DSM_273	PF00994	NC_007512.1_2044	2.2e-29	MPSI...
C_luteolum_DSM_273	PF00994	NC_007512.1_147	1.3e-28	MAFT...
C_luteolum_DSM_273	PF00994	NC_007512.1_148	3.7e-26	MLTS...
C_luteolum_DSM_273	PF01507	NC_007512.1_1607	6.1e-38	MSSA...
C_luteolum_DSM_273	PF01507	NC_007512.1_1606	5.4e-40	MSRI...

```
# Retrieve matches only for hits with the lowest E-value
# and for a single step
results.get_step_matches('GenProp1764', 1, top=True)
Out:
```

Sample_Name	**Signature_Accession**	**Protein_Accession**	**E-value**	**Sequence**
C_chlorochromatii_CaD3	PF00994	NC_007514.1_151	1.3e-31	MRAV...
C_luteolum_DSM_273	PF01507	NC_007512.1_1606	5.4e-40	MSRI...

```
# Retrieve matches only for hits with the lowest E-value
# for a single property and for only one sample

results.get_property_matches('GenProp1764',
                             sample='C_chlorochromatii_CaD3',
                             top=True)
Out:
```

Step_Number	Signature_Accession	Protein_Accession	E-value	Sequence
1	PF00994	NC_007514.1_151	1.300000e-31	MRAE...
2	PF01687	NC_007514.1_1520	3.100000e-33	MRLI...

```
# For a given property step, write the protein
# that is most likely to carry out the step in each
# organism. These proteins have the lowest E-value
# for their domain annotation match.
with open('proteins.fasta', 'w') as fasta_file:
    results.write_supporting_proteins_for_step_fasta(fasta_file,
                                                     'GenProp1757',
                                                      2,
                                                      top=True)
```

2.5.3 AssignmentCacheWithMatches and GenomePropertiesResultsWithMatches Performance

For 1877 proteins from *Pelodictyon luteolum* DSM 273 (NCBI Taxa ID: 319225) and 1774 proteins from *Chlorobium chlorochromatii* CaD3 (NCBI Taxa ID: 340177), it took 80.4 ms ±1.7 ms (N = 80) and 89.4 ms ±4.0 ms (N = 80) to parse the FASTA and InterProScan TSV files and generate two AssignmentCacheWithMatches objects. These caches were then combined into a single GenomePropertyResultsWithMatches object in 4.2 s ±0.2 s (N = 20). The two assignment caches were found to take 21.3 and 15.3 MB of memory, respectively. The final GenomePropertyResultsWithMatches object required only 29.8 MB of memory. At over 4.2 seconds for joining only two caches, scaling analyses to large datasets may become a challenge and opportunities for optimizing this step should be pursued. The GenomePropertyResultsWithMatches object generated was subsequently serialized to JSON in 7.1 s ±1.3 s (N = 20). This speed may need to be optimized in the future. The proteins most likely to carry out GenProp1757 step number two in each sample were written to a FASTA file in 21.7 ms ±0.8 ms (N = 80) using the same GenomePropertyResultsWithMatches object's **write_supporting_proteins_for_step_fasta** method (Table 2.12).

2.6 Development of a File Storage Format and Database Interface for Storing Genome Property Assignments and Supporting Information

Once Pygenprop generates assignments of support for individual genome properties and steps, users many find it useful to be able to store these assignments for later use or dissemination. With the Genome Properties Perl library, property and step assignments can be saved to text files written in either a custom human-readable format (see github.com/ Micromeda/ pygenprop/blob/master/pygenprop/testing/test_constants/ _chlorochromatii _CaD3.txt) or JSON format. Both of these file types are created per-organism and do not contain supporting information such as annotation match scores or protein sequences. Because the Perl library creates these files per-organism, a large number of records must be tracked and managed if a researcher wants to compare assignments across multiple organisms. Also, if a user wants to retain information about domain annotations and protein sequences that support these assignments, then they would have to track and manage a series of additional InterProScan TSV and FASTA files for each organism. Tracking, managing, and sharing all these files would be difficult, so Pygenprop supports the creation of Micromeda files that store the information held within these three file types in a single file. This new file type can store data for multiple organisms allowing the transfer of entire datasets between users or computer systems.

2.6.1 Selection of a Data Storage Format

Micromeda's assignment results file format is based on SQLite3 [118]. During Pygenprop's development, several file formats were reviewed before the selection of this format. The types considered included custom text formats, custom binary formats, JSON, YAML Ain't Markup Language (YAML) [14], and Hierarchical Data Format version 5 (HDF5). Custom text or binary files were passed over as they would provide minimal advantages over existing off-the-shelf file types that offer similar performance with little to no development overhead. JSON and YAML were not selected as they are both text encoded and take up substantially more disk space than equivalent binary formats. HDF5 [58] is a binary format used for storing enormous arrays of data, allowing a user to define data structures inside the file. SQLite3 was chosen over HDF5 for the following reasons:

- It would take less time to design SQLite3 database tables than it would be to define our own custom HDF5 structures

- SQLite3's compatibility with a broad range of tools and programming languages
- SQLite3 uses structured query language (SQL) [43] allowing for compatibility with larger server-based database systems such as MySQL [47] and PostgreSQL [109, 118]

2.6.2 The Usage of SQLAlchemy to Provide Expanded Database Connectivity

Traditionally a relational database management system (DBMS), such as MySQL and PostgreSQL, is a server process that continually runs on a computer system (*i.e.*, they are daemons [102]) waiting for input from other programs or human users [47, 109]. Such databases are designed to handle connections from hundreds of applications or users simultaneously. Users and software communicate with relational DBMSs via a domain-specific language (DSL) called SQL. The language allows a user to define the structure of a database and add, remove, and query data [43]. In contrast to traditional databases, SQLite3 does not run as a server process. Instead, SQLite3 is a software library and associated file type that takes SQL strings as input and manipulates a single small file on disk according to instructions found within these strings [118].

SQLAlchemy [11] is a tool that acts as a compatibility layer that allows users to write software that can query and store data found in multiple types of relational DBMSs (Fig. 2.7). The tool also allows the user to define a relational database schema in a series of Python classes, access individual database records as Python objects (Fig. 2.7) and query a database using Python idioms. SQLAlchemy allows a user to create, update, delete, and query data from an SQL compatible DBMS without writing a single line in the SQL language [11].

Pygenprop uses SQLAlchemy to write assignment data and supporting match information to SQLite3 files. In the context of Pygenprop, these files are called Micromeda files. Also, due to the use of SQLAlchemy and through the changing of a single line of code, Pygenprop can write assignments and supporting data to more extensive, daemon-based, high-performance databases such as PostgreSQL or MySQL (Fig. 2.7).

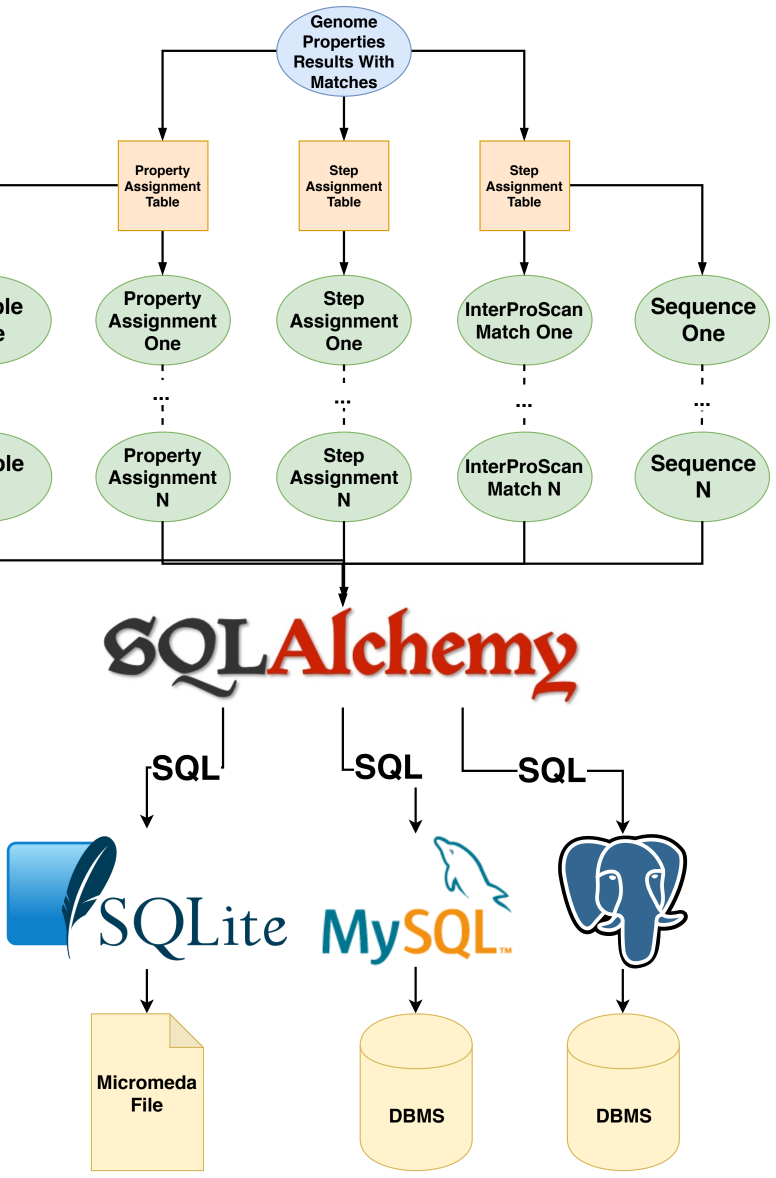


Figure 2.7: **How Pygenprop uses SQLAlchemy objects to write property assignments, InterProScan annotations, and protein sequences to Micromeda files and databases.** This process involves mapping GenomePropertiesResultsWithMatches objects to a series of SQLAlchemy objects representing individual relational database tables. The use of SQLAlchemy allows for GenomePropertiesResultsWithMatches objects to be written to a variety of relational DBMS.

2.6.3 The Schema of the Micromeda File Format

As with all relational database schemas, Micromeda's schema consists of a series of data tables. These tables were designed in such a way as to store property assignments, step assignments, and associated supporting information as compactly as possible. This compactness is essential as it allows users to keep files sizes, and thus transfer times, to a minimum allowing for quick dissemination of datasets. The optimizations that were chosen to support this goal are listed below:

- Only retain supporting information for steps that are assigned YES
- Only retain step assignments that are assigned YES (step assignments of NO are inferred using information from a GenomePropertiesTree object)
- Property and step assignments of support (*i.e.*, YES, NO, or PARTIAL) are stored as the numbers 0, 1, and 2, rather than strings, to save space
- The database schema was normalized to the 3rd normal form (3NF)[9] [13]

The Micromeda file's final relational table structure can be seen in the schema found in Fig. 2.8.

[9]Database normalization (see en.wikipedia.org/wiki/Database_normalization) is the process of splitting large data tables into smaller tables that are linked together and only store each piece of data in the dataset once. For example, a protein sequence only needs to be stored once if it is kept in a separate table.

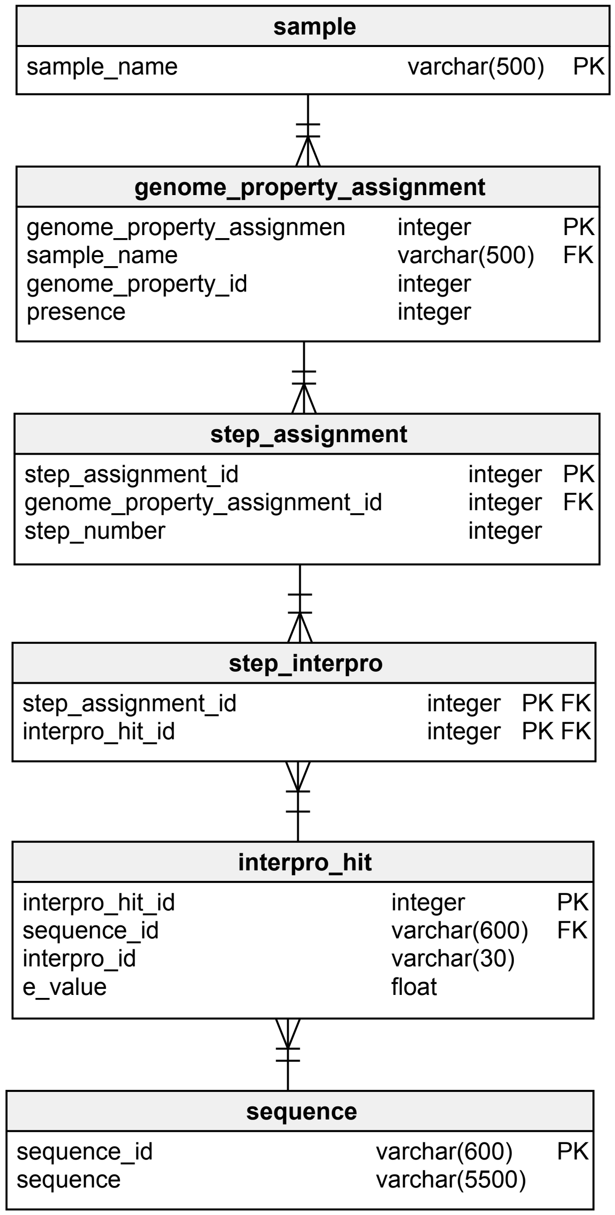

sample
sample_name varchar(500) PK
genome_property_assignment
genome_property_assignmen integer PK
sample_name varchar(500) FK
genome_property_id integer
presence integer
step_assignment
step_assignment_id integer PK
genome_property_assignment_id integer FK
step_number integer
step_interpro
step_assignment_id integer PK FK
interpro_hit_id integer PK FK
interpro_hit
interpro_hit_id integer PK
sequence_id varchar(600) FK
interpro_id varchar(30)
e_value float
sequence
sequence_id varchar(600) PK
sequence varchar(5500)

Figure 2.8: **The relational schema used by Micromeda SQLite3 files.** The schema consists of a series of data tables that are normalized (see Footnote 9) to 3NF. There are tables for samples, property assignments, step assignments, a mapping table between step assignments and InterPro annotations, InterPro annotations, and proteins sequences.

2.6.4 SQLAlchemy Classes Used By Micromeda

Pygenprop maintains five SQLAlchemy classes for representing relational tables and records. These objects represent individual property assignments, step assignments, InterProScan domain annotations, and protein sequences and are used to generate SQL statements for both creating SQL tables and database queries through SQLAlchemy's object-relational mapping (ORM) functionality. Attributes and Properties of these objects are detailed in Tables 2.13, 2.14, 2.15, 2.16, 2.17.

Table 2.13: Attributes of Sample objects.

Name	**Type**	**Description**
name	Attribute	Name of the sample (*e.g.*, an organism name)
property _assignments	Attribute	List of property assignment objects

Table 2.14: Properties and attributes of PropertyAssignment objects.

Name	**Type**	**Description**
assignment	Property	Return the property's assignment as YES, NO, or PARTIAL
identifier	Property	Return the property's identifier (*e.g.*, GenProp0078)
property _assignment _identifier	Attribute	Unique numeric identifier for a property assignment of a single sample
property _number	Attribute	Genome property identifier as a number (*e.g.*, the 0078 of GenProp0078)
numeric _assignment	Attribute	Property's assignment as the numbers 0, 1, or 2 (equal to YES, NO, or PARTIAL)
sample_name	Attribute	Name of the sample that the property assignment belongs to
sample	Attribute	Sample object that the property assignment belongs to
step _assignments	Attribute	List of step assignment objects belonging to a single property

Table 2.15: Attributes of StepAssignment objects.

Name	**Type**	**Description**
step _assignment _identifier	Attribute	Unique numeric identifier for a step assignment of a single sample
property _assignment _identifier	Attribute	Genome property identifier that the step belongs to as a number (*e.g.*, the 0078 of GenProp0078)
number	Attribute	Number of the step
property _assignment	Attribute	Property assignment object that the step assignment belongs to
interproscan _matches	Attribute	List of interproscan match objects that support the existence of property assignment

Table 2.16: Attributes of InterProScanMatch objects.

Name	**Type**	**Description**
interproscan _match _identifier	Attribute	Unique numeric identifier for an interproscan annotation of a single protein sequence
sequence _identifier	Attribute	Identifier of a protein sequence
interpro _signiture	Attribute	InterPro consortium signature accession of a domain found in a protein sequence
expected_value	Attribute	E-value of the match between a motif found in the protein and annotated domain
step _assignments	Attribute	List of step assignment objects that are supported by the InterProScan annotation
sequence	Attribute	Sequence object that the InterProScan annotation annotates

Table 2.17: Attributes of Sequence objects.

Name	**Type**	**Description**
identifer	Attribute	Identifier of a protein sequence
sequence	Attribute	Protein sequence of the protein

2.6.5 Reading and Writing Micromeda Files

Both GenomePropertiesResults and GenomePropertiesResultsWithMatches objects provide a **to_assignment_database** method (Tables 2.10 and 2.12). This method takes an SQLAlchemy engine object and uses it to write assignment and annotation information to a database. This engine object is created from a database uniform resource identifier (URI) [19] string that can point toward an SQLite3 file or a larger process-based relational database. Once called, this method converts the results object's pandas DataFrames to a series of SQLAlchemy objects and then use SQLAlchemy to write the information contained within these objects to the engine's connected database. Code for writing Micromeda files can be found below.

```
# A SQLAlchemy engine object can be created
# for a variety of SQL databases

# Write to a Micromeda file
engine = create_engine('sqlite:///data.micro')
results.to_assignment_database(engine)

# Write to a PostgreSQL database
db_uri = 'postgresql://scott:tiger@localhost:5432/mydatabase'
engine2 = engine = create_engine(db_uri)
results.to_assignment_database(engine2)
```

The reading of assignments from Micromeda files or databases is facilitated by the **load_assignment_caches_from_database** and **load_assignment_caches_from_database_with_matches** functions of Pyegenprop's results module. These functions produce lists of AssignmentCache or AssignmentCacheWithMatches objects, respectively. These lists can the be combined into GenomePropertiesResults or GenomePropertiesResultsWithMatches objects. Code for reading Micromeda files can be found below.

```
tree = parse_genome_properties_flat_file(properties_file_handle)
engine = create_engine('sqlite:///data.micro')

caches = load_assignment_caches_from_database_with_matches(engine)

results = GenomePropertiesResultsWithMatches(*caches,
                                             properties_tree=property_tree)
```

2.6.6 Micromeda File Performance

For 1877 proteins from *Pelodictyon luteolum* DSM 273 (NCBI Taxa ID: 319225) and 1774 proteins from *Chlorobium chlorochromatii* CaD3 (NCBI Taxa ID: 340177), a representative GenomePropertiesResultsWithMatches was generated. This object was found to be able to be serialized to a Micromeda file in 29.4 s ±0.8 s (N = 20). This Micromeda file was found to be approximately one fifth the size of the original files used to create the GenomePropertiesResultsWithMatches objects (Fig. 2.9). The same Micromeda file was found to be able to be reconstituted back into GenomePropertiesResultsWithMatches object in 6.3 s ±1.3 s (N = 20).

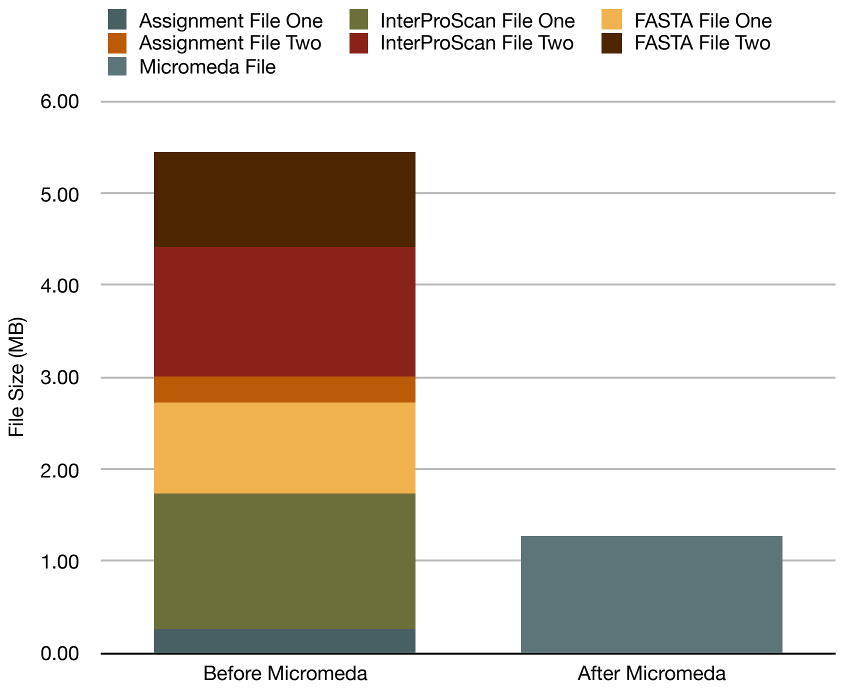


Figure 2.9: **Comparison of a Micromeda file's size to the cumulative size of the input files used to create it.** Micromeda files take up significantly less space than the input files used to make them. Previously, pathway analysis for two samples required the tracking of at least six files. With Micromeda, the information contained within these files is combined and reduced, allowing for property assignment and supporting information to be stored in a single file and one fifth of the disk space.

2.7 Development of an In-Memory Transfer Format for Property Assignments and Supporting Match Information

When writing software for pathway analysis on high-performance computing (HPC) infrastructure, it may be of interest to transfer pathway datasets between machines via memory-to-memory transfers. It may also be useful to be able to store pathway datasets in an in-memory cache to accelerate bioinformatics web application performance. Micromeda-Server uses such a cache (see Section 3.3). GenomePropertyResultsWithMatches objects can be serialized into a format optimized for these use cases. A key performance metric for both of these use cases is serialization/deserialization speed of GenomePropertyResultsWithMatches objects to and from the format. The memory space taken up by the format is less of a concern. The format ultimately chosen was MessagePack [59](see msgpack.org).

Several formats were reviewed to support memory-to-memory transfer and in-memory caching of GenomePropertyResultsWithMatches objects. These included: JSON, Google Protocol Buffers [157], and MessagePack. JSON has been the gold standard for transferring information between computer systems, especially over the internet, for over a decade [97]. Because JSON is a text format, several binary alternatives such as ProtoBuffs and MessagePack have emerged that can store the same amount data in less space, allowing for faster transfers [59, 157, 90, 38, 15]. These formats also have much better serialization performance than JSON [90, 38, 15]. For Pygenprop, MessagePack was selected over Protocol Buffers because pandas provide built-in methods, though currently experimental, for serializing DataFrames directly to MessagePack. MessagePack also serializes four times faster than Protocol Buffers [38].

GenomePropertyResults and GenomePropertyResultsWithMatches objects have a method called **to_msgpack** that supports serialization of these objects to a MessagePack binary string. Internally, this function calls pandas' **to_msgpack** function on a list of the object's pandas DataFrames (Tables 2.10 and 2.12), returning a MessagePack binary. For deserialization, the process is run in reverse, using the function **load_results_from_msgpack** of Pygenprop's results module, converting the MessagePack binary stream back to a GenomePropertyResultsWithMatches object. Code for serializing and deserializing GenomePropertyResultsWithMatches objects can be found below.

```
message = results.to_msgpack()

new_results = load_results_from_msgpack(message)
```

For 1877 proteins from *Pelodictyon luteolum* DSM 273 (NCBI Taxa ID: 319225) and 1774 proteins from *Chlorobium chlorochromatii* CaD3 (NCBI Taxa ID: 340177), a representative GenomePropertiesResultsWithMatches was generated. This object serialized to a MessagePack binary in 29.5 ms $\pm$0.3 ms (N = 80) and deserialized in 54.0 ms $\pm$0.9 ms (N = 80). The MessagePack binary required 4.2 MB of memory.

2.8 Verification and Automated Testing

To validate that property and step assignments of support were calculated correctly, the property and step assignments generated by Pygenprop were compared to those produced by the original Genome Properties Perl library. Based on a test proteome of 3,000 proteins, property assignments differed from those of the original Perl library by 2.9%, due to an error corrected Pygenprop (github.com/ebi-pf-team/genome-properties/issues/30).

The proteomes of *Pelodictyon luteolum* DSM 273 (NCBI Taxa ID: 319225) and *Chlorobium chlorochromatii* CaD3 (NCBI Taxa ID: 340177) were used to validate that assignments and supporting information did not change before and after serialization of GenomePropertiesResults and GenomePropertiesResultsWithMatches objects. It was found that Pygenprop could correctly reconstitute these objects (*i.e.*, they had the same cryptographic checksum value before and after) from both SQLite3 files and MessagePack binary strings.

Pygenprop has extensive end-to-end and unit tests for its codebase. There are currently 104 unit tests that validate the functionality of the majority of code functions. These tests cover 94% of lines in the Pygenprop's codebase. Pygenprop's end-to-end test involves the making of a JSON serialization of a GenomePropertiesResults object created from the InterProScan results of two organisms. This test can be run against newly released versions of the Genome Properties database to ensure compatibility over time.

2.9 Future Improvements

Although Pygenprop is currently feature-rich and available to the public, there are still several improvements that could be used to make the library more useful. Some of these are related to making the library more computationally efficient, whereas others are related to improving its analysis capabilities. Several potential improvements are highlighted below.

2.9.1 Adding The Ability To Serialize GenomeProperty Objects to DESC Files

The Genome Properties database consists of a series of flat files hosted in a GitHub repository (see github.com/ebi-pf-team/genome-properties). New properties are added to the database via adding additional property **DESC** files (see Section 2.1.1) to the repository. Information about existing properties can be changed by editing the database's existing **DESC** files. These **DESC** files are plain text and currently must be created and edited manually using a plain text editor. Changes to the database are added via fork and pull workflow (see atlassian.com/git/tutorials/comparing-workflows/forking-workflow) where a user makes a copy of the database on GitHub, makes changes to this copy, and then requests that their copy's differences be merged into the central Genome Properties database repository via a GitHub pull request. The manual creation and editing of genome properties **DESC** files require one to thoroughly understand the **DESC** format and write or edit property files with no errors. For example, before writing their first property file, users must know all twenty-three two-letter keycodes (see Subsection 2.1.1) used by the **DESC** file format. An easier way to create or edit property files, rather than manual editing, would be of use for those users who are not familiar with the **DESC** format.

GenomeProperty objects and their related child objects (see Section 2.2) are designed to represent individual genome properties and their steps, functional elements, and evidence. It is currently possible to take a Genome Properties **DESC** file (see Subsection 2.1.1) and use Pygenprop to turn the file's contents into objects, but it is not possible to do the reverse. Being able to serialize a GenomeProperty object and its children to a **DESC** file may be useful to users who want to add new genome properties to the Genome Properties database or edit the database's existing properties. For example, the user could use Pygenprop, loaded within a Jupyter Notebook [88], to parse an existing property's **DESC** file into a GenomeProperty object and its children. Afterwards and within the same notebook, the user could programmatically edit the newly loaded property objects' attributes. If the ability to write **DESC** files was added to Pygenprop, then this updated property object representing the edited property could then be written to an updated **DESC** file. This updated file could then replace the old version of the property's **DESC** file in the Genome Properties database, which would result in the property being updated. A user could also build, within a Jupyter Notebook, a new GenomeProperty object programmatically that represents a new genome property. This new object could be written to a new **DESC** file. The user could then add this new file to the database's existing pool of flat files, which would, in turn, add to a new property to the database. A property's steps and evidence can also be added or edited using the same object-oriented methods. Pygenprop writes these modified or added objects to the **DESC** file at the same time as their parent property object. The ability to add and edit properties of the database using a notebook interface would make it drastically more accessible for less experienced users to contribute to the Genome Properties database as they would not have to remember the details of the **DESC** format. The ability to write **DESC** files would also make it easier to generate new properties from data contained in other pathway databases by allowing the porting process to be more easily automated.

2.9.2 Adding A Method to Calculate Fold Change in the Presence of Genome Properties and Steps

The GenomePropertyResults class currently possesses a method called **get_results_summary** (Subsection 2.10) that is used to summarize the presence and absence of genome property or step assignments across multiple organisms. This method counts the occurrence of YES, NO, and PARTIAL assignments for each organism in a dataset. For example, when comparing the virulence properties of *E. coli* K12 (NCBI Taxa ID: 1010810) and O157:H7 (NCBI Taxa ID: 83334) (see github.com/Micromeda/pygenprop/blob/master/docs/source/_static/tutorial/tutorial.ipynb), this method would show users that O157:H7 has more of these properties assigned YES than K12. This increase could be further quantified by adding a **calculate_assignment_fold_change** method to the GenomePropertyResults class. This method could take the name of a single sample to use as a baseline and, for the other samples, calculate fold changes in the frequency of YES, NO, and PARTIAL assignments of support for specific sets of genome properties or steps.

2.9.3 Improving Pygenprop's Memory and Time Performance

Throughout this chapter, the performances of different components of Pygenprop have been reviewed. Two of these performance numbers were found to be quite weak and should be addressed. These are the times taken to read or write Micromeda files and the time required for serializing GenomePropertiesResultsWithMatches objects to JSON. Micromeda's server component uses both of these functions extensively, and they are a significant source of lag in Micromeda's current web interface.

The performance of reading and writing Micromeda files could be improved by reconfiguring how these files are written (*i.e.*, not writing assignments one at a time) or using an alternative faster file format such as MessagePack binary files. The creation of such MessagePack files would be facilitated by GenomePropertiesResultsWithMatches **to_msgpack** method.

The suggested improvements to Micromeda-Server discussed in Subsection 3.7.2 would change the requirements for the JSON generated from GenomePropertiesResultsWith-Matches objects. The object would only be required to generate JSON that contains property assignments or step assignments. This change would speed up JSON generation as assignments would no longer have to be added to a property tree nodes one at a time. Assignment DataFrames could be converted to JSON using methods built into pandas.

2.10 Summary

Pygenprop is a Python library for the programmatic utilization of Genome Properties data. The software has methods for both the exploration of the Genome Properties database and the property and step assignments of multiple organisms. It also provides ways of accessing the domain annotations and protein sequences that were used to calculate these assignments. Micromeda files can be used to store and transfer entire pathway analysis datasets allowing for increased reproducibility.

When compared to existing pathway analysis libraries, Pygenprop is unique because it is currently the only Genome Properties library written for Python. The previous Genome Properties library released by EBI was written in Perl. Unlike this Perl library, Pygenprop provides numerous methods for comparing property assignments across multiple organisms. Also, the library allows for the creation of Micromeda files, greatly reducing the number of files required for pathway analysis. There are a variety of pathway analysis software libraries written to support either the Pfam [167, 125, 165, 34] and Metacyc [75] databases. However, the goal of these libraries is to either help users download Pfam information via the database's web application programming interface (API) [34] (see en.wikipedia.org/wiki/Application_programming_interface) or help them generate graph visualizations of metabolic networks [125]. In terms of analysis functionality, Pygenprop is closest to clusterProfiler [165]. However, clusterProfiler is only compatible with the R programming language [128], whereas Pygenprop is built for Python. None of the libraries cited are integrated with Python's Scipy [78] or machine learning ecosystems in the same way as Pygenprop is, nor do they have its automation enabling features such as the tracking of supporting information or the serialization of entire datasets to files, databases, or in-memory transfer formats. This level of automation leaves Pygenprop in a unique position to be used for the development of future integrated pathway analysis tools, such as Micromeda.

With regards to Micromeda, Pygenprop provides the backbone to its server component. In this role, the library gives Micromeda's pathway visualization application JSON trees of properties and their assignments, JSON strings containing information about individual properties and FASTA files of protein sequences that support the existence property steps across multiple organisms.

Chapter 3

Development of a Web Application Programming Interface for Genome Properties Data

As discussed in Chapter 1, one of the goals of Micromeda is to provide users with an interface for visualizing the presence and absence of genome properties across multiple organisms. Users can access Micromeda's heat maps through a web application that generates them from uploaded Micromeda file data. The use of a web application has several advantages that are discussed in Subsection 1.4.1. Micromeda's web application consists of two components: client and server. The UI client, which draws the heat maps, is supported by a web server process that parses Micromeda files and provides a web API for accessing these file's contents. This chapter will discuss the server component of Micromeda, called Micromeda-Server, including the services it provides and its implementation. The client web application that uses these services is discussed in Chapter 4. Source code for Micromeda-Server is located at github.com/Micromeda/micromeda-server.

3.1 Overview of Web Servers

The World Wide Web and associated web applications are ideal delivery mechanisms for data analysis software such as Micromeda [18]. Such applications follow a client-server architecture [145] (see en.wikipedia.org/wiki/Client-server_model) where the code running in the user's browser is called the client. The user acquires this client by downloading it into their browser like any other webpage. If the client requires external data, then it can request this information from a server process running on a server computer system[1]. Requests to the server are made, via Hypertext Transfer Protocol (HTTP) [53], using a series of uniform resource locator (URL) addresses [17] (*i.e.*, web addresses) that return specific types of data. These addresses are known as **endpoints** (Section 3.4) and form a web API.

3.2 Micromeda-Server Workflow and Implementation

Micromeda-Server is designed to provide a web API to client applications that require access to information about the Genome Properties database. The server also provides an API for accessing temporarily stored property assignments, step assignments, and supporting information for user-supplied datasets. Micromeda-Server is written in Python and utilizes the Flask web development framework [63] (Fig. 3.1) to map Python functions for handling specific web API requests to server URL addresses (*i.e.*, "endpoints"). Information about the Genome Properties database is supplied to Micromeda-Server via a **genomeProperties.txt** release file (Fig. 3.1 and Subsection 2.1.1). Property assignments, step assignments, and supporting information for user datasets are supplied via user-uploaded Micromeda files (Fig. 3.1). These Micromeda files are parsed into GenomePropertyResultsWithMatches objects (Subsection 2.5.2) that are later stored in an in-memory Redis cache [67] in MessagePack format [59] (Fig. 3.1, Fig. 3.2 and Section 2.7). In the context of Micromeda-Server, the contents of each uploaded and cached Micromeda file is called a dataset. A single Micromeda file, stored on the same server computer system as Micromeda-Server, can also be provided to Micromeda-Server during start-up for use as a default dataset (Fig. 3.1). This default dataset is also parsed to a GenomePropertiesResultsWithMatches object and is used to supply data to Micromeda-Server's API if users have not uploaded any Micromeda files. The standard workflow for starting and then using Micromeda-Server is the following:

1. Start Micromeda-Server while providing a **genomeProperties.txt** file and an optional default Micromeda file (Fig. 3.1)
2. The **genomeProperties.txt** file is parsed to a GenomePropertiesTree object

[1]For the rest of the thesis, the term **server computer system** is used to refer to the physical hardware on which server software is run. The term **server** is used to refer to a software process that provides users or applications with data.

3. The default dataset Micromeda file is parsed to a GenomePropertiesResultsWithMatches object
4. The client application sends a user-supplied Micromeda file to the server via the upload endpoint (Fig. 3.2)
5. The user-supplied Micromeda files are parsed to a GenomePropertiesResultsWithMatches object that is later stored in the Redis cache in MessagePack format (Fig. 3.2)
6. The server supplies the client with a dataset key that is unique to each uploaded Micromeda file (Fig. 3.2)
7. The client can later supply this dataset key to the server during proceeding API requests to get information from the previously uploaded Micromeda file (Fig. 3.2)
8. If the client provides no dataset key then the server supplies information about the default dataset during API requests

Each GenomePropertiesResultsWithMatches object cached to Redis is given a time-to-live (TTL) value [64] (see en.wikipedia.org/wiki/Time_to_live). Users can set this value for any period, such as minutes or days. After the TTL of the cached object is exceeded, the object is flushed from the cache; the user will then have to re-upload their Micromeda file. The default TTL used is six hours. During each API request, if a dataset key is provided, the MessagePack-formatted GenomePropertiesResultsWithMatches object is grabbed from the cache and reconstituted into its original form (Fig. 3.2). During the API call this reconstituted GenomePropertiesResultsWithMatches object's methods are used to supply data to the client (Fig. 3.2 and Fig. 3.3). Further details on these endpoints are provided in Section 3.4.

Micromeda files contain information about both assignments and supporting information used in their creation. This supporting information, such as protein sequences, can take up substantial disk space. Permanently storing such information would be prohibitive in terms of both hardware and maintenance costs. In response, Micromeda-Server was designed to store uploaded datasets temporarily. Micromeda-Server does not have a user login system [2] and uploads to the server are done anonymously.

[2]A user login system is very complex to build and maintain. Code for tracking user names, passwords, and emails must be generated. This user information must be stored in a cryptographically secured and anonymized database. Code for handling logins, logouts, and secure password changes would also have to be implemented. Having Micromeda file upload be anonymous drastically reduced the complexity of Micromeda-Server's development and future deployment.

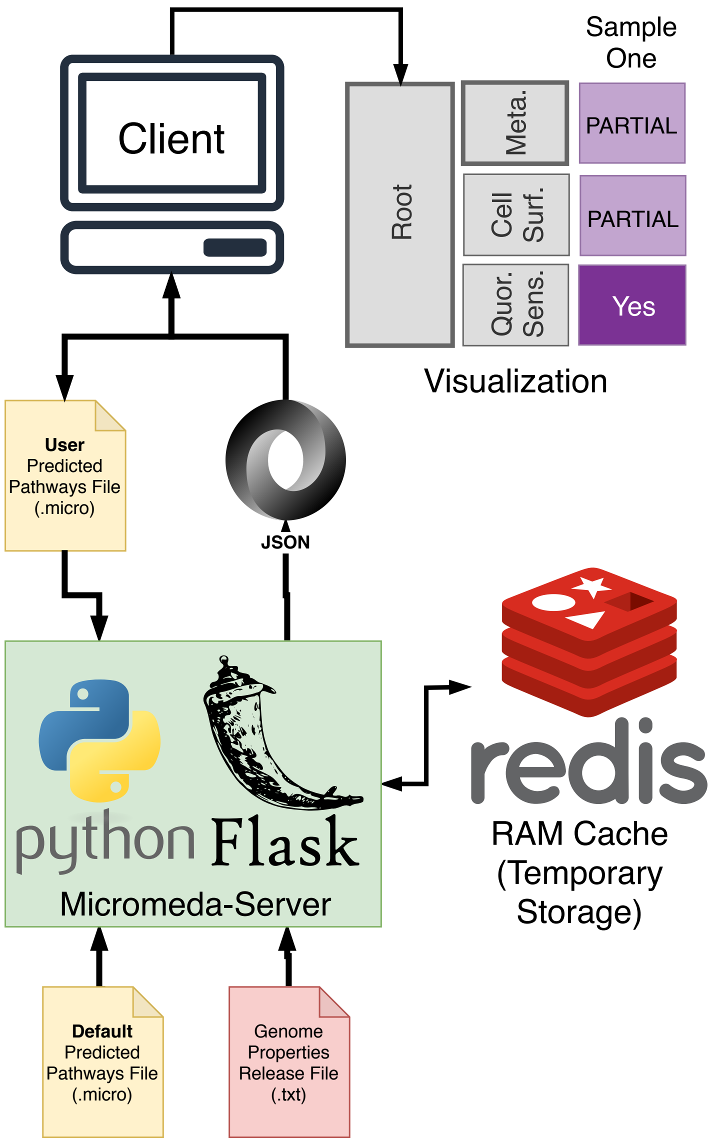

Client
Sample One
Root
Meta.
PARTIAL
Cell Surf.
PARTIAL
Quor. Sens.
Yes
Visualization
User Predicted Pathways File (.micro)
JSON
python Flask
Micromeda-Server
redis
RAM Cache (Temporary Storage)
Default Predicted Pathways File (.micro)
Genome Properties Release File (.txt)

Figure 3.1: **Components of Micromeda-Server's software architecture.** The web server code was written in Python using the Flask web framework [63]. Micromeda-Server is supported by a Redis cache and a series of text files. A genomeproperties.txt file supplies data for the generation of Genome Properties DAG. Micromeda files, either default or uploaded, provide information about property assignments, step assignments, and supporting information of multiple organisms.

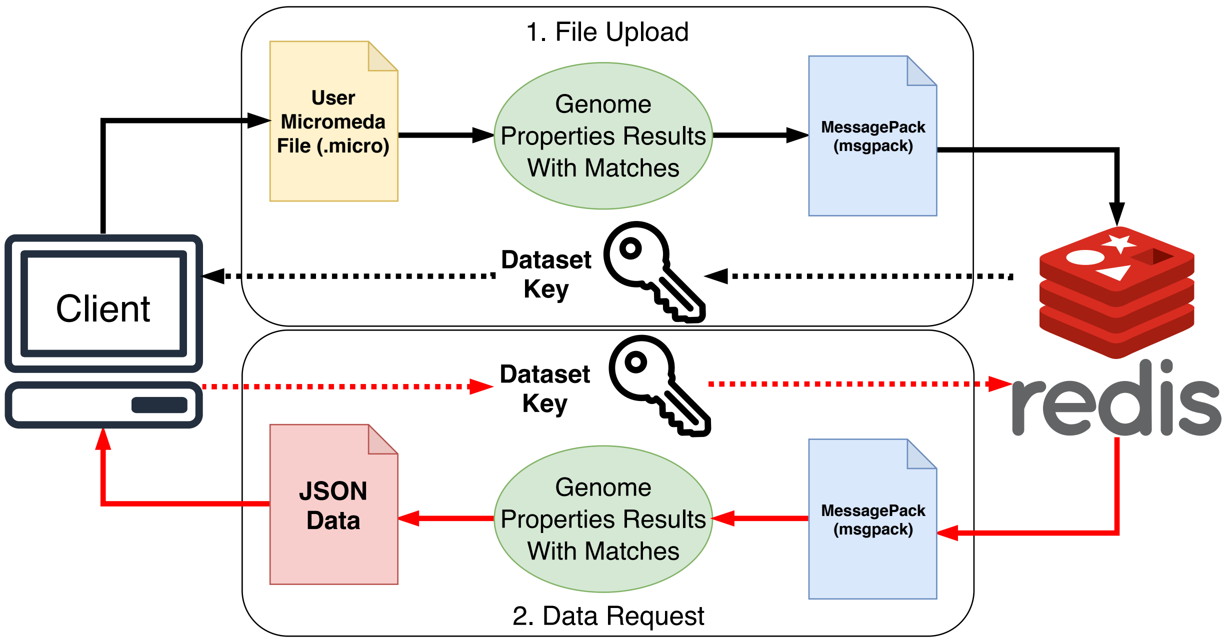


Figure 3.2: **How uploaded Micromeda files are cached to Redis by Micromeda-Server.** The caching process involves the generation of GenomePropertiesResultsWithMatches objects for each uploaded file. These objects are cached in Redis and reconstituted between API calls. Micromeda-Server uses methods possessed by these reconstituted GenomePropertiesResultsWithMatches objects to produce responses that are sent back to a web client application. Each uploaded file is assigned a dataset key that is provided to the client. The client later uses this key to request data from a specific Micromeda file.

The number of simultaneous users that Micromeda-Server can support almost entirely depends on the server computer systems the software is run on and the deployment strategy used (see Section 3.6). Micromeda-Server is horizontally-scalable (see en.wikipedia.org/wiki/Scalability#Horizontal_or_Scale_Out), which means its performance can be improved by running multiple copies of the software across multiple server computer systems. On a single computer, the main bottleneck of Micromeda-Server is the processing power required to parse uploaded Micromeda files. For example, it can take several minutes for a Micromeda file containing information from forty bacterial genomes to be parsed and stored within Redis. This file parsing process can be parallelized across CPU cores, with one core taking several minutes for parse each Micromeda file uploaded. Thus, the maximum number of simultaneous users who can upload Micromeda files in parallel is limited by the number of CPU cores inside the server computer system on which Micromeda-Server is

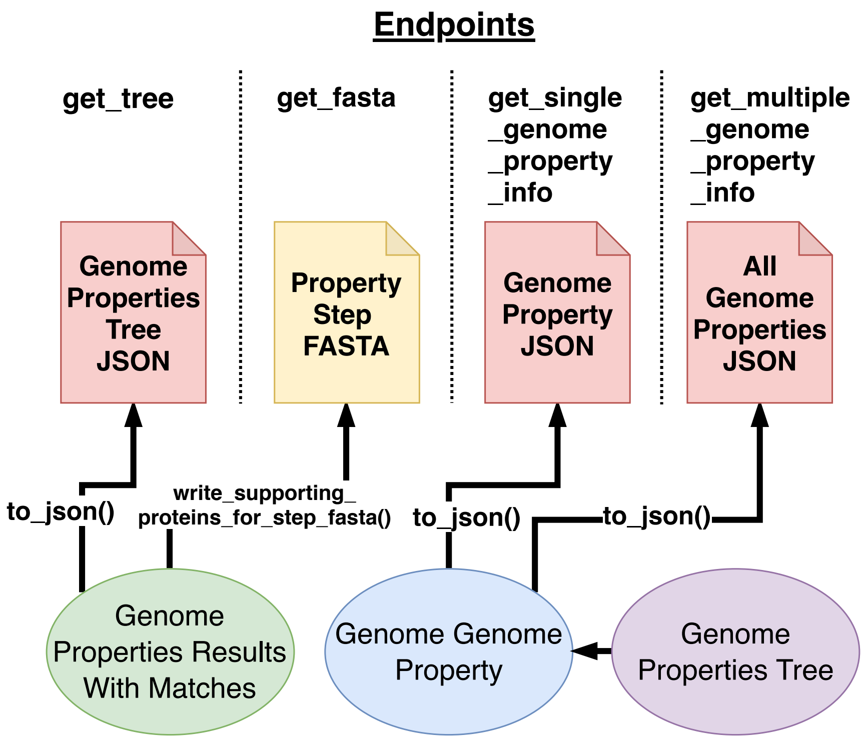

Endpoints
get_tree
get_fasta
get_single _genome _property _info
get_multiple _genome _property _info
Genome Properties Tree JSON
Property Step FASTA
Genome Property JSON
All Genome Properties JSON
to_json()
write_supporting_ proteins_for_step_fasta()
to_json()
to_json()
Genome Properties Results With Matches
Genome Genome Property
Genome Properties Tree

Figure 3.3: **Endpoints that Micromeda-Server presents for accessing Genome Properties and Micromeda file data.** These endpoints return JSON documents and FASTA files that are generated using methods of GenomePropertiesResultsWithMatches objects, GenomeProperty objects, and GenomePropertiesTree objects.

hosted. Once the contents of Micromeda files are stored in Redis, and information is being retrieved from this cache, the number of active users can be drastically increased as individual client endpoint HTTP requests take only a few CPU cycles to complete. In terms of memory usage, a Micromeda file containing information from forty bacterial genomes only takes up a few hundred MB of RAM once uploaded. A server with thirty-two GB of RAM should support close to one hundred simultaneous clients requesting information from previously uploaded Micromeda files.

3.3 Use of Redis for Dataset Caching

Python, due to limitations in its default cPython interpreter [156], is only capable executing one compute thread [133] (see en.wikipedia.org/wiki/Thread_(computing)) at a time [12]. This limitation causes problems for web server APIs that are required to handle multiple requests from clients simultaneously. In response, the majority of Python web frameworks, which provide boilerplate code for writing API endpoints, are designed to run multiple copies of the Python web server code, which each handle separate endpoint requests (Fig. 3.4). Flask is one such framework [63]. These codes are run in separate processes (see en.wikipedia.org/wiki/ Thread_(computing)#Threads_vs._processes) and do not share a memory space (Fig. 3.4). Thus, any in-memory objects created for one API request are not shared with the other parallel requests, which are being handled by different processes (Fig. 3.4). Also, there is no guarantee that subsequent API requests from a single web client will be mapped repeatedly to the same API server process (Fig. 3.4). This lack of mapping between subsequent requests causes a problem as a GenomePropertiesResultsWithMatches object created by the upload of a Micromeda file would be stored in one process and would not be available to other processes that future client HTTP requests may be mapped to (Fig. 3.4). One way of circumventing this process isolation issue is to store data to be shared between web server processes in an external process that is used as a cache (Fig. 3.2 and Fig. 3.4). This way, all web API processes have one place where they can request shared data. Micromeda-Server uses Redis as this caching process. Redis is a caching server that stores keyed Micromeda file data RAM.

Micromeda-Server uses Redis to cache GenomePropertiesResultsWithMatches objects, in MessagePack format, for use by multiple request handling processes (Fig. 3.2 and Fig. 3.4). Micromeda-Server generates these GenomePropertiesResultsWithMatches objects from Micromeda files uploaded to the server. During API requests where a client requires data from a specific uploaded file, responding API processes can each pull a MessagePack formatted GenomePropertiesResultsWithMatches object, representing the uploaded file, from the Redis cache. Subsequently, each GenomePropertiesResultsWithMatches object can be reconstituted within its process and its methods used to gather data for a API request response (Fig. 3.2 and Fig. 3.4). Rapid serialization of MessagePack to GenomePropertiesResultsWithMatches objects allows for this design pattern (see Section 2.7).

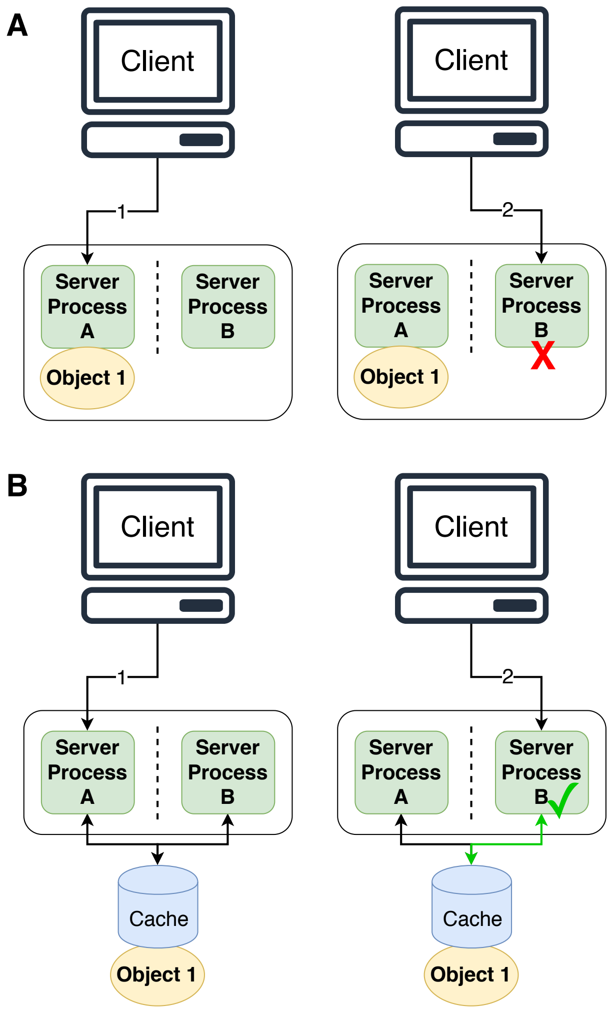


Figure 3.4: **How requests directed towards Python web endpoints are spread out across multiple processes and sharing data between these processes is difficult.** (A) Processes cannot share in-memory objects directly due to operating system enforced process boundaries. (B) In order for data to be shared between request handling processes, it must be stored by a third central process, such as a cache or database.

3.4 Application Programming Interface Endpoints

Micromeda-Server provides several endpoints for supplying web clients with information about individual genome properties and information from uploaded Micromeda files. These endpoints were written using the Flask Python web framework [63] and are represented by **clean URLs** (see en.wikipedia.org/wiki/Clean_URL) where some information that would normally be stored as HTTP GET parameters are stored in the URL path (Fig. 3.5). Flask was chosen due to its simplicity as compared to more comprehensive frameworks such as Django [69]. The endpoints also follow a representational state transfer (REST) architecture [52] (see en.wikipedia.org/wiki/Representational_state_transfer). These endpoints and their implementation are summarized in Table 3.1, Fig. 3.4, Fig. 3.5, and detailed in subsections below.

https://micromeda.uwaterloo.ca/fasta/genprop0526/1?dataset_key=FXDABADS&all=false

Figure 3.5: **Example of a URL that a client would use to request information from Micromeda-Server.** Micromeda-Server uses data held within URLs to figure out what information to return for a given HTTP request. The example URL depicted is used to download a FASTA file containing the top proteins (*i.e.*, those with the lowest E-value domains) that support GenProp0526 step one for dataset FXDABADS. The URL path variables are in blue and the HTTP GET parameters are in green. Note that the URL displayed is an example and does not point towards an active copy of Micromeda-Server or Micromeda-Client. Links to a demonstration of the client interface can be found in Section 4.5.

Table 3.1: Five endpoints of Micromeda's server component. By using these endpoints, clients can request data about individual genome properties, upload Micromeda files, and request information about stored assignment databases.

Python Function Name	**Endpoint URL**	**HTTP Request Types**	**URL Path Variables**	**GET Parameter Variables**	**Return Value**
upload	/upload	GET, POST	None	None	JSON containing a dataset key that can be used by future API requests to access information from the uploaded Micromeda file

Table 3.1 continued from previous page

Python Function Name	**Endpoint URL**	**HTTP Request Types**	**URL Path Variables**	**GET Parameter Variables**	**Return Value**
get_tree	/genome_properties_tree	GET	None	dataset_key (optional)	JSON tree representing all properties in the current Genome Properties database with each node annotated with a list of YES, NO, PARTIAL assignments for each organism in a dataset
get_single_genome_property_info	/genome_properties/<string: property_id>	GET	property_id	None	JSON containing information about a genome property such as a description of it and a list of equivalent records from other databases (*e.g.*, KEGG [86], MetaCyc [85])
get_multiple_genome_property_info	/genome_properties	GET	None	None	JSON array containing information about all genome properties in the database. Each property is given a description and a list of equivalent records from other databases (*e.g.*, KEGG, MetaCyc)

Table 3.1 continued from previous page

Python Function Name	**Endpoint URL**	**HTTP Request Types**	**URL Path Variables**	**GET Parameter Variables**	**Return Value**
get_fasta	/fasta/ <string: property_id>/ <int:step _number>	GET	property_id, step_number	dataset_key (optional), all (optional)	FASTA file containing either all or the top proteins (*i.e.*, those with the lowest E-value domain annotations) supporting the existence of a given property step. A dataset key can be provided to specify a dataset

The **upload** endpoint accepts the client upload of a Micromeda file and returns a hexadecimal encoded universally unique identifier (UUID) key [93] (see en.wikipedia.org/wiki/ Universally_unique_identifier) to the client. After upload, the Micromeda file is parsed and transformed into a GenomePropertiesResultsWithMatches object. This object is then serialized to MessagePack using the object's **to_msgpack** function (Table 2.12) and the resulting binary is cached in Redis using the Redis Python library [103] (Fig. 3.2). During the previous process, a UUID, to be used as a dataset key, is generated using Python's built in UUID generation function [149]. This UUID is used as the key for requesting the recently stored MessagePack serialization from the Redis cache (Fig. 3.2). Micromeda-Server returns the key to the client application in response to the file upload. The client can provide this key to other API endpoints to receive data from the uploaded Micromeda file (Fig. 3.2).

The **get_tree** endpoint provides the client with a JSON tree representing all properties and steps in Genome Properties database (Fig. 3.6). This tree represents parent-child relationships between properties. Step nodes are also attached to their parent genome property nodes and act as leaves (Fig. 3.6). Note that this endpoint returns a tree rather than a DAG (Fig. 3.6). In this tree, properties that would have had two parents in the Genome Properties DAG (Subsection 1.4.2) are duplicated (Fig. 3.6). Each property and step node in the tree is annotated by a list of assignments of support (*i.e.*, YES, NO, or PARTIAL), one for each sample in a previously uploaded or default Micromeda file (Fig.

3.6). The **get_tree** endpoint can take a **dataset_key** HTTP GET parameter variable (Table 3.1). If a dataset key generated by the previous upload of a Micromeda file is assigned to this variable, then the assignments of support stored in the key's associated Micromeda file are returned. The dataset key is used to request a MessagePack formatted GenomePropertiesResultsWithMatches object, representing the uploaded Micromeda file, from the Redis cache. Once reconstituted, the GenomePropertiesResultsWithMatches object's **to_json** method (Table 2.12) is called to generate the tree JSON provided by the endpoint. This newly built JSON is returned to the client. If no dataset_key is provided to the endpoint, then the default dataset GenomePropertiesResultsWithMatches object is used to generate the tree JSON for the endpoint. The default dataset GenomePropertiesResultsWithMatches object is built from a default dataset Micromeda file that is provided to Microemda-Server during the tool's initial start up.

The **get_single_genome_property_info** endpoint takes a genome property identifier as a URL parameter (Table 3.1). This genome property identifier is used query for a matching GenomeProperty object (Section 2.2.1), representing the property whose identifier is specified, from a global GenomePropertiesTree object (Section 2.2.7) created on Micromeda-Server's start up (Section 3.2). If found, this GenomeProperty object's **to_json** method (Table 2.2) is called to create a JSON document containing the property's information. The endpoint returns this document. The JSON document contains the property's name, its description, and a list of equivalent records for the pathway that property represents found in other pathway databases (Table 2.2).

When the **get_multiple_genome_property_info** endpoint is called, the **to_json** method (Table 2.2) is called for every GenomeProperty object (Section 2.2.1) that is a child of the GenomePropertiesTree object (Section 2.2.7) created on Micromeda-Server start up. Each of the resulting JSON strings generated is placed into a list within a single larger JSON document, which is then returned by the endpoint.

The **get_fasta** endpoint is used to send the client a FASTA file containing protein sequences that support the existence of a property step across multiple organisms in a specific uploaded dataset. The URL path of requests to this endpoint includes the genome property identifier and step number of the property step whose protein matches should be included in the returned FASTA file. The returned FASTA file can either contain all proteins that support the existence of a property step or a subset of these supporting proteins, one per sample, that have the lowest E-value match to the InterPro domain that is used to identify the given property step. These proteins are known as the **"top"** hits. The contents of the returned file is controlled by the presence of a HTTP GET parameter called **all** (Fig. 3.5). If **all** is set to true, then a FASTA file containing all proteins that support a step is returned. Otherwise, a FASTA file containing only the lowest E-value proteins is returned.

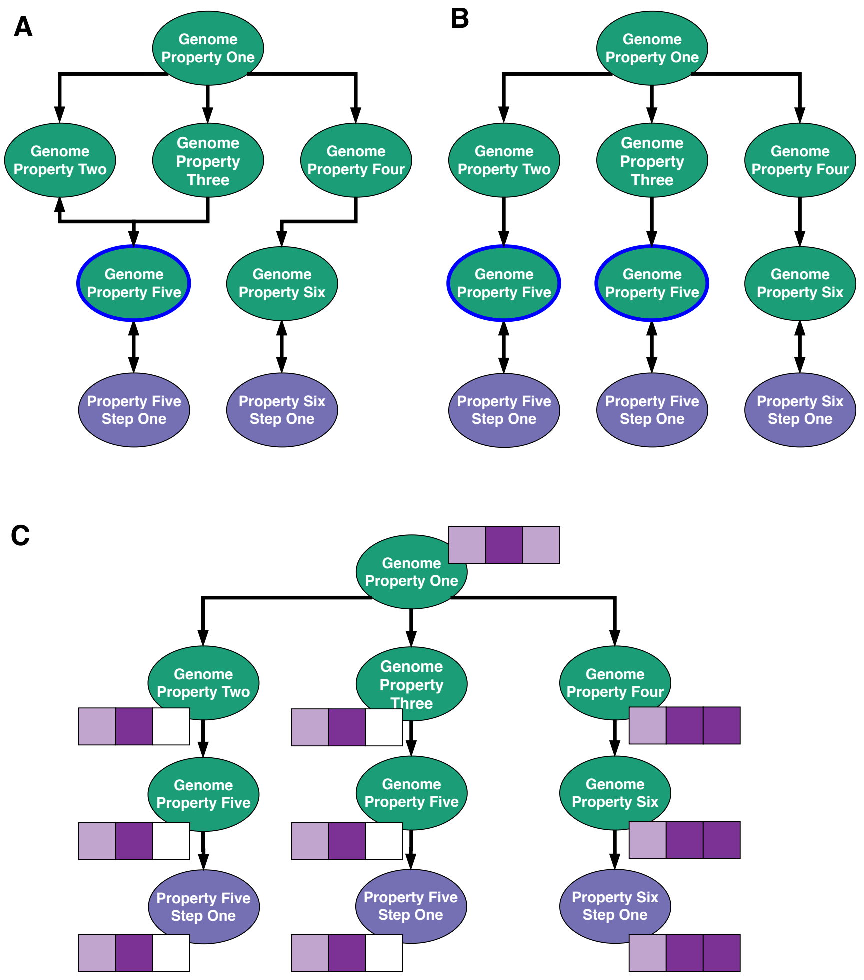


Figure 3.6: **Comparison of the tree data structures returned by Micromeda-Server's Get_Tree endpoint to the property DAG built by Pygenprop.** Unlike a DAG (A), a tree (B) cannot have branches that merge. Thus, the JSON returned by the Get_Tree can endpoint has some Genome Property nodes duplicated (B). In this JSON document, each node is tagged with a list of YES (Dark Purple), PARTIAL (Light Purple), and NO (White) assignments (C). Each assignment in this list belongs to a single organism in a dataset.

Like the **get_tree** endpoint, the **get_fasta** endpoint also accepts a **dataset_key** HTTP GET parameter (Fig. 3.5). The value of this variable is used to reconstitute a GenomePropertiesResultsWithMatches object representing a previously uploaded Micromeda file

(Fig. 3.2). This object's **write_supporting_proteins_for_step_fasta** method (Table 2.12) is used generate the FASTA file, which is sent to the client.

3.5 Micromeda Server Performance

The performance of Micromeda's endpoints was tested using a Micromeda file generated from the protein sequences and InterProScan annotations of *Chlorobium chlorochromatii* CaD3 (NCBI Taxa ID: 340177) and *Pelodictyon luteolum* DSM 273 (NCBI Taxa ID: 319225). The performance metrics discussed in this section were recorded using Safari's (version 13; apple.com/safari) Web Inspector. When this file was sent to the **upload** endpoint of Micromeda-Server, it was parsed and added to the Redis cache in 11.3 s $\pm$2.0 s (N = 3). The **get_tree** endpoint could create a genome property tree from this cached Micromeda file in 9.4 s $\pm$4.0 s (N = 3). The **get_fasta** endpoint could generate a FASTA file with the top supporting proteins for GenProp0633 step number two in 33.0 ms $\pm$4.0 ms (N = 10). A property information JSON file could be generated for GenProp0633 by the **get_single_genome_property_info** endpoint in 7.0 ms $\pm$2.0 ms (N = 10). The **get_multiple_genome_property_info** endpoint can generate a JSON information file for all properties in 23.0 ms $\pm$5.0 ms (N = 10). The slow execution time of the **upload** and **get_tree** endpoints caused latency in client application. For example, there was a noticeable delay in the rendering of visualizations in Micromeda's client application. Thus, the performance of these endpoints should be optimized. Potential optimizations to the endpoints are discussed in Section 3.7.

3.6 Micromeda Server Deployments

It is common to deploy web servers in different configurations depending on the expected request volume. Three deployment strategies of increasing size are discussed below.

If a user wishes to install and run Micromeda-Server on their desktop or laptop and only needs to visualize one dataset at a time, then a very simplified deployment strategy can be used (Fig. 3.7). This deployment, called a single-user deployment, uses Flask's built-in development HTTP server to respond to requests. The server is activated when Micromeda-Server's Python script is run directly from a CLI. This single-user deployment is slow and can only handle requests from a single client. In this configuration, the upload endpoint is turned off, and Redis is not used. As a result, users cannot upload Micromeda files. Instead, Micromeda-Server gets its assignment data from a single default Micromeda file stored on disk (Fig. 3.7). This deployment method is similar to the one used by Anvio's MAGs visualization and refinement server [49]. The single-user deployment configuration is also useful to developers who want to test newly developed features or bug fixes.

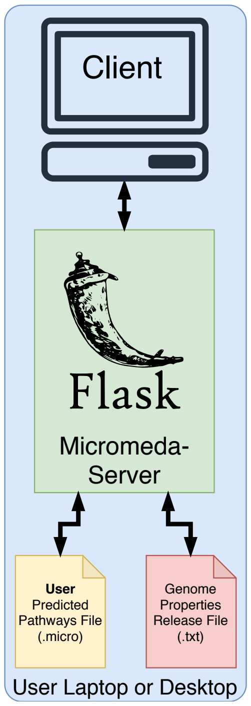

Client
Flask
Micromeda-Server
User Predicted Pathways File (.micro)
Genome Properties Release File (.txt)
User Laptop or Desktop

Figure 3.7: **How Micromeda-Server would be deployed to support a development or single-user environment.** In this deployment style, Micromed-Server uses Flask's builtin HTTP server, a genomeProperties.txt file, and a server-side Micromeda file. However, this deployment style encounters problems when used by multiple users because only a single copy of the Python Flask code is run at a time.

If a user requires Micromeda-Server to handle multiple users simultaneously, such as would be the case if the software was installed on a server computer system, a larger deployment must be used (Fig. 3.8). This deployment type, called a multi-user deployment, adds additional software layers that increase Micromeda-Server's scalability. As discussed in previous sections, multiple copies of Micromeda-Server's Python code must be run simultaneously to handle multiple client requests. This technique is done by putting the Micromeda-Server under the command of a master HTTP server that can route traffic to multiple copies of the Python Flask code running separate processes (Fig. 3.8). Examples of such master HTTP servers are Apache [54] and Nginx [129] HTTP servers. These master servers handle the parsing of HTTP requests. In addition to the master server, a middleware component, such asweb server gateway interface server (uWSGI) [153] or gunicorn [32], must also be used. Redis is used to cache GenomePropertiesResultsWithMatches objects between requests in this deployment type. All components of Micromeda-Server deployed on the same server computer system (Fig. 3.8).

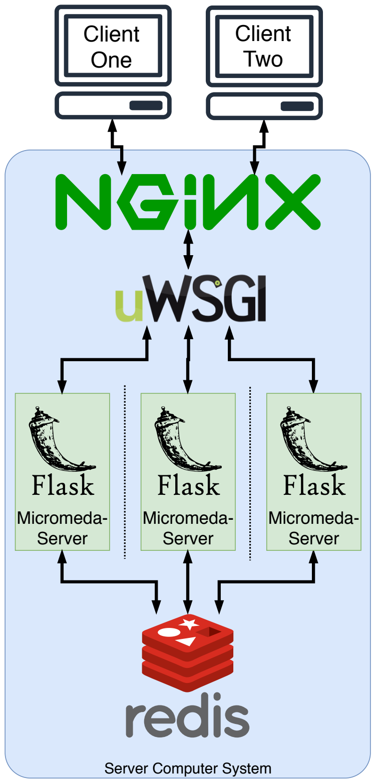


Figure 3.8: **How Micromeda-Server would be deployed to support multiple users using a single server computer system.** When deployed to handle the traffic of multiple clients, Multiple other types of software must support Micromed-Server. Multiple copies of the application's Python code are run simultaneously to handle multiple clients. Redis is used to provide shared data between these processes. A uWSGI middleware component is used to connect these copies to a high performance HTTP server, such as Nginx, that can handle high web traffic volumes. The genomeProperties.txt files and default Micromeda files are still used in this deployment style but were omitted from the diagram for simplicity.

For a large number of simultaneous users, Micromeda-Server may need to be scaled horizontally across multiple servers (Fig. 3.9). This deployment type is called a multi-server deployment. The scaling is facilitated by placing a load balancer (en.wikipedia.org/wiki/ Load_balancing_(computing)) out front of multiple copies of the multi-user deployment running on separate server computer systems (Fig. 3.9). Redis can also be run on a separate server computer system or computing cluster (Fig. 3.9). Such multiple server deployment strategies can be scaled horizontally by adding new server computer systems to the deployment (Fig. 3.9). These new server computer systems allow for increases in request volume.

Various cloud computing corporations have developed Platform as a Service (PaaS) [92] products (see en.wikipedia.org/wiki/Platform_as_a_service) that help users scale Python web applications without having to spend time setting up complex multi-server deployments. These PaaS, such as Google App Engine (cloud.google.com/ appengine) or Heroku (heroku.com), provide the simplicity of a single-user deployment with the scalability of multi-server deployment. When a developer's code is deployed to a PaaS, the resulting deployment, called a PaaS deployment, provides attributes of both single-user deployments and multi-server deployments. For example, developers are only required to upload their Python code files to the platform and, subsequently, the platform will automate the rest of the deployment process. For example, the PaaS will create load balancers and multiple copies of request handling Python processes automatically. Such platforms perform scaling tasks in the background and invisibly to the developer who uploaded the code.

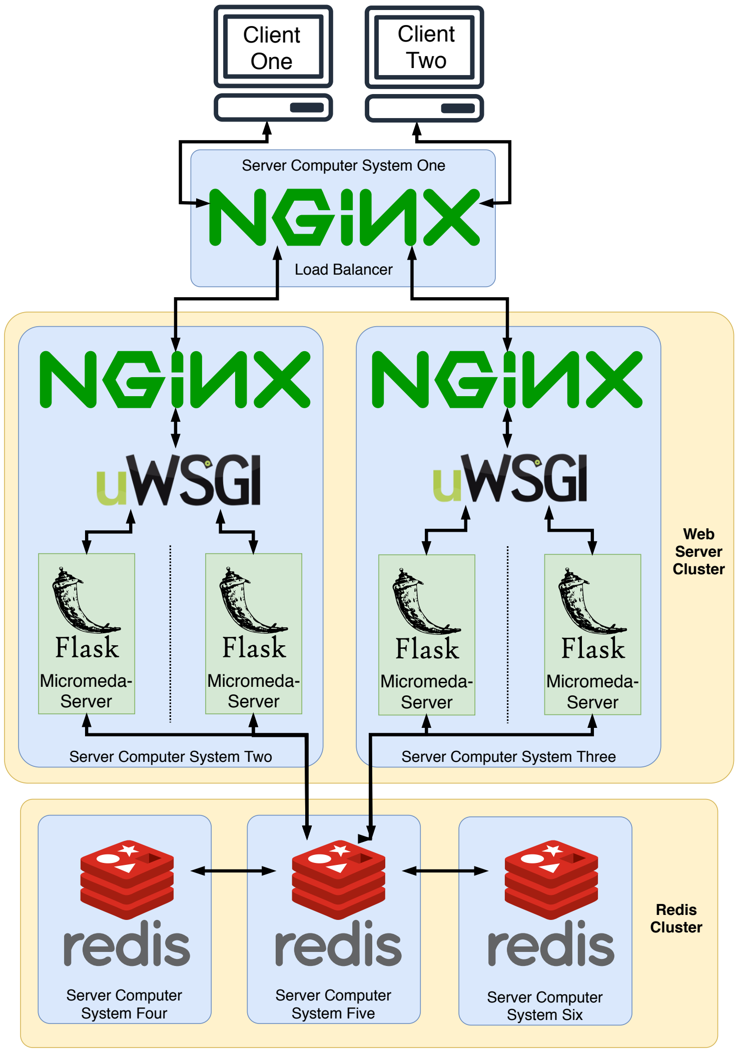


Figure 3.9: **How Micromeda-Server would be deployed to support multiple users using a cluster of computer systems.** When Micromeda is required to scale to handle hundreds or thousands of simultaneous users, its workload must be spread out across multiple caching and web server computer systems. Each node in the web server cluster runs additional copies of Micromeda-Server and supporting software. The performance of such deployments can be increased by adding hardware to either the caching or web server clusters. A copy of the genomeProperties.txt file and default Micromeda file would be stored in each server computer system in the web server cluster. These file have been omitted from the diagram for simplicity.

3.7 Future Improvements

Micromeda-Server currently possesses endpoints that provide all the information needed by Micromeda's web-based visualization client (see Section 3.4). One endpoint provides an annotated Genome Properties tree that includes information about both the structure of the Genome Properties database and assignments of supports for properties and steps. Other endpoints offer detailed information about individual properties and steps, and allow for the download of protein sequences that support property steps. In addition to the existing endpoints, new endpoints could be made that would provide new data for future versions of this client, allowing it to have expanded functionality. Performance optimizations for the existing endpoints could also be made, which could reduce latency in the client's UI by reducing the time spent waiting for responses from Micromeda-Server. Improvements to Micromeda-Server's existing endpoints and a potential new endpoint are discussed below.

3.7.1 Improving Performance of the Upload Endpoint

The **upload** endpoint takes a Micromeda file and saves the file's contents to a Redis cache. This endpoint was shown to have poor performance (see Section 3.5) and should be optimized to be more responsive. Most of the performance problems with this endpoint can be attributed to the performance of parsing Micromeda files into GenomePropertiesResultsWithMatches objects using the **load_assignment_caches_from_database_with_matches** function of Pygenprop's results module. The performance of this function was also weak (see Subsection 2.6.6). Performance improvements to this function are discussed in detail in Subsection 2.9.3. The performance optimizations recommended in that section should be adopted and will drastically improve the performance of the upload endpoint.

3.7.2 Improving Performance of the Get_Tree Endpoint and Building Endpoints for Returning Property and Step Assignments

The **get_tree** endpoint sends JSON to the client that contains both property tree and assignment data. This JSON is generated by calling a GenomePropertiesResultsWithMatches object's **to_json** method. This method is known to have speed issues (see Subsection 2.5.3 and Section 3.5) due to the method having to insert assignments into the JSON tree one at a time, rather than in batch. This speed issue could be addressed by reconfiguring Micromeda-Server's **get_tree** endpoint to no longer output a tree annotated by property assignments. Instead, new endpoints could be built that return step and property assignments separately (Fig. 3.10). New methods of GenomePropertiesResultsWithMatches objects would have to be developed to generate JSON for these new endpoints (Section 2.9.3). The code for generating the property tree JSON could be moved to the GenomePropertiesTree class (Fig. 3.10).

Caching the JSON generated by endpoints in Redis could also improve these endpoint's performance. On subsequent API calls, the cached data could be recalled from Redis and immediately returned to the client instead of being regenerated with every API call. For example, only having to generate a property tree once procedurally would significantly improve endpoint performance.

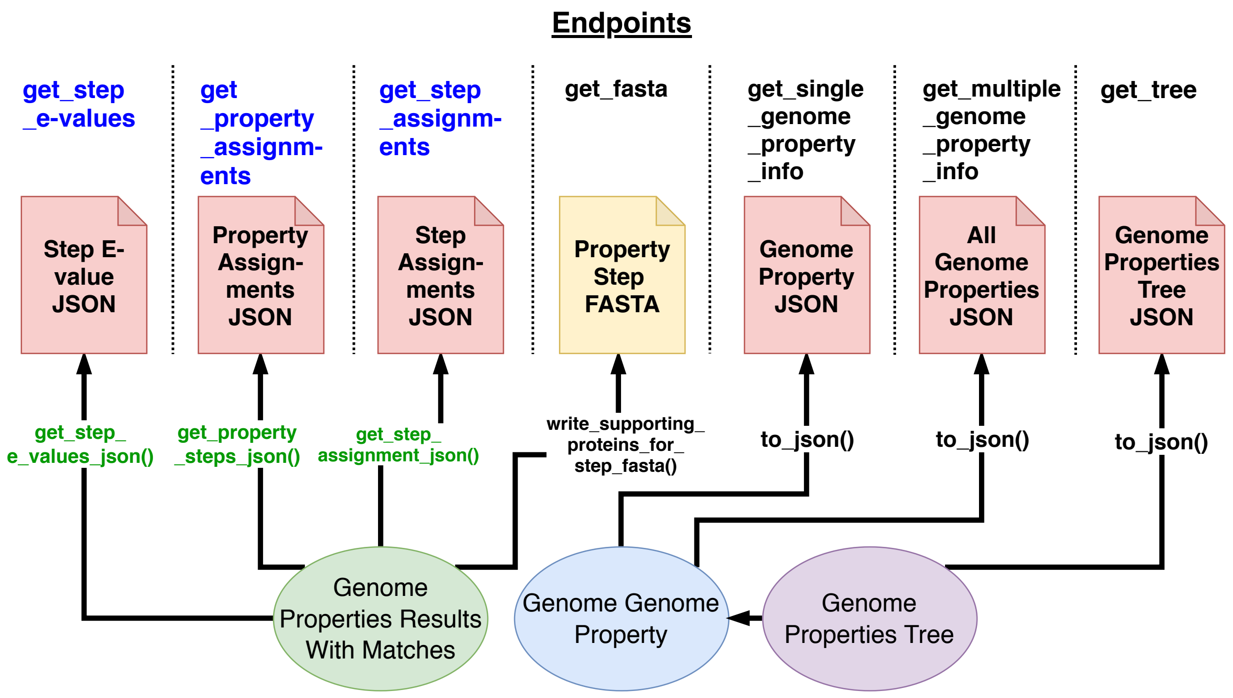

Endpoints
get_step_e-values
get_property_assignm-ents
get_step_assignm-ents
get_fasta
get_single_genome_property_info
get_multiple_genome_property_info
get_tree
Step E-value JSON
Property Assign-ments JSON
Step Assign-ments JSON
Property Step FASTA
Genome Property JSON
All Genome Properties JSON
Genome Properties Tree JSON
get_step_e_values_json()
get_property_steps_json()
get_step_assignment_json()
write_supporting_proteins_for_step_fasta()
to_json()
to_json()
to_json()
Genome Properties Results With Matches
Genome Genome Property
Genome Properties Tree

Figure 3.10: **New endpoints that could be added to Micromeda-Server to allow for expanded client functionality.** These endpoints (blue) would return step assignments, step supporting information, and property assignments. New endpoints would be supported by new JSON generating methods of GenomePropertiesResultsWithMatches objects (green).

3.7.3 Creation of an Endpoint for Returning Domain Annotations Supporting Property Steps

Micromeda files not only contain property and step assignments for a set of organisms but also contain additional information such as domain annotations and proteins sequences that support the existence of property steps. Micromeda-Server currently provides an endpoint for accessing protein sequences that support steps (see Section 3.4). However, there are no endpoints for accessing the stored InterProScan annotations of these proteins. Specifically, these annotations contain E-value scores representing how closely the domain in the protein matches to a model of a representative domain in a protein database (see Subsection 1.4.2). The E-value scores for these domains may be useful to client visualizations that compare not only the presence and absence of proteins that support property steps but also compare how close these matches are to existing domain models. Thus, it may be useful to create a new endpoint that returns domain annotation E-value scores for domains that support property steps (Fig. 3.10). This endpoint could generate its data from the **step_matches** DataFrame of reconstituted GenomePropertiesResultsWithMatches objects.

3.8 Summary

The creation of web server API's for accessing information about biochemical pathways, and even the precalculated presence and absence of these pathways across organisms, is quite common [164, 111, 124, 155, 9, 146, 110, 33]. Indeed, such web APIs have been developed by both the creators of KEGG [86] and MetaCyc [83], not only for these database's web client applications but also for academic use. A web server has also been built for Genome Properties database website [132] that is hosted by the EBI. However, the server's API is not publicly available and is only designed to support the Genome Properties website. The KEGG, MetaCyc, and Genome Properties website all contain precalculated pathway annotations for sets of reference organisms [80, 85, 83]. Many pathway annotation web sites such as From Metabolite to Metabolite (FMM) [33], KAAS [110] and MAPLE [146] can pathway annotate user-supplied genomes uploaded in FASTA format [121]. Such annotation servers are complex to build and host due to the relative computational complexity of scanning for genes that support the existence of pathway steps. In contrast, the Genome Properties website allows for the upload of user-created InterProScan annotation files. The creation of such annotations files pushes the most computationally complex part of the Genome Properties pipeline, domain annotation, off onto end-users [132]. As discussed in Subsection 1.4.1, Micromeda-Server follows a similar approach. However, unlike the Genome Properties website, Micromeda-Server takes the upload of Micromeda files. Micromeda files allow the upload of datasets consisting of pathway annotations from multiple genomes simultaneously, unlike other pathway servers, which often require uploading genomes or InterProScan annotations for organisms one at a time. Also, because Micromeda files contain protein sequences that support pathway annotations, Micromeda-Server can present these sequences for download by users, which is a feature of few other pathway annotation servers currently possess. With the avoidance of computationally complex annotation steps and a variety of horizontally scalable deployment options, Micromeda-Server should provide a reliable and sustainable API for Micromeda's client application.

Chapter 4

Development of a Web-Based Visualization Tool for the Comparison of Organism Genome Properties

As discussed in Chapter 1, a client web application provides Micromeda's UI. This client's role is to provide users with a streamlined interface for visualizing patterns of property and step assignment found across organisms. These assignment data are provided to the client in the form of a user-uploaded Micromeda file (Section 2.6). After upload, these files are sent to Micromeda-Server (Chapter 3) where they are parsed and used to provide data to the client. The client can request these data through a series of web API endpoints (Section 3.4) provided by Micromeda-Server. This chapter discusses the client web application, called Micromeda-Client, in detail. Links to a demonstration of the client interface can be found in Section 4.5. Source code for Micromeda-Client is located at github.com/Micromeda/micromeda-client.

4.1 Visualisation Design

One of the core uses of Genome Properties assignment data is to mine it for biologically relevant patterns in the presence and absence of biochemical pathways or structural features across organisms. One of the best ways to detect such patterns is through the use of data visualization. By visualizing Genome Properties assignment data, users can make comparisons between organisms, pathways, and steps. Several example comparisons and their research relevance are listed below and are displayed in Fig. 4.1.

- By looking at the assignments of a single property across organisms, users can select subsets of organisms that may possess a specific phenotype (Fig. 4.1a). Such comparisons are useful in scientific fields where specific traits must be identified. Such fields include pathogen research, microbial ecology, bioengineering, and bioprocess engineering, among others.

- By comparing the assignments of multiple genome properties, users can identify patterns of conservation across organisms in a dataset (Fig. 4.1c). In microbial ecology, such comparisons can be used to identify microorganisms that fit specific biochemically determined ecological niches (*e.g.*, photoferrotrophy).

- By looking at the step assignments of a single property and organism, users can evaluate the correctness of a property assignment (Fig. 4.1b). A property might be assigned NO or PARTIAL because it is missing only a few required steps; However, it may possess many other steps that are not required (Fig. 4.1b). Being able to look at all step assignments for the property would allow users to determine why a property has been given a specific assignment.

- By comparing step assignments of a single property across multiple organisms, users can see whether pathway steps are retained across organisms (Fig. 4.1d). If a property step is not retained in a large assortment of phylogenetically distant organisms, it may not be required by a pathway or may be carried out by proteins that are non-canonical in these phylogenetically distant organisms (Fig. 4.1d). Such non-canonical proteins may not possess domains used by Genome Properties but still carry out a missing pathway step.

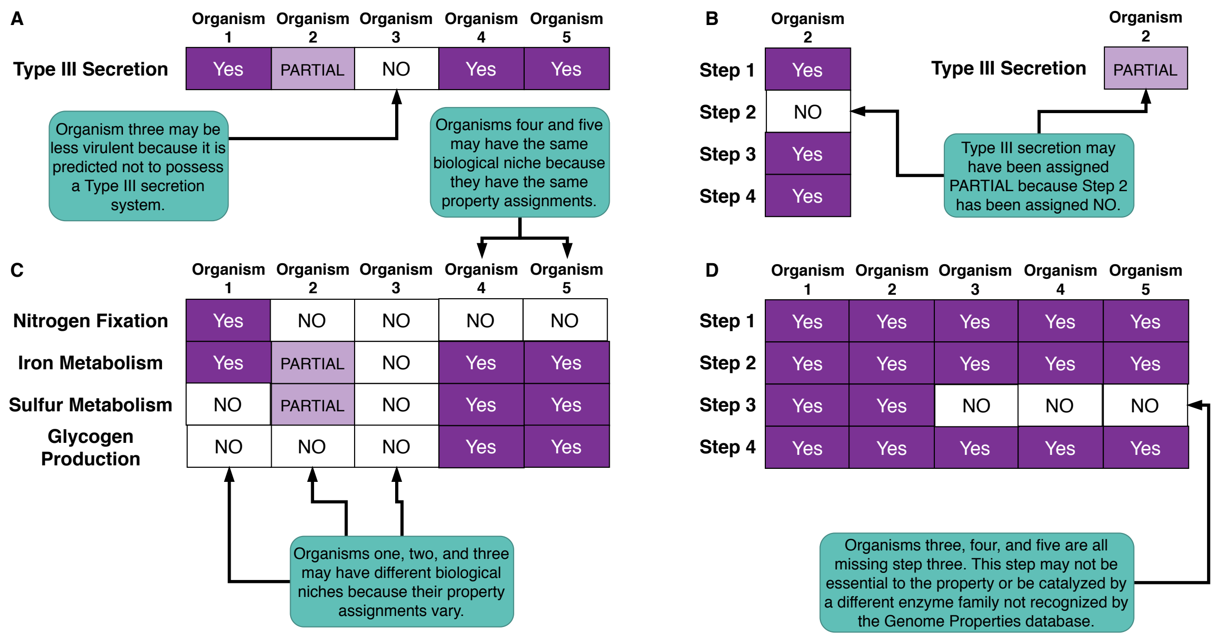

A
Organism 1
Organism 2
Organism 3
Organism 4
Organism 5
Type III Secretion
Yes
PARTIAL
NO
Yes
Yes
Organism three may be less virulent because it is predicted not to possess a Type III secretion system.
Organisms four and five may have the same biological niche because they have the same property assignments.
B
Organism 2
Step 1
Yes
Step 2
NO
Step 3
Yes
Step 4
Yes
Organism 2
Type III Secretion
PARTIAL
Type III secretion may have been assigned PARTIAL because Step 2 has been assigned NO.
C
Organism 1
Organism 2
Organism 3
Organism 4
Organism 5
Nitrogen Fixation
Yes
NO
NO
NO
NO
Iron Metabolism
Yes
PARTIAL
NO
Yes
Yes
Sulfur Metabolism
NO
PARTIAL
NO
Yes
Yes
Glycogen Production
NO
NO
NO
Yes
Yes
Organisms one, two, and three may have different biological niches because their property assignments vary.
D
Organism 1
Organism 2
Organism 3
Organism 4
Organism 5
Step 1
Yes
Yes
Yes
Yes
Yes
Step 2
Yes
Yes
Yes
Yes
Yes
Step 3
Yes
Yes
NO
NO
NO
Step 4
Yes
Yes
Yes
Yes
Yes
Organisms three, four, and five are all missing step three. This step may not be essential to the property or be catalyzed by a different enzyme family not recognized by the Genome Properties database.

Figure 4.1: **Examples of different pathway analysis tasks that can be supported by the visualization of Genome Properties data.** Comparisons can be made between organisms, pathways, or pathway steps.

If Micromeda-Client is to support these listed comparisons, the data visualization method used must allow users to perform the following tasks.

- Track assignments across organisms
- Assess the magnitude of assignments
- Aggregate assignments into summaries
- Explore how these aggregate summaries are derived
- Find assignments of interest

When a visualization approach was selected for use by Micromeda-Client, the compatibility of the visualization with these tasks was prioritized.

4.2 Overview of Genome Properties Data and How Micromeda-Client Visualizes This Data

Specific visualization techniques are better suited for presenting certain types of data over others, and it is essential first to discuss the nature of data to be visualized before discussing how a specific visualization approach was selected for integration into Micromeda-Client. At its core, the data presented by Micromeda-Client consist of assignments for genome properties and their steps. Such assignments are ordinal data [132, 3], as their states of YES, PARTIAL, and NO are ordered. It should be noted that properties are connected hierarchically [132]. Thus, this structure is hierarchical data [132, 134]. The property hierarchy influences the ordinal assignment data of each property because assignments of parent genome properties can be used to summarize the assignments of child genome properties or steps [132]. Each piece of assignment data that is presented by Micromeda-Client also belongs to a specific property or step and organism. Thus, the Genome Properties data can also be considered to be multidimensional [122].

When designing a data visualization, often specific visualization techniques present themselves intuitively as potential candidates, and this was the case while designing Micromeda-Client. An appropriate visualization for multidimensional datasets, such as Genome Properties assignments, is a heat map [163, 152](Fig. 4.1) and this visualization technique was the one chosen for the client (Fig. 4.1). A heat map was chosen over the competing visualization techniques such as bubble charts [152], circular maps [161, 143], or treemaps [139] due to the number of variables that needed to be plotted by the client and the superior space utilization of heat maps, which mimic the square dimensions of the computer monitors they are displayed on [114].

Micromeda-Client's heat map uses cell position to indicate what assignment belongs to what property or step and organism (Fig. 4.1). Because most Micromeda files contain information about fewer organisms than there are properties in the Genome Properties database (*e.g.*, approximately 40 organisms versus 1296 properties), assignments for the same property, but from different organisms, are positioned within the same heat map row (*i.e*, property labels are placed along y-axis rather the x-axis of the heat map). Columns are used to group assignments from the same organism across properties or steps. This configuration leads to a heat map that is much taller than it is wide and requires that users scroll vertically. Vertical scrolling is much more convenient than scrolling horizontally because it allows users to scroll the visualization quickly by using their mouse's scroll wheel.

The magnitude of each assignment is encoded using cell colour (Fig. 4.1). Because assignments are ordinal data, it makes sense to encode assignments using colour saturation, rather than hue [113]. For the heat map, a purple cell colour scheme was chosen to ensure that the diagram is interpretable by those with colour blindness. Specifically, colours were optimized for users with severe deuteranopia (*i.e.*, complete loss of green cones), which, though rarer, has stronger Red-green colour blindness affects (see wikipedia.org/wiki/Color_blindness#Classification). A web service called ColorBrewer (colorbrewer2.org) was used to select cell colours, and a tool called Sim Daltonism (github.com/michelf/sim-daltonism) was used to ensure colour blind compatibility of the entire diagram.

Any visualization technique used is also challenged by the magnitude of data to be presented by Micromeda-Client. For example, if the heat map described at the top of this section displayed all assignments for only a few organisms, then its size would prevent users from finding or tracking assignments quickly across organisms. As of version 2.0 of the Genome Properties database, there are 1296 properties and 6525 steps [132]. If the client were used to generate a single heat map representing the assignments of all properties, with each cell being 5 mm tall, then the heat map produced would be approximately 6.5 meters tall (*i.e.*, fifteen vertical pages on a 24" monitor). If a similar heat map was made, but for assignments of all property steps, then the heat map generated would be over 32.6 meters tall (*i.e.*, seventy-five vertical pages on a 24" monitor). If both of these heat maps were combined, then the resulting heat map would be even longer. Micromeda-Client's visualization interface addresses the length issue by using interactive aggregation and disaggregation [113] of assignment rows to reduce the overall length of the client's assignment heat map (Fig. 4.2). This reduced length facilitates rapid visual exploration of property and step assignments.

Micromeda-Client's UI allows users to manipulate the contents of the client's assignment heat map to show and hide properties and steps according to their position in the Genome Properties DAG [132] (Fig. 4.2). Because the Genome Properties DAG arranges individual properties hierarchically, the assignments of properties closer to the root can be used to summarize the assignments of properties closer to the DAG's leaves (Fig. 4.2). In the context of a heat map, this hierarchical assignment summarization means that a row of assignments for a parent property can be used to summarize the rows of assignments of this property's children, either other properties or steps (Fig.4.2 and Fig. 4.1). Micromeda-Client provides mechanisms to expand and collapse heat map rows to display either parent summary assignments or more detailed child assignments (Fig. 4.2).

One critical design decision for Micromeda-Client was how to let users control the aggregation and disaggregation of heat map rows. To support this usage, a new visualization component to Micromeda-Client was designed, in addition to the assignment heat map.

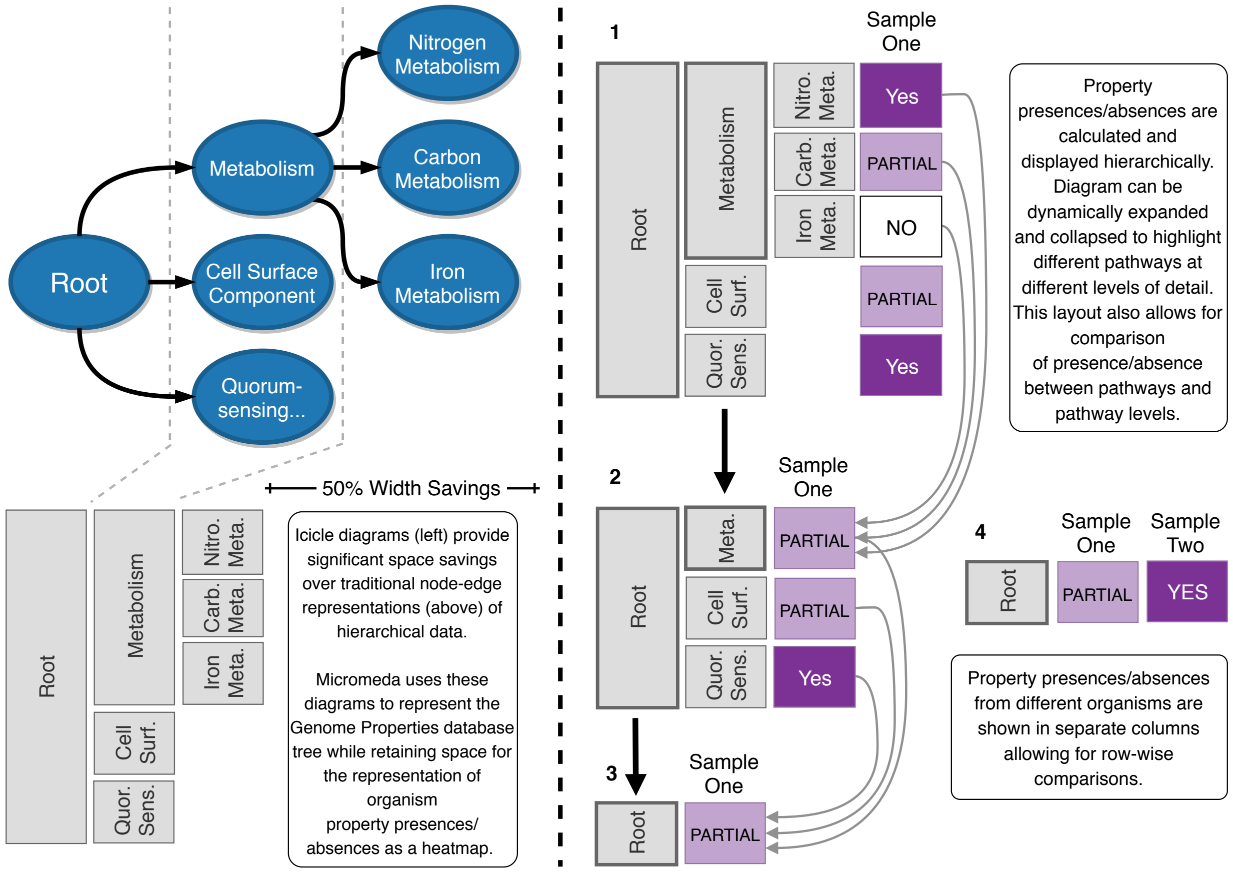

Nitrogen Metabolism
Metabolism
Carbon Metabolism
Root
Cell Surface Component
Iron Metabolism
Quorum-sensing...
50% Width Savings
Root
Metabolism
Nitro. Meta.
Carb. Meta.
Iron Meta.
Cell Surf.
Quor. Sens.
Icicle diagrams (left) provide significant space savings over traditional node-edge representations (above) of hierarchical data.
Micromeda uses these diagrams to represent the Genome Properties database tree while retaining space for the representation of organism property presences/ absences as a heatmap.
1
Sample One
Yes
PARTIAL
NO
PARTIAL
Yes
Property presences/absences are calculated and displayed hierarchically. Diagram can be dynamically expanded and collapsed to highlight different pathways at different levels of detail. This layout also allows for comparison of presence/absence between pathways and pathway levels.
2
Meta.
PARTIAL
PARTIAL
Yes
3
Root
PARTIAL
4
Sample One
Sample Two
YES
Property presences/absences from different organisms are shown in separate columns allowing for row-wise comparisons.

Figure 4.2: **Overview of the components of Micromeda's visualization design that are critical to user interactivity.** The use of an icicle diagram allows the final visualization to be more compact. Clicking nodes in the icicle diagram changes the shape of the heat map by adding and removing rows.

Specifically, the client uses a horizontal icicle diagram[1] placed left of the heat map. Icicle diagrams were chosen over other ways of visualizing hierarchical data, such as trees, due to their spacial compactness (Fig. 4.2). The icicle diagram is used to control the heat map's content (Fig. 4.2). Icicle diagrams are used to display hierarchical data, such as the parent-child relationship between properties and between properties and their steps. With Micromeda-Client, each genome property and step in the Genome Properties database is assigned a node in the icicle diagram (Fig. 4.2). Nodes that represent properties are

[1]Traditional icicle diagrams have leaf nodes facing downwards. The icicle diagram used by Micromeda-Client is rotated 90 degrees and has its leaf nodes facing to the right.

coloured gray and nodes that represent steps of leaf properties are coloured green. The leaf nodes of the icicle diagram are aligned to rows in the adjacent heat map (Fig. 4.2) that contain the assignments for the property or step that the node represents.

With Micromeda-Client, the nodes in the icicle diagram are given a state of either on or off. When a user clicks a node in an off state, a new column is added to the icicle diagram. This column's contents includes new icicle nodes representing the clicked node's children. Simultaneously, new assignment rows are added to the heat map. These rows belong to the same properties or steps that the newly added icicle diagram nodes represent. The new heat map rows replace the clicked node's assignment row. The new icicle nodes and heat map rows are vertically aligned. Each new node in the icicle diagram has the same height as the heat map row to which the new node is aligned. The clicked node's shape is expanded vertically to align with the top and bottom of its first and last child node cells, respectively. The overall length of the icicle diagram and heat map is extended. If the user clicks the previously clicked node once again, then the expansion process is reversed. The child nodes and their assignment rows are removed from the visualization (Fig. 4.2), the clicked cell returns to its original shape, and its matching summary assignment row is placed back into the heat map (Fig. 4.2). Child assignment rows are also removed from the heat map. Columns in the icicle diagram are deleted, upon child cell removal, only if all sibling or cousin nodes to the clicked node also have no children displayed.

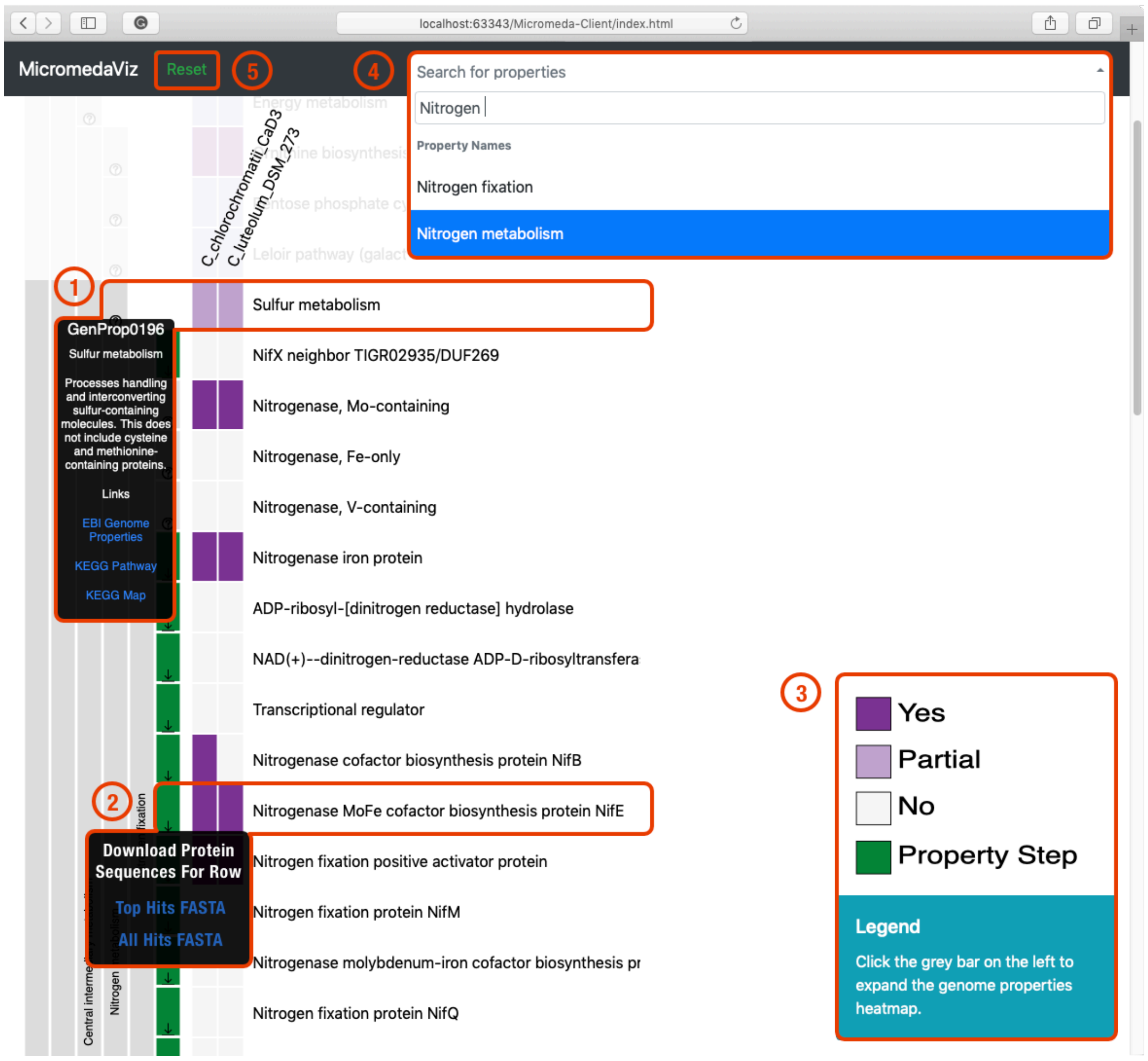
The visualization strategy chosen for Micromeda-client supports the required tasks presented at the top of this section. The heat map allows users to track assignments across organisms and assess their magnitude. The interactive aggregation control provided by the icicle diagram allows users to aggregate step and property assignments into summaries. These aggregate assignments can later be disaggregated to show child assignments, allowing users to explore how the parent assignments were derived. As the icicle diagram follows the structure of the Genome Properties DAG, specific assignments can be found quickly by following the DAG's structure from parent to child. Searching for properties is further enhanced by Micromeda-Client's ability to search for properties by name. This search functionality is discussed in the next section.

4.3 Additional Features of Micromeda-Client's Interface

In addition to the visualization capability presented in Section 4.2, the client's interface also possesses several other features that help users explore their data. In the top right corner of the UI is a text-based search box (Fig. 4.3). This box allows users to search for properties by entering a text string containing either a property name or identifier. As the user enters this string, a list of matching property names are displayed in a drop-down menu (Fig. 4.3). As the user enters additional information, this list of possible matches shrinks. If the user clicks one of these property names, then Micromeda-Client will automatically scroll to the heat map row where assignments for the clicked property are located. If the property is not shown in the current version of the heat map, then it will be added to the heat map by disaggregating heat map rows in a path towards the property. This path is built recursively by disaggregating heat map rows of parent properties along a path from the root of the Genome Properties DAG to the property that was clicked in the search menu. As the user scrolls the client diagram, the X-axis labels of the diagram remain in a fixed position at the top of the web browser window (i.e., the X-axis scrolls with the diagram and does not become hidden during page scrolling) to provide users with sample/genome name information. A similar scrolling behaviour as to when a search bar item is clicked is also activated when a user clicks a node in the icicle diagram. The diagram's X-axis is scrolled to align with the clicked node's associated heat map row.

While users explore the heat map, it may be useful for them to be able to access more context about the displayed properties. The icicle diagram possesses question mark glyphs[2] at the bottom of each property node that facilitates access to additional information about the property that the node represents (Fig. 4.3). When a user places their cursor over one of these glyphs, a pop-up box appears displaying information about the property that the glyph's node represents (Fig. 4.3). This pop-up box includes the name of the property, a description of it, a link to the property on the EBI Genome Properties website (*e.g.*, ebi.ac.uk/interpro/genomeproperties/ #GenProp0867), and a list of links to equivalent records in other pathway databases such as KEGG [80] and MetaCyc [85]. The box is hidden once again when the user's cursor leaves the glyph or the pop-up box.

[2]A glyph is an elemental component of a data visualization. A data visualization is composed of a series of glyphs. Often glyphs are directly mapped to data points [31].


localhost:63343/Micromeda-Client/index.html
MicromedaViz
Reset
5
4
Search for properties
Nitrogen
Property Names
Nitrogen fixation
Nitrogen metabolism
C_chlorochromatii_CaD3
C_luteolum_DSM_273
Energy metabolism
Pentose phosphate cycle
Leloir pathway (galact
1
Sulfur metabolism
GenProp0196
Sulfur metabolism
Links
EBI Genome Properties
KEGG Pathway
KEGG Map
NifX neighbor TIGR02935/DUF269
Nitrogenase, Mo-containing
Nitrogenase, Fe-only
Nitrogenase, V-containing
Nitrogenase iron protein
ADP-ribosyl-[dinitrogen reductase] hydrolase
NAD(+)--dinitrogen-reductase ADP-D-ribosyltransfera
Transcriptional regulator
Nitrogenase cofactor biosynthesis protein NifB
2
Nitrogenase MoFe cofactor biosynthesis protein NifE
Download Protein Sequences For Row
Top Hits FASTA
All Hits FASTA
Nitrogen fixation positive activator protein
Nitrogen fixation protein NifM
Nitrogenase molybdenum-iron cofactor biosynthesis pr
Nitrogen fixation protein NifQ
Central interme
Nitrogen
fixation
3
🟪 Yes
▨ Partial
⬜ No
🟩 Property Step
Legend

Processes handling and interconverting sulfur-containing molecules. This does not include cysteine and methionine-containing proteins.

Click the grey bar on the left to expand the genome properties heatmap.

Figure 4.3: **Web-browser window containing Micromeda's heat map visualization and interface annotated to highlight the interface's search and information gathering features.** The interface provides functionality for getting additional information about properties and steps (1) and provides the ability to download protein sequences that support step assignments (2). A legend provides context to the heat map's colour scheme (3). The interface also supports property searching (4) and possesses a reset button that allows users to reset the heat map to its initial loaded state (5).

Some leaf property steps have a different glyph at the bottom of their nodes that is shaped like a download symbol (Fig. 4.3). This glyph allows users to download protein sequences that support the existence of property steps. These download glyphs are only placed on nodes whose associated property steps are assigned YES in at least one organism displayed on the heat map. A pop-up box is also generated when these download glyphs are hovered over. This box contains two download links (Fig. 4.3). The first link downloads a FASTA file containing the protein sequences that are most likely to carry out the step for each organism in a dataset[3]. The second link downloads a FASTA file containing any protein that could have carried out the step across all organisms in a dataset[3]. When the cursor is removed from this download pop-up box or the glyph, then the pop-up box is hidden.

Micromeda-Client's interface also includes a reset button. This button resets the heat map and icicle diagram back to their starting configuration where only the top-level properties are shown (*i.e.*, one level below the root of the Genome Properties DAG) to the user. Users can click this button to reset the diagram to allow them to search for other properties.

4.4 Implementation

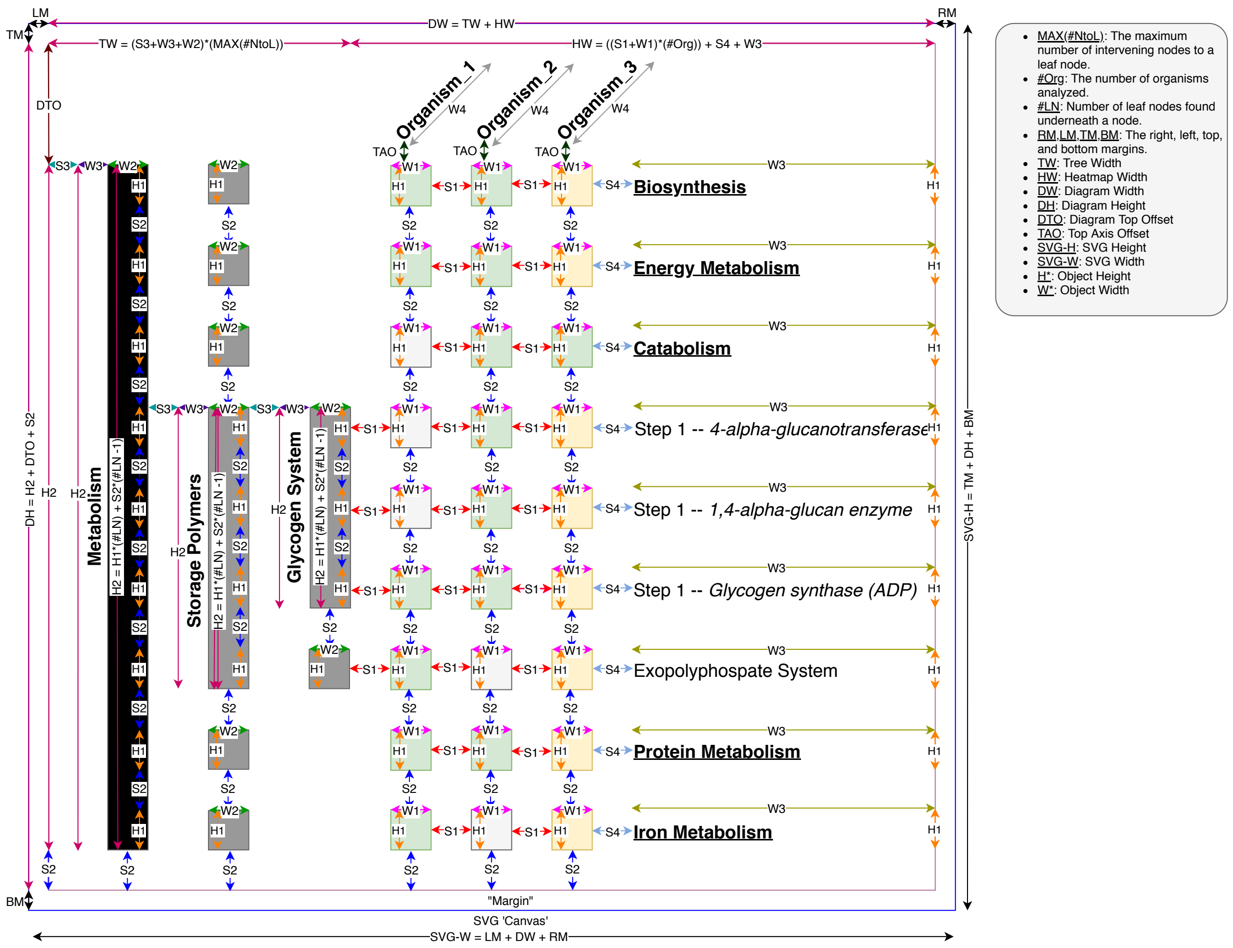
Micromeda-Client's interface consists of two web pages that were structured using Hypertext Markup Language version 5 (HTML) [99], styled using Cascading Style Sheets version 3 (CSS) [22], and scripted via European Computer Manufacturers Association (EMCA) script version 6 (JavaScript) [57]. Users access one page for uploading user-generated

[3]As discussed in Subsection 1.4.2, each member database of the InterPro consortium possesses custom software that filters domain matches. These filtering tools implement rules, such as minimum match E-value scores or minimum match alignment lengths, that are used to remove false-positives matches. These rules are unique to each member database and are custom to the sequence search tool used by each database. InterProScan implements all sequence search tools and the false-positive filtering methods used by member databases. All domain matches identified by InterProScan are considered to be true positives, as they have passed scrutinous match filtering rules. Pygenprop utilizes these true-positive InterProScan matches to predict genome properties. For a given dataset, there are often multiple proteins with predicted true-positive domain matches that support a single genome property step. However, of this set matching proteins, the protein with the lowest match E-value score for the step's supporting domain is most likely to carry out the step. As discussed in Section 3.4, Micromeda-Server's **get_fasta** endpoint returns the FASTA formatted sequences of either all proteins containing a match to a domain that supports a property step, regardless of the organism, or a single protein per organism, each with the lowest E-value match to the domain that supports a property step. Micomeda-Client's step protein download pop-up boxes provide web links that allow users to download FASTA files generated by the **get_fasta** endpoint.

Micromeda files, and another for presenting the visualizations of the file's data. To use Micromeda-Client, users must first navigate to the upload page and upload a Micromeda file. After the upload is complete, the user's browser will automatically redirect to the visualization page. Both pages are styled using the Bootstrap 3.0 CSS framework [141]. Bootstrap is used to set up page elements such as header navigation bars and drop-down menus. Bootstrap also makes each page compatible with tablet computers and phones as it will automatically restyle non-visualization page elements to fit on these device's smaller screens.

4.4.1 Core Data Structures

The client visualization page uses two core data structures and they are both accessed during visualization generation. One is a diagram configuration array that contains a series of measurements. These measurements are used during diagram drawing and control spacing between heat map cells, heat map cell dimensions, the offset of axis labels, and other visualization details (Fig. 4.4). The contents of this setting array are stored in a JSON file [25], called **diagram_configuration.json**, which is deployed alongside the HTML files of the client. The second data structure is a copy of the Genome Properties DAG in the form of a tree (*i.e.*, nodes with two parents are duplicated) of JavaScript objects. Each of these objects represents a genome property or step and are linked together in parent-child relationships. This data structure is analogous to the one used by Pygenprop (Section 2.2.7). Each of these objects possesses an attribute, called assignments, containing a list of property assignments for the organism in a dataset. Micromeda-Client later uses these assignments during the generation of its heat map. These property and step objects also have an attribute, called **enabled**, containing a boolean (*i.e.*, true or false). Elements of Micromeda's UI manipulate the **enabled** attribute of objects in the property tree and this, in turn, manipulates the contents of the visualization (Fig. 4.5). The **enabled** boolean of each property object tree determines whether the children of the property should be displayed in client visualization (Fig. 4.5). The property tree is placed within a parent JavaScript object. This property tree JavaScript object is analogous to objects instantiated from Pygenprop's GenomePropertiesTree class (Section 2.2.7) and, as such, also possesses methods for property identifier-based lookups of the tree's underlying property objects and methods for extracting lists of the tree's root and leaf property objects.



- MAX(#NtoL): The maximum number of intervening nodes to a leaf node.
- #Org: The number of organisms analyzed.
- #LN: Number of leaf nodes found underneath a node.
- RM,LM,TM,BM: The right, left, top, and bottom margins.
- TW: Tree Width
- HW: Heatmap Width
- DW: Diagram Width
- DH: Diagram Height
- DTO: Diagram Top Offset
- TAO: Top Axis Offset
- SVG-H: SVG Height
- SVG-W: SVG Width
- H*: Object Height
- W*: Object Width

Figure 4.4: **How pre-defined spacing, width, and length values determine the layout of Micromeda-Client's diagrams.** Changes in these values can shift the size and spacing of the heat map's cells and axes. Dimension values are stored in an external file that is loaded upon diagram generation. The contents of this file can be modified to change the layout of Micromeda-Client's heat map.

4.4.2 Loading the Visualization

As mentioned previously, requests for data from Micromeda-Server supports much of the functionality of Micromeda-Client. All these requests are done through a technique known as Asynchronous JavaScript and XML (AJAX) [77, 95] (see en.wikipedia.org/wiki/Ajax_(programming)). AJAX allows requests to be made to the server, using JavaScript

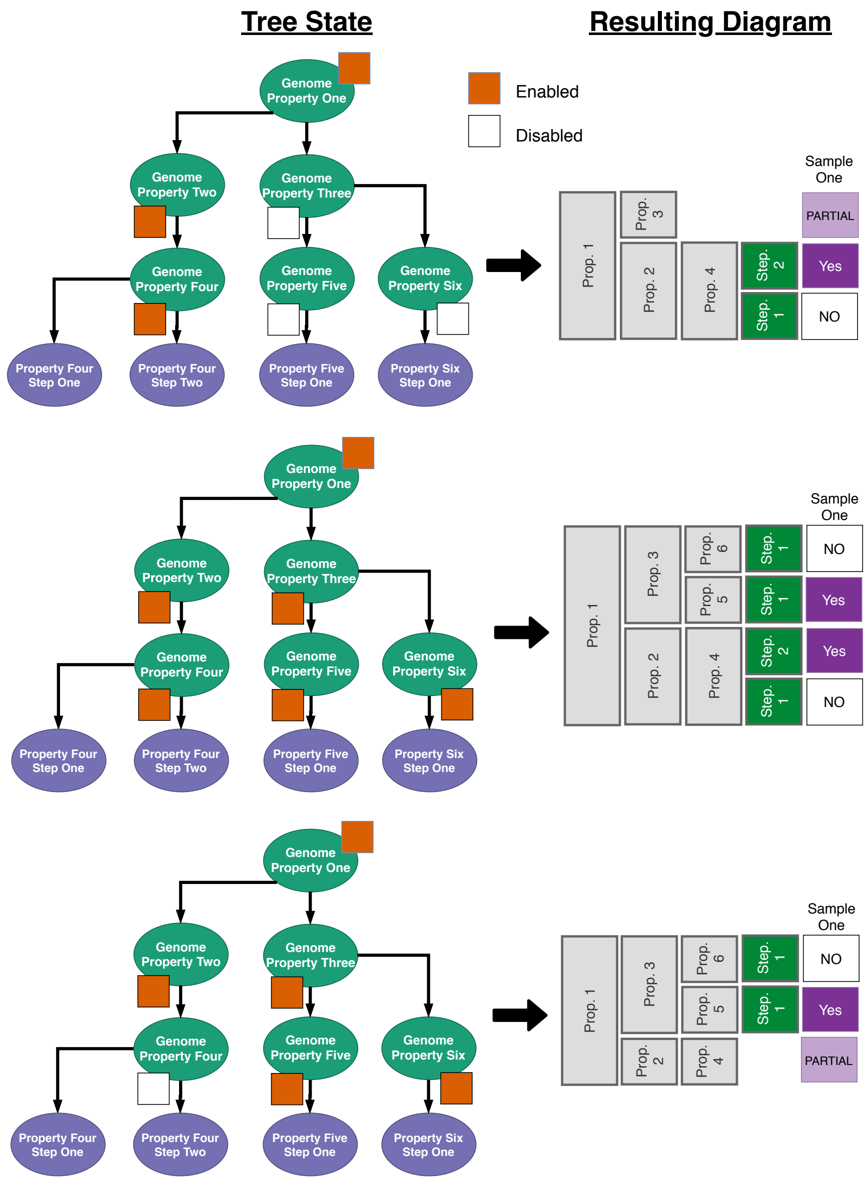


Figure 4.5: **How the state of Micromeda-Client's heat map diagram is controlled by the state of a client-side in-memory property tree.** Changing the state of enabled attributes of property objects in the tree causes the heat map and icicle diagram to be generated differently on subsequent renderings. When a property is enabled, then its children are rendered in the heat map.

[57], without the need for a web page reload. This technique allows Micromeda-Client to remain asynchronous to the server where it was downloaded. AJAX requests are made via HTTP [53], through a series of URL addresses [17] that map to Micromeda-Server endpoints (see Section 3.4). All AJAX requests to the server were made using the JQuery JavaScript library [30, 95]. The URL address of Micromeda-Server, where AJAX requests are made to, is found in a JSON file called server_config.json. This JSON configuration file is deployed with the HTML files of the client (i.e., the HTML files for the client's visualization page and upload page). Both the file upload and visualization page call this JSON file upon their initial load to acquire the location of Micromeda-Server.

The upload page contains a file drag and drop zone. When a user drops a Micromeda file onto this zone, the file is sent via AJAX to Micromeda-Server using the server's **upload** endpoint (see Section 3.4). The drag and drop zone was implemented using DropzoneJS [106]. After the file upload is complete, Micromeda-Server returns the file's associated dataset key to Micromeda-Client. This dataset key is stored in the browser's web local storage [68] (en.wikipedia.org/wiki/Web_storage) using a library called localForage [36]. Once the dataset key is stored, the client redirects the browser to the visualization page where visualizations of uploaded file's contents are generated.

As the visualisation page loads, the client requests a JSON tree from Micromeda-Server's **get_tree** endpoint (see Subsection 3.4) via AJAX. The dataset key saved in the browser's local storage is provided with this request and ensures that Micromeda-Server returns data from the browser's most recently uploaded Micromeda file. Micromeda-Server will respond to the **get_tree** request by returning a JSON tree containing assignments for all properties and steps within this uploaded Micromeda file. The client parses this JSON file into a JavaScript object tree and uses it as the property tree data structure mentioned in Subsection 4.4.1.The visualization is then built using the dimensions from the diagram configuration array, the structure of the property tree, and the **enabled** attribute of property objects contained within the property tree (see Subsection 4.4.1). The visualization itself is generated using functions from version 3.0 of the D3.js visualization library [23] and custom JavaScript code.

4.4.3 Interactivity After Initial Visualization Load

One of the critical properties of the client's visualizations is that they are interactive. This interactivity is provided by manipulating the JavaScript property tree (Fig. 4.5). When a user clicks a property node in the icicle diagram, a JavaScript onclick event [94] (en.wikipedia.org/ wiki/DOM_events) is triggered that changes the state of the equivalent property object in the property tree data structure. Specifically, the clicked node's equivalent property object's **enabled** attribute is inverted upon icicle node click (Fig. 4.5) and the entire visualization is subsequently re-rendered. Because the enable attribute has changed on the property object, the heat map visualization is re-rendered in a different configuration (Fig. 4.5).

In the case where a user clicks a leaf node of the icicle diagram, the client changes the matching property object's **enabled** attribute from false to true. Then, when the diagram is subsequently re-rendered, the children of the clicked node are also rendered (Fig. 4.5). The client software then automatically scrolls the page, via JavaScript, so that the bottom of the X-axis labels are aligned with the top of the clicked node. All scrolling behaviour in the client is facilitated by JQuery [95]. The opposite occurs when users click a non-leaf node of the icicle diagram. The client changes the associated property object's **enabled** attribute from true to false and, after re-rendering, the node's children are removed from the diagram. The client then scrolls so that the X-axis labels are aligned with the top of the clicked node. When a parent property object's **enabled** attribute is changed, this change is not cascaded to its children. This lack of transfer provides the visualization with a sort of "memory" where the visibility state of each grandchild property is retained (Fig. 4.5).

At page load, the property tree is used to create a JavaScript array of pairs of property names and identifiers. After the visualization generation, an interactive search menu is created in the top right-hand corner of the Micromeda-Client interface (Fig. 4.3). This menu is built using the Select2 library [162] (select2.org) and uses the previously mentioned pairs of property identifiers and names as search data. When a user searches for a property name or identifier in the search menu, properties whose names or identifiers are similar to the search query are displayed in a drop-down menu (Fig. 4.3). When the user clicks one of these menu properties, then another JavaScript event occurs where the parent of the matching property object in the JavaScript property tree is modified. All property objects along the property tree in a path from the root to the parent of matching property object have their **enabled** attribute set to true. When client visualization is subsequently re-rendered, a path to the assignments of the clicked property is displayed along with the property's assignments. The client software then automatically scrolls the page, via JavaScript, so that the bottom of the X-axis labels are aligned with the top of the icicle diagram cell representing the clicked property.

After each re-rendering of the assignment visualization, Micromeda-Client sends a series of AJAX requests to Micromeda-Server. These are requests for information about each property that is visible in the icicle diagram (Fig. 4.3). These requests are sent to Micromeda-Server's **Get_Single_Genome_Property _Info** endpoint (see Section 3.4). The endpoint returns a JSON document containing information about each property that is later displayed in property information pop-up boxes (Fig. 4.3). The contents of these documents are cached in web local storage using localForage [36]. During the subsequent diagram renderings, requests are not made for properties whose data is already cached. Caching property information client-side and only requesting information about visible properties reduces the number of requests sent to Micromeda-Server and the overall load to be handled by the server computer system that Micromeda-Server is run on.

Both property information and FASTA download pop-up boxes are generated upon activation of JavaScript onmouseover events [94] (en.wikipedia.org/wiki/DOM_events) of the question mark and step download glyphs, respectively. When a question mark glyph of a property icicle diagram node is hovered over, information about the property is retrieved from web local storage using localForage. This property information was cached during the previous diagram render.If no information for a property was previously cached, a property information template is used. The cached or template property information is used to generate the contents of the property informationpop-up box. The FASTA download pop box, when generated, contains links for downloading FASTA files. These links point to URLs provided by Micromeda-Server's **get_fasta** endpoint (see Section 3.4). When the"get all proteins" link is clicked (Fig. 4.3), the endpoint URL's **all** HTTP GET parameter is set to true during the request and a FASTA file containing all proteins that support the step is returned (see Section 3.4). The **all** HTTP GET parameter is not added to requests initiated by clicking the "get top proteins" link of the pop-up. This lack of a parameter causes the return of a FASTA file containing only the proteins that are most likely to support a step. During both of these types of sequence file requests, the dataset key stored in local web storage is attached. This key tells the Micromeda-Server to generate FASTA files from sequences found in the browser's most recently uploaded Micromeda file. Both pop-up boxes are hidden upon removal of the user's cursor via the triggering of an onmouseout event [94] (en.wikipedia.org/wiki/DOM_events).

4.5 Performance Testing, End-User Testing, and Demonstration of the Client User Interface

Performance testing found that the client's visualization was capable of being re-rendered almost instantaneously even when several hundred rows of assignments were displayed. All interactive components of Micromeda-Client had practically no visual lag attributed to their construction. All UI delay was caused by waiting for data from Micromeda-Server endpoints. Ways of speeding up these endpoints are discussed in Section 3.7.

A demo of Micromeda's client interface can be found at tundra-pear.glitch.me. This demo is hosted in a way that certain features that require Micromeda-Server, such as Genome Property pop-ups and FASTA file downloads, are disabled. A video that displays these disabled features can be found at tinyurl.com/wlgelmg. This video also demonstrates the process of uploading a Micromeda file. A tutorial that covers how to generate Micromeda files can be found at github.com/Micromeda/micromeda-workflow.

Three potential users of Micromeda-Client were provided with a customized version of the demo interface linked in the preceding paragraph. These interface demos were customized to contain data relevant to each user's research project. The potential users were required to provide the author with multiple microbial genomes or MAGs in FASTA format. The author put these sequence files through the Micromeda pipeline (*i.e.*, protein prediction using Prodigal, scanning for Interpro domains with InterProScan, and Genome Properties assignment with Pygenprop). The resulting property and step assignments were integrated into a customized version of the client demo. The author generated one demo for each user. The users that were provided with demos were from both academia and industry. Specifically, they consisted of the following:

- An academic researcher interested in using Micromeda to analyze a series of MAGs of putatively ammonia-oxidizing organisms found in wastewater treatment plant samples.
- An academic researcher that wanted to use Micromeda to analyze a series of MAGs of anoxygenic phototrophic bacteria gathered from boreal shield lake water samples.
- A commercial entity interested in using Micromeda to assist them with the trait-based down-selection of plant growth-promoting microbes from an existing genetically sequenced culture collection. They hoped to use the down-selected organisms in agricultural field trials.

These users were interviewed via video conference after they were provided with their customized demo. The interviews were freeform and open-ended. However, the following key questions were asked:

- Was the potential user happy with how Micromeda-Client presents property and step assignment information?
- How intuitive did they find the client's UI and were there any changes to the interface that could be made to make it more intuitive?

- Is there any aesthetic changes to Micromeda-Client that the user would recommend?
- What additional features should be added to the client to improve its data analysis capabilities?

After user interviews, suggestions for improving the client's UI were recorded. Several of the potential user's recommendations are discussed in Section 4.7.

4.6 Deployment

Before Micromeda's UI can be used, it must be downloaded into a web browser. A web server must be made available to send the client code to the user's browser. Micromeda-Client can be deployed in two ways. The client can be served from a web server, such as NGINX [129], running on the same server computer system as Micromeda-Server (Fig. 4.6). Such a web server is already a component of the medium-scale deployment of Micromeda-Server described in Section 3.6. Alternatively, Micromeda-Server can be served from a content delivery network (CDN) [50] (en.wikipedia.org/wiki/Content_delivery_network) such as Amazon Cloudfront [158] (aws.amazon.com/cloudfront) or Cloudflare Anycast [27] (cloudflare.com/cdn). In this deployment configuration, the end-user downloads the client code from the nearest content delivery server in the CDN, rather than the same server where Micromeda-Server is hosted (Fig. 4.6). After loading, the client would then send API requests to Micromeda-Server, which would be hosted outside the CDN (Fig. 4.6). During end-user testing, a version of Micromeda-Client was deployed via Amazon Cloudfront CDN. The files for Micromeda-Client could also be downloaded and opened by a developer's web browser for development and testing purposes.

4.7 Future Improvements

Several improvements could be made to the client that would increase its overall usefulness. These features would make the client more natural to use and provide users with more information about their datasets. Several of these potential improvements were derived from feedback gathered during end-user testing.

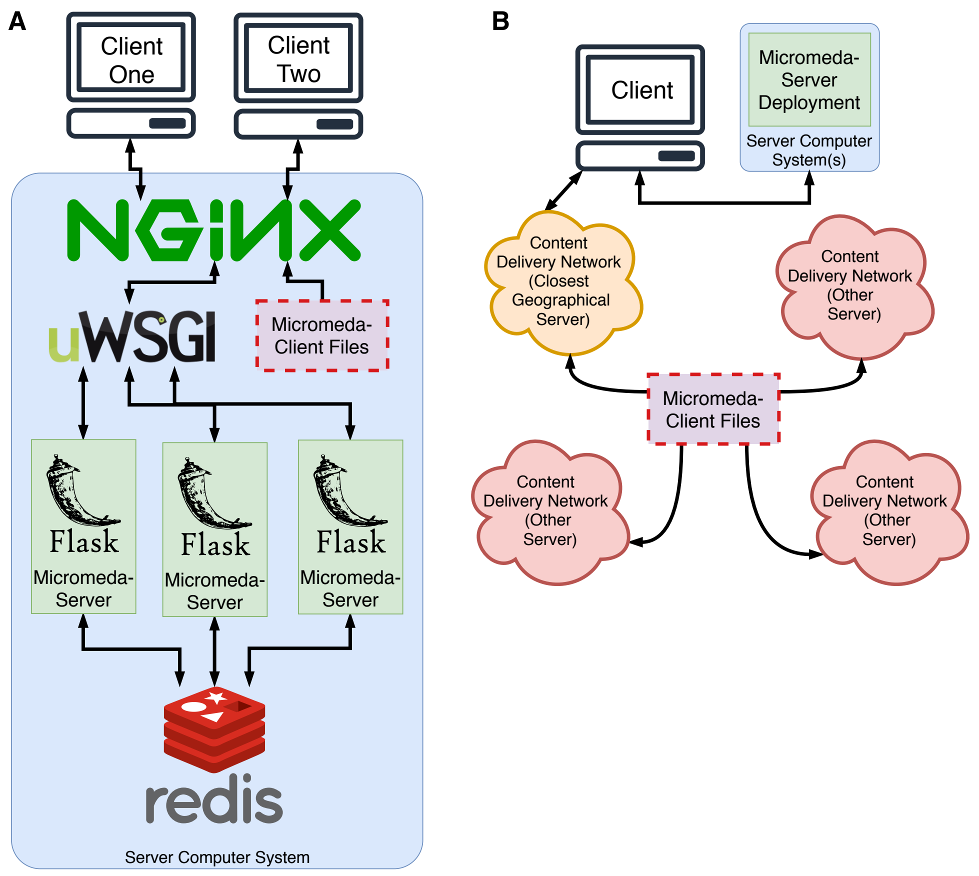

A
Client One
Client Two
NGINX
uWSGI
Micromeda-Client Files
Flask
Micromeda-Server
Flask
Micromeda-Server
Flask
Micromeda-Server
redis
Server Computer System
B
Client
Micromeda-Server Deployment
Server Computer System(s)
Content Delivery Network (Closest Geographical Server)
Content Delivery Network (Other Server)
Micromeda-Client Files
Content Delivery Network (Other Server)
Content Delivery Network (Other Server)

Figure 4.6: **The two primary deployment strategies for Micromeda-Client.** The client files for Micromeda-Client, in production, can either be deployed on the same server computer system as Micromeda-Server (A) or deployed via a CDN (B).

4.7.1 Providing More Information About Property Steps

In the current version of Micromeda-Client, there are pop-up boxes that provide additional information about individual genome properties, such as property descriptions and links to equivalent records in other pathway databases. However, there are no equivalent pop-up boxes for providing more detailed information about property steps. The Genome Properties database includes additional information about steps that could be displayed in another set of pop-up boxes. For example, using information about what domains support a step could be used to generate links to domain records on the InterPro website. These links could be added to the suggested step pop-up boxes. Also, such pop-up boxes could provide

GO terms and links to details about these terms on the GO website because individual steps are associated with such terms. The information in step information pop-up boxes would provide additional context to heat map step assignments. The boxes would be activated by hovering over a question mark glyph placed slightly above the download glyph of each step node in the client's icicle diagram (Fig. 4.3). The addition of step information pop-up boxes would require an additional endpoint to be added to Micromeda-Server.

4.7.2 Providing Improvements to Search Capability

Currently, the search menu in the top right corner of the client interface allows users to search for genome properties by name or identifier (Section 4.3 and Fig. 4.3). The ability to search for properties could be expanded by allowing users to search for properties by the identifier of equivalent records from KEGG [80] or MetaCyc [85]. These new search terms would allow users, who are familiar with the identifiers of specific KEGG or MetaCyc pathways, to rapidly find the equivalent pathway's genome property and its assignments in the Micromeda-Client heat map. It may also be useful to be able to search for property steps, rather than just properties, by name and have the visualization open a path and scroll to the assignment row for a searched step. Steps could also be searched for by their associated InterPro domain signature identifiers or GO term numbers. Improved ability to search for properties and steps would require additional endpoints to be added to Micromeda-Server.

4.7.3 Automating the Scaling of the Visualization for Different Screen Sizes

As mentioned in Subsection 4.4.1, the visualization generated by Micromeda-Client relies upon a file called **diagram_configuration.json** that contains a series of measurements. These measurements consist of length, width, and spacing values that control the layout of the client's heat map and icicle diagram (Fig. 4.4). The default values for these measurements, as stored in **diagram_configuration.json**, facilitate the generation of an adequate diagram layout for most datasets. However, for several datasets, such as those with long organism names or large numbers of samples, the default diagram measurements can cause visual anomalies such as text clipping between diagram cells and organism names being displayed off-page.

To fix such anomalies, future versions of Micromeda-Client could use JavaScript to automatically adjust diagram configuration measurements to fit different datasets or browser window sizes. For example, X-axis spacing values could be changed dynamically based on the length of the longest organism name in a dataset. Alternatively, the heat map cell width could be adjusted based on the number of samples in a dataset. By adjusting the default diagram layout values dynamically, Micromeda-Client could generate diagrams that better fit a variety of datasets and user devices.

4.7.4 Modification of Micromeda-Client to Collect Assignment Data From a Separate Endpoint

Currently, Micromeda-Client gathers its assignment data from the **get_tree** endpoint of Micromeda-Server (see Section 3.4). There are problems with this approach, as discussed in Subsection 3.7.2, and it would be more appropriate to have a separate server endpoint for returning assignments for properties and steps. In addition to solving the problems mentioned, having the client make a separate API call to gather these assignment data would be useful as it would allow the client to request for Micromeda-Server to rearrange the data before it is returned. This reconfigurability would also allow for step assignments that are supported by protein domains to be replaced by match E-value scores or for assignments of organisms to be returned in a different order.

4.7.5 Clustering Heat Map Columns by Assignment Similarity

Columns in the heat map contain assignments from different organisms. Currently, these columns are ordered alphanumerically by organism name. Users may find it useful to be able to cluster these columns by assignment contents rather than by name. Clustering by assignment contents would group organisms that have similar assignments and potentially similar metabolic, physiological, or structural characteristics. Columns could be clustered either globally by the assignments of all properties and steps or locally by only those properties and steps that are visible in the current diagram rendering. The simplest solution for clustering these columns would be to use Micromeda-Server. As noted in Subsection 3.7.2, if the return of property and step assignments were pushed to a separate endpoint, then the data returned would have to be generated by serializing assignment DataFrames of GenomePropertiesResultsWithMatches objects to JSON [25]. If clustering columns in the heat map was required, then Micromeda-Server could accomplish this visual clustering by clustering the DataFrames of GenomePropertiesResultsWithMatches objects column-wise using Scikit-learn [123] and pandas. Afterwards, these clustered DataFrames would then be converted to JSON and sent to the client. Returning clustered assignment JSON data could be controlled by an HTTP GET parameter in an endpoint request.

4.7.6 Visualizing Step Assignments by E-value Score Rather Than by Categorical Assignment

Leaf genome properties are supported by steps that use InterProScan matches to InterPro consortium domain signatures as evidence. These matches, between a motif found in a protein of an organism and an InterPro domain, have an E-value score. Micromeda-files store these scores. Matches with E-value scores that are below a specific per-InterPro member database threshold are filtered out by InterProScan automatically. Matches that remain are likely to be true positives (*i.e.*, the motif matched is an ortholog to the domain from the database). However, there is still some E-value score variation among the remaining matches. Motifs with matches that have lower E-value scores are closer in sequence to domains in the database and are more likely to be orthologous. These lowest E-value matches are be said to be the“strongest” matches.

During end-user testing, several potential users were interested in having a way to compare the relative strength of step assignments between organisms. For example, if a property is assigned YES in two organisms, which organism is more likely to possess a step? Step assignments are currently assigned a binary YES or NO, and thus the relative strength of these assignments cannot be compared. One way to compare step assignments between organisms would be to compare the strength of these assignments' supporting domain matches. For example, Micromeda-Client could display the E-value score of the closest match supporting each step in its heat map in place of a binary YES or NO value. These E-value scores could be coloured by shades of green along a continuous scale. Pop-up boxes could also be generated by hovering over each cell in the heat map cell and would display match info, such as the E-value score, protein name, and matched domain identifier. No E-value data would be presented for NO assignments. An interface switch could be implemented that would be used to switch property assignments between binary YES or NO values and continuous E-value scores. A Micromeda-Server endpoint would have to be built to provide these E-value score data, as discussed in Subsection 3.7.3.

4.7.7 Improving Filtering Capability of the Client

During end-user testing, users requested a way to identify assignments that differ between organisms. Such differences could be highlighted by removing assignment rows from the heat map that possess the same assignment across all organisms in the dataset, leaving only those that differ. For example, a UI switch would have to be implemented that would allow users to switch between a complete assignment view of all assignment rows and only those that differ. The assignment rows would have to be dropped based on those that are displayed in the current version of the heat map. Each node in the JavaScript property tree could be given a property called **differing** that returns true if the property's associated assignments are differing between organisms and false otherwise. Children of nodes on the JavaScript property tree whose state is set to **enabled** would only be rendered if their **differing** property is set to true.

The icicle diagram could be re-rooted based on the assignments and child assignments of a specific property. For example, having the visualization show the property for iron metabolism and its children. This feature could be implemented by having a glyph on each node of the icicle diagram that, when clicked, allows for the diagram to be re-rooted at the node. In the future, this feature could be expanded for users to select multiple properties to display, for example by clicking three of the re-root glyphs and an interface button. Being able to perform selections in this way would allow users only to compare the assignments of two or more properties that are distant in the Genome Properties DAG.

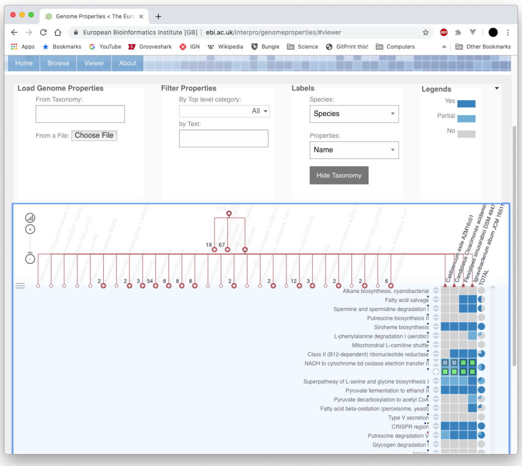
4.8 A Comparison of Micromeda-Client to Other Pathway Visualization Software

Several web-based software tools already exist for visually comparing the presence and absence of biochemical pathways across organisms. However, while interviewing potential users of Micromeda-Client, the interviewees conveyed their frustration with these existing tools. Users mentioned how these tools forced them to navigate through multiple web pages to gather the information needed to perform their analyses. This process is time-consuming and error-prone.In response to this information, Micromeda-Client was designed to integrate as much information as possible into a single view and interactivity is used to allow users to show and hide property assignment information as needed. Below is a detailed comparison between Micromeda-Client and three similar visualization tools.

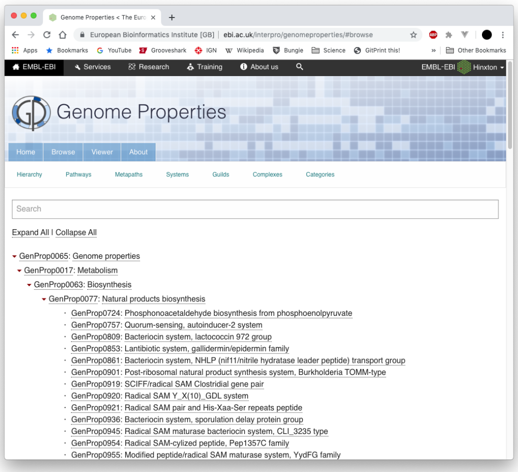
As of fall 2019, there is only one Genome Properties assignment visualization tool available other than Micromeda-Client. This tool is part of the EBI Genome Properties website [132] (ebi.ac.uk/interpro/genomeproperties) and has an assignment viewing page that displays a heat map similar to the one generated by Micromeda-Client [132] (Fig. 4.7a and see ebi.ac.uk/interpro/genomeproperties/#viewer). In contrast to Micromeda-Client, information about the properties displayed is not accessible from the heat map itself and must be viewed in a secondary web page. This page contains a property browser (Fig. 4.7b and see ebi.ac.uk/interpro/genomeproperties/#browse). Even when a user finds a property on this browsing page, they must open a third page that contains the property's information such as a description and links to equivalent records. Thus, with the Genome Properties website, if users need a more detailed description of a property, then they are forced to swap between multiple web pages. To effectively use the EBI tool, from a UI perspective, users should place both property browser and assignment viewer windows side by side. However, using both of these windows simultaneously requires either a large monitor or two monitors placed side by side because the assignment heat map page of the website does not scale down well with a shrinking window size. If both property information and assignment information pages are open simultaneously on a 14" laptop, then the controls of the assignment heat map page are cut off, and the visualization becomes unusable. Unlike Micromeda-Client, the Genome Properties website's assignment viewer only displays a subset of leaf genome properties and their steps, not all properties and steps (Fig. 4.7a). The website also does not perform any aggregation of assignments to reduce the length of its heat map (Fig. 4.7a). Like the client, users can upload their data. In the case of the EBI assignment viewer, users can upload InterProScan TSV files instead of Micromeda files [132]. Thus, there is no way for users to access the underlying protein sequences that support each step.

In contrast to the property assignment heat map generated by the Genome Properties website, Micromeda-Client uses pop-up boxes to display its property information. Users of Micromeda-Client do not have to search for information about a property of interest in an entirely new window, nor do they need to swap between multiple windows to find this information. Having all the information in a single view saves users time and reduces mistakes where users view information about the wrong property. Also, unlike theEBI Genome Properties website, Micromeda-Client displays assignments for all properties, not just leaf properties. The icicle diagram that is present in the client's visualization helps organize property assignments, so users more easily find these assignments. Unlike the EBI tool, the client's assignment viewer does use interactive aggregation and disaggregation, which decreases the length of its heat maps substantially.

Other websites that visualize pathway annotation information do use aggregation and disaggregation in the same way as Micromeda-Client [154, 42]. However, some of these tools have interface issues. For example, Microscope [154], a pathway annotation website, also presents heat maps conveying levels of completeness for KEGG [80], MetaCyc [85], and antibiotics & Secondary Metabolite Analysis Shell (antiSMASH) [20] pathways (Fig. 4.8). Like the Genome Properties database, both KEGG and MetaCyc organize their


Genome Properties < The Euro
European Bioinformatics Institute [GB] | ebi.ac.uk/interpro/genomeproperties/#viewer
Home
Browse
Viewer
About
Load Genome Properties
From Taxonomy:
From a File: Choose File
Filter Properties
By Top level category:
All
by Text:
Labels
Species:
Species
Properties:
Name
Hide Taxonomy
Legends
Yes
Partial
No
Caldisericum exile AZM16c01
Candidatus Cloacimonas acidaminovorans
Flexistipes sinusarabici DSM 4947
Ignavibacterium album JCM 16511
TOTAL
Alkane biosynthesis, cyanobacterial
Fatty acid salvage
Spermine and spermidine degradation I
Putrescine biosynthesis II
Siroheme biosynthesis
L-phenylalanine degradation I (aerobic)
Mitochondrial L-carnitine shuttle
Class II (B12-dependent) ribonucleotide reductase
NADH to cytochrome bd oxidase electron transfer II
Superpathway of L-serine and glycine biosynthesis I
Pyruvate fermentation to ethanol II
Pyruvate decarboxylation to acetyl CoA
Fatty acid beta-oxidation (peroxisome, yeast)
Type V secretion
CRISPR region
Putrescine degradation V
Glycogen degradation I

(a) The Genome Properties viewer displays heat maps containing assignments for both reference and novel organisms.



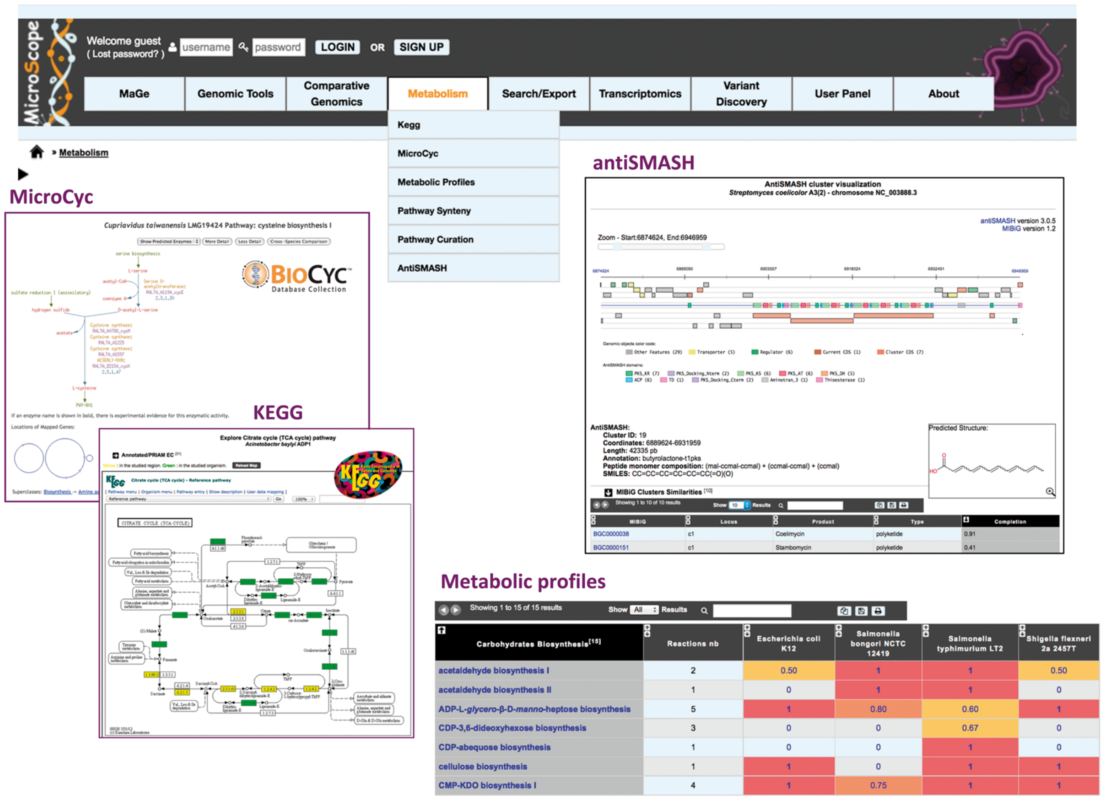
- GenProp0065: Genome properties
  - GenProp0017: Metabolism
    - GenProp0063: Biosynthesis
      - GenProp0077: Natural products biosynthesis
        - GenProp0724: Phosphonoacetaldehyde biosynthesis from phosphoenolpyruvate
        - GenProp0757: Quorum-sensing, autoinducer-2 system
        - GenProp0809: Bacteriocin system, lactococcin 972 group
        - GenProp0853: Lantibiotic system, gallidermin/epidermin family
        - GenProp0861: Bacteriocin system, NHLP (nif11/nitrile hydratase leader peptide) transport group
        - GenProp0901: Post-ribosomal natural product synthesis system, Burkholderia TOMM-type
        - GenProp0919: SCIFF/radical SAM Clostridial gene pair
        - GenProp0920: Radical SAM Y_X(10)_GDL system
        - GenProp0921: Radical SAM pair and His-Xaa-Ser repeats peptide
        - GenProp0936: Bacteriocin system, sporulation delay protein group
        - GenProp0945: Radical SAM maturase bacteriocin system, CLI_3235 type
        - GenProp0954: Radical SAM-cylized peptide, Pep1357C family
        - GenProp0955: Modified peptide/radical SAM maturase system, YdFG family

(b) The Genome Properties browser provides information about individual properties and their hierarchy.

Figure 4.7: **Browser windows containing the two main webpages of the Genome Properties website.** The viewer page (a) is used for viewing property and step assignments. The browser page (b) helps users learn more about individual genome properties. The viewer page (a) allows for the upload of InterProScan TSV files of novel organisms. These files are used to calculate assignments for these organisms. With Micromeda-Client, the same information presented by these two pages is integrated into a single page.

pathways hierarchically. However, with Microscope, when a user clicks the visualization interface to gain access to the completeness of child pathways, an entirely new heat map is generated on a separate page. Child pathway assignments are not displayed within the same heat map view (Fig. 4.8). When navigating results from high-level pathways to low-level pathways, users are required to open several heat map pages. To access a heat map containing results for pathway steps, users may have to open five or more separate pages. In addition to opening each level of the visualization on a separate page, if users need to find additional information about a pathway, then they need to click on links that take them to a separate page. This page mirrors a page on the KEGG or MetaCyc website that describes the pathway (Fig. 4.8). There is no way to compare the cousin pathway results within the same heat map (Fig. 4.8).

In contrast to Microscope, Micromeda-Client displays its aggregated and disaggregated heat map rows within the same view. Thus, users are not required to swap back and



AntiSMASH:
Cluster ID: 19
Coordinates: 6889624-6931959
Length: 42335 pb
Annotation: butyrolactone-t1pks
Peptide monomer composition: (mal-ccmal-ccmal) + (ccmal-ccmal) + (ccmal)
SMILES: CC=CC=CC=CC=CC=CC(=O)(O)

MIBiG Clusters Similarities [10]

Showing 1 to 10 of 10 results

MIBiG	Locus	Product	Type	Completion
BGC0000038	c1	Coelimycin	polyketide	0.91
BGC0000151	c1	Stambomycin	polyketide	0.41

Showing 1 to 15 of 15 results

Carbohydrates Biosynthesis[15]	Reactions nb	Escherichia coli K12	Salmonella bongori NCTC 12419	Salmonella typhimurium LT2	Shigella flexneri 2a 2457T
acetaldehyde biosynthesis I	2	0.50	1	1	0.50
acetaldehyde biosynthesis II	1	0	1	1	0
ADP-L-*glycero*-β-D-*manno*-heptose biosynthesis	5	1	0.80	0.60	1
CDP-3,6-dideoxyhexose biosynthesis	3	0	0	0.67	0
CDP-abequose biosynthesis	1	0	0	1	0
cellulose biosynthesis	1	1	0	1	1
CMP-KDO biosynthesis I	4	1	0.75	1	1

Figure 4.8: **User interface of and analysis services provided by the MicroScope annotation server.** The server presents both pathway assignment heat maps and pathway diagrams. The heat maps show the presence and absence of pathways across reference organisms and novel organisms whose genomes have been uploaded in FASTA format. The pathway diagrams allow users to explore further the pathways annotated by the server. Figure is from [154].

forth through multiple pages to find child pathway results and to understand how results for parent properties are calculated. Information about the properties is also accessible through pop-ups from within the same view. The assignment of cousin properties is easily seen within the same heat map view.

In contrast to the Genome Properties website and Microscope, some alternative pathway annotation visualization tools do present their data hierarchically within the same visualization [42]. For example, FuncTree2 [42] allows users to plot KO annotation [100, 81] frequency (i.e, the number of proteins of an organism that that possess a specific KO an-

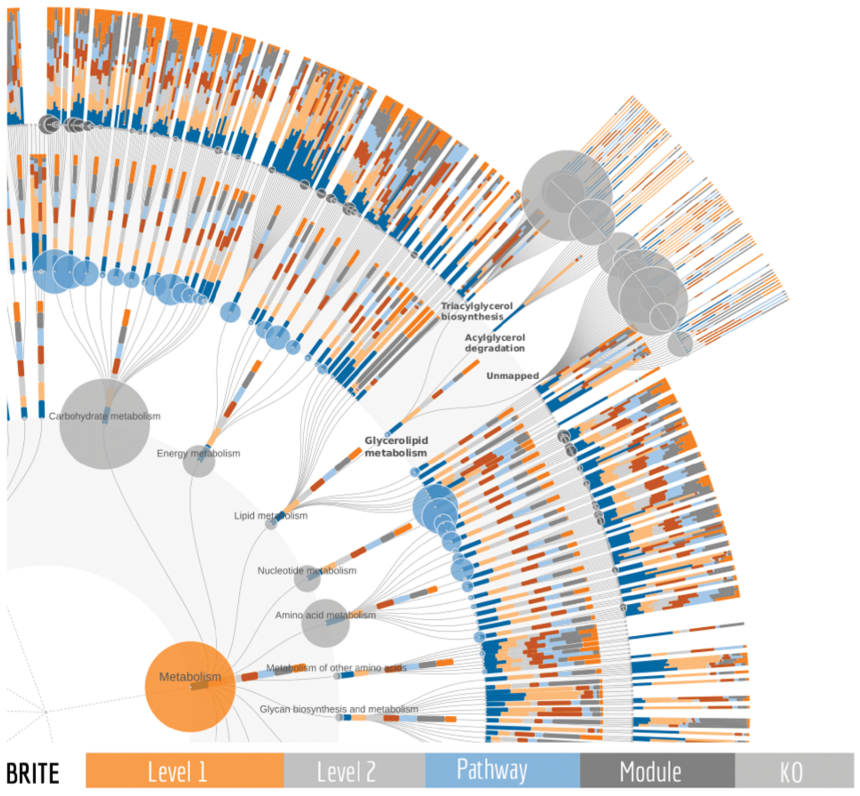


Figure 4.9: **Overview of the KEGG annotation visualization produced by FuncTree2.** FuncTree2 presents a radial tree diagram that shows the abundance of KEGG Orthology (KO) annotations within multiple organisms' predicted proteomes. In diagrams created by the tool, leaves of the radial tree represent individual KO annotations. In other words, each leaf node represents a type of protein that can be found in a cell. Nodes closer to the root represent different levels of classification that group these KO annotations according to the hierarchy of pathways in the KEGG database. A stacked bar chart annotates each leaf node. This chart possesses coloured bars representing the number of proteins in each organism's proteome with a specific KO annotation. These bar charts are oriented radially. Stacked Bar charts also annotate other nodes in the radial tree. The stacked bar charts on higher-level nodes represent reciprocal summations of KO counts of child nodes. Nodes can be clicked to remove child nodes and change the shape and size of the visualization. Figure is from [42].

notation) across multiple organisms (Fig. 4.9). Frequencies are visualized using a radial tree with nodes annotated by stacked bar charts. The bar charts of leaf nodes of the tree represent counts for a singular KO, whereas charts annotating nodes closer to root display summed KO counts for child nodes (Fig. 4.9). Unlike Micromeda-Client, aggregated and disaggregated data are presented simultaneously. Nodes in the tree can be clicked to add and remove child nodes from the visualization. Instead of the default KEGG-based tree, FuncTree2 also accepts the upload of custom built trees and annotation frequencies derived from other pathway databases.

One of the key design differences between FuncTree2 and Micromeda-Client is their choice of radial and linear layouts, respectively. In the application note for FuncTree2, its authors note that they chose to use a radial tree over horizontal trees in their visualization to save screen space. Radial trees are more space-efficient than horizontal trees when presenting large numbers of nodes [26]. However, at least one study has shown, via eye-tracking, that radial trees underperform traditional and orthogonal tree layouts for a variety of visual search tasks [26]. Another study has shown that icicle diagrams (as used by Micromeda-Client) allowed users to have higher interaction accuracy and efficiency [114] than radial sunburst diagrams (as used by tools such as Krona [116] (github.com/marbl/Krona)). Such gains in user interaction capability should also carry over to Micromeda-Client, which uses a rectangular, rather than a radial layout[4].

Rather than defaulting to a horizontal tree or radial tree, micromeda's client replaces both of these trees with an icicle diagram. This icicle diagram provides superior spacial compactness to either tree type as no edges have to be rendered and nodes can be placed adjacently in a compact fashion. Rectangular diagrams have better space utilization than radial diagrams on modern computer monitors [114]. In contrast to FuncTree2, the visualization of Micromeda-Client only allows users to show either aggregated assignments for a parent node or the disaggregated assignments of its children, not both simultaneously. Both the decision to use icicle diagrams and to either present parent or child assignments allow the client's diagram to retain the same square layout as an annotated horizontal tree with superior space utilization to a radial tree.

4.9 Summary

The web client of Micromeda allows users to visually explore and compare assignments for genome properties and their steps across organisms. The software also provides information about these properties and steps and provides links between them and equivalent records in

[4]I assert that radial layouts are most useful when they show connections between data on opposite sides of the layout. This design pattern is used by visualization tools such as Circos [91].

other databases. Due to the use of Micromeda files, the client also allows users to download protein sequences that support the displayed assignments. These functionalities directly support the required tasks listed in Section 4.1.

As discussed in the Section 4.8, the client's visualization addresses many interface issues that afflict other pathway annotation visualization software tools. As discussed in the summary section of Chapter 3, another feature that sets Micromeda apart from these tools is the access to supporting information used in property and step assignments such as protein sequences. In the future, and as discussed in Subsection 4.7.6, the data presented by Micromeda-Client could be further expanded to display more information such as E-value scores. Annotation frequency, as displayed by default by FuncTree2, could also be readily displayed by the client. Overall, Micromeda-Client will increase users' ability to utilize pathway annotation information and set a new standard for pathway annotation visualizations.

Chapter 5

Conclusions

Micromeda is designed to help researchers compare the genomically predicted functional characteristics across multiple organisms. Two key elements facilitate this capability. Firstly, Pygenprop, a library that assists Micromeda's other components, can be used to compare functional characteristics across organisms programmatically. Secondly, Micromeda-Client can be used to generate an interactive heat map that allows users to perform the same comparisons visually. Micromeda-Server was developed to provide an API that provides data to this client. Micromeda files allow for the transfer of entire datasets of pathway predictions and supporting information such as protein sequences to Micromeda-Client. These files also enable the rapid transfer of datasets between researchers. Pygenprop is used to generate these Micromeda files, and the client uses their contents to draw its heat maps. The functional predictions made and displayed by Micromeda are derived from the information found within the Genome Properties database and the domain annotations of an organism's proteins.

5.1 Micromeda in the Context of Previous Work

In contrast to many pathway annotation systems (see Sections 3.8 and 4.8) that are either run remotely or locally on a researcher's computer, Micromeda has both local and remote components. InterProScan and Pygenprop are run locally on users' computers and are used to generate Micromeda files. These files are later uploaded to Micromed-Client, which is deployed on a remote server along with Micromeda-Server. The main reason for this split is to shift the computational cost of generating pathway annotations onto users while allowing UI components, such as Micromeda-Client, to be centrally hosted. This central hosting will allow for the rapid deployment of updated versions of Micromeda with new UI features and bug fixes.

From a human-computer interaction perspective, Micromeda provides a much-improved UI for visualizing the difference in predicted function characteristics (Section 4.8). Unlike other tools, Micromeda-Client uses interactivity to dynamically switch between displaying summaries of the presence of multiple pathways and the presence of individual pathways steps. The client also integrates various pieces supporting information, such as descriptions of select genome properties, directly into its interface. This level of integration is in contrast to other tools that require users to view such information in separate web pages.

Another key feature of the Micromeda that makes it stand out from other tools is the platform's ability to connect pathway annotations to the predicted protein sequences that support them. Both Pygenprop and Micromeda-Client allow users to generate FASTA files that contain proteins, from all organisms in a dataset, that support a property step. These FASTA files could be used to build phylogenetic trees. This feature is not available in other tools.

Pygenprop is one of the first libraries designed to support the programmatic comparison of genomically predicted functional characteristics. When combined with tools such as Jupyter Notebooks, the library provides a powerful tool for both developing pathway analysis tools and performing rapid analyses. Pygenprop is the second library that is compatible with Genome Properties data and is one of the only pathway comparison libraries written for Python. The library is also one of the first pathway analysis tools to integrate with the Python data science ecosystem.

As the number of sequenced genomes increases and the cost of sequencing becomes more affordable, there will be an increasing need for tools that can rapidly compare the pathways of multiple organisms or even entire genome-resolved microbial communities. Micromeda provides tools that can scale with these increasing datasets. Micromeda-client can be used to visually compare the pathways of multiple organisms, and Pygenprop can examine the possessed pathways of thousands. Micromeda's improved user interface design and ability to connect pathway annotations to protein sequences will improve the throughput at which researchers can perform pathway analysis.

5.2 Future Improvements to Micromeda

As discussed in Section 1.5, the summary of each chapter of the thesis examines potential improvements to the component that the chapter reviews. However, some substantial improvements require modifications to multiple parts of the Micromeda platform or are outside the scope of previous chapters. These changes are discussed in the subsections below.

5.2.1 Creation of an Automated Pipeline for Rapid and Easy Generation of Micromeda Files

Currently, there is no automated pipeline that users can apply to generate Micromeda files. As detailed in Section 1.4, users are required to install and run three CLI bioinformatics tools on their organisms' raw genome sequences to build Micromeda files. For convenience, an automated bioinformatics pipeline should be developed that would first install all these bioinformatics tools and later run them in series on user-supplied genomes. Components of the pipeline could be deployed via Conda (conda.io) and run in Conda environments to prevent these components from interfering with software installed previously. Pipeline automation tools such as Snakemake [89] or Nextflow [46] could be used to develop such a pipeline. These tools ensure that steps in the pipeline are executed in the correct order and allow for these steps to be scaled out in parallel if multiple organism's genomes are to be processed.

5.2.2 Add the Option to Assign Properties According to Percentage Completeness Rather Than Categorically.

Currently, the heat map generated by Micromeda is coloured according to property assignments that follow a discrete scale. Each property is assigned YES, PARTIAL, or NO support and is tinted to match. It should be noted that the algorithm employed to generate these assignments uses only the presence of required steps for a property in its calculations (see Subsection 2.3.2). If even a single required step is not supported, then the parent property is assigned PARTIAL support. If Micromeda is applied to incomplete genomes, such as those generated from metagenomes, then many genes encoding for proteins that support property steps may be missing. These missing genes would result in many properties being assigned PARTIAL support.

Users may find it useful to be able to quantify the level of support for each assigned property along a continuous scale. This change would address these potentially high levels of PARTIAL property assignments created by running Micromeda on incomplete genomes. For example, properties that have more of their steps supported could be given higher values than those that are incomplete. Non-required steps could also influence such a continuous scale. In the context of a heat map, these continuously scaled assignments of support would be assigned different shades of the same colour.

Algorithms for calculating levels of PARTIAL support for biochemical pathways have been developed before. For example, MAPLE [146], a KEGG-based pathway annotation server, can calculate the module completion ratio (MCR) (*i.e.*, continuous levels of support based on the presence of pathway steps) for each module in the KEGG database. KEGG modules are roughly equivalent to higher level genome properties. MCRs are calculations of the level of completeness of individual pathways that take into account that multiple different enzymes may catalyze some pathway steps. These ratios are calculated using custom "boolean algebra-like equations" generated for each KEGG module [147]. These equations take KO annotations (see Section 4.8, Fig.4.9, and [100]) of an organism's proteins as input. KO annotations are equivalent to the step evidence used to support genome property steps.

Pygenprop could also implement a similar "completeness ratio" algorithm to that used by MAPLE. Much of the logic encoded in MAPLE's "boolean algebra-like equations" is already built into the Genome Properties database itself. For example, the record for each genome properties already records the InterPro domains, which represent multiple enzyme families, that can be used to support a single step. The steps required for individual properties are also codified. Calculation of "completeness ratios" for genome properties may be complicated by the fact that several genome properties rely on the presence of others (see Subsection 1.4.2), unlike KEGG modules, which are only supported by KO annotations.

Pygenprop could implement such a "completeness ratio" based assignment system by providing the library with a separate set of functions for calculating genome property assignments. The library could then be modified to calculate either discrete or continuous assignments for individual properties. These continuous assignments could be stored in Micromeda files or generated dynamically. Micromeda's interface could be modified to have a user interface switch that would allow users to switch their current heat map between displaying property assignments as either YES, NO, or PARTIAL or as a percentage of step completeness. Micromeda-Server would have to be modified to provide an endpoint (Section 3.4) for these serving completeness-based property assignments to Micromeda-Client.

5.3 Recommendations for Future Development

Researchers are scaling up their research projects from studying the capabilities of individual microorganisms to looking at the functional capabilities of entire microbial communities from a systems biology perspective. Due to its ability to programmatically compare thousands of pathway presence and absences from multiple organisms simultaneously, Pygenprop has the potential to form the basis of future bioinformatics tools that support the emerging community-scale research. Such tools could use patterns in the presence and absence of pathway steps found in organisms from a single environment to detect patterns of interspecies cross-feeding. Pygenprop's integration with the Python data science ecosystem could be leveraged to build classifiers that automatically identify organisms predicted to carry out specific ecosystem functions. When combined with culture condition information found in databases such as the Bacterial Diversity Metadatabase (BacDive) [130], Pygenprop could be used to develop classifiers that use the presence and absence of genome properties to predict an organism's favoured growth conditions. If such a tool was applied to multiple organisms from an environment, the data produced could be potentially used to build tools that predict how microbial communities will shift in response to changes in environmental conditions. It is recommended that future users of Pygenprop explore the possibility of building such tools.

5.4 Summary

Micromeda is a set of tools that allow users to make rapid comparisons of the pathways possessed by multiple organisms. Although there are many improvements to be made to the software, the current version of the tool is robust and provides users with new capabilities with regards to pathway analysis. The platform should be maintained and further expanded upon in terms of both user interface and overall features. Micromeda already provides users with a faster way to perform pathway analysis and future improved versions, with improved performance and capabilities, will help users even more. It is hoped that Micromeda will garner wide adoption by the scientific community.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Sahar Abubucker, Nicola Segata, Johannes Goll, Alyxandria M Schubert, Jacques Izard, Brandi L Cantarel, Beltran Rodriguez-Mueller, Jeremy Zucker, Mathangi Thiagarajan, Bernard Henrissat, et al. Metabolic reconstruction for metagenomic data and its application to the human microbiome. *Plos Computational Biology*, 8(6):e1002358, 2012.

[3] Alan Agresti. *Analysis of ordinal categorical data*, volume 656. John Wiley & Sons, 2010.

[4] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal Of Molecular Biology*, 215(3):403–410, 1990.

[5] Tiago Antao. *Bioinformatics with Python Cookbook: Learn how to use modern Python bioinformatics libraries and applications to do cutting-edge research in computational biology.* Packt Publishing Ltd, 2018.

[6] Rolf Apweiler, Terri K. Attwood, Amos Bairoch, Alex Bateman, Ewan Birney, Margaret Biswas, Philipp Bucher, Lorenzo Cerutti, Florence Corpet, Michael D. R. Croning, et al. Interpro—an integrated documentation resource for protein families, domains and functional sites. *Bioinformatics*, 16(12):1145–1150, 2000.

[7] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T

Eppig, et al. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25, 2000.

[8] Terri K Attwood, Michael DR Croning, Darren R Flower, AP Lewis, JE Mabey, Philip Scordis, JN Selley, and W Wright. Prints-s: the database formerly known as prints. *Nucleic Acids Research*, 28(1):225–227, 2000.

[9] Ramy K Aziz, Daniela Bartels, Aaron A Best, Matthew DeJongh, Terrence Disz, Robert A Edwards, Kevin Formsma, Svetlana Gerdes, Elizabeth M Glass, Michael Kubal, et al. The rast server: rapid annotations using subsystems technology. *Bmc Genomics*, 9(1):75, 2008.

[10] Alex Bateman, Lachlan Coin, Richard Durbin, Robert D Finn, Volker Hollich, Sam Griffiths-Jones, Ajay Khanna, Mhairi Marshall, Simon Moxon, Erik LL Sonnhammer, et al. The pfam protein families database. *Nucleic Acids Research*, 32(suppl_1):D138–D141, 2004.

[11] Michael Bayer. Sqlalchemy. *The Architecture Of Open Source Applications*, 2:291–314, 2014.

[12] David Beazley. Understanding the python gil. In *PyCON Python Conference. Atlanta, Georgia*, 2010.

[13] Catriel Beeri, Philip A Bernstein, and Nathan Goodman. A sophisticate's introduction to database normalization theory. In *Readings in Artificial Intelligence and Databases*, pages 468–479. Elsevier, 1989.

[14] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml™) version 1.1. *Yaml. Org, Tech. Rep*, page 23, 2005.

[15] Peter Bengtsson. Msgpack vs json (with gzip), Jul 24AD.

[16] Lee H Bergstrand, Josh D Neufeld, and Andrew C Doxey. Pygenprop: a python library for programmatic exploration and comparison of organism genome properties. *Bioinformatics*, 2019.

[17] Tim Berners-Lee. Rfc 1738: Uniform resource locators (url). *Ftp://Ftp. Internic. Net/Rfc/Rfc1738. Txt*, 1994.

[18] Tim Berners-Lee, Dimitri Dimitroyannis, A John Mallinckrodt, and Susan McKay. World wide web. *Computers In Physics*, 8(3):298–299, 1994.

[19] Tim Berners-Lee, Roy Fielding, Larry Masinter, et al. Uniform resource identifiers (uri): Generic syntax, 1998.

[20] Kai Blin, Simon Shaw, Katharina Steinke, Rasmus Villebro, Nadine Ziemert, Sang Yup Lee, Marnix H Medema, and Tilmann Weber. antismash 5.0: updates to the secondary metabolite genome mining pipeline. *Nucleic Acids Research*, 2019.

[21] Grady Booch. Object-oriented development. *Ieee Transactions On Software Engineering*, SE-12(2):211–221, 1986.

[22] Bert Bos and Eric Meyer. CSS3 introduction. W3C working draft, W3C, May 2001. http://www.w3.org/TR/2001/WD-css3-roadmap-20010523/.

[23] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE Transactions On Visualization And Computer Graphics*, 17(12):2301–2309, 2011.

[24] Robert M Bowers, Nikos C Kyrpides, Ramunas Stepanauskas, Miranda Harmon-Smith, Devin Doud, TBK Reddy, Frederik Schulz, Jessica Jarett, Adam R Rivers, Emiley A Eloe-Fadrosh, et al. Minimum information about a single amplified genome (misag) and a metagenome-assembled genome (mimag) of bacteria and archaea. *Nature Biotechnology*, 35(8):725, 2017.

[25] T Bray. Rfc 7159: The javascript object notation (json) data interchange format. *Internet Engineering Task Force (Ietf)*, 2014.

[26] Michael Burch, Natalia Konevtsova, Julian Heinrich, Markus Hoeferlin, and Daniel Weiskopf. Evaluation of traditional, orthogonal, and radial tree diagrams by an eye tracking study. *Ieee Transactions On Visualization And Computer Graphics*, 17(12):2440–2448, 2011.

[27] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 Internet Measurement Conference*, pages 531–537. ACM, 2015.

[28] K Canese. Pubmed celebrates its 10th anniversary. *Nlm Tech Bull*, 352(352), 2006.

[29] Minh Duc Cao, Son Hoang Nguyen, Devika Ganesamoorthy, Alysha G. Elliott, Matthew A. Cooper, and Lachlan J.M. Coin. Scaffolding and completing genome assemblies in real-time with nanopore sequencing. *Nature Communications*, 8:1–10, 2017.

[30] Jonathan Chaffer. *Learning jQuery.* Packt Publishing Ltd, 2013.

[31] Chun-houh Chen, Wolfgang Hrdle, Antony Unwin, Chun-houh Chen, Wolfgang Hrdle, and Antony Unwin. *Handbook of Data Visualization (Springer Handbooks of Computational Statistics).* Springer-Verlag TELOS, Santa Clara, CA, USA, 1 edition, 2008.

[32] Benoit Chesneau. Gunicorn is a python wsgi http server, Jul 2018.

[33] Chih-Hung Chou, Wen-Chi Chang, Chih-Min Chiu, Chih-Chang Huang, and Hsien-Da Huang. Fmm: a web server for metabolic pathway reconstruction and comparative analysis. *Nucleic Acids Research*, 37(suppl_2):W129–W134, 2009.

[34] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[35] UniProt Consortium. Uniprot: a hub for protein information. *Nucleic Acids Research*, 43(D1):D204–D212, 2014.

[36] localForage Contributors. localforage documentation.

[37] Charles E Cook, Mary Todd Bergman, Robert D Finn, Guy Cochrane, Ewan Birney, and Rolf Apweiler. The european bioinformatics institute in 2016: data growth and integration. *Nucleic Acids Research*, 44(D1):D20–D26, 2015.

[38] Peter Cooper. Messagepack: Efficient, cross language binary object serialization, Mar 2010.

[39] Zak Costello and Hector Garcia Martin. A machine learning approach to predict metabolic pathway dynamics from time-series multiomics data. *NPJ systems biology and applications*, 4(1):1–14, 2018.

[40] David Croft, Antonio Fabregat Mundo, Robin Haw, Marija Milacic, Joel Weiser, Guanming Wu, Michael Caudy, Phani Garapati, Marc Gillespie, Maulik R Kamdar, et al. The reactome pathway knowledgebase. *Nucleic Acids Research*, 42(D1):D472–D477, 2013.

[41] Miroslava Cuperlovic-Culf. Machine learning methods for analysis of metabolic data and metabolic pathway modeling. *Metabolites*, 8(1):4, 2018.

[42] Youssef Darzi, Yuta Yamate, and Takuji Yamada. Functree2: an interactive radial tree for functional hierarchies and omics data visualization. *Bioinformatics*, 2019.

[43] Chris J Date and Hugh Darwen. *A Guide to the SQL Standard*, volume 3. Addison-Wesley New York, 1987.

[44] Carlotta De Filippo, Matteo Ramazzotti, Paolo Fontana, and Duccio Cavalieri. Bioinformatic approaches for functional annotation and pathway inference in metagenomics data. *Briefings In Bioinformatics*, 13(6):696–710, 2012.

[45] Valeria De Fonzo, Filippo Aluffi-Pentini, and Valerio Parisi. Hidden markov models in bioinformatics. *Current Bioinformatics*, 2(1):49–61, 2007.

[46] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316, 2017.

[47] Paul DuBois and Michael Foreword By-Widenius. *MySQL*. New riders publishing, 1999.

[48] Sean R Eddy. Accelerated profile hmm searches. *Plos Computational Biology*, 7(10):e1002195, 2011.

[49] A Murat Eren, Özcan C Esen, Christopher Quince, Joseph H Vineis, Hilary G Morrison, Mitchell L Sogin, and Tom O Delmont. Anvi'o: an advanced analysis and visualization platform for ‘omics data. *Peerj*, 3:e1319, 2015.

[50] David A Farber, Richard E Greer, Andrew D Swart, and James A Balter. Internet content delivery network, November 25 2003. US Patent 6,654,807.

[51] Alan Fersht. *Structure and mechanism in protein science: a guide to enzyme catalysis and protein folding*. Macmillan, 1999.

[52] Roy Fielding. Representational state transfer. *Architectural Styles And The Design Of Netowork-Based Software Architecture*, pages 76–85, 2000.

[53] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1, 1999.

[54] Roy T. Fielding and Gail Kaiser. The apache http server project. *Ieee Internet Computing*, 1(4):88–90, 1997.

[55] Deborah L Finke and William E Snyder. Niche partitioning increases resource exploitation by diverse communities. *Science*, 321(5895):1488–1490, 2008.

[56] Robert D Finn, Teresa K Attwood, Patricia C Babbitt, Alex Bateman, Peer Bork, Alan J Bridge, Hsin-Yu Chang, Zsuzsanna Dosztányi, Sara El-Gebali, Matthew Fraser, et al. Interpro in 2017—beyond protein family and domain annotations. *Nucleic Acids Research*, 45(D1):D190–D199, 2016.

[57] David Flanagan. *JavaScript: the definitive guide*. O'Reilly Media, Inc., 2006.

[58] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47. ACM, 2011.

[59] Sadayuki Furuhashi. Messagepack specification, 2013.

[60] Michael Y Galperin, D Roland Walker, and Eugene V Koonin. Analogous enzymes: independent inventions in enzyme evolution. *Genome Research*, 8(8):779–790, 1998.

[61] Charles Gawad, Winston Koh, and Stephen R Quake. Single-cell genome sequencing: current state of the science. *Nature Reviews Genetics*, 17(3):175, 2016.

[62] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna, and Sean R Eddy. Rfam: an rna family database. *Nucleic Acids Research*, 31(1):439–441, 2003.

[63] Miguel Grinberg. *Flask web development: developing web applications with python*. O'Reilly Media, Inc., 2018.

[64] James Gwertzman and Margo I Seltzer. World wide web cache consistency. In *USENIX annual technical conference*, pages 141–152, 1996.

[65] Daniel H Haft, Brendan J Loftus, Delwood L Richardson, Fan Yang, Jonathan A Eisen, Ian T Paulsen, and Owen White. Tigrfams: a protein family resource for the functional identification of proteins. *Nucleic Acids Research*, 29(1):41–43, 2001.

[66] Daniel H. Haft, Jeremy D. Selengut, Roland A. Richter, Derek Harkins, Malay K. Basu, and Erin Beck. TIGRFAMs and Genome Properties in 2013. *Nucleic Acids Research*, 41(Database issue):D387–95, jan 2013.

[67] Jing Han, E Haihong, Guan Le, and Jian Du. Survey on nosql database. In *2011 6th international conference on pervasive computing and applications*, pages 363–366. IEEE, 2011.

[68] Ian Hickson. Web storage (second edition). W3C recommendation, W3C, April 2016. http://www.w3.org/TR/2016/REC-webstorage-20160419/.

[69] Adrian Holovaty and Jacob Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.

[70] Philippe Horvath and Rodolphe Barrangou. Crispr/cas, the immune system of bacteria and archaea. *Science*, 327(5962):167–170, 2010.

[71] Laura A. Hug, Brett J. Baker, Karthik Anantharaman, Christopher T. Brown, Alexander J. Probst, Cindy J. Castelle, Cristina N. Butterfield, Alex W. Hernsdorf, Yuki Amano, Kotaro Ise, Yohey Suzuki, Natasha Dudek, David A. Relman, Kari M. Finstad, Ronald Amundson, Brian C. Thomas, and Jillian F. Banfield. A new view of the tree of life. *Nature Microbiology*, 1(5):1–6, 2016.

[72] Sarah Hunter, Rolf Apweiler, Teresa K Attwood, Amos Bairoch, Alex Bateman, David Binns, Peer Bork, Ujjwal Das, Louise Daugherty, Lauranne Duquenne, et al. Interpro: the integrative protein signature database. *Nucleic Acids Research*, 37(1):D211–D215, 2008.

[73] Sarah Hunter, Rolf Apweiler, Teresa K. Attwood, Amos Bairoch, Alex Bateman, David Binns, Peer Bork, Ujjwal Das, Louise Daugherty, Lauranne Duquenne, Robert D. Finn, Julian Gough, Daniel Haft, Nicolas Hulo, Daniel Kahn, Elizabeth Kelly, Aurélie Laugraud, Ivica Letunic, David Lonsdale, Rodrigo Lopez, Martin Madera, John Maslen, Craig Mcanulla, Jennifer McDowall, Jaina Mistry, Alex Mitchell, Nicola Mulder, Darren Natale, Christine Orengo, Antony F. Quinn, Jeremy D. Selengut, Christian J.A. Sigrist, Manjula Thimma, Paul D. Thomas, Franck Valentin, Derek Wilson, Cathy H. Wu, and Corin Yeats. InterPro: The integrative protein signature database. *Nucleic Acids Research*, 37(SUPPL. 1):211–215, 2009.

[74] Doug Hyatt, Gwo-Liang Chen, Philip F LoCascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *Bmc Bioinformatics*, 11(1):119, 2010.

[75] SRI International. Pythoncyc: A python interface for pathway tools.

[76] Miten Jain, Hugh E Olsen, Benedict Paten, and Mark Akeson. The oxford nanopore minion: delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17(1):239, 2016.

[77] Jesse James Garrett. Ajax: A new approach to web applications, 02 2005.

[78] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed ¡today¿].

[79] Philip Jones, David Binns, Hsin-Yu Chang, Matthew Fraser, Weizhong Li, Craig McAnulla, Hamish McWilliam, John Maslen, Alex Mitchell, Gift Nuka, et al. Interproscan 5: genome-scale protein function classification. *Bioinformatics*, 30(9):1236–1240, 2014.

[80] Minoru Kanehisa and Susumu Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.

[81] Minoru Kanehisa, Susumu Goto, Yoko Sato, Miho Furumichi, and Mao Tanabe. Kegg for integration and interpretation of large-scale molecular data sets. *Nucleic Acids Research*, 40(D1):D109–D114, 2011.

[82] Peter D Karp, Christos A Ouzounis, Caroline Moore-Kochlacs, Leon Goldovsky, Pallavi Kaipa, Dag Ahrén, Sophia Tsoka, Nikos Darzentas, Victor Kunin, and Núria López-Bigas. Expansion of the biocyc collection of pathway/genome databases to 160 genomes. *Nucleic Acids Research*, 33(19):6083–6089, 2005.

[83] Peter D Karp, Suzanne Paley, and Tomer Altman. Data mining in the metacyc family of pathway databases. In *Data mining for systems biology*, pages 183–200. Springer, 2013.

[84] Peter D Karp, Suzanne Paley, and Pedro Romero. The pathway tools software. *Bioinformatics*, 18(suppl_1):S225–S232, 2002.

[85] Peter D Karp, Monica Riley, Suzanne M Paley, and Alida Pellegrini-Toole. The metacyc database. *Nucleic Acids Research*, 30(1):59–61, 2002.

[86] Shuichi Kawashima, Toshiaki Katayama, Yoko Sato, and Minoru Kanehisa. Kegg api: A web service using soap/wsdl to access the kegg system. *Genome Informatics*, 14:673–674, 2003.

[87] Markus A Keller, Gabriel Piedrafita, and Markus Ralser. The widespread role of non-enzymatic reactions in cellular metabolism. *Current Opinion In Biotechnology*, 34:153–161, 2015.

[88] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[89] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[90] Bruno Krebs. Beating json performance with protobuf, Jan 2017.

[91] Martin Krzywinski, Jacqueline Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J Jones, and Marco A Marra. Circos: an information aesthetic for comparative genomics. *Genome Research*, 19(9):1639–1645, 2009.

[92] George Lawton. Developing software online with platform-as-a-service technology. *Computer*, 41(6):13–15, 2008.

[93] P Leach, Michael Mealling, and Rich Salz. Rfc 4122: A universally unique identifier (uuid) urn namespace. *Proposed Standard, July*, 2005.

[94] Travis Leithead, Gary Kacmarcik, and Doug Schepers. UI events. W3C working draft, W3C, November 2018. https://www.w3.org/TR/2018/WD-uievents-20181108/.

[95] Jingjing Li and Chunlin Peng. jquery-based ajax general interactive architecture. In *2012 IEEE International Conference on Computer Science and Automation Engineering*, pages 304–306. IEEE, 2012.

[96] Tania Lima, Andrea H Auchincloss, Elisabeth Coudert, Guillaume Keller, Karine Michoud, Catherine Rivoire, Virginie Bulliard, Edouard De Castro, Corinne Lachaize, Delphine Baratin, et al. Hamap: a database of completely sequenced microbial proteome sets and manually curated microbial protein families in uniprotkb/swiss-prot. *Nucleic Acids Research*, 37(suppl_1):D471–D478, 2008.

[97] D4 Software Ltd. A brief history of json, Jun 2017.

[98] Hengyun Lu, Francesca Giordano, and Zemin Ning. Oxford Nanopore MinION Sequencing and Genome Assembly. *Genomics, Proteomics And Bioinformatics*, 14(5):265–279, 2016.

[99] Eve Maler, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau, and Tim Bray. Extensible markup language (XML) 1.0 (third edition). first edition of a recommendation, W3C, feb 2004. http://www.w3.org/TR/2004/REC-xml-20040204.

[100] Xizeng Mao, Tao Cai, John G Olyarchuk, and Liping Wei. Automated genome annotation and pathway identification using the kegg orthology (ko) as a controlled vocabulary. *Bioinformatics*, 21(19):3787–3793, 2005.

[101] Aron Marchler-Bauer, Myra K Derbyshire, Noreen R Gonzales, Shennan Lu, Farideh Chitsaz, Lewis Y Geer, Renata C Geer, Jane He, Marc Gwadz, David I Hurwitz, et al. Cdd: Ncbi's conserved domain database. *Nucleic Acids Research*, 43(D1):D222–D226, 2014.

[102] Eddie Martin. *Computer jargon dictionary and thesaurus*. Beecroft Publishing, 2006.

[103] Andy McCurdy. Redis, Aug 2019.

[104] Scott McGinnis and Thomas L Madden. Blast: at the core of a powerful and diverse set of sequence analysis tools. *Nucleic Acids Research*, 32(suppl_2):W20–W25, 2004.

[105] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[106] Matias Meno. dropzone.js.

[107] Huaiyu Mi, Betty Lazareva-Ulitsky, Rozina Loo, Anish Kejariwal, Jody Vandergriff, Steven Rabkin, Nan Guo, Anushya Muruganujan, Olivier Doremieux, Michael J Campbell, et al. The panther database of protein families, subfamilies, functions and pathways. *Nucleic Acids Research*, 33(suppl_1):D284–D288, 2005.

[108] Gerhard Michal and Dietmar Schomburg. *Biochemical pathways: an atlas of biochemistry and molecular biology*. John Wiley & Sons, 2012.

[109] Bruce Momjian. *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001.

[110] Yuki Moriya, Masumi Itoh, Shujiro Okuda, Akiyasu C Yoshizawa, and Minoru Kanehisa. Kaas: an automatic genome annotation and pathway reconstruction server. *Nucleic Acids Research*, 35(suppl_2):W182–W185, 2007.

[111] Yuki Moriya, Daichi Shigemizu, Masahiro Hattori, Toshiaki Tokimatsu, Masaaki Kotera, Susumu Goto, and Minoru Kanehisa. Pathpred: an enzyme-catalyzed metabolic pathway prediction server. *Nucleic Acids Research*, 38(suppl_2):W138–W143, 2010.

[112] Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists.* " O'Reilly Media, Inc.", 2016.

[113] T. Munzner. *Visualization Analysis and Design.* AK Peters Visualization Series. CRC Press, 2015.

[114] Sujay Muramalla, Ragaad Al Tarawneh, Shah Rukh Humayoun, Ricarda Moses, Sven Panis, and Achim Ebert. Radial vs. rectangular: Evaluating visualization layout impact on user task performance of hierarchical data. *Iadis International Journal On Computer Science & Information Systems*, 12(2), 2017.

[115] Allen Newell, J Cliff Shaw, and HA Simon. *Programming the logic theory machine.* Rand Corporation, 1957.

[116] Brian D Ondov, Nicholas H Bergman, and Adam M Phillippy. Interactive metagenomic visualization in a web browser. *Bmc Bioinformatics*, 12(1):385, 2011.

[117] Ross Overbeek, Tadhg Begley, Ralph M Butler, Jomuna V Choudhuri, Han-Yu Chuang, Matthew Cohoon, Valérie de Crécy-Lagard, Naryttza Diaz, Terry Disz, Robert Edwards, et al. The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes. *Nucleic Acids Research*, 33(17):5691–5702, 2005.

[118] Mike Owens. *The definitive guide to SQLite.* Apress, 2006.

[119] Donovan H. Parks, Christian Rinke, Maria Chuvochina, Pierre Alain Chaumeil, Ben J. Woodcroft, Paul N. Evans, Philip Hugenholtz, and Gene W. Tyson. Author Correction: Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life. *Nature Microbiology*, 903:1, 2017.

[120] Adam Paszke, Gregory Chanan, Zeming Lin, Sam Gross, Edward Yang, Luca Antiga, and Zachary Devito. Automatic differentiation in pytorch. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017) Workshop on Automatic Differentiation.* Long Beach, CA, 2017.

[121] William R Pearson. [5] rapid and sensitive sequence comparison with fastp and fasta. *Methods In Enzymology*, 1990.

[122] Torben Bach Pedersen and Christian S Jensen. Multidimensional data modeling for complex data. In *Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337)*, pages 336–345. IEEE, 1999.

[123] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal Of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[124] Luca Pireddu, Duane Szafron, Paul Lu, and Russell Greiner. The path-a metabolic pathway prediction web server. *Nucleic Acids Research*, 34(suppl_2):W714–W719, 2006.

[125] Joram M Posma, Steven L Robinette, Elaine Holmes, and Jeremy K Nicholson. Metabonetworks, an interactive matlab-based toolbox for creating, customizing and exploring sub-networks from kegg. *Bioinformatics*, 30(6):893–895, 2013.

[126] Morgan N Price, Grant M Zane, Jennifer V Kuehl, Ryan A Melnyk, Judy D Wall, Adam M Deutschbauer, and Adam P Arkin. Filling gaps in bacterial amino acid biosynthesis pathways with high-throughput genetics. *Plos Genetics*, 14(1):e1007147, 2018.

[127] Christopher Quince, Alan W Walker, Jared T Simpson, Nicholas J Loman, and Nicola Segata. Shotgun metagenomics, from sampling to analysis. *Nature Biotechnology*, 35(9):833, 2017.

[128] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019.

[129] Will Reese. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, 2008(173):2, 2008.

[130] Lorenz Christian Reimer, Anna Vetcininova, Joaquim Sardà Carbasse, Carola Söhngen, Dorothea Gleim, Christian Ebeling, and Jörg Overmann. Bac dive in 2019: bacterial phenotypic data for high-throughput biodiversity analysis. *Nucleic Acids Research*, 47(D1):D631–D636, 2018.

[131] Siyuan Ren, Guang Yang, Youyu He, Yiguo Wang, Yixue Li, and Zhengjun Chen. The conservation pattern of short linear motifs is highly correlated with the function of interacting protein domains. *Bmc Genomics*, 9(1):452, 2008.

[132] Lorna J Richardson, Neil D Rawlings, Gustavo A Salazar, Alexandre Almeida, David R Haft, Gregory Ducq, Granger G Sutton, and Robert D Finn. Genome properties in 2019: a new companion database to interpro for the inference of complete functional attributes. *Nucleic Acids Research*, 47(D1):D564–D572, 2018.

[133] Jerome Howard Saltzer. *Traffic control in a multiplexed computer system.* PhD thesis, Massachusetts Institute of Technology, 1966.

[134] Hanan Samet. *Applications of spatial data structures.* Citeseer, 1990.

[135] Ailen M Sánchez, George N Bennett, and Ka-Yiu San. Novel pathway engineering design of the anaerobic central metabolic pathway in escherichia coli to increase succinate yield and productivity. *Metabolic Engineering*, 7(3):229–239, 2005.

[136] scikit-bio development team. scikit-bio, 2014. Python package version 0.5.5.

[137] Torsten Seemann. Prokka: rapid prokaryotic genome annotation. *Bioinformatics*, 30(14):2068–2069, 2014.

[138] Y. Shafranovich. Common format and mime type for comma-separated values (csv) files. RFC 4180, RFC Editor, October 2005. `http://www.rfc-editor.org/rfc/rfc4180.txt`.

[139] Ben Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *Acm Trans. Graph.*, 11(1):92–99, January 1992.

[140] Alan Snyder. Encapsulation and inheritance in object-oriented programming languages. In *ACM Sigplan Notices*, volume 21, pages 38–45. ACM, 1986.

[141] Jake Spurlock. *Bootstrap: responsive web development.* " O'Reilly Media, Inc.", 2013.

[142] Jörg Stelling, Steffen Klamt, Katja Bettenbrock, Stefan Schuster, and Ernst Dieter Gilles. Metabolic network structure determines key aspects of functionality and regulation. *Nature*, 420(6912):190, 2002.

[143] Paul Stothard and David S Wishart. Circular genome visualization and exploration using cgview. *Bioinformatics*, 21(4):537–539, 2004.

[144] William R Strohl. Biochemical engineering of natural product biosynthesis pathways. *Metabolic Engineering*, 3(1):4–14, 2001.

[145] Liba Svobodova. Client/server model of distributed processing. In *Kommunikation in Verteilten Systemen I*, pages 485–498. Springer, 1985.

[146] Hideto Takami, Takeaki Taniguchi, Wataru Arai, Kazuhiro Takemoto, Yuki Moriya, and Susumu Goto. An automated system for evaluation of the potential functionome: Maple version 2.1. 0. *Dna Research*, 23(5):467–475, 2016.

[147] Hideto Takami, Takeaki Taniguchi, Yuki Moriya, Tomomi Kuwahara, Minoru Kanehisa, and Susumu Goto. Evaluation method for the potential functionome harbored in the genome and metagenome. *Bmc Genomics*, 13(1):699, 2012.

[148] Olga Khersonsky Tawfik and Dan S. Enzyme promiscuity: a mechanistic and evolutionary perspective. *Annual Review Of Biochemistry*, 79:471–505, 2010.

[149] Python Team. Uuid objects according to rfc 4122, Jul 2019.

[150] D Tenenbaum. Keggrest: Client-side rest access to kegg. r package version 1.24. 0. 2019, 2019.

[151] Ines Thiele and Bernhard Ø Palsson. A protocol for generating a high-quality genome-scale metabolic reconstruction. *Nature Protocols*, 5(1):93, 2010.

[152] Edward R Tufte. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 2001.

[153] unbit. The uwsgi project, Feb 2019.

[154] David Vallenet, Alexandra Calteau, Stéphane Cruveiller, Mathieu Gachet, Aurélie Lajus, Adrien Josso, Jonathan Mercier, Alexandre Renaux, Johan Rollin, Zoe Rouy, et al. Microscope in 2017: an expanding and evolving integrated resource for community expertise of microbial genomes. *Nucleic Acids Research*, 45(D1):D517–D528, 2016.

[155] David Vallenet, Stefan Engelen, Damien Mornico, Stéphane Cruveiller, Ludovic Fleury, Aurélie Lajus, Zoé Rouy, David Roche, Gregory Salvignol, Claude Scarpelli, et al. Microscope: a platform for microbial genome annotation and comparative genomics. *Database*, 2009, 2009.

[156] Guido Van Rossum and Fred L Drake Jr. *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[157] Kenton Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available At Least As Early As Jul*, 72, 2008.

[158] Jinesh Varia, Sajee Mathew, et al. Overview of amazon web services. *Amazon Web Services*, pages 1–22, 2014.

[159] Andreas Wagner. Metabolic networks and their evolution. In *Evolutionary systems biology*, pages 29–52. Springer, 2012.

[160] Larry Wall et al. The perl programming language, 1994.

[161] Matthew O Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1(3-4):194–210, 2002.

[162] Alexander Weissman. Select2 documentation.

[163] Leland Wilkinson and Michael Friendly. The history of the cluster heat map. *The American Statistician*, 63(2):179–184, 2009.

[164] Jianmin Wu, Xizeng Mao, Tao Cai, Jingchu Luo, and Liping Wei. Kobas server: a web-based platform for automated annotation and pathway identification. *Nucleic Acids Research*, 34(suppl_2):W720–W724, 2006.

[165] Guangchuang Yu, Li-Gen Wang, Yanyan Han, and Qing-Yu He. clusterprofiler: an r package for comparing biological themes among gene clusters. *Omics: A Journal Of Integrative Biology*, 16(5):284–287, 2012.

[166] Jianzhi Zhang. Evolution by gene duplication: an update. *Trends In Ecology & Evolution*, 18(6):292–298, 2003.

[167] Jitao David Zhang and Stefan Wiemann. Kegggraph: a graph approach to kegg pathway in r and bioconductor. *Bioinformatics*, 25(11):1470–1471, 2009.