A network approach to the specification, enhancement and representation of synthetic biology designs

by

Matthew Crowther

Submitted for the degree of Doctor of Philosophy in the School of Computing, Newcastle University

Supervised by Dr Ángel Goñi Moreno and Professor Anil Wipat

January 2024

Declaration

I declare that this thesis is my work unless otherwise stated. No part of this thesis has previously been submitted for a degree or other qualification at Newcastle University or any other institution.
Matthew Crowther

Acknowledgements

I am immensely grateful to Professor Anil Wipat and Dr Angel Goñi-Moreno for their guidance and invaluable support throughout my research. Their insights and encouragement have been fundamental to my work. Special thanks to my peers for their camaraderie and my family for their support.

Publications arising from this thesis

- Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "A Network Approach to Genetic Circuit Designs". *ACS Synthetic Biology* 11:9, 2022. PMID: 36044984, pp. 3058–3066. DOI: 10.1021/acssynbio.2c00255. eprint: https://doi.org/10.1021/acssynbio.2c00255. URL: https://doi.org/10.1021/acssynbio.2c00255

- Matthew Crowther et al. "ShortBOL: A Language for Scripting Designs for Engineered Biological Systems Using Synthetic Biology Open Language (SBOL)". *ACS Synthetic Biology* 9:4, 2020. PMID: 32129980, pp. 962–966. DOI: 10.1021/acssynbio.9b00470. eprint: https://doi.org/10.1021/acssynbio.9b00470. URL: https://doi.org/10.1021/acssynbio.9b00470

- Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "GENETTA: a Network-Based Tool for the Analysis of Complex Genetic Designs". *ACS Synthetic Biology*, 2023

- Hasan Baig et al. "Synthetic biology open language (SBOL) version 3.0.0". *Journal of Integrative Bioinformatics* 17:2-3, 2020, p. 20200017. DOI: doi:10.1515/jib-2020-0017. URL: https://doi.org/10.1515/jib-2020-0017

- Esteban Martínez-García et al. "SEVA 4.0: an update of the Standard European Vector Architecture database for advanced analysis and programming of bacterial phenotypes". *Nucleic Acids Research* 51:D1, 2022, pp. D1558–D1567. ISSN: 0305-1048. DOI: 10.1093/nar/gkac1059. eprint: https://academic.oup.com/nar/article-pdf/51/D1/D1558/48441442/gkac1059.pdf. URL: https://doi.org/10.1093/nar/gkac1059

Abstract

Synthetic biology incorporates many existing biological fields of study with an engineering approach to constructing new or redesigning physical parts into devices and systems, focusing on assembling sets of standardised genetic components. A principle of the field is to structure biological systems into a hierarchy of abstract entities in a standardised fashion. However, in reality, ad hoc procedures, colloquial languages, and informal data are still ubiquitous in the field. These practices are acceptable when working with relatively simple systems, but as the field progresses and complexity increases, they quickly become impractical, limiting the scale and complexity of design. Efforts to implement standards in the field have thus far been limited. These efforts have been limited in overcoming the complexities introduced by standards and the corresponding disruption to existing working practices, as they require a knowledge of often complex data structures. Synthetic biologists with limited knowledge of data representations may struggle to use these standards without support. This research focuses on four areas that enable robust genetic designs to be defined, captured, enhanced and explored using standard data structures without exposing data complexity to practitioners of synthetic biology. Firstly, research was carried out to enable the specification of genetic designs using a common language for use within synthetic biology. The outcome was ShortBOL, an extensible language backed by a standard which mapped the fabricated language produced by standard formats to a more naturally understood vocabulary. Secondly, approaches to improve the accessibility of design data were explored. Existing design data was integrated and enhanced to form a weighted knowledge graph (WKG) enriched with dynamic metadata. The metadata was designed to solve issues of uncertainty commonly encountered within existing databases. The WKG is designed to learn and evolve through interactions with human users and interfacing tools, fostering a dynamic exchange of insights. After establishing the weighted knowledge graph and processes to maintain rich and structured underlying data, two examples were used to demonstrate the utility of the WKG from both human and computational perspectives. The first use case illustrates the advantages a user can gain, enhancing the query system by offering tailored results, reducing the time needed to identify the desired entity by ranking results and enabling feedback to improve future results. The second use case demonstrated how the weighted knowledge graph could be harnessed computationally to improve existing designs by automatically integrating existing data to reduce the manual burden of retroactively upgrading design data. In this context, we explored introducing functional information into existing designs, addressing the typical absence of such data. The final section researched and developed a graph-based methodology for representing and visualising circuit design information. The approach transformed the design into dynamic network structures, which could be automatically modified on demand according to user specifications. A significant focus was on scale abstraction, providing an automatic sliding level of detail that further tailors the visualisations to a given situation. In the ever-evolving field of synthetic biology, this research aims to pave the way for a future where standardised language, enriched knowledge graphs, and dynamic

visualisations empower scientists to unlock the full potential of genetic design, bridging the gap between complexity and practicality.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

1 Background

Synthetic Biology (SynBio) seeks to engineer either fully synthetic biological material or redesign systems already found in nature[6]. Many principles adopted by SynBio are taken from other engineering fields, namely, software engineering and computing science, by applying techniques to build up levels of abstraction, such as parts, devices, and systems, using standardised and interchangeable building blocks[7]. This idea of programming living systems is a fundamental goal of SynBio[8], and it involves applying the principles of information processing, control, and design commonly used in computer programming to the field of biology. However, design data is frequently captured within adhoc flat text formats, which cannot capture the information required to realise these principles truly. Data standard formats have been developed specifically for SynBio, for example, the Synthetic Biology Open Language (SBOL)[4], which aims to address these issues. These formats are designed to replace older formats and specifically target the requirements within synthetic biology, namely the ability to interface computationally with the data and the capacity to define abstract modules. Data standards have been limited in widespread adoption because of the inherent complexity introduced, burdening practitioners, the departure from established working procedures and a lack of clarity on the benefits of implementing standards. Instead of developing a genetic design at the sequence level with handwritten annotations, the intent of the design must be encoded with precise labels specified within the standard[9]. While some efforts have been made to reduce these barriers via specialised tooling, they may be too complex, may not consider the community's wide range of knowledge levels and may not provide sufficient motivation for such a considerable change to working practises.

2 Aims and Objectives

Research Question : This thesis investigates how standard data structures can simplify the specification, integration, enhancement, and representation of design data in synthetic biology, ultimately reducing the burden on users. Aim: The main goal is to enhance the usability of synthetic biology data by employing standard data structures that minimise the need for users to interact directly with complex underlying systems.

Objectives : First, the thesis focuses on developing abstract methods for specifying design data. This aims to simplify the complexities typically associated with such tasks, thus encouraging wider adoption of standards like SBOL. Secondly, it seeks to create a centralised dynamic data system that integrates and enhances existing design data. This system is designed to evolve and adapt, reducing uncertainties within the existing knowledge base and making data management more straightforward. The research also explores implementing user-centric interaction models.

Using approaches such as the WKG querying technique, the system provides tailored search results, handles abstract queries, ranks resources based on quality, and incorporates user feedback to refine future outputs. Another objective is to facilitate automatic data integration. The study investigates methods that allow the knowledge system to automatically integrate and update existing design data, thereby reducing the manual effort required and enhancing the efficiency of data workflows. Lastly, the thesis examines the development of multiple representational methods. This involves creating various ways to view data derived from a single design to address the needs of different users, tackling the challenge posed by the multi-dimensional nature of synthetic biology data.

Research Principles : The thesis adheres to three main principles. It treats all data as networks to facilitate complex analysis and easy modifications. It extends these networks into knowledge graphs to enhance data consistency and tractability. Finally, it emphasises automation and minimal human input to avoid comprehension issues and reduce the need for direct interaction with data structures.

3 Contribution

3.1 Research

The first research outcome explored how the natural language used within synthetic biology and the language introduced by standards can be mapped together to provide user-facing language interfaces. Next, the weighted knowledge graph (WKG) enabled the ability to normalise, canonicalise, and enhance existing synthetic biology databases. Once the WKG was established, two use cases were examined from a human and computational perspective. From a human view, to provide better access to data via an improved query system exploiting the features and metadata encoded within the WKG. From a computational approach to remove requirements imposed on a user by standards by automatically introducing extra information into an existing design which is not commonly described. The final research output explored how standards can increase perceived complexity and provide insight by using network projections to create new graphs containing data relevant to a specific design aspect, such as graphs representing interactions between biological or chemical entities.

3.2 Tools

During this research, two tools were developed. ShortBOL is a tool developed parallel with abstracting language research (Chapter 3). The tool contains a new language which abstracts much of the unfamiliar language introduced with standards. Furthermore, it provides an easily extensible template system that increasingly allows abstract representations to be captured given the user's requirements. Genetta is a tool that combines all methods developed during network-centric research (Chapters 4,5, and 6), namely enhancing, validating, and representing genetic designs within networks. The tool includes methods for user-customisable network visualisation, evaluation, canonicalisation, enhancement and validation of genetic designs.

4 Structure

The remainder of this thesis is divided into seven chapters. Chapter **two** is the background, which expands on the motivation, including the literature review and approaches to tackle the identified research areas. Chapter **three** explores efforts to simplify and democratise language, specifically new language not typical within synthetic biology introduced when data standards are implemented. Chapter **four** explores how existing databases can be automatically canonicalised, that is, the ability to create virtual analogues and introduce provenance into design data. From this, the centralised database is used to explore how it can be expanded to include new information which is not explicitly defined. Chapter **Five** covers how a user's access to data can be improved and how existing genetic designs can be enhanced using the centralised weighted knowledge graph established previously. Chapter **six** combines the previous efforts of specification, canonicalisation, and enhancement to represent genetic designs dynamically using network visualisation so that information can be tailored to a broad range of people. Finally, in chapter **seven**, the conclusion summarises the research presented in the thesis and describes possible ideas for future work.

Chapter 2: Background

1 Introduction

This section comprises a review and explanation of five topics that underpin the research. First, an overview of synthetic biology is presented, encompassing its core principles, an analysis of deviations from these ideals in practice, and an exploration of the underlying reasons. Furthermore, the functioning of the design, build, test, and learn (DBTL) cycle in synthetic biology is explained, along with the discussions surrounding automation within the DBTL framework and the necessary conditions for successfully implementing this iterative process. Secondly, this introduction covers the representation of synthetic biology information, particularly design data, as a network of interconnected data points and explores the potential for data analysis. The background of networks encompasses the core principles of network science and numerous established methods utilised throughout multiple research chapters. Next, this section explores the role of knowledge graphs in computer science, their application to data in synthetic biology, and the benefits of knowledge graphs compared to more loose data structures. Additionally, it explains ontologies, a specific type of knowledge graph that can impose a structured framework and semantic guidelines on individual data sources. Finally, this section examines the general concept of standards and their successful application across various fields to accelerate progress. The Synthetic Biology Open Language (SBOL)[10] standard is reviewed as the primary standard used in this research. This review discusses the benefits of utilising standards within synthetic biology and highlights the challenges introduced during adoption. This section reviews existing data sources that capture structured and unstructured design data, shedding light on fundamental issues regarding standardised design data.

2 Synthetic Biology

2.1 Core Principles

Synthetic biology applies engineering principles to biology, emphasising forward engineering[11] to design or redesign biological entities as systems that either do not exist in nature or perform functions not naturally carried out by biological systems[12]. Synthetic biology draws upon principles from engineering disciplines, including computer science, and practical knowledge from traditional biological fields like systems biology and molecular biology. Three core principles underpin this field[13]. Modularisation concerns constructing a system from interchangeable building blocks, allowing for the substitution of biological components or constructs to ensure compatibility with the host or to alter local functionality within synthetic biology. Abstraction simplifies or conceals complexity, presenting only relevant information to reduce the perception of intricacy. In synthetic biology, abstraction is achieved by encapsulating sequence data into biolog-

ical parts, which, in turn, are encapsulated into devices and can increase arbitrary levels of detail. Standardisation entails establishing a shared set of operational principles, terminology, processes, structures, or protocols to facilitate interoperability among different entities. An example of standards in synthetic biology is the SEVA format[5], which provides a framework for assembling components inside plasmid vectors, allowing for the exchange of genetic designs across various host systems. When applied in the context of synthetic biology, these engineering principles promote the reusability of biological components and enhance the predictability and robustness of design operations.

2.2 Genetic Circuits

The design and implementation of genetic circuits [14, 15, 16] that allow cells to perform predefined functions lie at the core of synthetic biology[17, 18]. Furthermore, the engineering of genetic circuits involves various genetic parts that play crucial roles in ensuring the functionality and reliability of these systems. Key components include:

- **Promoters and Operators:** These DNA sequences regulate the transcription of genes. The choice of promoter determines the conditions under which a gene is activated or repressed, which is fundamental in creating circuits that respond to environmental stimuli or internal cellular states.
- **Ribosome Binding Sites (RBS):** These sequences influence the translation initiation rate of mRNA into protein, which is crucial for adjusting the protein output levels of genetic circuits.
- **Coding Sequences:** These are the regions of DNA that are transcribed and translated into protein, serving as the functional machinery or reporters in genetic circuits.
- **Terminators:** These sequences signal the end of transcription, ensuring proper demarcation of genetic messages and preventing run-on transcriptions, which could disrupt circuit functionality.
- **Insulators and Scaffold Matrices:** These elements help mitigate the effects of surrounding genetic elements, reducing 'crosstalk' between adjacent modules and enhancing circuit predictability and stability.

Genetic circuits are an instance of applying engineering principles such as modularisation and abstraction to biology. Biological parts are assembled inside the cell to perform pre-defined functions. An example is the engineering of increasingly complex Boolean logic circuits[19] that use cascades of transcriptional regulators. Other circuits, such as switches[20], counterss[21] and memories[22], are routinely engineered using transcriptional and post-transcriptional processes[23]. Different host organisms such as bacteria[24], yeasts[25] and mammalian[26] cells are used to test circuits in several applications[27], ranging from pollution control[28] to medical diagnosis[29]. Furthermore, the functionalities of genetic circuits will only improve as scientists control the information processing abilities of biological systems: signal noise[30, 31] metabolic dynamics[32, 33], context-circuit interplay[34, 35], stability[36] and more[37].

2.3 Design data

Design data in synthetic biology refers to the information and specifications crucial for designing and engineering biological systems for a particular purpose. Design data within synthetic biology will contain a consortium of data not captured within traditional biology[38]. Here are some critical aspects of design data in synthetic biology:

- **Structural Information:** Fundamentally, design data includes genetic sequences that encode the functions or characteristics of interest. However, the sequence data can be abstracted into the component genetic parts, such as promoters, terminators, and other genetic elements, which can then be further abstracted into more comprehensible modules[39]. Furthermore, the parts may reference a catalogue of standardised biological parts.

- **Functional Information:** Synthetic biologists design genetic circuits, which are combinations of genes and regulatory elements that perform specific functions within a biological system. Design data for genetic circuits includes information on the arrangement and regulation of these components[9].

- **Computational Models:** Computational models can predict the behaviour of biological systems based on design data[40].

- **DNA Synthesis and Assembly Instructions:** Design data provides instructions for synthesising and assembling DNA sequences[41].

- **Characterisation Data:** Designs are often context-dependent, and information on how specific biological parts or genetic circuits behave under different conditions is essential[42].

Overall, design data in synthetic biology serves as the blueprint for creating and engineering biological systems. Design data in synthetic biology differs significantly from capturing natural biological systems in one significant way. Design data in synthetic biology focuses on the intentional design, construction, and manipulation of biological systems for practical purposes while capturing natural biological systems, which involves studying and documenting existing ecosystems and organisms in their natural state without significant alterations. Although natural and synthetic biology complement one another, the type of information captured differs. While attempting to reduce complexity, the genetic circuit approach exposes some unique challenges. As circuits increase in complexity, unintended interactions between genetic elements can occur, leading to unpredictable behaviours. This complexity management is a significant hurdle in synthetic biology. Also, different host organisms might respond variably to the same genetic circuit due to differences in their cellular machinery, impacting the performance consistency across different biological contexts. Genetic circuits can be prone to mutations or epigenetic modifications, potentially leading to circuit degradation or failure, especially in long-term applications. Finally, scaling up from small test circuits to larger, more complex systems often introduces new challenges, including maintaining the functionality of each component and integrating multiple circuits without interference.

Intent and Reality

While modularisation and abstraction are at the core of driving aims of synthetic biology more often, the reality of applied synthetic biology is the opposite. Synthetic biology processes are more commonly still considered at the primary structure level with no abstraction. Furthermore, data is captured and stored in ad hoc and free-form formats, and processes can be specific to an individual lab. For example, all descriptive data is stored as unconstrained written text in a Genbank file. For example, experimental, functional, conformational, epigenetic, contextual, quantitive or metadata can be captured formally within these files. Furthermore, a hierarchy of information, such as abstract modules or even higher-order genetic information, such as cell-cell information, cannot be encoded within a Genbank file. This type of information cannot be explicitly encoded within these formats because they do not contain the infrastructure to define this type of information, as they were initially described to capture natural sequence data, which will not be changed over time. Furthermore, Genbank annotations are largely computationally intractable because the context regarding what the annotations mean is written within free, meaning consistent computational interpretation is impossible. While this works for traditional biology, where the focus is on the sequence data, this goes directly against the principles of modularity and abstraction of synthetic biology. In conclusion, while the intent is to express biological systems as genetic circuits where modules can be swapped, the reality is that this is seldom achieved, even in theory, when capturing the intent of a design.

2.4 Design, Build, Test, Learn (DBTL)

Design, Build, Test and Learn (DBTL)[43] is a circular workflow where each iteration aims to learn from the previous and improve the biological system by the desired metric. DBTL is often discussed with automation because the iterative nature combined with the ever-increasing complexity of biological systems makes manual processes more unfeasible. For example, a biologist could often manually carry out hundreds of pipetting steps during the build phase. Combined with this, the cyclical nature of DBTL, where certain functions may be performed multiple times over multiple iterations, is prime for automation, which excels at parallel or stable operations[44]. DBTL and automation will undoubtedly be crucial components for advancing synthetic biology. However, like DBTL and automation, they are closely related, and automation and standards are also commonly coupled. Because automation approaches do not benefit from a human understanding of abstract concepts, the data and processes must be formalised, i.e. a computer must understand information and instructions unambiguously. Therefore, another reason for synthetic biology standards is automation, where the whole process is built from formalised data and processes. When standards are applied, abstraction can be applied to the workflows and technologies, allowing DBTL workflows to be used more generically among different labs. Because this research is focused on standard data, the following sections discuss the computing science principles central to this work and how they can be applied to represent standard synthetic biology design data.

3 Graph Theory and Network Science

3.1 Introduction

"Networks are at the heart of complex systems" - Barabási Complex systems are massively challenging to understand by their nature[45]. Representing data that can be conceptualised as a network helps considerably with comprehension. Networks are becoming common knowledge mainly because of social media and the network of interconnected people. However, networks have also been applied to solve problems with other complex datasets within numerous fields, including biology, security, city infrastructure and the World Wide Web. Networks can expand as new information is discovered and change shape to meet requirements. Because of the multi-variate nature of the data, a flexible structure is critical. Fundamentally, networks encode the interconnections between a set of entities. Graphs are represented in the form of nodes (individual points of data) and edges (relationships between the data) [46]. A node represents each entity, and edges represent connections between nodes. For example, when building networks from circuit designs, a repression relationship edge links two nodes representing a regulator protein (e.g. aTc) and its cognate promoter (pTet).

3.2 Network Science in Biology

Graph theory methods can assist the interrogation of network structures in several ways for circuit designs and produce sub-networks of particular interest hidden within design formats. What has been termed network biology deals with the quantifiable representation of complex cellular systems in graphs and their study to characterise functional behaviour. Furthermore, a network approach is often successfully implemented within systems biology to represent both simulation models and knowledge models [47] for example, to depict multiple omics data within a single network. Within complex systems, networks exist that encode interactions between elements. Genetic designs model complex systems that contain multiple networks such as transcriptional, parts hierarchies or protein interaction networks. Network science has precedence as it has been applied successfully to complete many tasks and provides confidence in using a network approach to genetic designs.

3.3 Fundamental methods and terms

Many existing and fundamental network science methods have been combined to overcome specific challenges. This section will briefly discuss some of the methods used.

Table 1: Keywords, definitions and small methods for networks

Name	Description	Example
Graph	A set of nodes N connected by edges E.	A graph could represent the synthetic biology design.
Structural metrics	Provide a measurement of a structural property of a graph.	The number of genes required to activate a specific protein.

Global metrics	Provide measurements of the whole graph.	To identify highly coupled entities which are all responsible for the production of some chemical.
Local metrics	Provide measurements on an individual node.	The number of coding regions activated by a specific promoter.
Walk	A set of edges within a graph.	The effects of a repressed promoter on a regulatory network.
Path	A walk in which all the edges and all the nodes are different.	The effects of a repressed promoter on a regulatory network without feedback loops.
Path Length	The number of edges the path contains.	The number of entities modified by a change in a regulatory network.
Shortest Path	Shortest distance between nodes i and j. Notation: $d\|d_{ij}$ Never contains self-loops or intersections.	The most efficient path in a regulatory network to enable protein production.
Diameter	Longest shortest path in a graph Notation: d_{max} Distance between the two furthest nodes.	The maximum number of biochemical reactions required to convert one metabolite into another within the organism.
Average path length	Average of the shortest paths between all pairs of nodes. Notation:$< d >$ Formula: $$\frac{1}{N(N-1)} \cdot \sum_{i,j=1,Ni!=j} d_{ij}$$	The average number of biochemical reactions required to convert one metabolite into another within the organism.
Cycle	A path with the same start and end node.	To identify if a path within a regulatory network feeds back on itself.
Eulerian Path	A path that traverses each edge exactly once.	In a network where each node represents a nucleotide and edges represent the order of assembly, the path would reconstruct the entire synthetic DNA molecule without overlaps or gaps.
Hamiltonian Path	A path that visits each node exactly once.	In a network, each node represents an amino acid residue, and edges represent potential interactions between adjacent residues. The path through this graph visits each residue exactly once.

Connectivity	The minimum number of elements that need to be removed to separate the remaining nodes into two or more isolated subgraphs.	Could be used to identify functional modules within a genetic design.
Clique	A subset where an edge connects all nodes.	May refer to a functional module with complementary interactions.
Connected	There is a path between every two nodes.	A connected regulatory network ensures that every gene can potentially influence or be influenced by others.
Strongly connected	There is a route between every two nodes.	A path indicating a feedback loop within the regulatory network.
Complete	There is an edge between every pair of nodes.	Complete protein-protein interaction would imply that every protein interacts directly with every other protein in the network.
Disconnected	At Least one instance of no path between nodes.	This could indicate that there are isolated metabolic pathways that do not interact with each other.
Components	Disconnected Sub-Graphs.	The isolated metabolic pathways within a disconnected network.
Weighted Graphs	Network edges can have properties providing information about individual edges. Weight is a specific numerical property representing the cost of traversing that relationship, whether distance, time, cost or any other factor.	A weight may indicate the levels of expression of a given protein.

3.4 Types of Graph

Based on the type of data represented and the resultant shape, graphs can be split into three categories: monopartite, bipartite and multipartite. A multipartite graph is a graph that contains multiple types of data, for example, parts, interactions, measurements, and simulation data, and it is the most likely type of graph for capturing design data. With large datasets, a multipartite graph is the most likely result. However, a bipartite (two data types) or monopartite (a single data type) graph is more desirable for analysis and comprehensible representation. Graph algorithms do not consider (or can understand unless explicitly explained) the information (labels and properties), and many datatypes complicate representation with large networks. Therefore, it is commonly a requirement before analysis on graphs is performed to produce projections of a graph and perform analysis on this projection. A projection is a compression of information by abstracting two or more nodes (including edges) into a single node. Figure 1 displays a small example of how a multipartite graph (many datatypes in a single graph) can be projected into bipartite (two datatypes in a graph) and monopartite (single datatype in a graph).

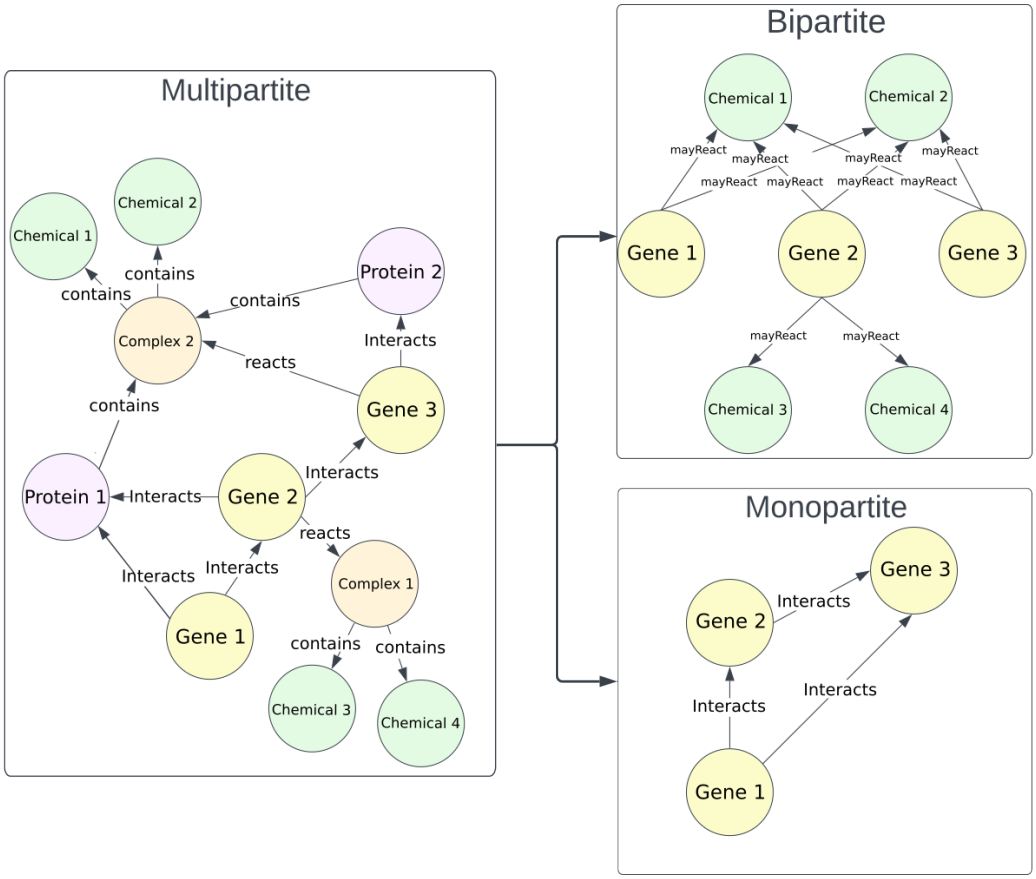

Figure 1: Multipartite graph representing four datatypes (genes, proteins, complexes and chemicals) with edges relating connections between types. The bipartite graph is projected by compressing proteins and complexes into edges representing genes' potential reaction to individual chemicals. The monopartite projection compresses all types apart from genes and relationships between them. Furthermore, new edges are inferred by the indirect connection between nodes.

3.5 Problem Classes

Here, we briefly discuss some general groups of problems that can be solved using a network approach. These are problems that have been solved using networks and are often combined to solve specific and more significant challenges. In each group, the type of problem being solved is discussed within the context of synthetic biology. Each group will be used several times within the upcoming chapters, and the specifics will be discussed where relevant.

Similarity

The similarity is how alike the two nodes are based on neighbouring nodes or node or edge properties. The similarity is a fundamental problem within network science because, without a solution, no methods for comparing within the network exist. Similar algorithms are commonly used within a more extensive pipeline to quantify node similarity for some grander purpose. However,

one potential use of node similarity within synthetic biology could be identifying potentially homologous genes. A similarity measurement may be used based on the interaction of a gene with other entities. If two genes have many of the same connections, that may describe some homogeneity. Node similarity compares nodes based on the number of shared neighbour nodes. This type of similarity helps identify structural equivalence within a network. Node Similarity computes pair-wise similarities based on the Jaccard metric[48](comparison between the sets relative to the total number of unique elements in both sets) or the Overlap coefficient[49](comparison between the number of shared elements between the sets without considering the size of the sets or the number of unique elements). Figure 2 displays a simple example of node similarity using both metrics. The premise is based on the number of similar constituent parts between genes. The Jaccard metric considers Gene2 and Gene1 to have 2/3 similarity because they share two of the same parts out of a maximum possible three. However, the overlapping similarity provides a score of one because all parts in Gene1 are also in Gene2.

- **Jaccard metric (Jaccard Similarity Score)**

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

- **Overlap coefficient (Szymkiewicz–Simpson coefficient)**

$$O(A,B) = \frac{|A \cap B|}{min(|A|, |B|)}$$

Path Finding

Pathfinding walks the network to find a path between two or more nodes. It can also be used to find the shortest path or evaluate the availability and quality of paths between nodes in a network. Pathfinding algorithms use the concept of cost, which is the cost for the pathfinding algorithm to pass an edge. The cost could be the number of edges passed or based on a property on the edge. In the upcoming examples, the weight is explained further. Like similarity algorithms, pathfinding within networks is a fundamental task commonly combined with other algorithms to solve numerous problems. Shortest path algorithms (a subset of pathfinding) could be used in isolation to identify the most optimal pathway within metabolic pathway engineering. Determining the shortest path could indicate an optimal pathway in a network where many paths exist from some input chemicals to some output compound. Many algorithms exist to solve several problems by focusing on different subproblems. A simple pseudo-gene network displays pathfinding between genes in the upcoming explanations of the specific algorithms.

Dijkstra's Shortest Path The Shortest Path algorithm [50] calculates the shortest (weighted) path between a pair of nodes. The shortest path is desirable for multiple usages. Figure 3 displays the shortest path algorithm using both weights and the number of edges. For example, if weight is not being used and simply reducing the number of edges is required, the algorithm finds a path

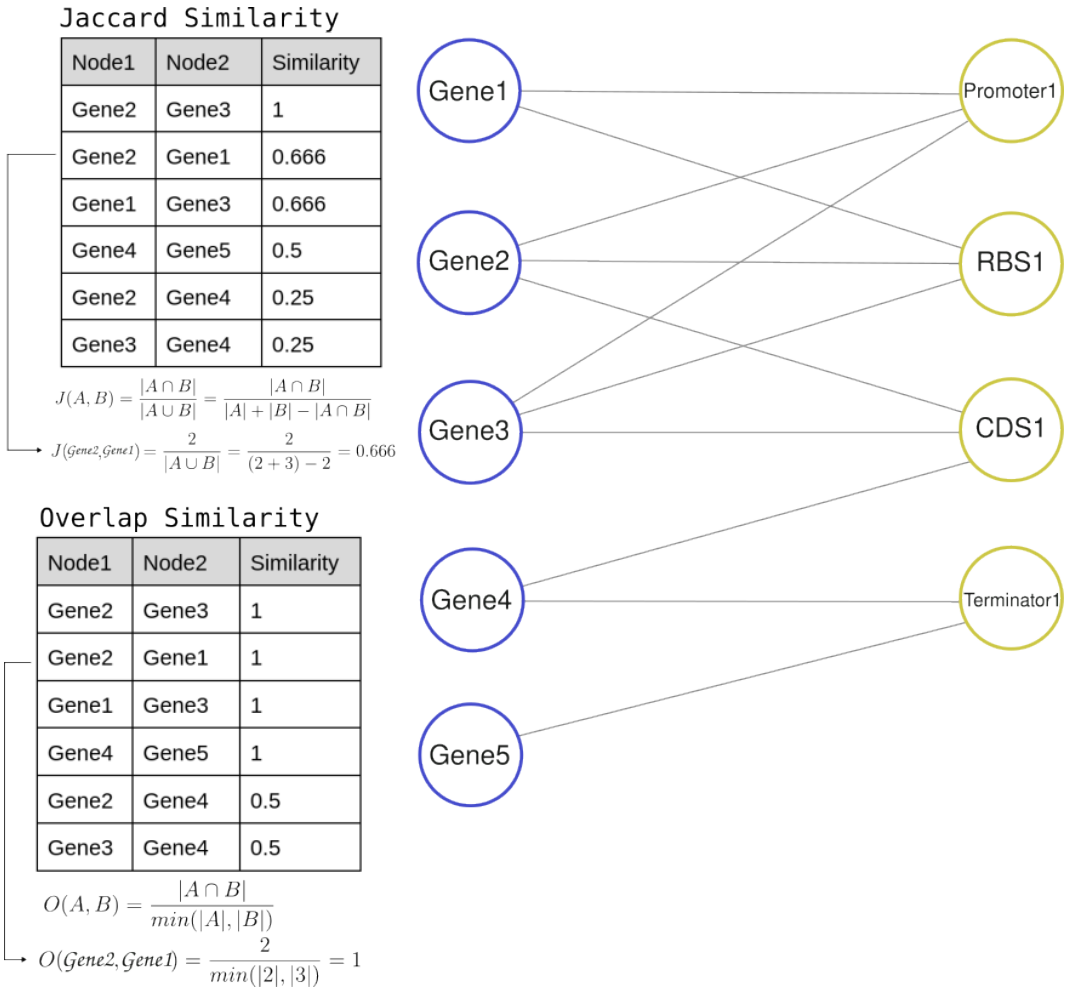

Figure 2: A simple bipartite network displaying Genes and the constituent parts. Right - The visualisation of this network. Left-Top - The results for the Jaccard similarity. Left-Bottom - The results for the Overlap similarity. For both algorithms, only the top six results are displayed.

of length three, but if weight is taken into account, then the result is a length of 4 with a weight cost of 90.

Breadth-First Search The Breadth First Search (BFS) algorithm starts at a source node and traverses to nodes in order of increasing distance. BFS is useful when searching for nodes that fill criteria, and the assumption is that closer nodes are more likely to match. The alternative to BFS is Depth First Search (DFS), which traverses each path before returning. Figure 3 displays a BFS where the source node (Gene1) traverses to Gene2 before backtracking to Gene3, performing the rest of the search in one traversal.

Minimal Spanning Tree The Minimum Spanning Tree begins from a source node, finds all reachable nodes, returns the edges between the source and destination, and the minimal cost

associated with reaching the node. Figure 3 displays the minimal spanning tree using the source node (Gene1) and finding the minimal cost to reach each node within the network.

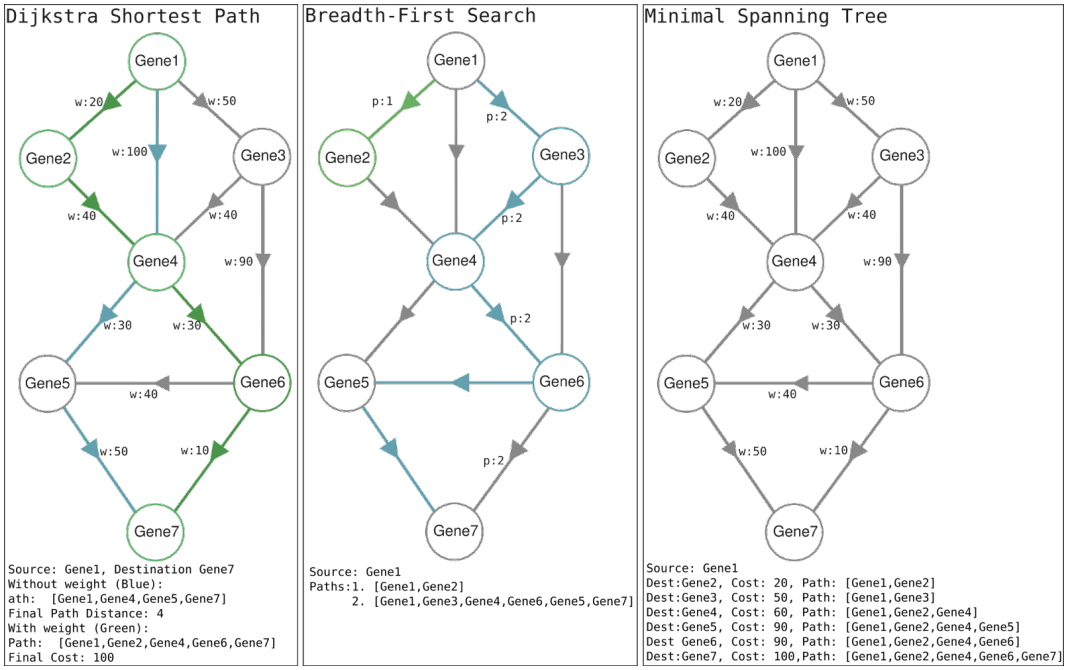

Figure 3: Three examples of pathfinding algorithms. Left - Dijkstras shortest path between the source (Gene1) and destination (Gene2). The algorithm can use edge weight or the number of connections as a metric. For example, blue represents when weights are not considered, and green is the shortest path with the least weight cost. Middle - Breadth-First Search, the algorithm finds its nearest neighbours and fans out, increasing the maximum distance each iteration. The algorithm considers the nodes closest before the ones further away. Right - Minimal Spanning Tree uses weight to calculate the least costly path to each node.

Centrality

Graph topology can be measured using centrality[51], which helps identify the most influential nodes and their role in a network. These measures help identify which nodes play pivotal roles in network structure, communication, or information flow. This information can be used to understand group dynamics, for example, the lynchpin in a metabolic network given some predicate. Centrality can be divided into four categories, each defining "important" nodes.

Degree Degree Centrality is the most straightforward measurement of centrality and is the total direct links with the other nodes. It deems that popularity is the measurement of importance and counts the number of incoming and outgoing edges from a node (degree). Degree Centrality tells us how many direct connections a node has. This type of centrality can be used to find highly

connected nodes within networks, i.e. popular nodes. The degree centrality of node n can be calculated by:

$$C_D(n) = deg(n)$$

, where deg is the node n's degree. Figure 4 displays the degree centrality for all nodes within the network. "Gene2" is the node with the highest degree centrality value because its degree is 3.

Closeness The Closeness Centrality detects nodes that can spread information efficiently through a subgraph. It measures the average distance from a node to all other nodes. A node's score is the sum of all shortest paths to other nodes, and the nodes with the best closeness score are likely the shortest distances to all other nodes. This type of centrality can be used to find the node that can affect the most change in the network when removed or modified. The closeness centrality of node n can be calculated by:

$$C_C(n) = \frac{N}{\sum_n d(n,v)}$$

. The variable d_{nv} represents the shortest path length between node n and v, while N indicates the total number of nodes in the network. Figure 4 displays the closeness centrality with "Gene2" and "Gene3", providing the highest scores as the aggregation of the shortest paths to all reachable nodes is 7.

Betweenness The concept of Betweenness Centrality involves measuring a node's influence on the flow of a graph and its ability to connect other nodes on a path. It helps identify nodes that act as bridges between other nodes, which are essential in a network. This measure can determine which nodes have the most influence on the flow of the network. To calculate the Betweenness Centrality of a node, the formula

$$C_B(n) = \sum_{j<k} \frac{g_{jk}(n)}{g_{jk}}$$

is used, where g_{jk} represents the number of shortest paths connecting j and k, and g_{jk} stands for the number of shortest paths that include the node n. Within Figure 4, the highest betweenness centrality is "Gene6" because all nodes must pass it to reach "Gene7".

4 Interaction Networks

Interaction networks are powerful tools in biology, providing a comprehensive view of how molecules interact and regulate biological processes. This section explores gene regulatory, protein-protein interaction, probabilistic functional integrated and metabolic networks, each focused on specific aspects of molecular interactions within living systems.

Node	Degree	Closeness	Betweenness
Gene1	2	0	0
Gene2	3	1	3.8
Gene3	1	1	1.1
Gene4	1	0.66	1.6
Gene5	1	0.75	4.3
Gene6	1	0.66	5
Gene7	0	0.4	0
Gene8	0	0.66	0

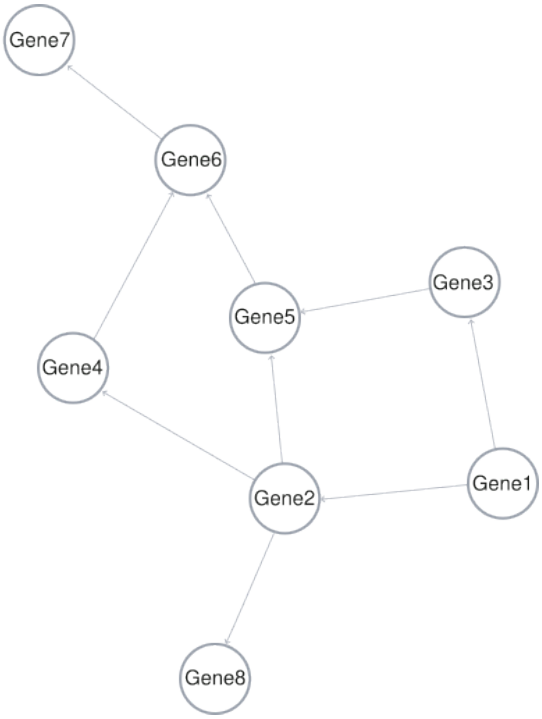

Figure 4: A small pseudo-gene network to display types of centrality (Degree, Closeness and Betweenness). The table represents the centrality of each node under the given predicates.

4.1 Genetic regulatory network (GRN)

A Gene Regulatory Network (GRN) is a network that comprises genes and their interactions, to describe the regulation of gene expression and other critical cellular processes[52]. These networks aim to understand how genes, proteins, and other molecular components work together to perform biological functions[53] by capturing the interactions between genes and their regulatory elements. GRNs show how genes and their regulatory elements interact and demonstrate the dynamic nature of these interactions, revealing how biological systems respond to internal and external stimuli. GRNs help reveal the regulatory hierarchies, feedback loops, and signalling pathways that modulate gene expression patterns and play a crucial role in understanding cellular differentiation, development, response to environmental stimuli, and disease progression. Moreover, GRNs provide a framework for understanding biological processes at a systems level. They help to identify the emergent properties that arise from the interconnectedness of genes and molecular components within regulatory interactions.

4.2 Protein-protein interaction network (PPI)

Protein-protein interaction (PPI) networks are higher order interaction networks describing interactions between[54]. These networks provide a blueprint for the connections between proteins, offering an understanding of cellular function. PPI networks can help understand how cells communicate with each other, how metabolic processes work, and how regulatory cascades per-

form[55]. They help identify the proteins necessary for cellular function. They can reveal the interactions between structural and regulatory proteins, giving insight into the emergent properties of proteins in a biological environment[56]. Moreover, PPI networks offer a rich data source for computational modelling and can help predict protein functions, identify new protein-protein interactions, and simulate cellular behaviour under different conditions. By combining experimental data with computational approaches, PPI networks provide a holistic view of cellular interactions, making them useful for drug discovery[57], biomarker identification[58], and the design of therapeutic interventions[59].

4.3 Integrated networks

Integrated networks are computational models integrating omics data such as genomic, transcriptomic, proteomic, and metabolomic information[60]. These networks provide a unified and multi-layered graph that captures the complexity of biological systems. They represent a comprehensive approach to understanding the interplay of molecular components and regulatory networks that underlie biological functions. Furthermore, integrated networks combine diverse omics data streams, including genes, gene expression, proteins, and metabolites. This integration helps comprehend interactions and signalling pathways that control cellular functions and behaviours.

4.4 Probabilistic functional integrated networks (PFIN)

Probabilistic Functional Integrated Networks (PFINs) are Integrated network that also encodes probabilistic methodologies with functional genomics data[61]. They specifically focus on incorporating probabilistic methods, which capture and represent the uncertainty or probabilities associated with the relationships between biological entities in the network. Like integrated networks, PFINs integrate various omics with a probabilistic context, allowing uncertainty estimation and robust modelling of complex biological phenomena, resulting in a more comprehensive and nuanced representation of biological systems' dynamics[62].

4.5 Metabolic networks

Metabolic networks display the biochemical reactions that capture the flow of metabolites (small molecules crucial for cellular functions) as they perform substrate transformations[63]. They reveal the interconnected reactions that modulate cell biosynthesis, degradation, and conversion of metabolites[64]. Moreover, metabolic networks offer a perspective, showcasing the relationships between cellular components within the metabolic context, such as genes, proteins, and enzymes. They enable systems-level analyses, allowing researchers to unravel emergent properties, identify key metabolic hubs, and decipher the underlying regulatory mechanisms directing cellular metabolism. Metabolic networks integrate experimental data, computational modelling, and network analyses to provide insights into metabolic fluxes[65], regulatory control[66], and adaptive responses under varying environmental conditions[67].

5 Networks in synthetic biology

The discussed networks have been employed successfully within systems biology, exploring natural systems, but they have also been adapted for engineered systems within synthetic biology. Networks within synthetic biology play crucial roles in designing and optimising synthetic systems, which are not requirements when working with natural systems. This section explores some synthetic biology-specific applications of a network approach or usage of the previously discussed networks for a synthetic biology approach.

5.1 Networks for standardised exchange

Genetic designs are a uniquely synthetic biology type of data structure because inherently natural systems being characterised do not need an associated design. Networks are employed in rational design approaches, facilitating the prediction and optimisation of engineered biomolecules such as enzymes, ribozymes, or aptamers for specific functions. As discussed, synthetic biology-centric information is commonly captured as a network (or in a format that can easily be transformed into a network). For example, as discussed, SBOL is captured as RDF, which is already a graph-based format. However, different types of information encoded within different formats can also be converted. Some other formats within synthetic biology can be encoded as a network.

Build protocols

As well as efforts to standardise genetic design, build protocols have been targeted for standardisation. For example, AutoProtocol[68] is a standardised language and framework designed to facilitate the communication of experimental protocols and instructions between software systems and laboratory automation platforms. AutoProtocol is most commonly encoded within the JavaScript Object Notation (JSON)[69], a human-readable data interchange format widely used for data transmission and storage in computer systems. It is often used to represent structured data objects in a format that is easy for humans and machines to read and write. JSON can be a hierarchical network by representing nodes as JSON objects; edges represent hierarchy relationships between the objects.

Simulation data

Simulations play a considerable role within synthetic biology to predict the function and characteristics of engineered systems before building and testing a physical construct[70], reducing both time and financial costs. While initially designed for systems biology, Systems Biology Markup Language (SBML)[71] plays a crucial role in synthetic biology by providing a standardised format for representing computational models of biological systems. SBML is designed to describe biological processes, interactions, and networks in a machine-readable and exchangeable format. SBML, much like SBOL, can be easily represented as a network because it is captured within an XML-like serialisation format and based on the RDF data model.

5.2 Regulatory network and control design

Regulatory network and control design in synthetic biology involves engineering genetic circuits and biological systems to control and manipulate cellular functions[72]. These systems typically consist of regulatory elements, such as promoters, transcription factors, genetic switches, and signalling pathways, designed to achieve specific control over gene expression, cellular processes, or behaviour of biological systems. Currently, synthetic regulatory networks are struggling to reach the complexity of natural regulatory systems[73] because of the inherent complexity in designing functional systems at that scale. Therefore, synthetic regulatory networks are often smaller than natural ones.

Genetic circuits

Genetic circuits are synthetic biology-specific regulatory networks. They are designed systems that mimic the behaviour of electronic circuits by controlling gene expression and cellular functions[74]. These circuits were proposed to enable the engineering of biological systems with controllable and predictable behaviours. They stemmed from the need to design and manipulate cellular functions analogous to electronic circuits because biological systems' analogue nature introduces enormous complexity that is challenging to predict and comprehend. Genetic circuits integrate various genetic elements and regulatory components to process input signals and produce specific output responses within living cells[75]. These circuits can be designed to perform diverse functions, such as logic operations, signal amplification, temporal control of gene expression, and feedback regulation.

5.3 Networks to generate and validate designs

Modelling biological systems using networks is an established approach within synthetic biology[76]. It involves creating computational representations or mathematical descriptions of biological processes to simulate, analyse, and predict their behaviour[77]. These models aim to capture the complexity of biological phenomena. Below is a discussion of two ways of modelling biological systems using networks. One usage is validating existing designs by establishing the systems' functionality, accuracy, and reliability before building them within a lab[78]. Validation ensures the designed systems perform as intended and align with the expected behaviour[79]. For example, biological networks allow for comparing simulated predictions and experimental data by validating computational models against experimental results[80]. This comparison between computational and experimental can be used to increase the accuracy and reliability of both the design and model over time[81]. Instead of validating existing designs, creating novel designs has also been targeted using a network approach[82]. Design generation involves developing, refining, and enhancing biological systems to achieve some abstract function usually described by a practitioner[83]. Automatic design creation is still an emerging technology within synthetic biology, and the main successful approaches adopt a semi-automated approach to guide a practitioner's actions during development. For example, they have been used to help optimise metabolic pathways for producing specific compounds by representing the pathways as networks which can help predict cellular behaviours such as substrate availability, enzyme kinetics, and metabolic fluxes[84].

5.4 Hierarchical networks

In synthetic biology, hierarchies refer to hierarchically organising and arranging biological components, systems, and processes[85]. The goal is to understand better, engineer, and control biological systems at various scales to design biological systems with desired properties and behaviours across multiple scales, from molecular interactions to whole organisms.[86] This approach allows for more efficient and effective design and engineering of biological systems. As discussed, a hierarchy is a common network topology and can be easily represented as one. Two types of hierarchies are common within synthetic biology: structural and functional.

Structural networks

The structural hierarchy in synthetic biology refers to the hierarchical arrangement of biological components and systems based on their physical or spatial levels of organisation[87]. It describes how biological entities are classified in scale, from smaller molecular components to larger structures. Below is an example of a structural hierarchy. **Molecular Level**: At the lowest level, synthetic biology deals with molecular components such as nucleotide or amino acid sequences or individual small molecules. **Genetic parts**: Modular components at the molecular level include DNA sequences (Promoter, CDS, RBS, for example), regulatory elements, and molecular components that perform specific functions within genetic circuits or biological pathways[27]. **Constructs**: Refers to organised assemblies of genetic parts designed and engineered to perform specific functions or tasks. It involves the combination and arrangement of genetic parts into functional units. **Systems**: A structural hierarchy can arbitrarily increase abstraction by defining increasingly large structural entities.

Functional networks

The functional hierarchy in synthetic biology involves organising biological components and systems based on their functional roles, interactions, and behaviours rather than their physical structure[88]. It focuses on how biological systems operate and perform specific functions. Below is an example of a functional hierarchy. **Basic Functionality**: Functional hierarchies begin with understanding and engineering basic biological functions such as gene expression and regulation, signalling, metabolism, and cell growth. **Module and Subsystem Functions**: Modular design principles allow the construction of functional modules or subsystems[89]. These components, comprising genetic circuits or pathways, perform specific tasks or processes within a larger system. **System-Level Functions**: Synthetic biology assembles functional modules into larger systems capable of performing complex functions or behaviours. Systems may include regulatory networks, metabolic pathways[90], or signalling cascades[91] engineered to achieve specific objectives. **Application-Level Functionality**: Like structural hierarchies, functional hierarchies have room for growth. It may describe much higher-order information, such as cellular interactions[92], complete gene networks or cellular computers[37].

6 Knowledge Graphs

Early efforts in modelling biological systems using networks had a limitation: the lack of structure and the meaning of nodes and connections. For example, if two nodes representing proteins are linked, it is helpful to know what type of connection this is, like a binding, repression, or activation interaction. Some recent efforts are based on knowledge graphs[93], networks that model a real-world environment and can capture noisy real-world information in a structured domain, allowing more abstract, unanticipated questions. Two main aspects make knowledge graphs, semantic labels and rules regarding what data can connect.

- **Semantic Labelling** - Refers to entities within a knowledge graph containing standard known labels which can be computationally detected and used, providing semantic standardisation.

- **Relationship Rules** - Refers to what types of data can connect by specific connection types and provides structural standardisation to data, i.e. all potential connection types are known beforehand.

Knowledge graphs have been widely used in biological sciences, from building knowledge bases to predicting biological reactions and are used here as the basis for structuring input data[94]. The addition of semantics allows for more complex control over the underlying data, e.g., the ability to arrange information into several layers of abstraction.

6.1 The Resource Description Framework (RDF)

As discussed, file formats such as Genbank are semantically and structurally ambiguous and incompatible with a standard approach to synthetic biology[4]. For example, if a genetic design reaches a level of complexity that considering it at the level of interacting genetic parts is no longer possible, and comprehension is only achievable at some level of abstraction, explicitly storing this detail is impossible; this is nothing to say of multi-cellular systems which intercellular interactions cannot be defined. Furthermore, attributes of entities, such as genetic roles or interactions, are captured within free-text annotations; therefore, dissemination is challenging. For example, a representation of a coding region could be named CDS, Coding Region, Coding Sequence or Protein Coding Region. As humans, we understand these are synonymous, but without the explicit knowledge, software attempting to use this knowledge would assume these to be different genetic roles. Therefore, Genbank files have two fundamental flaws: the inability to capture abstract structures, such as interactions or structural or functional modules and the inherent semantic ambiguity that comes with free text. Resource Description Framework (RDF)[95] is a model to represent data about physical objects and abstract concepts and express relationships between entities using a graph format. RDF has been designed for the representation of corresponding data on the web. Fundamentally, RDF is a foundational technology for defining standard models for unambiguous data exchange. Models defined in RDF primarily provide two mechanisms that make it preferable over free-text descriptions: structure and semantics. Structure refers to the shape of a dataset, that is, how entities within that set can connect. The Genbank format makes the structure flat because it lists features related to a sequence. Where the structure is concerned with

how data can connect, semantics is the meaning of the underlying data. The Genbank format's semantics are not constrained; therefore, any characters in any order can be specified. The semantic range of a Genbank file is largely unconstrained. While it is true that annotation qualifiers can be loosely interpreted because they are contained within a large domain of values, the values of these qualifiers are written annotations which cannot be reliably interpreted. This difference can be seen when the data is represented as graphs within Figure 5; with Genbank, only the connection between the document and annotations can be specified, but with RDF, the data points can interconnect with different types of connections. The ability to define arbitrary edge types produces more robust networks that encode much more information than the flat structure of a Genabank file. RDF allows describing anything: people, genes, objects, and concepts. RDF represent information by statements in the following format: **<subject> <predicate> <object>**
As seen within Figure 6, concerning RDF in the context of a network, both the subject and object are equivalent to nodes, and the predicate is equivalent to an edge between these nodes. Those statements express a relation between the subject and the object, both resources in a dataset and are known as triples. Table 2 displays the three triples, where each row within the subject and object columns maps to a node and the predicate maps to an edge between the entities in that row. Resources can be present in multiple triples but perform different functions. For example,

Table 2: Example of three triples defining a simple LacI regulation system. Subject and Object columns define entities within the design and the predicate column defines relationships between the subject and object.

Subject	Predicate	Object
<IPTG>	<bindsTo>	<LacI>
<LacI>	<bindsTo>	<pLac>
<LacI>	<description>	"Coding region for the LacI protein."

take the LacI resource used within multiple triples as the subject and object and perform different functions. The resources in the example triples are all placeholder references to objects for this example. However, in true RDF, resources can be IRIs, literals, blank nodes, and predicate IRIs. IRI (Internationalized Resource Identifier) is a protocol standard and allows containing characters from the Unicode character set. Also, literals are fixed values, including strings, dates, and numbers. Finally, blank nodes are placeholder nodes for either temporary or unknown values. Table 3 removes the placeholders and introduces the full IRIs. While this appears more complex,

Table 3: Example of three triples defining a simple LacI regulation system. Subject and Object columns define entities within the design and the predicate column defines relationships between the subject and object. Some resources (IRI) define accessible resources which reference external resources.

Subject	Predicate	Object
<https://openwetware.org/wiki/IPTG>	<http://identifiers.org/biomodels.sbo/SBO:0000177>	<http://parts.igem.org/Part:BBa_C0012>
<http://parts.igem.org/Part:BBa_C0012>	<http://identifiers.org/biomodels.sbo/SBO:0000177>	<https://parts.igem.org/Part:BBa_R0010>
<http://parts.igem.org/Part:BBa_C0012>	<http://purl.org/dc/terms/description>	"Coding region for the LacI protein."

the only change is that local terms have been swapped with links to online resources. Components

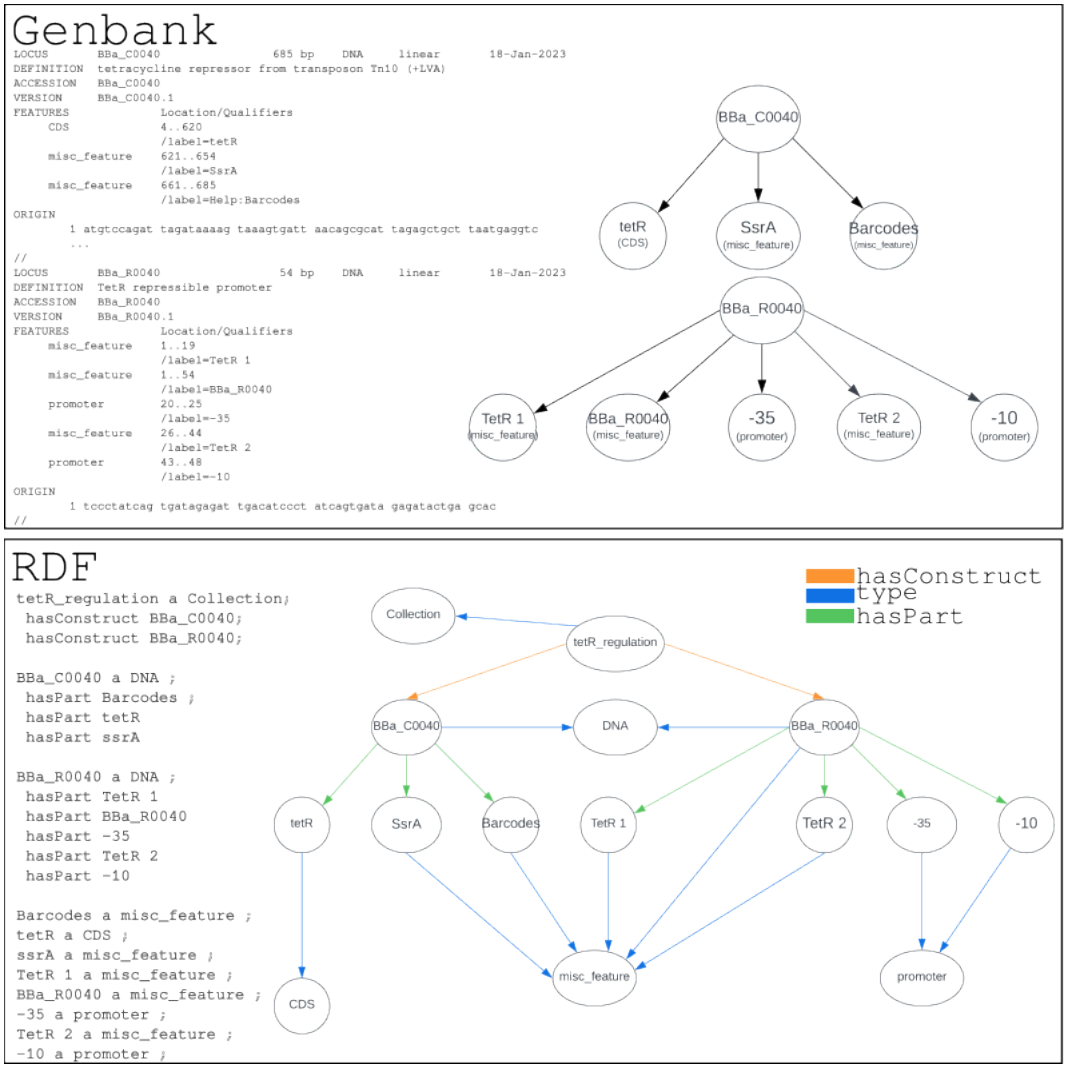

Figure 5: Comparison between the representation of GenBank file and the same data represented as RDF when visualised as a graph. The files define the same TetR regulatory system taken directly from the IGEM parts repository. The Genbank (TOP) can only encode a flat structure where the record links directly to direct physical parts. In contrast, RDF (BOTTOM) can represent an arbitrary depth structural hierarchy. Genbank also cannot explicitly encode the types of connections. However, RDF can explicitly explain the types of relationships between entities. Hence, the node type can be expressed, creating a new node.

representing resources display the power of RDF, combining and unifying information from different datasets.

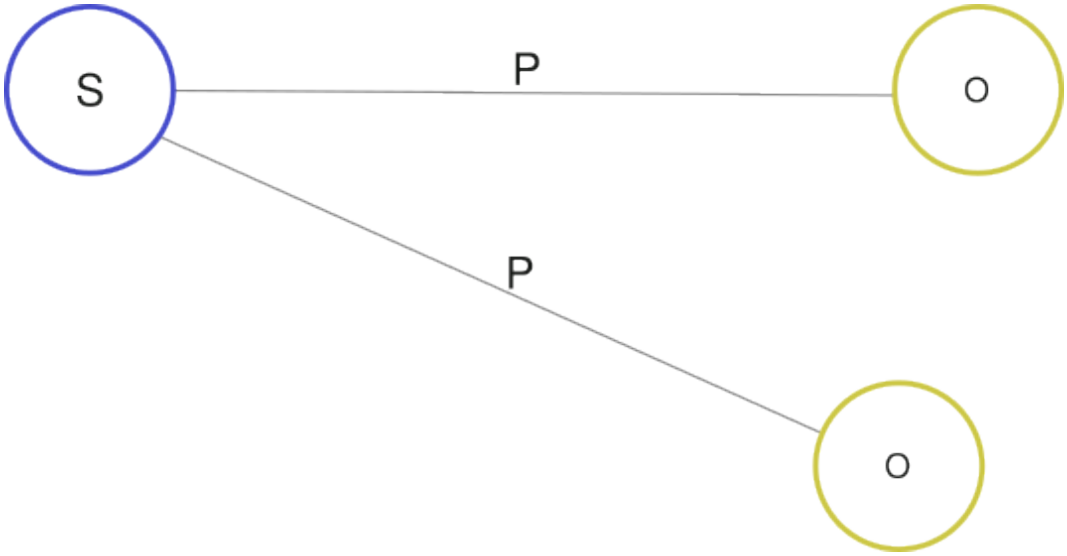

Figure 6: RDF structure is described as "(s,p,o)", subject, predicate and object.

6.2 Ontologies

As discussed, RDF alone does not inherently provide structural and semantics standardisation, as seen within 5; despite being captured in RDF, there is no guarantee that another RDF file does not use a different structure or semantics. Therefore, the shape or semantic labels between datasets are not guaranteed to be unified. Ontologies[96, 97, 98] are generalised knowledge graphs, meaning they only model general types of things that share certain properties but do not include information about specific individuals. Ontologies can be conceptualised as blueprints (a contract on structure and shape) to produce instances that are also knowledge graphs. It defines a set of classes that can be instantiated and rules that dictate what objects can connect and in what ways. For example, ontologies are used within synthetic biology to provide design blueprints. It may describe the labels which pertain to biological roles, such as a promoter or RBS or types of biological parts that can interact. In short, an ontology defines a common vocabulary for researchers who need to share information in a domain. Ontologies provide several benefits:

- Information structure contracts are agreed upon among people or software agents, enabling interoperability.
- Ontologies can be reused, i.e. ontologies can use or extend other ontologies as needed.
- Domain assumptions can be easily changed irrespective of implementation if the knowledge of the domain changes.
- Ontologies capture domain knowledge that software agents can use to specify operational knowledge. The separation of the domain and operational knowledge allows operations to be implemented irrespective of the domain.

In conclusion, ontologies limit complexity, organise data and enable computational tractability. Therefore, if each individual agrees to a blueprint, this provides a standard approach to data capture, ensuring information can be easily captured, integrated and shared between potentially disparate institutions or people.

RDF VS Labelled graphs for Ontologies

So far, a question may be raised comparing the RDF graph to the labelled graph. All RDF graphs can be represented as a directed labelled graph, but not all labelled graphs can be represented as an RDF graph. An issue that means that all labelled graphs can not be represented as an RDF graph is that predicates (edges) cannot encode annotations, as displayed in Figure 6. For example, weight is commonly used within network science to encode the strength of an edge relative to some metric. Therefore, we define the knowledge graph during research as a directed labelled graph where nodes and edges contain an arbitrary number of properties, as seen in Figure 7.

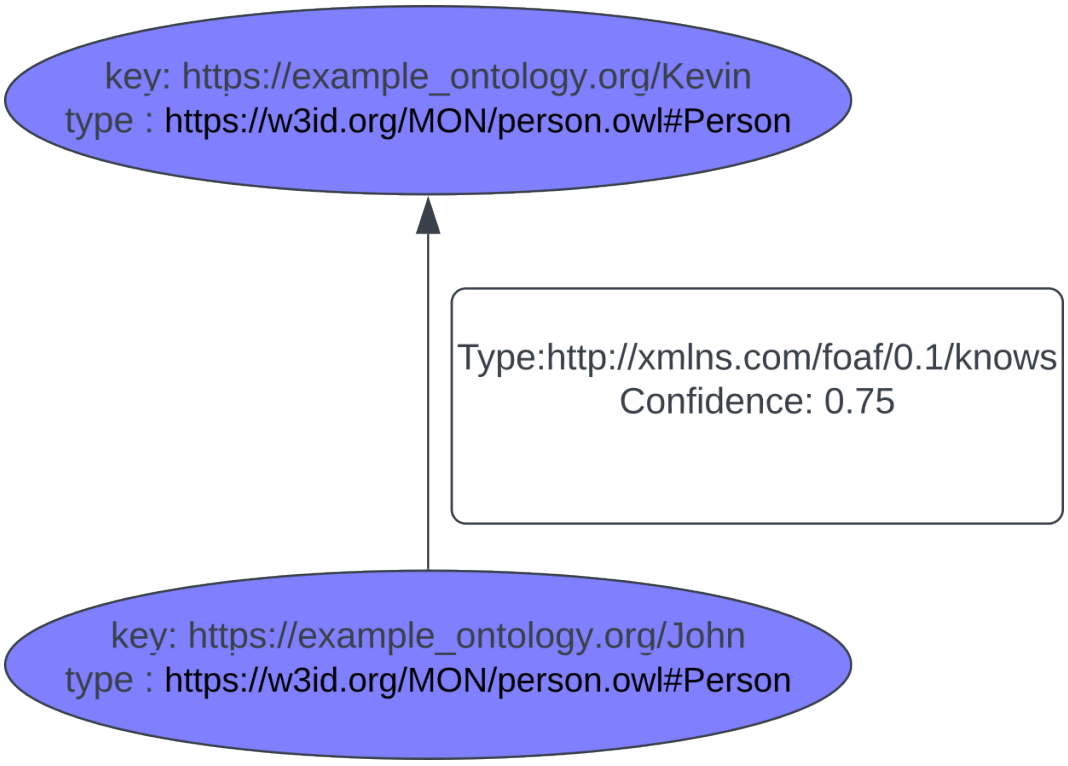

Figure 7: Example of how RDF can be expanded to incorporate properties. Each node contains a key and a type, and edges contain a type and confidence. Despite nodes and edges not explicitly linking to a single URI, URIs are still used as property values.

7 Data Standards

Data formats such as Genbank are ubiquitous within synthetic biology. They allow simple annotations to be defined but are often open to interpretation due to a lack of complete standard semantics and structure. Therefore, they are not conducive to the iterative nature of synthetic biology because the static, non-standard nature makes development challenging and is accentuated with automation because computational handling is convoluted.

7.1 Synthetic Biology Open Language (SBOL)

Data formats that effectively capture and represent increasingly complex designs have emerged. A leading example is a standard to implement a synthetic biology-centric knowledge graph: Synthetic Biology Open Language[10] (SBOL), which describes both structural (e.g. DNA sequences) and functional (e.g. regulation interactions) information. The SBOL community describes the model as "An open standard for the representation of in-silico biological designs". SBOL provides several specific advantages:

- SBOL can be captured as RDF; therefore, all the advantages of RDF exist.
- SBOL provides a contract so all agents can comprehend others' data.
- SBOL is an open language, and all features are free. A language which is synthetic biology specific, catering to the specific requirements.
- SBOL can be used as a general container for different data types. For example, SBOL does not have the mechanisms to define simulation models (this is not the focus of SBOL). However, Systems Biology Modelling Language (SBML), which defines simulation models, can be linked to an SBOL document.

As explained, SBOL is captured using RDF; therefore, the model inherently has a high affinity with the network approaches discussed. Practically, SBOL is a specification that provides a contract for all users to abide towards if they want interoperability with other users. Figure 8 displays an example of the SBOL ontology providing classes for all users to create instances. If the same terms and connections are adhered to, then the underlying semantics of all designs will be the same. To this end, the Synthetic Biology Open Language (SBOL) data standard will capture genetic design data. Below, the data model is explained in more detail to provide insight into its usage. However, it must be noted that despite using the model for data capture, this research focuses not on the model itself but on using it to reach our goals. Therefore, the data model will be hidden for the most part.

SBOL Versions

- **SBOL 1** - The initial release of SBOL, primarily a structured version of a Genbank file where sequence data and annotations can be defined.
- **SBOL 2** - Introduced the ability to define functional aspects of a design, such as interactions.

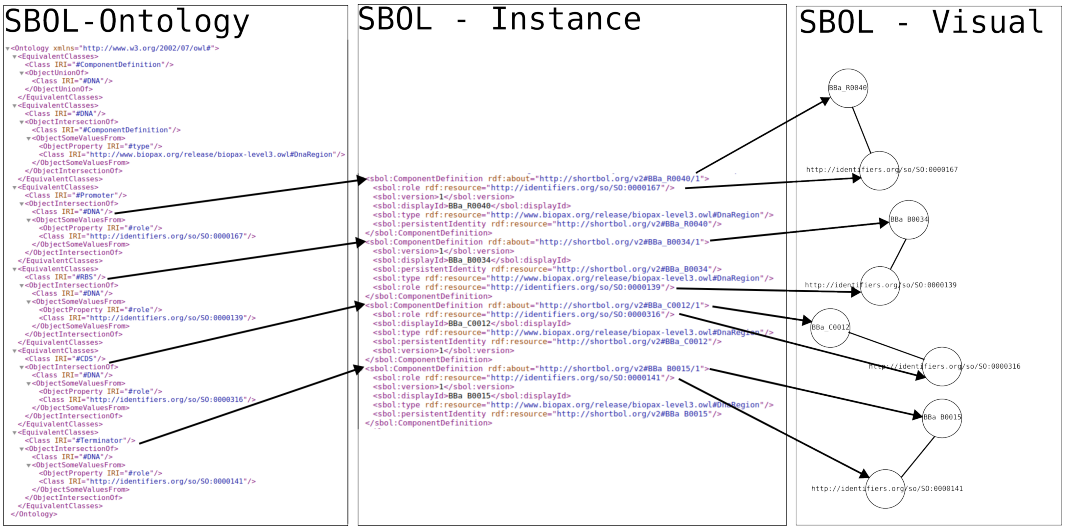

Figure 8: Simple visualisation of how SBOL designs can be created from the SBOL ontology. Classes are defined within the SBOL ontology (ComponentDefinition, DNA, Promoter, RBS, CDS and Terminator). These classes can then be instantiated within a design by providing a specific name (part names, for example). The instance of a design could then be visualised as the underlying network.

- **SBOL 3** - The most recent SBOL iterations, a simplified version of SBOL 2 designed to be more accessible.

Below is the core classes for the SBOL3 version displayed in figure 9.

7.2 Advantages of standard data

Standardisation of data within synthetic biology provides many benefits when correctly implemented and adopted. Here is an overview of some advantages of standardisation within the context of design data.

Tractability

Increased tractability means software and databases can comprehend the underlying data more efficiently and in greater detail. Currently, designs are most commonly shared via Genbank unstructured and ambiguous files, often containing large amounts of free text data. Therefore, if one agent (programmatic or human) creates a design, another agent must derive the encoded detail from a vast range of potential values in a large domain of potential datatypes. When design data is captured within a standard format, the structure is introduced by constraining the domain, and semantic tags are used instead of free text to narrow the potential range of values.

Interopabilility

Interoperability[99] refers to software, users and databases' ability to exchange data to understand incoming and outgoing data. Data must be tractable because each agent that exchanges data must

Table 4: Core components and description of SBOL3 data model.

Class	Description
Component	The **Component** is the core class within SBOL and can represent any entity. The primary usage of this class is to represent entities with designed sequences, such as DNA, RNA, and proteins. However, it can also represent any other entity part of a design, such as simple chemicals, molecular complexes, strains, media, light, and abstract functional groupings of other entities.
Interaction	The **Interaction** class describes the desired functional behaviour of how the Feature objects of a **Component** interact. For example, this class can represent different forms of genetic regulation, processes from the central dogma of biology and other fundamental molecular interactions.
Participation	Each **Participation** represents the role of features within an **interaction**.
Feature	The **Feature** class composes Component objects into a structural or functional hierarchy.
Location	The **Location** class represents the location of **Features** on a **Sequence**.
Sequence	The **Sequence** class represents a **Component** object's primary structure.
Constraint	The **Constraint** class assert restrictions on the relationships of pairs of **Feature** objects relative to the **Sequence** they are attached to.

understand what is being received. Interoperability within design data is especially desirable because a single agent is unlikely to develop a design from beginning to end. Instead, multiple tools focusing on a specific aspect combined with multiple humans with specific expertise will likely interface with the design.

Abstraction via Modularity

Modularity[100] refers to composing genetic parts into modules to perform a specific function. Abstraction concerning design data is hiding genetic and physical parts with larger constructs that perform a specific function. Therefore, modularising genetic designs increases abstraction because modules to perform a specific function can be swapped for another module, which differs in implementation but is similar in function. Design standards such as SBOL enable grouping

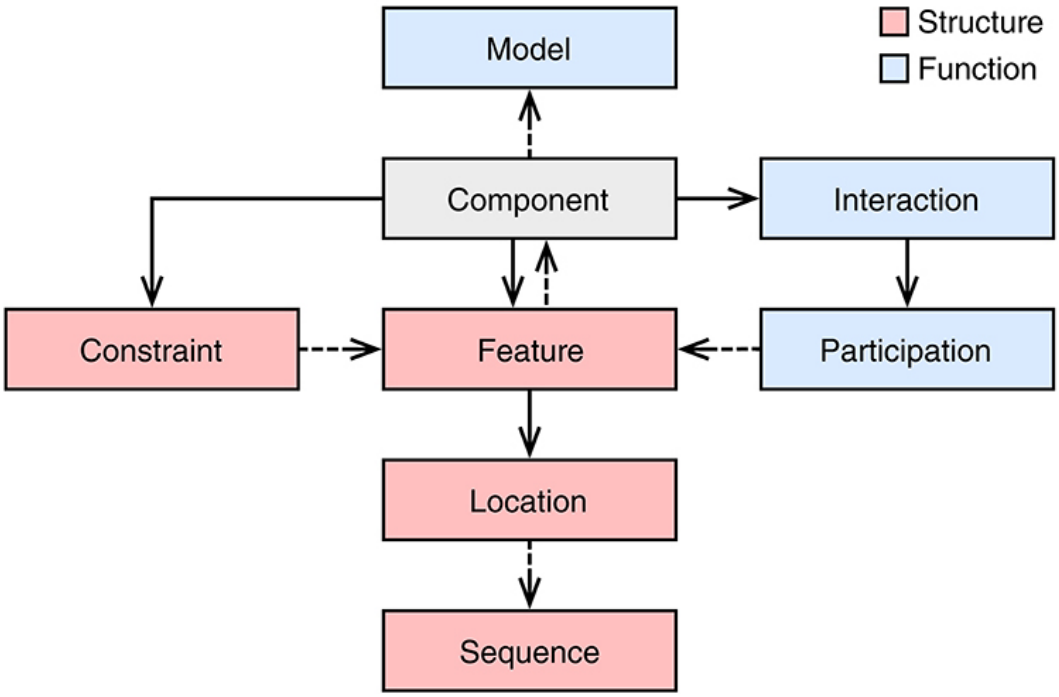

Figure 9: Overview of SBOL3 data model. The component is the core class with functional (Blue) and structural (Red) classes directly or indirectly interfacing with a component.[4]

genetic parts into collections based on physical structure or function, which is a step towards abstraction via modularity.

Reduced Human Error

The human error introduced during design can be costly if the error persists in the construction of the system[101]. For example, the designer may take the sequence for a genetic part from a database with expectations about function during design. However, with most data sources currently, there is no guarantee that this is the exact desired function or that the advertised function is correct. While standards will certainly not remove all human error when design data is structured with semantics errors due to ambiguity, it will undoubtedly be reduced.

7.3 Challenges introduced by standards

An issue introduced with SBOL is the increased complexity of the underlying data structure. For instance, consider the difference between a Genbank file and an SBOL file. A Genbank file is a simple sequence combined with some textual annotations that describe the location of the sequence in a common language. On the other hand, an SBOL file contains the same data but encodes all the information to make the data computer-tractable. The Genbank file is less complex to read because it only requires biological knowledge; there is no pre-existing knowledge of the data standard encoded. Because SBOL RDF/XML is too verbose and complex for humans to edit designs

manually, particularly those involving many components and features, tools must be developed to abstract this complexity. This difference can be seen in Figure 10, where the same small construct is described within Genbank and SBOL. While technically describing the same information, the SBOL version is much larger and less user-friendly. Software tools and libraries have been developed to manipulate SBOL, for example, libSBOLj[102] and pySBOL[103], which can be linked to other software, enabling them to read, write, and manipulate SBOL data. These libraries support tool developers and others with programming skills; using them presents a challenging learning curve for most synthetic biologists.

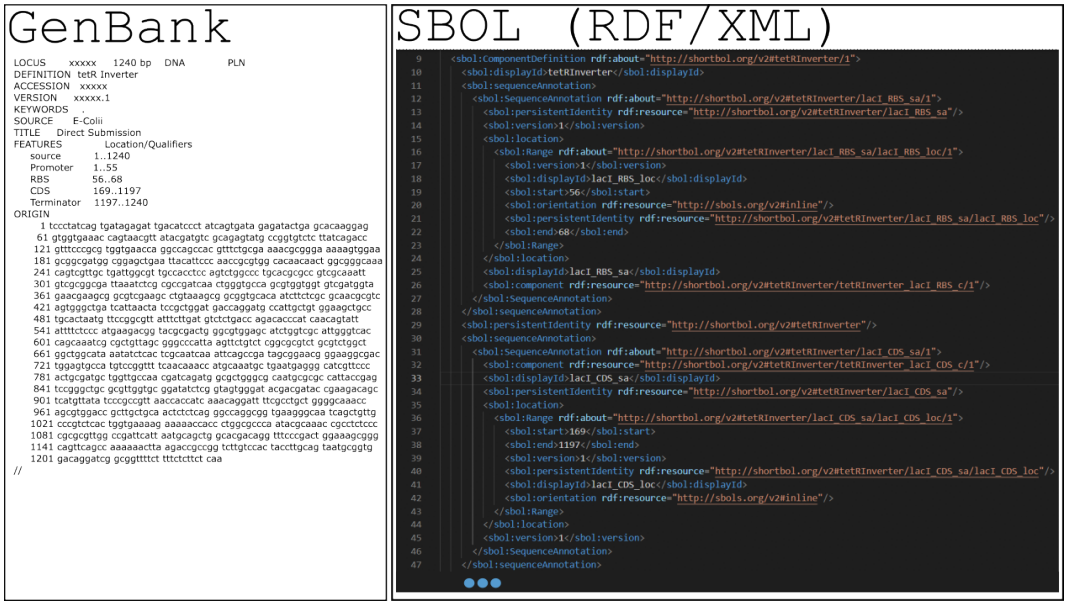

Figure 10: A GenBank file (Left) and SBOL encoded XML (Right) describing the same Gene (promoter, RBS, CDS and Terminator). The SBOL is cut off due to the large size of the document.

The background chapter serves as the foundation for the thesis, providing a comprehensive overview of existing knowledge relevant to the research topic. It introduces synthetic biology, which draws on engineering principles and borrows from diverse fields like computer science, systems biology, and molecular biology. It applies forward engineering to design novel biological systems not found in nature or to perform functions beyond natural capabilities. This field operates on three core principles: modularisation, abstracting complexity by encapsulating sequence data into biological parts, abstraction, and standardisation, establishing shared principles and protocols to facilitate interoperability, fostering the reuse of biological components, and improving design predictability. Next, graph theory and network science were established by introducing their purpose: understanding and analysing complex systems. Furthermore, an introduction to network science in biology to understand complex biological systems was outlined. Subsequently, the fundamental terminology, methods and processes commonly used to solve challenges in many domains were discussed. Finally, for the network background, a review of existing work of networks in Synthetic biology was reviewed. Once networks were established, knowledge graphs were enabled, addressing the limitations of conventional networks by incorporating semantic labelling

and relationship rules, enhancing structural and semantic standardisation within the data and enabling more comprehensive control over the information, particularly relevant in biological sciences for organising and predicting complex reactions. Next, the limitations of GenBank files and the advantages of the Resource Description Framework (RDF) in effectively representing biological data were explored. Finally, ontologies, as conceptualised blueprints providing structure and shared vocabulary, facilitate standardisation and computational tractability, allowing easy data integration and sharing. Finally, data standards, such as SBOL, were shown, including usage, advantages, and disadvantages and the core of the SBOL data model was described. From this, challenges introduced by standards were identified, namely the introduction of considerable complexities.

Chapter 3: ShortBOL - A language to specify standard design data via an extensible and user-facing language.

Publications arising from this chapter

- Matthew Crowther et al. "ShortBOL: A Language for Scripting Designs for Engineered Biological Systems Using Synthetic Biology Open Language (SBOL)". *ACS Synthetic Biology* 9:4, 2020. PMID: 32129980, pp. 962–966. DOI: 10.1021/acssynbio.9b00470. eprint: https://doi.org/10.1021/acssynbio.9b00470. URL: https://doi.org/10.1021/acssynbio.9b00470

Software arising from this chapter

- ShortBol

1 Introduction

Synthetic biology, like in many domains, contains highly specialised language[104]. This language, combined with the number of colloquialisms and aliases where different people use many synonymous names, means that language within synthetic biology is highly nuanced. However, an unconstrained language domain is an issue for attempts at standardisation because it is inherently ambiguous [105]. However, while well-suited for precise machine communication, introducing standards does little to reduce unconstrained language problems because forcing a blanket language change within synthetic biology is impossible. Furthermore, the semantics of the data model introduced by SBOL are too verbose, complex and far removed from the existing language for humans to manually edit designs, particularly those involving many components and features.

Software tools and libraries have been developed to manipulate SBOL, which can support tool developers and others with solid programming skills; using them presents a highly challenging learning curve for most synthetic biologists, not helped by the non-familiar semantics of the data model. Computer-aided Design (CAD) and visualisation tools have also been developed to visualise designs and make the designs easier for humans to communicate. However, these visual design tools are often limited in the features of the representation they can access, and visual editing is often a slow and somewhat manual process. Therefore, there is a need for a standard method for specifying genetic designs, with the ability to define an array of datatypes at a scalable level of abstraction using a system with a low barrier to entry.

1.1 Existing specification methods

Methods for specifying genetic designs are not new concepts. Visual methods for specifying genetic designs and domain-specific languages (DSL) exist within synthetic biology and related fields[106, 107]. DSLs are programming languages for specific domains, making it easier for non-programmers to communicate ideas by providing higher abstraction and readability.

However, in synthetic biology, a language backed by a standard such as SBOL that can describe an arbitrary level of detail is missing. This section reviews several existing methods for specifying genetic designs, both visually and typed. During this review, the existing design of a NOR gate (consisting of two transcription factors of bidirectional gene regulation of another regulatory gene which represses a constituent promoter that co-regulates three genes, one output and two others which regulate the initial transcription factors) built by Nielsen and colleagues[19] will be used to display the visualisation and typed techniques.

Visual Methods

Computer-aided Design (CAD) and visualisation tools have also been developed to visualise designs and make the designs easier for humans to communicate. [108, 109] These visual design tools, however, are often limited in the features of the representation that they can access. Furthermore, visual editing is usually slow, scales poorly with size and complexity and is generally automation-unfriendly because, inherently, the visual process is designed for human use. Here we will briefly discuss several different methods for visual design specification.

SBOL Visual formalises the ubiquitous glyph method of describing genetic designs at the sequence level. Figure 1 displays this concept where each genetic part is represented as an icon on a linear sequence and may include abstract interactions illustrated by lines between glyphs. Many tools exist that implement this specification, such as SBOL Canvas [110] and VisBol [109]. However, the glyph approach is far more commonly a manual and physical process where a person quickly sketches on a physical whiteboard or lab book to convey a design to others, for example.

Advantages This approach offers the advantage of swiftly conveying a substantial amount of information. For instance, small-scale design concepts can be visualised quickly, making it a faster option when compared to similar methods. As previously mentioned, it is easily adaptable for manual creation, making it accessible without familiarity with the SBOL data model, software, or language. When applied with a standardised framework, this mapping facilitates the creation of formal representations from a less structured approach, eliminating unnecessary complexities while retaining the advantages of standardised data. Additionally, the glyph approach is already an established method for prototyping designs, further reducing unfamiliarity issues and streamlining its adoption.

Disadvantages This approach becomes infeasible when dealing with complex designs due to the glyphs representing the primary structure, which may not provide an adequate level of abstraction. For instance, in the case of a genome-scale design comprising thousands of genetic parts represented as glyphs, comprehension becomes virtually impossible. Also, it primarily offers a perspective of design data limited to the primary structure and simple interactions. Consequently, it cannot effectively visualize experimental data or abstract functions, limiting its utility for broader

applications. While common glyphs like Promoter, RBS, CDS, and Terminator are easily recognizable, more specialized glyphs may not be immediately identifiable to all users, potentially causing interpretation challenges.

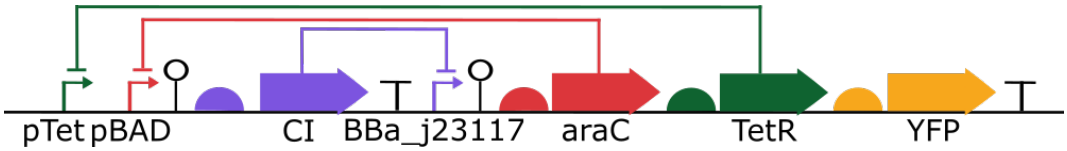

Figure 1: NOR gate[19], represented within the SBOL visual. The design involves two transcription factors that regulate genes in two directions, along with another regulatory gene that represses a promoter that regulates three genes. One of these genes is an output gene, and the other two regulate the initial transcription factors.

Sequence Map The most common approach for developing genetic designs involves sequence-level editing and conceptualising them as plasmid maps. Benchling[111] is the most common tool for sequence-focused editing and represents this as a linear or circular plasmid. When exported, the design is captured within the Genbank file format. Figure 2 displays how the abstract NOR gate can be edited at the individual nucleotide level. In this representation, parts or annotations are displayed as coloured regions, which may be encoded by the user or automatically identified.

Advantages Currently, conceptualising a design at the sequence level is the most widely used and accessible method for examining and making edits. Consequently, there are no challenges or learning curves associated with adopting new techniques. Also, editing a design at the sequence level offers the highest degree of control and precision possible. Moreover, tools like Benchling even provide automated annotation of detailed features, including restriction sites, enhancing the user's capabilities in fine-tuning their designs.

Disadvantages Making alterations to a design at the sequence level can contradict the concept of modularity within synthetic biology, as it often involves modifying individual nucleotides rather than working with abstract modules, which hinders the principle of reusability and standardisation. Attempting to understand large designs at the sequence level is unfeasible due to the complexity involved. For instance, deducing the function of a structure comprising 1000+ parts would be an exceptionally labour-intensive process. Furthermore, this representation does not inherently align with a standardised format. It predominantly encourages the use of flat file formats like Genbank, as it primarily focuses on the primary sequence data without capturing the modular and hierarchical aspects of the design.

Existing languages

Languages to abstract complexity are not new. In large part, computing science achieves abstraction using programming languages that hide specific complexities. The process of creating abstraction languages is the history of programming languages and has significantly contributed to the rapid progress of silicon-based technology over the last decades. For example, the C programming language was initially developed due to the complexities and difficulties of engineering using assembly language[112]. The synthetic biology community has long recognised the utility of such

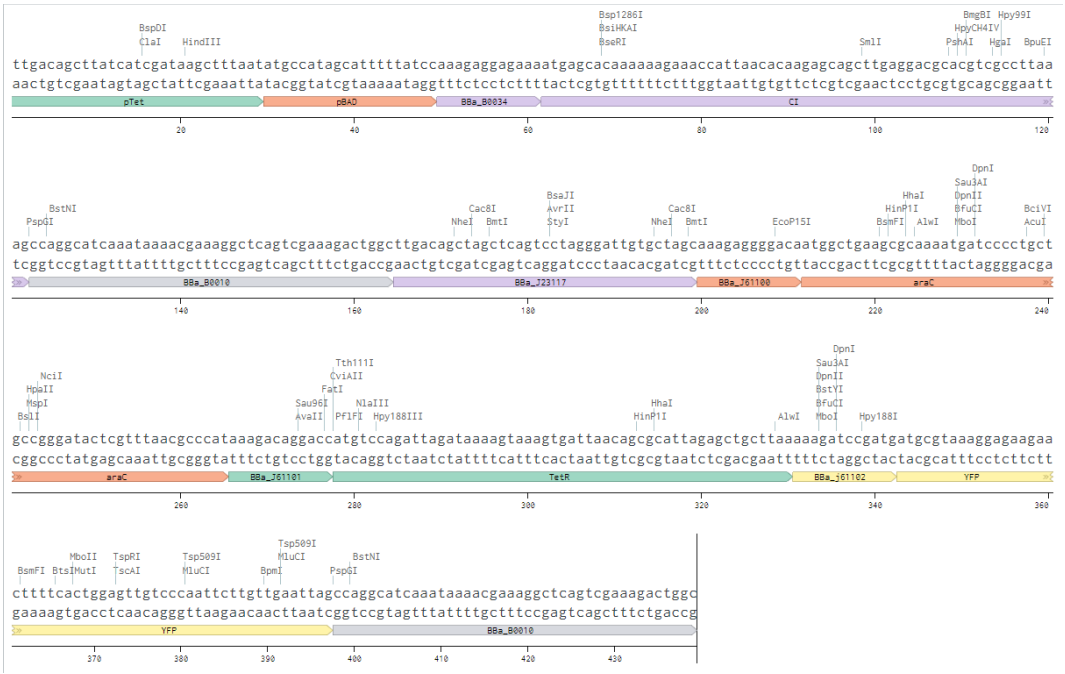

Figure 2: NOR gate[19] implementation represented as a sequence map, the arrowed squares represent user-annotated genetic parts on the sequence, and colour denotes genetic roles. Created using Benchling[111]. For convenience, large sequences have been truncated.

domain-specific languages. Languages such as the Genotype Specification Language [113] and Eugene[114] have previously been developed, in particular, to enable automated assembly and the exploration of the synthetic biological design space. Here we will briefly discuss several different methods for language-based design specification.

Genetic Engineering of Living Cells (GEC) Genetic Engineering of Cells (GEC) [115] is a language for defining transcriptional genetic designs. The language asks the user to define abstract parts and interactions, which are then swapped with the standard MIT Registry of Standard Biological Parts to define genetics parts. To complement the language is a tool specifically designed for the language. Furthermore, programs can also be translated into reactions, allowing simulations to be carried out.

Advantages The approach can seamlessly interact with databases integrated into the language implementation, which means that instead of users creating local terms, they can access virtual counterparts from databases, ensuring standardised referencing. Additionally, GEC can generate multiple designs for a given specification. This versatility in offering several potential solutions can identify the most viable or optimised candidates to fulfil a specific function, enhancing the design process.

Disadvantages The practical use of the language is currently challenging due to the absence of active development and support for the software. Furthermore, incorporating the ability to perform simulations into a language can introduce complexity that may not be necessary for all

users. This inherent complexity is typically associated with languages that allow the definition of complex features, potentially making the language less accessible to a broader user base.

Eugene Like SBOL, Eugene[114] is both a human and machine-readable language for specifying the design of biological systems. Furthermore, like GEC, the language and underlying tooling can generate solutions from abstract design specifications. The language uses constraints that must be fulfilled, such as containing specific genetic parts within the final design. Figure 3 displays the NOR gate within Eugene. This definition is the minimally viable implementation and could be expanded to produce multiple variations of the NOR gate, further constraints on part position or interactions between parts.

Advantages An expressive language that enables the definition of logic can facilitate the generation of multiple design variants. Therefore, instead of users creating localized terms, they can access virtual counterparts from databases, ensuring standardized references. Also, this approach is particularly effective when dealing with larger and more complex designs, as it allows for the modularisation and reuse of logic within a design file, streamlining the design process and promoting modularity and reusability.

Disadvantages The language can be intricate and may pose challenges in interpretation due to the multitude of features within Eugene, particularly for specifying constraints and logic. The capability to define logical constructs can lead to a verbose language, which may demand more manual effort for smaller designs when compared to similar tools. Furthermore, it is essential to note that the language is still under development, and it may undergo significant changes in the future, which could impact its stability and functionality.

Programming language implementation While not DSLs, programming language implementations of the SBOL data model, such as pySBOL[103], libJS[116] and libSBOLj3[117], exist, which are third-party libraries created within the given programming language. The languages use the native features of the language to enable SBOL specification. For example, within object-orientated languages such as Python, the classes are defined for each SBOL class which can connect with other instances.

Advantages Libraries written within an established programming language are typically more reliable than domain-specific languages (DSLs) since they often benefit from larger user and support bases, contributing to their stability and robustness. Furthermore, programming language implementations offer the advantage of integrating the standard model into a broader system, making it compatible with automation pipelines and other specification tools, thereby enhancing its versatility and utility within various contexts.

Disadvantages Libraries can sometimes appear verbose compared to custom, domain-specific languages because the underlying general-purpose language must support a broader range of features that may not be essential to the specific domain. For instance, a language designed to specify the structure of genetic designs does not require capabilities for arithmetic operations. To utilise these libraries, individuals need to have a working knowledge of the programming language and be able to execute scripts, along with any additional processes inherent to the language. For example, Figure 4 illustrates the NOR gate described in all figures in this review. Despite its small and straightforward design, the associated file size can be relatively large, highlighting the potential for increased complexity when using these libraries within a broader programming context.

```
Property Name(txt);
Property Represses(txt);
Property InducedBy(txt);
Property PromoterType(txt);
Property Pigeon(txt);
PartType SmallMolecule(Name);
PartType Promoter(Name, Pigeon);
PartType RBS (Name, Pigeon);
PartType Repressor(Name, Pigeon);
PartType CDS(Name, Pigeon);
PartType Terminator(Name, Pigeon);

Promoter p1(.Name("p1"),.Pigeon("p p1 1"));
Promoter p2(.Name("p2"),.Pigeon("p p2 2"));
Promoter p3(.Name("p3"),.Pigeon("p p3 3"));
Promoter p4(.Name("p4"),.Pigeon("p p2 4"));
RBS r1("r1","r r1 5");
RBS r2("r2","r r2 6");
RBS r3("r3","r r3 7");
CDS c1("c1","c c1 8");
CDS c2("c2","c c2 9");
CDS c3(.Name("c3"), .Pigeon("c c3 10"));
Terminator t1("t1", "t t1 11");
Terminator t2(.Name("t2"), .Pigeon("t t2 12"));
Terminator t3("t3","t t3 13");

c1 REPRESSES p3;
c2 REPRESSES p4;

Device RepressingDevice1(Promoter, Promoter, RBS, Repressor, Terminator);
Device RepressingDevice2(Promoter, Promoter, RBS, Repressor, Terminator);
Device ReportingDevice1(Promoter, RBS, CDS, Terminator);
Device NorGate(RepressingDevice1, RepressingDevice2, ReportingDevice1);


Rule r(
    ON NorGate:
        p1 BEFORE p3
        p2 BEFORE p3
        p1 BEFORE p4
        p2 BEFORE p4
);

Device[] lst = product(NorGate, strict);
for(num i=0; i<lst.size; i++) {
    println(lst[i]);
}
pigeon(lst);
```

Figure 3: NOR gate[19] implementation within Eugene programming language. The **Property** keyword defines objects linked to defined PartTypes. **PartType** defines classes (Promoter, RBS, CDS and Terminator). Interactions such as "REPRESSES" can be defined between PartTypes. **DEVICES** are defined by providing constraints such as composition (What the structure of the device should be). Rule sets can be defined further by providing functional and structural constraints. In this case, relative location requirements of component parts. Finally, a set of Device objects are created, specific variations of the design that adhere to all constraints.

```
p1 = sbol3.Component('p1', sbol3.SBO_DNA,roles=[sbol3.SO_PROMOTER])
p2 = sbol3.Component('p2', sbol3.SBO_DNA,roles=[sbol3.SO_PROMOTER])
p3 = sbol3.Component('p3', sbol3.SBO_DNA,roles=[sbol3.SO_PROMOTER])
p4 = sbol3.Component('p4', sbol3.SBO_DNA,roles=[sbol3.SO_PROMOTER])

rbs1 = sbol3.Component('rbs1', sbol3.SBO_DNA,roles = [sbol3.SO_RBS])
rbs2 = sbol3.Component('rbs2', sbol3.SBO_DNA,roles = [sbol3.SO_RBS])
rbs3 = sbol3.Component('rbs3', sbol3.SBO_DNA,roles = [sbol3.SO_RBS])

cds1 = sbol3.Component('cds1', sbol3.SBO_DNA,roles = [sbol3.SO_CDS])
cds2 = sbol3.Component('cds2', sbol3.SBO_DNA,roles = [sbol3.SO_CDS])
cds3 = sbol3.Component('cds3', sbol3.SBO_DNA,roles = [sbol3.SO_CDS])

term1 = sbol3.Component('term1', sbol3.SBO_DNA,roles = [sbol3.SO_TERMINATOR])
term2 = sbol3.Component('term2', sbol3.SBO_DNA,roles = [sbol3.SO_TERMINATOR])
term3 = sbol3.Component('term3', sbol3.SBO_DNA,roles = [sbol3.SO_TERMINATOR])

circuit = sbol3.Component('NOR_GATE', sbol3.SBO_DNA)
circuit.roles.append(sbol3.SO_ENGINEERED_REGION)
p1 = sbol3.SubComponent(p1)
p2 = sbol3.SubComponent(p2)
p3 = sbol3.SubComponent(p3)
p4 = sbol3.SubComponent(p4)

rbs1 = sbol3.SubComponent(rbs1)
rbs2 = sbol3.SubComponent(rbs2)
rbs3 = sbol3.SubComponent(rbs3)

cds1 = sbol3.SubComponent(cds1)
cds2 = sbol3.SubComponent(cds2)
cds3 = sbol3.SubComponent(cds3)

term1 = sbol3.SubComponent(term1)
term2 = sbol3.SubComponent(term2)
term3 = sbol3.SubComponent(term3)

circuit.features = [p1, p2, p3, p4, rbs1, rbs2,
                    rbs3, cds1, cds2, cds3, term1, term2, term3]

circuit.constraints = [sbol3.Constraint(sbol3.SBOL_PRECEDES, p1, rbs1),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, rbs1, cds1),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, cds1, term1),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, term1, p2),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, p2, rbs2),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, rbs2, cds2),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, cds2, term2),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, term2, p3),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, p3, p4),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, p4, rbs3),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, rbs3, cds3),
                       sbol3.Constraint(sbol3.SBOL_PRECEDES, cds3, term3)]

part1 = sbol3.Participation(sbol3.SBO_INHIBITOR, cds1)
part2 = sbol3.Participation(sbol3.SBO_INHIBITOR, cds2)
part3 = sbol3.Participation(sbol3.SBO_INHIBITOR, p3)
part4 = sbol3.Participation(sbol3.SBO_INHIBITOR, p4)
r1 = sbol3.Interaction(sbol3.SBO_INHIBITION, [part1, part3])
r2 = sbol3.Interaction(sbol3.SBOL_INHIBITION, [part2, part4])
circuit.interactions = [r1, r2]

doc = sbol3.Document()
doc.add(circuit)
doc.write('nor.nt', sbol3.SORTED_NTRIPLES)
```

Figure 4: Abstract NOR gate[19] described using pySBOL. Components refer to entities within the design (genetic parts here). SubComponents are structural instances of the Components added as features on the larger NOR circuit. Furthermore, constraints are defined by the relative orders of the parts of the structure. Two interactions are defined with the participating entities. Finally, the document (design) is created and written into a file.

1.2 Aims and Objectives

As discussed, there are straightforward mechanisms for defining genetic designs, such as SBOL visual, more detailed methods like Eugene and complete programmatic representations like pySBOL. However, a mechanism which bridges the gap between manual visual editing tools, complex DSLs and verbose programming language libraries that can resolve to a high-quality data standard such as SBOL is missing. The inspiration for this research is from programming languages within electronics that exist for one reason: to reduce complexity. Figure 5 shows that programming languages abstract machine code (the instructions given to the hardware), providing a more accessible interface that focuses on the algorithms and applications instead of the underlying process. The process within computing to abstract detail for a higher-level user is directly applicable here and critical to enabling synthetic biology standards.

Aims

This chapter aims to explore how existing DSLs are implemented. Specifically, how DSLs provide a domain-specific interface without a complete language. Next, explore how programming languages abstract technical complexity and provide user-friendly interfaces specifically for the natural language used within synthetic biology, including any common terminology that may be used. Finally, examine how object-oriented programming languages implement classes and inheritance to reduce complexity and promote reuse.

Objectives

The outcomes from this research can be broken down into the language and the tooling. The language outcome is to develop an abstraction language that captures arbitrary datatypes, enabling any synthetic biology design to be defined. Also, a direct mapping to the SBOL data model ensures all designs are SBOL compliant and, therefore, interoperable. The language must be easily extensible to increase abstraction and promote reuse and must be simple by removing many usual programmatic norms with more natural language features. For the tooling, the outcome is to develop a tool to parse the language and generate SBOL complaint RDF graphs. Furthermore, it must validate the generated graphs for SBOL compliance. The tools must remove technical requirements to provide access to non-technical users and a user experience that makes the underlying language accessible. The tooling must be robust so users can learn and use the language without issue.

2 Results

As discussed, there is a need for a simple and extensible language backed by a standard such as SBOL, which sits in between abstract specification methods and complex written methods. Furthermore, the language must bridge the gap between the manufactured language of a standard and the natural language used within synthetic biology and related communities.

ShortBOL is a human-readable/writable shorthand for describing biological designs in SBOL. This language allows SBOL data to be generated easily and quickly from simple textual descriptions sharing many design aims and characteristics with existing languages. The fundamental abil-

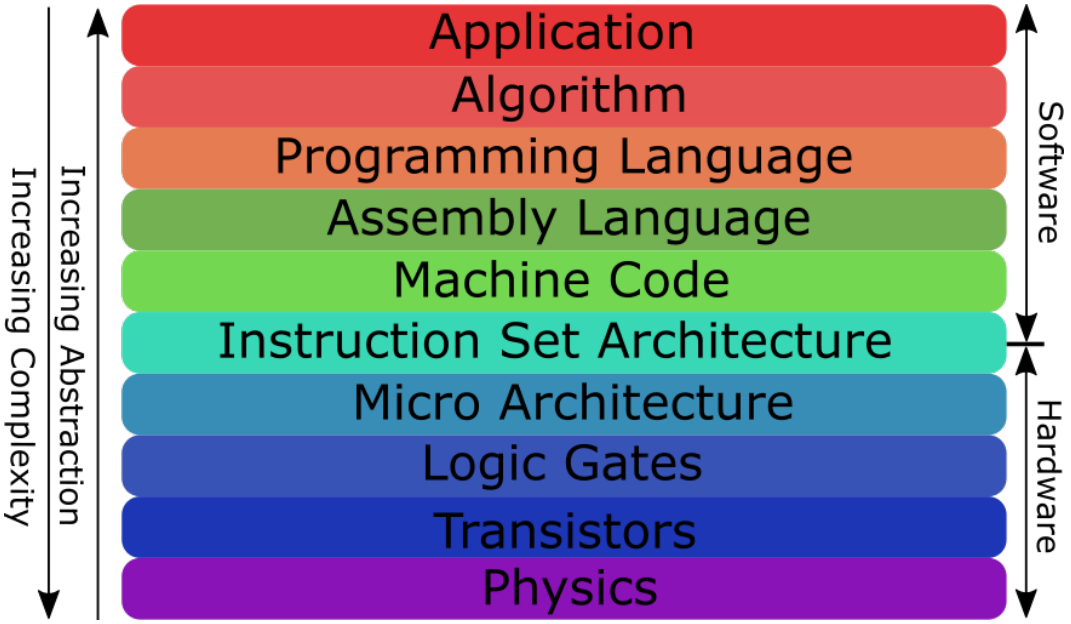

Figure 5: Abstraction layer within electronic, instruction and programming languages.

ity of ShortBOL is to define a hierarchy of abstraction within the language, which provides a user with the choice of high levels of control over the underlying model or to abstract the model entirely, thereby simplifying the process considerably.

This language is for those who wish to rapidly sketch synthetic biology designs using a simple, text-based scripting language instead of writing code that utilises the SBOL libraries. However, being an abstraction of SBOL data, ShortBOL inherits the richness of the SBOL data model and the ability to encapsulate design information of unique importance to synthetic biological constructs. Moreover, the ability to describe arbitrary RDF data in ShortBOL provides flexibility and extensibility to produce greater abstraction, modularity, and concision. ShortBOL is designed to be easy for synthetic biologists with minimal amounts of software development knowledge. Those with software development training can also find ShortBOL useful as a rapid method of producing SBOL more simply than by writing code that uses the SBOL libraries.

The core of the language is data structures similar to classes within traditional object-orientated programming languages called templates which encapsulate data within a single object, which is initially directly to the SBOL data model but eventually towards even more abstract representations focused more towards natural language within the domain. In detail, we will discuss the template system, including the more abstract composite templates. It must be noted, however, that defining these templates is usually not a task for the average user. Instead, these templates will likely be defined within a template library (see template libraries).

2.1 Templates

The language of ShortBOL is built around templates, constructs denoting SBOL classes, or groups of SBOL classes (see composite template) to provide abstraction and reduce complexity. Tem-

plates are language constructs conceptually similar to classes within programming languages, a blueprint that defines the properties that objects of that class will possess. Without templates, a document describing a design would be required to define each RDF triple explicitly. Figure 6 displays four templates (TopLevel, Component, DNA and Promoter) which define SBOL constructs at different levels of specificity. The templates within the language have several features which enable the customisation and specialisation of templates.

Inheritance

Inheritance is a mechanism to derive a template from another template for a hierarchy of templates that share properties. Inheritance allows new templates to extend existing templates, providing consistency between the underlying data and the reuse of existing templates. Figure 6 displays the inheritance between all four templates where the Promoter template inherits from DNA from Component and, in turn, from TopLevel. For example, Figures 6 1 & 2 display how the "Component" template defines the "TopLevel" template within its body. This statement tells the interpreter that the "Component" template inherits all of the properties of the "TopLevel" template. Templates are critical within ShortBOL to provide abstraction and reduce complexity. Without templates, a document describing a design would be required to define each RDF triple explicitly. However, these features allow custom templates to be defined. However, they must be expanded into RDF triples, constituting an SBOL complaint document (section expanding templates display how templates are instantiated and converted into graphs).

Aliases

Entities within an RDF graph must be resources that refer to web entities called Uniform Resource Identifiers (URIs) and are essential for the interoperability and connectedness of a standard format, namely SBOL. However, URIs can appear complex and verbose to those unfamiliar with the RDF methodology. Therefore, requiring a user to define full URIs within templates (and later instances of designs) contrasts with the motivations of SBOL. Aliases that map URIs to human-readable names can be defined to address this. Two examples shown in Figure 6 2 establish a namespace alias called "sbol_3" and a suffix alias "<type>". Using the sbol3 alias maps to the prefix `http://sbols.org/v3#`. It adds the "<type>" suffix to make a complete URI of `http://sbols.org/v3#type`, which is a standard predicate URI defined within the SBOL ontology to define entity types (DNA or protein, for example). Figure 6 4 also defines another alias as "i_promoter", which maps directly to the URI: `http://identifiers.org/so/SO:0000167`, the sequence ontology definition of a promoter.

Parameters

Arguments can be provided by either the user, an inheriting template or within a composite template (see composite templates) to provide a mechanism to customise a template further. Figure 6 1 & 2 requires a parameter named "type", which can be used to specify the RDF type of this template. In this case, this parameter will likely be fulfilled by another template which extends this template as opposed to a user providing a value within the design document.

Assignment

An assignment is a statement or operation that assigns a value to a variable or a data structure. It is a fundamental concept in programming and allows the storage and manipulation of data during the execution of a program. Within the context of the template system, ShortBOL uses the "=" operator and allows properties to be set on a template. Assignment operations connect the predicates to the objects within an RDF triple. For example, figure 6 2 assigns the parameter "type" value to the "<type>" alias of the template "Component". Also, figure 6 4 assigns the role of a template as a promoter.

2.2 Template Instantiation

So far, ShortBOL has been described as templates which can be defined to encapsulate entities. However, the templates alone do not define a design document. Templates within ShortBOL act as classes which can be instantiated to create instances of that template. Many instances of the same template can be (and most likely will) defined within a design. For example, multiple Promoters may be used within a single design. Therefore a mechanism is needed to create instances of classes and the resultant RDF triples. The primary operator within ShortBOL is the "is a", which defines an instance of a template. For example, within Figure 7, "BBa_B0034 is a RBS" introduces a new identifier, "BBa_B0034", whose properties will be set according to the pattern described by the RBS template. Instance definitions can also contain a block of ShortBOL expressions known as the body, which are the same bodies as within template definitions. These are used to declare additional properties and their values. For example, within figure 7, the template named "BBa_B0034" declares an additional property specifying the primary sequence. One or more arguments can also parametrise templates. For example, the DNASequence template expects a single argument containing a DNA string. This mechanism allows typical design and composition patterns to be easily captured within templates without requiring a complete programming language.

Prefix

RDF resources that refer to web entities are Uniform Resource Identifiers (URIs). However, as Figure 7 shows, subjects within a ShortBOL design document do not contain the full URI. For example, BBa_R0040 alone does not reference a resource but implicitly references <https://synbiohub.org/public/igem/BBa_R0040/1. However, much like the aliases explain in the templates section, for a user to input the full URI for each template produces more difficult-to-comprehend documents. Therefore, the default prefix can be set using the syntax "@prefix <prefix_uri>" and is appended to all template names within a document. For example, "@prefix <https://synbiohub.org/public/igem/>" will prefix all subjects with that partial URI, thus removing the requirement for all namespaces to be typed.

However, different template subjects may have different prefixes. For example, a template may reference a resource from <https://synbiohub.org/public/igem/ and another from <http://www.virtualparts.org/terms#. Therefore, named prefixes must be enabled, which allows explicit definitions on a template name. The syntax for a named prefix is: "@prefix prefix_name ≤prefix_uri>". For example "@prefix sbh ≤https://synbiohub.org/public/igem/>". would make templates using

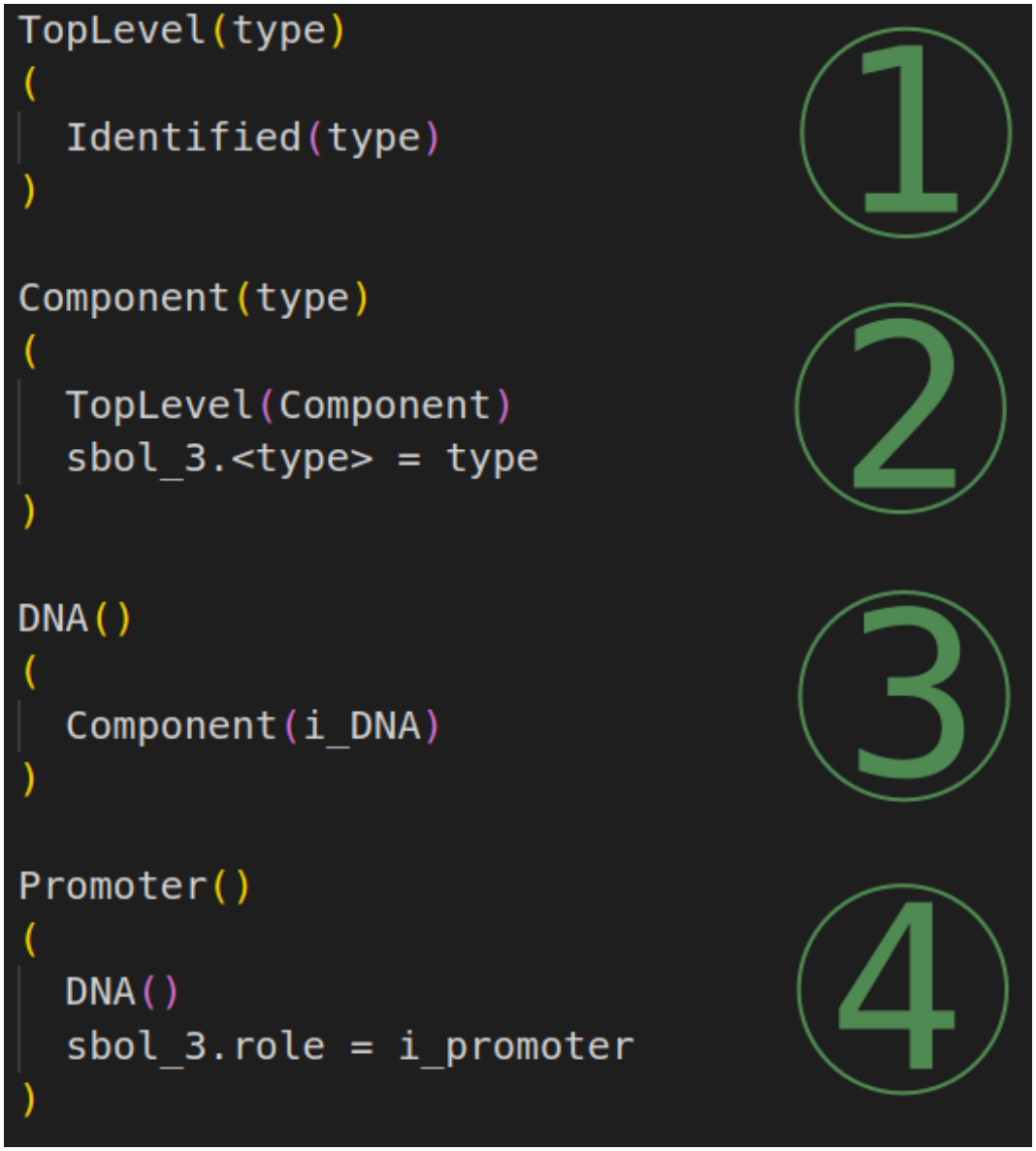

Figure 6: Template definition hierarchy from general to specialised. **1** - The TopLevel class is an abstract template representing all classes relating to top-level objects of an SBOL document, and it inherits from the Identified template, which is not displayed here but is the based template for all templates. **2** - The SBOL component class requires a type parameter (within brackets to distinguish it from the parameter name). It inherits from the TopLevel class (1). **3** - The DNA class is an abstraction of the Component (2) providing the type (DNA). **4** - A specialised DNA (3) class which sets the optional role property to a promoter.

```
BBa_B0034_seq is a DNASequence("aaagaggagaaa")
BBa_B0034 is a RBS()
(
  sequence = BBa_B0034_seq
)
```

Figure 7: A single part (BBa_B0034) defined within ShortBOL. It consists of the instantiation of a sequence template where the subject is assigned to the body of the RBS template using the sequence alias predicate.

this prefix follow the syntax"sbh.BBa_R0040 is a RBS()", which enables parts to be used from different sources.

Extensions

ShortBOL, the language, cannot perform computation. It provides a mechanism to capture and the potential to parse user-defined templates and generate general RDF graphs. Therefore it cannot manipulate or comprehend the underlying logic of the defined document. For example, it does not guarantee that the RDF graphs generated are SBOL, and a user could create a template library that is not SBOL compliant (either invalid entities or connections). The final graph would still be generated. An extended feature of ShortBOL not displayed here is extensions (see methods for a deeper discussion of extensions). Extensions provide a way to execute Python code to do additional, more complex processing of the RDF triples generated by expanding templates. The syntax for calling extensions is: " @extension <extension_names>()". An extension used within any ShortBOL document is "sbol3". In this extension, all SBOL objects in the final graph have SBOL-compliant URIs, and if not, they attempt to modify the triples such that they are. The "sbol3" extension implements specific data model knowledge during validation. Therefore, by decoupling all data model knowledge into a single extension, the language and underlying tooling do not need to be refactored in case of changes to the data model.

2.3 Composite Templates

An issue with the standard template system is a one-to-one mapping between templates and the entities they define. Therefore, each object must be explicitly invoked within the design document. Without being able to cluster multiple objects within a single template limits the potential to increase abstraction. For example, suppose a design defines an interaction within the SBOL data model. In that case, the creator must define several objects to ensure SBOL compliance displayed within Figure 8. For an Interaction with two reactants, two Participations and two Subcomponents must be defined from two initial physical entities. While these objects serve a purpose, they can often be inferred from other design aspects. For example, within SBOL, if a repression object is defined, it can be assumed that there will be a repressor entity and a repressed entity

without a user being required to define the participants. These inferences cannot be encoded using the standard template system within ShortBOL; therefore, a new template type is introduced. Composite templates are an advanced extension of the template system where template instances

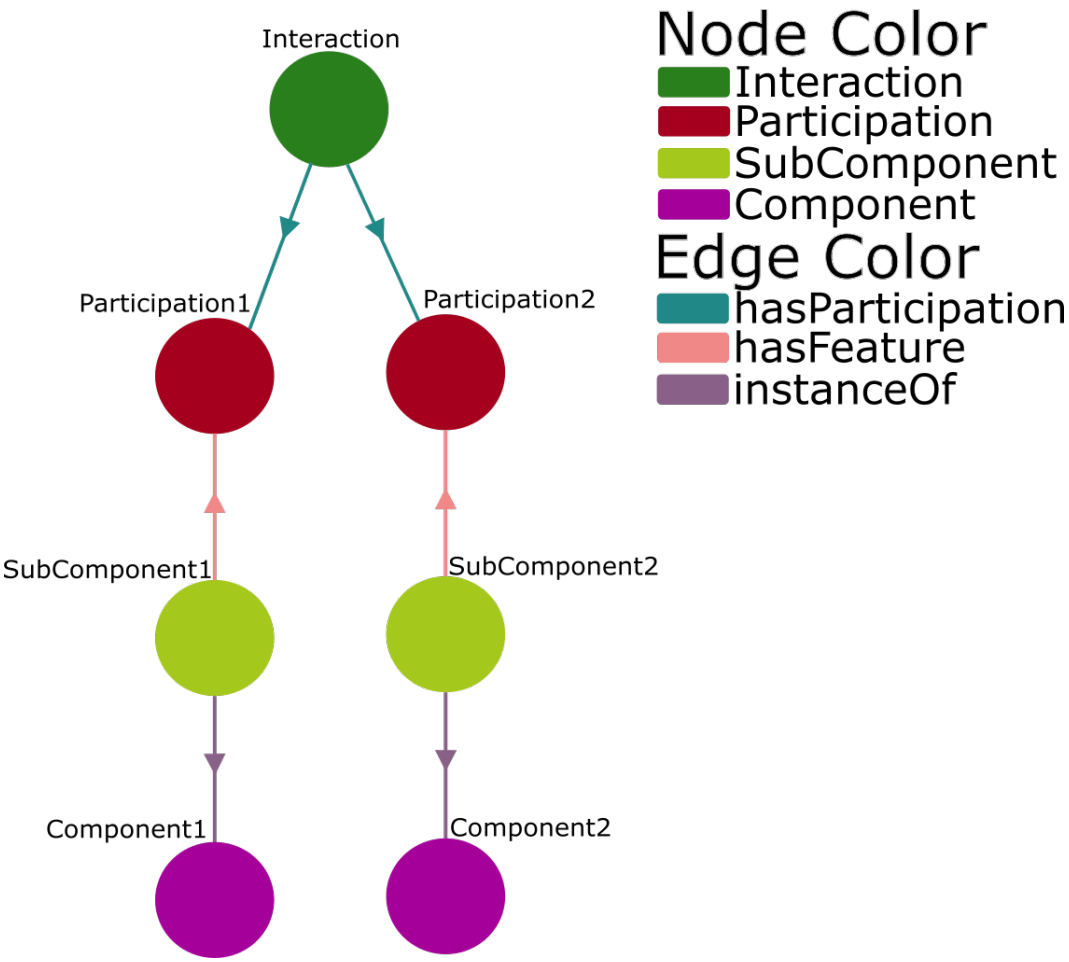

Figure 8: Graph representation of the requirements to define a two reactant interaction within the SBOL3 data model. The Interaction requires two participation instances, each requiring a SubComponent with definitions of Components.

can be defined within other templates and assigned. Composites aim to increase abstraction by removing SBOL-specific constructs and the required writing on the user side.

In Figure 9 1, the standard templates (Interaction, Feature, SubComponent and Participation) are initially defined in line with the SBOL data model. Next, figure 9 2 represents a composite template (hasInteraction) which appears much more complex within the definition. This template takes five parameters, the components, participation types and interaction type. Initially, the template instantiates two SubComponents. Internally instantiated templates do not have user-provided names and must be created automatically. The subject here is "self.part1._sc", which concatenates three values to create a URI as a name.

- "self" is a keyword used to refer to the parent object within the design. The parent is the subject name of the template instances within the design document.
- "part1" refers to the parameter given to the template.
- "_sc" is a literal string value which ensures the name is unique.

Next, the SubComponent instances are assigned as properties to the parent. The parent refers to the template instance within the design document. Next, an Interaction template is instantiated (and all intermediate templates) inside the hasInteraction template (figure 9 3). Again, the name of the Interaction and Participation templates must be calculated based on the inputs, ensuring a unique name is generated. However, when the instances of the Participation objects are assigned, they are set to the new Interaction instance instead of the parent. It should be noted that this composite template is unlikely to be instantiated by the user. Instead, it would be inherited by a specialised template, as in figure 9 4. This template is an example of a specialised version of the composite template, which defines a repression interaction by providing the participation and interaction types using aliased URIs. It must be noted that templates cannot accept an arbitrary number of arguments. Therefore a base composite template must be written for each number of participants within an interaction. For example, this template cannot define an interaction with three participants.

2.4 Template libraries

As discussed, templates are structures defining an object within a given domain. However, users of ShortBOL are not required to define their templates for every design. Instead, template libraries exist which define multiple templates for use.

Currently, ShortBOL contains two libraries (Developer and User) which define genetic parts, interactions, proteins and other non-genetic materials at two different levels of abstraction. These libraries differ by using composite templates in the case of the "User" library.

Figure 10 displays the usage of both libraries when defining the same design, a simple system describing the lacI & pLlac regulatory mechanism. The "User" templates provide a much more streamlined definition of a genetic design, removing the requirements to explicitly define the verbose methods of describing interactions within SBOL. However, the "Developer" mode does provide some benefits that being finer control. For example, take an interaction which contains an arbitrary number of participants. With the "User" mode, a template would be required for every potential number of participants. However, with the former, a single template can account for any number of participants. Furthermore, the composite templates require context (whom the internal templates connect to) and, therefore, must be instantiated inside a parent's body, as seen within Figure 10, where the "inhibition" composite template is instantiated inside the parent FunctionalEntity. Therefore, while the more abstract "User" mode is the most applicable in most circumstances, in exceptional cases, the "Developer" templates should be used. One template library does not need to be used exclusively within a document. Instead, both can be used simultaneously. The ability to mix and match libraries is a crucial feature of the underlying language because it allows the user to choose what level of detail they need to encode a piece of information. (see discussion for an extension of this point).

```
Interaction(type)
(
  Identified(Interaction)
  sbol_3.<type> = type
)

Feature(type)
(
  Identified(type)
)

SubComponent(component)
(
  Feature(SubComponent)
  instanceOf = component
)

Participation(subComponent, role)
(
  Identified(Participation)
  sbol_3.<role> = role
  participant = subComponent
)

hasInteraction(part1, part_1_type, part2, part_2_type, type)
(
  self.part1._sc is a SubComponent(component)
  self.part2._sc is a SubComponent(component)
  hasFeature = self.part1._sc
  hasFeature = self.part2._sc
  self.part1._.type._with_.part2 is a Interaction(type)
  (
    self._.part1._.part_1_type is a Participation(part1, part_1_type)
    self._.part2._.part_2_type is a Participation(part2, part_2_type)
    hasParticipation = self._.part1._.part_1_type
    hasParticipation = self._.part2._.part_2_type
  )
  hasInteraction = self.part1._.type._with_.part2
)

inhibition(inhibitor, inhibited)
(
  hasInteraction(inhibitor, i_inhibitor, inhibited, i_inhibited, i_inhibition)
)
```


Figure 9: A more complex instance of specialised templates. **1** - Interaction, Feature, SubComponent and Participation are standard specialised templates inheriting from a general template and assigning variables. **2** - A composite template that performs expansions and URI assignments within the template. **3** - The participations are created and assigned to the Interaction instead of the parent template. **4** - A specialised template inherits from the hasInteraction (3) template.

A

```
pLlac is a Promoter()
lacI_p is a Protein()

pLlac_sc is a SubComponent(pLlac)
lacI_p_sc is a SubComponent(lacI_p)

lacI_inhibitor is a Inhibitor(lacI_p_sc)
pLlac_inhibited is a Inhibited(pLlac_sc)
pLlac_inhibition is a Inhibition()
(
  hasParticipation = lacI_inhibitor
  hasParticipation = pLlac_inhibited
)

LacI_inhibition_module is a FunctionalEntity()
(
  hasFeature = pLlac_sc
  hasFeature = lacI_p_sc

  hasInteraction = pLlac_inhibition
)
```

B

```
pLlac is a Promoter()
lacI_p is a Protein()

LacI_inhibition_module is a FunctionalEntity()
(
  inhibition(lacI_p,pLlac)
)
```

Figure 10: A comparison of the A: “Developer mode” and the B: “User mode” defining the same protein-promoter inhibition. The composite (inhibition) within the “User mode” is defined within another template as the context is required.

Import

Template libraries and existing ShortBOL objects do not need to be within the same document as the design. Import statements can import external documents using the syntax: "use <document_name>". For example, "use <sbol3>" will import all SBOL3 templates and aliases defined within the library available for use. Furthermore, multiple imports can be used within a document enabling templates from different domains to be used within the same design (see future work). Furthermore, importing template libraries enables reuse and standard language interfaces instead of each user defining a set of local templates.

2.5 The Document

So far, we have discussed templates, composite templates, instantiation of templates and other language features. However, it is only when all of these are combined that a ShortBOL document is created. Figure 11 displays the NOR gate built by Nielsen and colleagues[19] as a ShortBOL design document, which was used when discussing previous specification methods.

The statements contained in shorthand documents are interpreted sequentially, and an RDF graph is generated from each template statement. The union of these graphs is then serialised as RDF/XML to produce a valid SBOL document. Here we will briefly discuss the statements sequentially from figure 11.

- **Import statements**: Import URIs are resolved to ShortBOL documents. For example, figure 11 imports the SBOL3 template library.
- **Prefix statements**: Prefixes identifiers map to their aliases (None if default prefix) available for use within the design document. Figure 11 displays the default prefix referring to `https://synbiohub.org/public/igem/` and a named prefix (lcp) which aliases the URI `https://synbiohub.programmingbiology.org/public/`.
- **Template instantiation**: Templates are associated with their subject identifier. If the name of a template application matches a registered template, this template will be expanded (see template expansion). Figure 11 displays many templates and composite templates. Some templates have bodies which contain extra information such as sequence information, descriptions or connections to other template subjects in the case of the NOR module, which defines structural and functional relationships between its constituents.
- **Extension statements**: The extension name is looked up, and the appropriate Python function is found and executed when the graph is generated. Figure 11 displays the sbol3 extension, which ensures the final graph (providing the ShortBOL document is valid) is SBOL compliant.

While all extra language features (import, prefix and extensions) can be used to customise further, a ShortBOL document, like in Figure 11, is unlikely to be the document a user would see if they were using the tooling developed alongside the language(see methods for a discussion of the tooling around the language). The import, prefix and extension statements do not need to be explicitly defined, as they would be inferred within the tool and inserted automatically into the

document before it is parsed. These insertions aim to reduce the user's requirements to interface with them because they introduce unneeded complexity.

2.6 Template expansion

So far, we have discussed how templates are defined and instantiated. While a user can create and develop ShortBOL documents using existing templates or create new ones without knowledge of the underlying processes, we will discuss how ShortBOL templates are compiled to RDF triples. Figure 12 will be referenced during the explanation as an outline of how this process occurs where templates are expanded and substituted when a design document (figure 12) references these templates, and then RDF triples are created. The generation of the RDF graph can be broken down into two stages.

Template table creation

Before any design documents are considered, the first stage is to generate the template table from the imported templates or locally defined templates. A template table maps each template type defined in the template library or the design document to the RDF graph for that template. Template tables enable substitution within the design document where instantiations are looked up in this table. The first stage is to generate to gather all properties of a template. For example, figure 12 1 displays the inheritance feature of the ShortBOL language. When a template inherits from another, all properties are substituted inside the template's body. As seen within Figure 12 2, all properties from the Promoter template and all parent classes are inserted inside the body. Furthermore, the keyword "self" is used. Within this context, "self" can be considered a Blank node, a substitute until an instance of this template is created.

Finally, the template table is generated by adding each template type as the keys and the RDF graph as the object. The result of a single table entry is represented in Figure 12 3. The RDF graph is generated by resolving each property within the template's body, where each property corresponds to a triple. For example, "sbol_3.role = i_promoter" resolves to the triple "self" - `http://sbols.org/v3#role` - `http://identifiers.org/SO:0000167` where aliases (sbol3, role and i_promoter) map to full or partial URI's.

Design Generation

The final design RDF graph can be generated by substituting template types with the corresponding triples and generating triples from properties. Figure 12 4 + 5 displays how instances of templates can be converted into RDF. The first step is using the type of template (in this case, Promoter) and searching the template table for the template definition. If a match is made, the RDF graph value is taken, and the Blank node is substituted with the template instance subject name. Secondly, the custom properties defined within the body of the template instance are translated into triples.

```
@use <sbol3>
@prefix <https://synbiohub.org/public/igem/>
@prefix lcp = <https://synbiohub.programmingbiology.org/public/>

lcp.arabinose is a SmallMolecule()
lcp.ATC is a SmallMolecule()
lcp.TetR is a Protein()
lcp.CI_p is a Protein()
lcp.AraC is a Protein()

BBa_R0040 is a Promoter()
(
  hasDNASequence("tccctatcagtgatagagattgacatccctatcagtgatagagatactgagcac")
)

BBa_I0500 is a Promoter()
(
  description = "pBad promoter"
)
BBa_B0034 is a RBS()
BBa_C0051 is a CDS()
BBa_B0010 is a Terminator()
BBa_J23117 is a Promoter()
BBa_J61100 is a RBS()
BBa_K1088005 is a CDS()
BBa_J61101 is a RBS()
BBa_C0040 is a CDS()
BBa_j61102 is a RBS()
BBa_K592101 is a CDS()

NOR_gate is a FunctionalEntity()
(
  precedes(BBa_R0040, BBa_I0500)
  precedes(BBa_I0500, BBa_B0034)
  precedes(BBa_B0034, BBa_C0051)
  precedes(BBa_C0051, BBa_B0010)
  precedes(BBa_B0010, BBa_J23117)
  precedes(BBa_J23117, BBa_J61100)
  precedes(BBa_J61100, BBa_K1088005)
  precedes(BBa_K1088005, BBa_J61101)
  precedes(BBa_J61101, BBa_C0040)
  precedes(BBa_C0040,BBa_j61102)
  precedes(BBa_j61102,BBa_K592101)
  precedes(BBa_K592101,BBa_B0010)

  inhibition(ATC,TetR)
  inhibition(arabinose,araC)
  inhibition(TetR,BBa_R0040)
  inhibition(AraC,BBa_I0500)
  inhibition(CI_p,BBa_J23117)
  geneticProduction(BBa_C0051,CI_p)
  geneticProduction(BBa_K1088005,araC)
  geneticProduction(BBa_C0040,TetR)
)

@extension <sbol3>()
```

Figure 11: NOR gate described within ShortBOL. From top to bottom: the document imports the SBOL3 template library and defines the default prefix, a named prefix, molecules and proteins using resources from a named prefix namespace. Next, the genetic parts are described; the subjects do not need to be prefixed with a namespace as they come from the default prefix namespace. Within some genetic parts, extra information is defined, namely DNA sequences and descriptions. The NOR_gate module is defined, which encapsulates the design. It first specifies the parts' relative positions using the precedes composite template and then defines several interactions using multiple specialised interaction composite templates. Finally, the SBOL3 extension is defined, which ensures the design document is SBOL compliant.

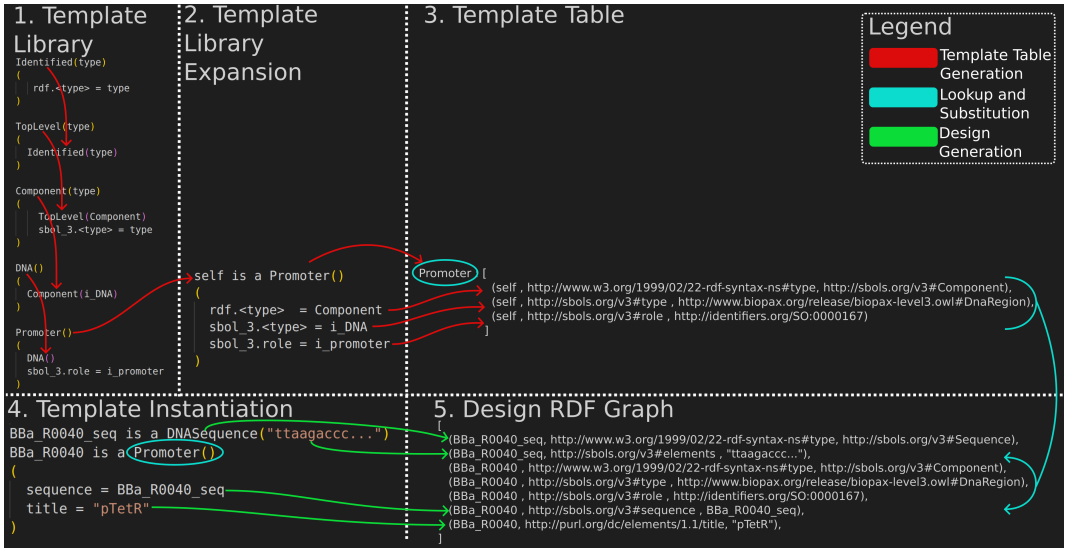

Figure 12: The overview of RDF graph generation from ShortBOL templates. **1** - Template inheritance, the Promoter template inherits from DNA which inherits from Component and the rest, taking the properties defined in the hierarchy. **2** - The self keyword is inserted as a placeholder for the Promoter's name, which is unknown at this point. Also, all properties (RDF type, SBOL type and SBOL role) are inserted into the body of the promoter template. Note: the property names and values are aliases for URIs. For example, i_promoter is an alias for the URI `http://identifiers.org/SO:0000167`. **3** - A template table is a dictionary with keys as the Promoter type, and values are the triples constituting a Promoter as defined within the template. The subject uses self as a placeholder. **4** - The design document defining a sequence and promoter template. **5** - The final RDF graph where all templates have been expanded, and substitutions made.

3 Methods

When designing the ShortBOL language, a software package and web application were also developed to complement the language. The software package can be used from the command line (`https://github.com/intbio-ncl/shortbol`) and primarily contains the compiler to produce SBOL from ShortBOL documents. However, it also includes documentation to compile ShortBOL text files to SBOL XML files. Furthermore, a web application (`http://shortbol.org/`) was developed to reduce the barrier to entry that a manual installation presents and provides several utilities. Here we will discuss the tool's features that are not inherent to the language discussed.

3.1 Editor

Like all DSLs and programming languages, Integrated development environments (IDEs) provide several utilities, namely, writing assistance[118]. IDEs are designed to improve productivity by reducing the time required to focus on trivial tasks and increasing comprehension of the written document. The ShortBOL web application editor provides two utilities to assist with writing ShortBOL, autocompletion and syntax highlighting. Firstly, the autocompletion looks up potential values from the current document (the text inside the editor) and the template libraries that have been imported. This feature provides the user quick access to existing templates and instances of templates and property names. Secondly, syntax highlighting will colour text depending on what it represents, for example, templates, connections, comments and literals, which increases comprehension, especially on large designs. Figure 13 displays the editor, containing two templates with comments where each type of text is coloured differently.

3.2 Validation

Two cases may occur that causes errors within the ShortBOL application:

- If a user makes a syntactic mistake in the code, i.e. an incorrect argument or missing brace, the language parser will be unable to deconstruct templates. ShortBOL can identify syntactical errors within the code during runtime and return user-friendly errors when the language parser detects an error, such as line numbers, positions and the type of error.

- If the document is syntactically valid but, when compiled, is not SBOL compliant. For example, two entities are connected via an invalid predicate. When ShortBOL code is executed, the output is validated for compliance with the SBOL specification, ensuring ShortBOL output will interoperate with other SBOL tooling. The errors are checked relative to the SBOL best practices.

3.3 Extensions

As discussed, extensions are a feature of ShortBOL that provides a way to execute Python code to do additional, more complex processing of the RDF triples generated by expanding templates. The combinatorial derivation extension is an implemented example where the extension will expand a Combinatorial Derivation SBOL object and generate all possible construct variants. An

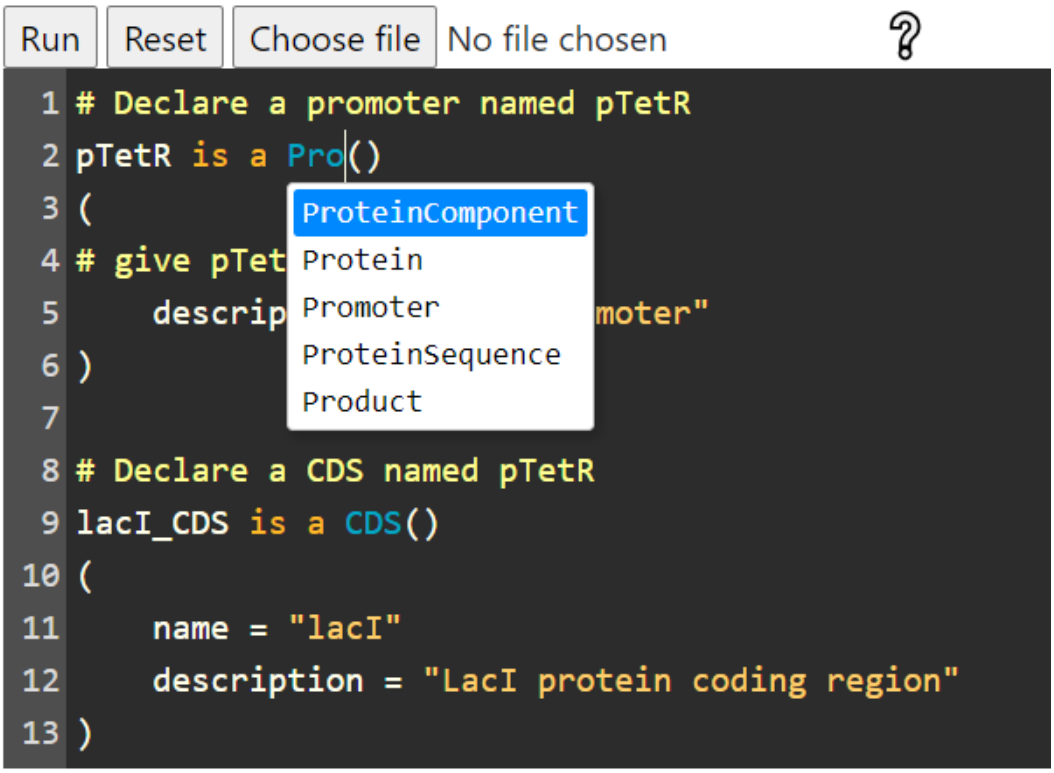

Figure 13: The ShortBOL web application editor. The editor enables the compilation of ShortBOL documents and contains autocompletion and syntax highlighting.

extension can be written to perform any modifications, analysis, or validation on the graph. The software package contains examples, each with an existing extension and its usage.

3.4 Converter

ShortBOL can be used to develop new genetic designs but has no inherent mechanism to edit existing designs captured within SBOL. Therefore, the ShortBOL software package contains a tool to generate ShortBOL from SBOL enabling a "round trip" between the two formats. In conjunction with SBOL, the converter can convert Genbank and FASTA files. However, as discussed, the resultant ShortBOL will likely be of lower quality using GenBank and FASTA files due to the inferior data formats.

Briefly explained, the converter produces the expanded template table, which contains the triples a template would generate when expanded. From this, a requirements hierarchy is formed where each template is positioned relative to its parent and derived templates. Figure 14 displays a sample of the template hierarchy where each node represents a template and the requirements for a template to be upgraded from its parent template. Each object within an SBOL graph (the existing design data) then traverses this hierarchy, comparing itself against the requirements and becoming the most specialised template possible. While a conversion can occur, the resultant ShortBOL will be of the lower abstraction "developer" code. "User" mode conversion is not possible because some structural data is lost when attempting to derive the abstract constructs. For example, if a composite template represents inhibition between two entities, one being the inhibitor and the other inhibited. In that case, which component performs which role without specialised data model knowledge is unclear.

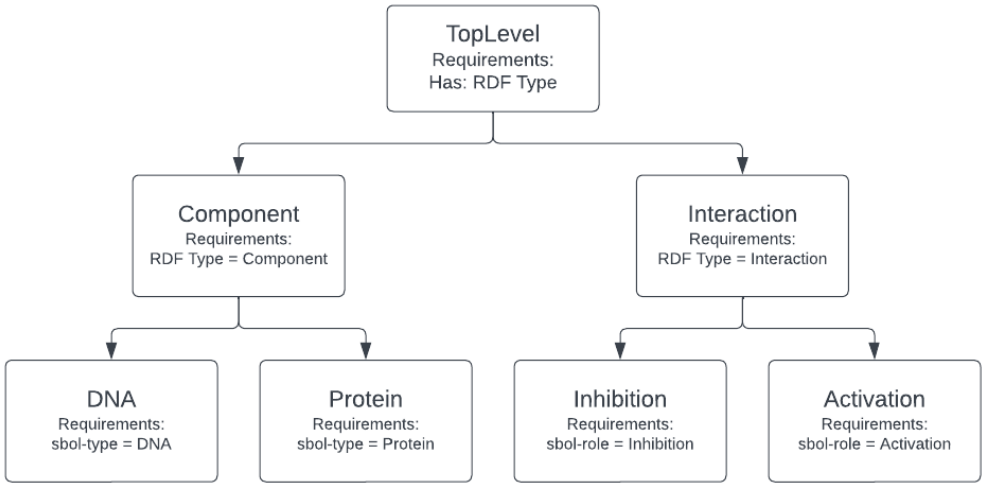

Figure 14: Example of a partial template hierarchy. Each node in the hierarchy contains requirements for a candidate to be valid as this template. If a pass occurs, the same comparison is made with children nodes.

3.5 Tutorial Series

A fundamental issue with any new language is the steep learning curve when adapting to a new paradigm. Tutorials and examples are excellent for providing templates to learn a language; they are practical examples that map the new language to the specific domain and can be used as a starting point for another project. Therefore, a tutorial describing how to use ShortBOL introduces features of the SBOL data model. This tutorial begins with the smallest possible examples and expands with increasing complexity. Secondly, an example series contains examples for each template within the library combined with complete design examples, such as a NOR gate implementation. Furthermore, tutorials and examples exist for both "Developer" and "User" template libraries.

3.6 Documentation

While tutorials provide specific examples of initial usage, they cannot be relied on to explain each aspect exhaustively. Therefore, along with the tutorials and examples, the ShortBOL web application contains exhaustive documentation on all templates within the underlying library. The documentation page is a self-generating document based on the documentation within the template library. Therefore it does not need to be updated if the ShortBOL libraries are expanded. Figure 15 displays the generating documentation page for both user and developer modes. The documentation contains names, parameters and descriptions of the templates. Furthermore, it describes the specialised template (inherited templates) and small usage examples.

3.7 Non-textual additions

Another resource for novice users is the ability to create instances of templates without typing using the forms system where each template within the library is displayed, allowing users to insert them. This system was implemented as a teaching mechanism to provide a custom example to users without being required to modify the existing examples. Figure 16 displays the system where each template class is offered and can then be customised relative to a user's requirements. Furthermore, specialised templates of the base template can be used. Figure 16 A displays all template base templates from the SBOL template libraries. When a template is chosen, figure 16 B displays the required and optional parameters and any specialised templates that inherit from this template.

3.8 Alternative representation

One advantage of compiling to a standard format is that other tools can also use this data. The ShortBOL web application also presents some visualisations of genetic designs discussed during the review. These additions are so that new users can see how changes to the design affect the final SBOL when presented in the formats they may be more familiar with. Here we briefly discuss these two alternative visualisation methods.

Component

The fundemental templates for creation of enitites in a biological design.

Usage: Use to represent parts in a design such as DNA or a Protein.

Parameters

Name	Possible Values	Description
type	- i_DNA - i_RNA - i_protein - i_smallMolecule - i_complex - i_functionalEntity	Category of the Component such as DNA or Protein.

Specialised Components: - DNAComponent - DNA - RNAComponent - RNA - ProteinComponent - Protein - SmallMoleculeComponent - SmallMolecule - ComplexComponent - Complex - FunctionalEntity - Promoter - RBS - CDS - Terminator - Operator - EngineeredGene - mRNA - CDS_RNA - sgRNA - Effector - TranscriptionFactor

Description:
These templates create a blueprint of a part, an instances of these are created using FunctionalComponents.

Usage Example:

```
        # Creates Four Components.
# Components can be thought as blueprints.
# It can be noted that the two RBS Components
# although written differently are functionally the identical.
pTetR is a Promoter()
(
    description = "pTet promoter"
)

lacI_CDS is a CDS()

lacI_RBS is a RBS()
lacI_RBS_2 is a Component(i_DNA)
(
  role = i_rbs
)

lacI_term is a Terminator()
```

Figure 15: The automatically generated ShortBOL documentation page. Each template from both levels of abstraction is displayed with a description, usage, description of parameters, specialised components and a small example.

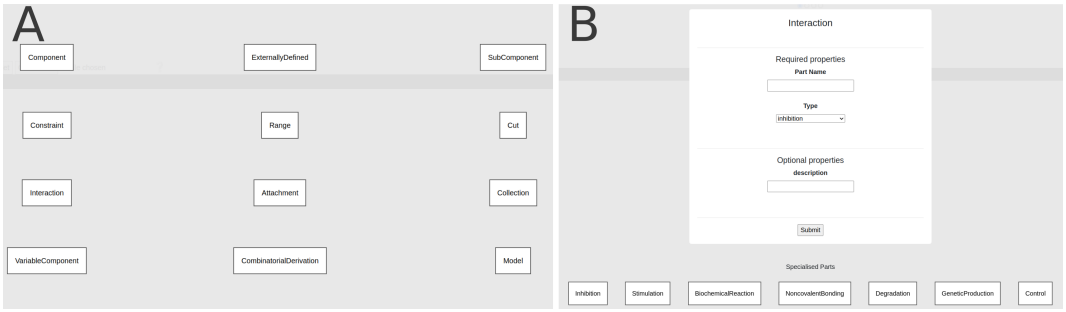

Figure 16: **A**) The options screen to choose a component to add to a document. **B**) The customisation screen for the Interaction template. Enables additional information and to choose specialised templates.

Glyph visualisation

As discussed in the background, the glyph representation is a powerful method for fast comprehension of relatively small designs. Therefore, to supplement the language, the web application implements a simple glyph visualisation mechanism that displays the compiled SBOL. Figure 17 displays a generated representation of a single gene defined within ShortBOL.

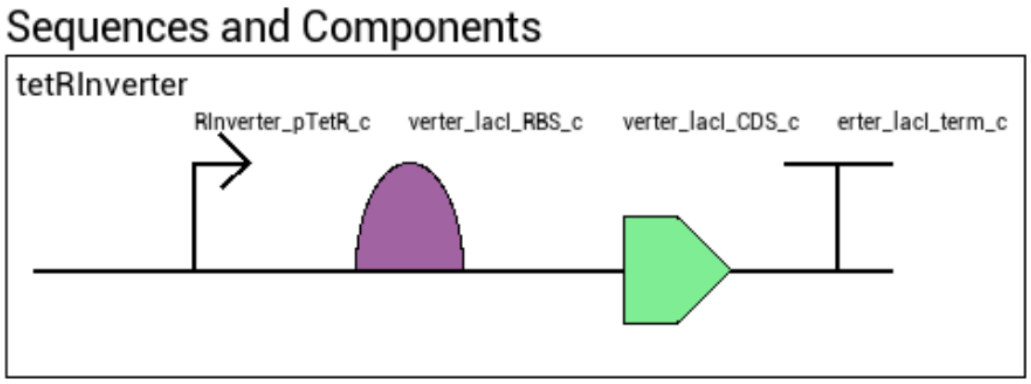

Figure 17: The glyph representation generated by the ShortBOL web application. A simple four-part construct is displayed as an example.

Sequence representation

Also discussed in the background, the sequence representation is comfortable for biologists. Therefore, to supplement the language, the web application implements a linear sequence view that includes the regions within the DNA of parts and constructs. Figure 18 displays the sequence representation where each component in the design is displayed as a coloured region.

Figure 18: The sequence representation of an example design from the ShortBOL web application. Each colour represents a genetic role and is contained within the length of the region within the sequence.

4 Discussion

This work aimed to explore the gap between natural language and language introduced by a standard and how mappings can be made to reduce the barrier to entry to define genetic designs stored in s standard format within synthetic biology. While existing language mappings and DSLs exist, this work focused on mapping to standard data, a simple syntax, and a more natural language. This simplified approach contrasts with the often more verbose languages that implement more complex syntax and language to provide extended functionality. The outcome of this research was a new language called ShortBOL that employs an extendable template system where templates can be defined to represent any physical or conceptual entity and encapsulate detail. Furthermore, templates can be defined and instantiated inside other templates. The mechanism to recursively define templates means an abstraction hierarchy of arbitrary depth can be defined. Moreover, this section introduced two initial template libraries that map to the SBOL data model in two different levels of abstraction, one of which is a close mirror to SBOL and a more abstract library aimed to reduce language dissonance. With the language of ShortBOL, a software package and web application were developed to reduce further some of the challenges a new user may find when using the language.

This section will discuss the design choices, strengths and limits of ShortBOL. Furthermore, the potential future directions for research will be discussed based on the challenges faced in this work.

4.1 Design choices

Decoupling logic and data via an object approach

The first design decision was identifying how SBOL classes would be captured within this language. The choice to employ an object approach instead of a functional one was made for two primary reasons. Firstly, the logic and the data model would be more likely to be coupled because a functional approach would consist of functions which are static blocks of computation with immutable data. Secondly, because the logic and data are coupled, the functional approach requires a more extensive and complex language because the language contains mechanisms to perform computation.

Once the decision to employ an object approach is made, the following design decision is the implementation detail. A complete class system, such as objects with member functions, is unnecessary because it would increase language complexity and promote logic encoded within the methods. Therefore, the minimal required features are added to templates.

- **Instantiation** refers to the production of instances of a template. ShortBOL uses the keywords "is a" to instantiate instances. The design decision is not to implement the instantiation system alone because it is a required feature but to decide on the keyword. The "is a " keyword was decided over symbols such as "=" because it makes the language appear more like a natural language than a programming language.

- **Inheritance** is creating new specialised templates that extend existing templates. Inheritance was decided as a language feature because it enables consistency between levels of the abstract hierarchy present within template libraries and increases the reuse of existing templates.

- **Aliases** are synonyms for URIs. For example, within ShortBOL, the alias "description" is synonymous with the URI `http://purl.org/dc/elements/1.1/description`. Aliases are used because many URIs within a document quickly make documents incomprehensible. Alias names are designed to be similar to the resource subject without creating naming collisions inside the libraries.

- **Properties** are values assigned inside the body of a template. In ShortBOL, the "=" syntax is implemented to allow further customisation by the user because defining a template for every possible property is infeasible.

- **Parameters** are user-provided arguments that further customise a template. Parameters are needed for undeterminable properties, such as sequence data or free text descriptions, or for required properties, such as template types.

- The **Context** within ShortBOL is the template name on which the property is being set. The keyword "self" refers to the direct template that a property is set inside and was implemented to provide control within composite templates. Context should not be a feature for users only using and not writing templates.

Targetted terms for different users

The first template library, "developer", had few design decisions because it was simply a translation of the SBOL data model. The library mainly defines the classes within SBOL and specialised classes, which map to the recommended URIs specified within the SBOL specification. The second template library, "user", was created to increase abstraction further and move away from the SBOL data model. The primary design decision was to introduce composite templates, which enabled existing templates to be instantiated within a template definition, removing the user's requirements to keep the design document in line with the SBOL structure. Furthermore, some non-SBOL templates, such as "hasDNASequence", were introduced to make the language appear more naturally written.

Removing complexity to enable user-friendliness

A conscious effort was made to reduce the complexity via templates. However, another design choice was to remove the requirements for a detailed understanding of the language. For example, the language initially required the namespace (the template library name) to prefix all templates with a dot operator. The requirement to prefix templates gave the user more control because multiple template libraries could be used within a single document without naming collision. However, the document appeared very verbose, and it was clear that this would alienate the user base to whom this language was designed. Many decisions like this were made, which resulted in a language with a small and straightforward syntax.

Active development and support of ShortBOL

A longstanding issue of academic software, especially within a smaller and less mature field such as synthetic biology, is the inability to provide development and support once the research is concluded. This issue is mainly because of the limited resources that can be allocated once the original research goals are finished. The result is that software can often not advance with changes in the field and provide active support to new users. This issue was considered from the inception of the implementation of ShortBOL, and design choices were made to address it. A common principle within software engineering is the separation of concerns, which involves keeping entities separate so that changes to one component have minimal impact on others, thereby increasing modularity and ease of maintenance. This principle was achieved within shortBOL by decoupling the model (SBOL), interface (Software implementation) and logic (ShortBOL libraries). Practically, this means that if one of the entities changes, all others can remain unchanged and still be functional. For example, if changes are made to the SBOL data model, the software system, i.e. the method to transform ShortBOL documents to SBOL-RDF, will not need to be changed. This approach substantially reduces the maintenance requirements of ShortBOL because the logic is contained within the libraries and will be the only part that needs to be changed. Currently, ShortBOL has two usage methods. Firstly, the live web application enables users to access the tool quickly. However, resource constraints may mean that this application cannot be maintained in the future. In this case, an offline version, which can be installed easily on an individual system, is available. Furthermore, several educational and training resources were created during development, including various examples, a tutorial system, a dynamic documentation system and videos. When com-

bined, these resources will reduce the likelihood that a user will be required to interface with the developers of ShortBOL.

4.2 Strengths

ShortBOL has three strengths that make it viable for defining genetic designs.

Abstraction Hierarchy for consistency and reuse

Templates are at the core of ShortBOL. As discussed, they are constructs similar to classes within programming languages and ontologies where templates can extend existing templates to provide further specialisation. Furthermore, templates can be composed within other templates abstracting detail from a user. Therefore, ShortBOL can define an arbitrary level of abstraction and can be used to target any level of knowledge. Furthermore, by extending existing templates, consistency is kept within the ShortBOL libraries and can clarify the levels of specialisation.

Decoupled logic and data for reusability and adaptability

Template libraries (the data) provide two main benefits, independent of the underlying language (mechanisms to generate graphs). Firstly, the language syntax, parsing of documents and the generation of RDF graphs do not have any SBOL-specific information (this is present within the existing template libraries and extensions). Therefore, the underlying mechanisms can be reused for any data instead of rewriting logic coupled to the data model. Secondly, changes to the data, such as changes to the SBOL data model, do not require changes to the system's internal logic. Instead, changes to the library are required, which is a considerably smaller operation.

Enables the definition of multivariate data

Many of the discussed design specification mechanisms can only capture a small number of data types. For example, the SBOL glyph approach can only capture sequence data and abstract interactions. However, the SBOL data model has classes encapsulating different datatypes such as experimental data, combinatorial designs and simulation models. ShortBOL implements all classes within the SBOL ontology and provides mechanisms to connect these datatypes. Furthermore, expanding the previous two strengths, ShortBOL has mechanisms to define facades where non-SBOL and SBOL data are contained within abstract templates providing a unified interface.

4.3 Limitations

This section discusses some shortfalls of the ShortBOL language and tooling stemming from design choices or are inherent to the process.

Manual approach to design specification

While ShortBOL can define implementation agnostic designs, details must be specified to generate a complete SBOL design. However, synthetic biology is moving towards automation due to the increased size and complexity of designs. Therefore, it is questionable how much further

purely manual design specification will be feasible. For example, defining a library of designs containing many constructs would be infeasible within ShortBOL. However, this is not inherent to ShortBOL. Manual specification techniques such as SBOL visual will also be infeasible if no automatic mechanism exists to generate the data.

User dependence to provide quality input

The ShortBOL language and libraries can produce compact documents that promote modularity and are comprehensible by people unfamiliar with the SBOL data model. However, like any specification or programming language, this depends on the person writing the document. The writer must intend to employ good practices while creating designs, such as reusing existing templates and specifying all possible data. ShortBOL does have mechanisms to promote good standards, such as the templating system but does not have any inherent mechanisms to ensure these.

Limited Resources for development

Programming languages and DSLs require continuous support, especially if mapping to a standard. For example, if the developers of SBOL decide to change the model, the ShortBOL libraries will need to update. Furthermore, despite creating a simplified language which focuses on accessibility and a shallow learning curve, a user must still interface with a software application. Most applications require maintenance, user testing over a long period and iterative improvement, especially with user feedback. While these are not inherent to the ShortBOL language or implementing tools, they seriously reduce the likelihood of longevity when resources do not exist for maintenance.

4.4 Future work

This section discusses three potential extensions for this work based on limitations or improvements identified after the project.

Using virtual analogues to remove referential incompatibility

ShortBOL allows users to define local terms (template names) without validating whether the resultant URI is a real resource. However, local terms which do not resolve to a shared resource are adverse to the intent of RDF, where each element references a resource. Furthermore, this breaks the idea of virtual analogues within synthetic biology, where each part or module within a design references the same virtual analogue. Therefore, in future, the mechanisms which implement the language could either provide a mechanism for the user to interface with standardised databases such as Synbiohub or automatically find the virtual analogues.

New template libraries

Currently, two libraries at different abstraction levels exist. While the second template library increases abstraction and reduces complexity, the language still holds artefacts from the language defined within SBOL. Therefore, a more abstract and robust template library that captures actual terms used within synthetic biology would be beneficial. Furthermore, providing libraries at

several layers of abstraction would provide more options for a user based on their requirements and knowledge level. To extend this future work, to decide upon a set of universal terms is not a trivial task and is a challenging social task requiring input from multiple members of the synthetic biology community, which specialise in many sub-domains within the field.

Different applications of the template system

As discussed, a design choice during development was to decouple the language and implementation from the data model (SBOL). Therefore, while the language (template libraries) are genetic design focused, the underlying technology could be applied to specify any datatype provided the template libraries are written. For example, to establish build protocols or designing and specifying experiments.

Generating Designs from abstract specifications

As discussed within the limitations, design specification is moving towards automated and semi-automated approaches. ShortBOL can define designs without implementation details, that is, the shape of the design without the biological mechanisms. Therefore, intent can be specified without the knowledge of the implementation. A potential future project for ShortBOL would be to take the existing specification and automatically generate a set of designs that fulfil the requirements specified within the structure. For example, a user could provide a partial parts list, required interactions or high-level functions such as logical gate function, i.e. NOR or OR gates.

5 Conclusion

In conclusion, this chapter explored the abstraction of unfamiliar language to be more accessible to a larger group of synthetic biologists. Furthermore, the ability for language to be often more expressive within a smaller footprint provides a new mechanism for designing complex biological systems which cannot be as quickly met using visual means. During this research, we developed ShortBOL, the language with a web application with several benefits over similar languages, primarily the backing of SBOL, which allows the underlying data to be structured and can benefit from existing semantic web tooling due to being based on RDF. Furthermore, the ability to create template libraries that can abstract previous template libraries provides the mechanism for hierarchical abstraction levels based on the requirements.

Chapter 4: Using weighted knowledge graphs to quantify uncertainty and enable community-based feedback

Publications arising from this chapter

- Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "GENETTA: a Network-Based Tool for the Analysis of Complex Genetic Designs". *ACS Synthetic Biology*, 2023

Software arising from this chapter

- Genetta

1 Introduction

Synthetic biology aims to compose sequence data into structural and functional modules to enable forward engineering[119]. Defining reusable and abstract modules introduces unique challenges not present when working solely at the sequence level. Standards such as the SBOL move towards this goal by enabling the construction of these modules so that they can be unambiguously connected using semantic labels and rules regarding how data connects, enabling the potential for standardised data. While this focus has been achieved to a mixed degree[120], as genetic circuits increase in size and complexity, abstraction away from a sequence-centric approach will become required. In an ideal case, a modular system allows the reuse of existing functions explicitly validated by use in past circuits and designs.

However, some issues are not inherently fixed by employing standards. Firstly, the employment of standard structures does not force certain types of information to be described. For example, sequence data overwhelmingly covers the data encoded within datasets and abstract functional information is seldom described despite being crucial for comprehension. Secondly, if a composite is stored in a database for sharing, standards may not prevent collisions with other composites. For instance, identical composite names can make it difficult to determine the original entity, even if they are conceptually synonymous. Also, if a composite is stored for reuse, there is no assurance that the advertised entity functions as intended. Often, a part may be labelled as a specific component in a database but might include extra sequence data, rendering it non-functional, which could be exasperated when used in a different context, such as a different host. This discrepancy, likely stemming from human error, can result in costly consequences for those who rely on the data as-is. Finally, The data can be in a standard format but captured within unstructured free text blocks that humans can interrogate but are unusable within computation approaches.

The result is large databases with existing biological and design knowledge that vary hugely in accessibility, accuracy and type of information encoded[121]. While this may be an acceptable process now, for synthetic biology to generate increasingly complex constructs, unreliable and flat data sharing must be reduced. Therefore, there is a need for datasets which encode more abstract and functional-centric information alongside the sequence data and features to increase the reliability of the underlying data.

1.1 A review of existing synthetic biology databases

In this context, a database refers to the software system that captures datasets, which are specific collections of information. Databases can store multiple datasets and export them in various formats. This analysis will focus on existing databases to identify their advantages and disadvantages. Reviewing these databases helps ascertain which features are beneficial and where potential shortcomings lie. Additionally, understanding the structure of the information provides insights into its potential utilisation and manipulation, particularly in computational approaches.

Synbiohub

SynBioHub is a web-based platform for storing, retrieving, and sharing synthetic biology designs[122]. It aims to facilitate the sharing and exchange of biological design data and promote the development of synthetic biology. Synbiohub holds existing biological constructs encoded within SBOL from individual parts to full designs, including translations of the IGEM parts repository[123] and the Cello parts library[124]. All information within Synbiohub is stored within a graph database in the form of an RDF triplestore in SBOL format but can be exported using several formats such as Genbank and FASTA but also different formats such as plasmid maps or CSV sheets.

Advantages Synbiohub offers various instances, including the original Synbiohub, the living computer project (A project to apply computer science principles to synthetic biology), and Sevahub[5] (storage for SEVA vectors, the standard representation of vector plasmids within the SBOL framework). These instances allow for customising specific databases, tailoring them to specialise in particular data types. SBOL-encoded data within Synbiohub facilitates programmatic access, enabling handling large volumes of data. Moreover, Synbiohub provides an application programming interface (API) that grants programmatic access to its contents, a crucial feature for automated processes. These Synbiohub instances house a wide range of datasets, some capturing libraries of full designs while others focus on genetic parts. As a result, a substantial amount of valuable information can be extracted from these instances. Being created around a standard data format (SBOL), the database can handle robust data types such as functional and structural information. Therefore, the mechanisms to access the data can take advantage of this.

Disadvantages While the datasets within Synbiohub instances are SBOL encoded, they typically contain information similar to that of a Genbank file, encompassing sequence data and annotations. Also, Synbiohub lacks a mechanism for verifying the presence of existing elements when incorporating new designs into the database, potentially resulting in the same entity under multiple resource names. The software implementation of Synbiohub is not particularly robust and suffers from numerous faults, which can pose challenges for users. Furthermore, Synbiohub's in-

terface often has the underlying data model bleeding out into the controls, which can introduce a steep learning curve to users uncomfortable with SBOL.

Inventory of Composable Elements

Inventory of Composable Elements (ICE)[125] is a registry platform that stores DNA components. ICE differs from Synbiohub primarily because it is a traditional relational database with entities that contain properties. Information can be exported using several formats, such as SBOL, Genbank and FASTA.

Advantages The database offers numerous analysis tools, including the capacity to edit constructs stored in the database, making it a more comprehensive software package than Synbiohub. Additionally, it allows users to share constructs with others or keep them private as needed.

Disadvantages The database faces limited support and has not gained widespread adoption. Its programmatic access is restricted, making seamless integration into automated pipelines impractical. The underlying database employs flat structures with keywords representing abstract information, including abstraction levels and functional categorisation.

AddGene

Addgene[126] is a platform primarily designed for scientists to share DNA, particularly plasmids. While it does function as a repository, its core purpose is to facilitate the exchange of plasmid DNA among researchers rather than to serve as a traditional database. These plasmids, extensively used in molecular biology and genetics, are primarily represented in a circular format at the primary sequence level in Addgene. The software system around the database attempts to self-annotate new plasmids when added to the repository. Additionally, the repository provides valuable resources like protocols and educational materials.

Advantage AddGene is frequently used by scientists across diverse fields. Its extensive collection of plasmids and vectors, numbering approximately 130,000. The representations revolving around a plasmid map are familiar to practitioners and do not require changes to current working practices.

Disadvantages AddGene lacks standardised data export formats and primarily emphasises the primary structure, exporting sequences without annotations. The database employs a flat organisational structure with broad categorisations like "Plasmid," "Antibodies," and "Preps." Notably, AddGene does not specifically cater to the objectives of synthetic biology, such as modularity and abstraction, as it is not a synthetic biology-centric database. Additionally, the software associated with the database does not provide an application programming interface (API), rendering programmatic access unfeasible.

Kyoto Encyclopedia of Genes and Genomes (KEGG)

KEGG[127] provides information on biological pathways, diseases, drugs, and organisms. The data can be exported as an image (PNG, JPEG), text-based (KEGG Markup Language, plain text), database (TSV, SQL), data exchange (JSON, XML) or bioinformatic (FASTA, GFF) format depending on the information explored.

Advantages KEGG provides detailed information on metabolic, regulatory, and signalling pathways and integrates various types of biological data, including genomic, chemical, and systemic information. Furthermore, it includes information on diseases, associated pathways and drug information, chemical structures and interactions with biological molecules.

Disadvantages KEGG specialises in natural systems, which may not always be appropriate for all synthetic biology applications. Also, KEGG requires a paid subscription for bulk download, which may be an issue with computational approaches. While KEGG covers a wide range of organisms, the depth of information may vary. Some species may have more comprehensive data than others, and the information for non-model organisms might be limited. Also, it primarily relies on curated data, and the availability of experimental data might vary.

1.2 A review of existing datasets

Numerous datasets exist in synthetic biology and related areas. Some of these datasets specialise in providing information about specific data types, such as metabolic networks, while others serve as general-purpose datasets, containing sequence information and general biological features. Even within general-purpose datasets, unique characteristics may be observed in terms of data capture and stored information, and all datasets exhibit variations in quality. Reviewing datasets becomes relevant in this context, as some databases, like Synbiohub, host multiple datasets. However, the focus will be on the format and structure a dataset may assume when extracted, as this aspect plays a critical role in integration and computational applications.

iGEM Registry of Standard Biological Parts

The IGEM parts repository[123] contains over 50,000 categorised records, including individual genetic parts, composite constructs, experimental data, plasmid details, cellular attributes, and protein information. The IGEM parts repository's representation is exported from the SynBiohub database encoded within SBOL for this review. However, it must be noted that SBOL encoding does not guarantee quality data that is tailored for the task, i.e. that each entity is referentially valid and encodes a broad domain of information.

Advantages The repository hosts detailed information about some of the most thoroughly tested and experimentally characterised genetic parts. Notably, iGEM records include additional metadata not typically found in the Synthetic Biology Open Language (SBOL) format, providing insights into whether a part is untested, low-quality, or unused. This additional metadata is based on usage and feedback within the iGEM parts repository.

Disadvantages iGEM information is captured ad-hoc within free-text descriptions, making extracting structured data beyond sequence information challenging. The quality of records within the iGEM parts repository exhibits significant variation, with a notable presence of unusable records. Additionally, the repository includes numerous duplicate and redundant records, which can complicate data management and utilisation.

Cello

The Cello parts are a collection of genetic components used within the Cello tool (an application that converts design specifications to transcriptional logic circuits)[124] to build a series of

genetic circuits. This collection is a small but highly characterised collection of frequently used genetic entities. Furthermore, multiple datasets exist, with the primary general dataset and other variants targeting different hosts. The Cello dataset can be exported from the LCP instance of the SynBiohub database encoded within SBOL.

Advantages The components in this collection are extensively characterised and frequently employed, thus increasing their reliability. The SBOL encoding utilised for these parts is of high quality. It provides additional information about interactions, functional attributes, and structural details, enhancing their utility in synthetic biology research and applications.

Disadvantages The dataset is relatively small, 300, unlike the thousands contained within other datasets. Although explicitly encoding interactions, sometimes they are highly abstracted, often obscuring critical interactions.

Virtual parts repository (VPR)

The virtual parts repository (VPR)[128] is a dataset of virtual genetic parts. This dataset aims to create extensive models that assist in designing large-scale biological systems through model-driven methods. Genetic components within the VPR dataset are expressed as a mathematical model in the form of the Systems Biology Markup Language (SMBL)[71] and as SBOL.

Advantages Every record in the dataset refers to a unique canonical entity, ensuring the absence of duplicates. The VPR dataset encompasses a diverse range of genetic parts and their interactions, providing a comprehensive genetic research and analysis resource.

Disadvantages Most of the dataset primarily focuses on interactions between sigma factors and proteins, which may not align with the needs of a more abstract network. Furthermore, the dataset adopts a localised approach, employing terms and terminology specific to VPR, rendering the records less interoperable with external sources. Consequently, when entities from different datasets correspond, the only commonality often resides at the sequence level.

1.3 What standards cannot provide

Standards like SBOL, while valuable, do not inherently eliminate all the issues they aim to address. Even when design data is expressed within a standard data format, issues like low quality, inaccuracies, inconsistencies, and ambiguities can persist. Consequently, the user is responsible for employing standards to produce reliable data. Moreover, as standards such as SBOL progress in defining reusable and abstract modules, they introduce unique challenges that do not arise when working solely at the sequence level. This section explores the nuances of standardisation, highlighting instances where consistent enforcement of standards may be lacking and the limitations in using standards to enforce certain data features.

Types of standardisation

Even when data is encoded within a standard like SBOL, encoding does not guarantee that data is entirely standardised. The following sections will discuss three distinct definitions of a standard for an SBOL document. Figure 1 features a detailed network representation of PhlF protein degradation, encoded using SBOL. In this figure, it is necessary to use complete IRIs to describe

specific standardisations fully. Regarding edges, the entire IRI is directly associated with a particular colour. However, the colour determines the common prefix with nodes and is combined with the node's label to generate the full IRI. The standardisation of this SBOL representation will be analysed concerning all three definitions within their respective sections.

Semantic standardisation concerns utilising a predetermined collection of ontological terms to describe data in a design[129]. It enables entities and their properties to be measured relative to others. For example, if the genetic role of two entities is defined within a finite and known domain, they can be compared for similarities or differences. SBOL enforces this standardisation by defining ontology terms to describe specific features. Figure 1 is semantically standardised because each object within the network references known SBOL types (Interaction, Participation, Feature and Component from the SBOL ontology) via the RDF type predicate. Furthermore, some objects also reference external ontologies, namely the systems biology ontology (SBO), to encode information such as genetic type (Protein) and interaction type (Degradation).

Structural standardisation ensures that every component within a design is connected only to other valid components. This structure normalisation enables predictable assumptions during analysis, allowing for traversal based on the rules defined within the standard. For instance, when an Interaction is specified, it is expected to have participants, and this understanding guides the analysis of the graph because computational approaches can make these assumptions. SBOL enforces structural standardisation by defining which components can connect and how. In Figure 1, structural standardisation is displayed by the connections between components that adhere to SBOL standards. For instance, a Participation object representing an entity's involvement in an interaction must reference a Feature through the hasFeature predicate from the SBOL ontology.

Referential standardisation involves each entity referencing a virtual analogue, a URI, which, in turn, points to an actual record within an accessible dataset. It provides a framework for representing, linking, and exchanging data on the web, enabling interoperability, machine readability, and flexible data modelling. Specifically, this standardisation facilitates computational comparisons among instances within a design. For instance, if two promoters reference the same virtual part, computational approaches can identify them as identical. While SBOL offers the means to name entities using these resources, it does not mandate referential standardisation, and an SBOL-encoded design can be considered SBOL valid without it. However, the instance of an SBOL-encoded system depicted in Figure 1 adheres to referential standardisation as it draws objects from the "programmingbiology" (LCP) instance of Synbiohub. Consequently, the URIs resolve to real and accessible resources, enabling an interconnected system beyond the local network.

Features unenforceable by a standard

Some features can be explicitly enforced or implicitly promoted by a standard, such as the three types of standardisation discussed previously. However, despite these features being critical for the robust functioning of databases, they cannot be enforced by a standard because they are products of how an instance is implemented. The following explores some of these features.

Collisions and misses between entities Even if a standard enforces referential standardisation, there is no assurance that the references are canonical, i.e., each part refers to a single virtual

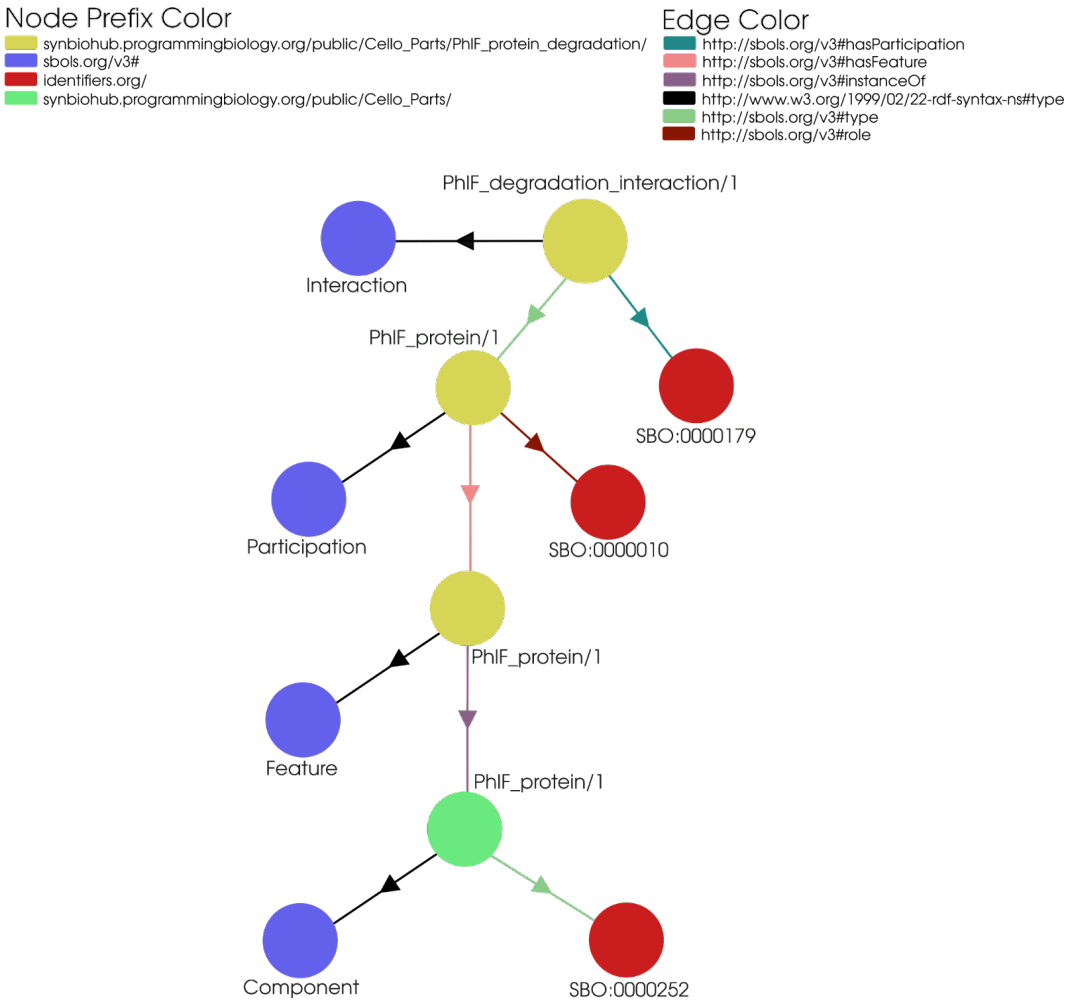

Figure 1: A network representation of the PhlF protein degradation described within SBOL. Each instance (node) references an accessible part of the web. Also, each instance contains edges pointing to descriptive information such as RDF types or via external ontologies referencing genetic roles or types.

entity. In this case, the IRIs of the entities within a design may differ. However, the data they point to is conceptually identical, for example, the same sequence in the case of sequence-based entities (see methods for a discussion on identifying conceptual equivalence for different datatypes). For example, as seen in Figure 2, dataset 1 represents a short regulatory mechanism of LacI -pLac repression, and Dataset 2 represents the pLac promoter in isolation. While the sequences for both pLac from D1 and BBa_R0100 from D1 are identical, an issue of specificity arises. The LacI protein from dataset 1 should repress the promoter from D2 but does not because the network is unaware that the two promoters are synonymous. Because of this, computational tractability is challenging. This issue is guaranteed when comparing datasets where design elements are defined locally, i.e. the names are only known within the design. Therefore, a computational approach must identify that these two virtual parts reference the same physical entity.

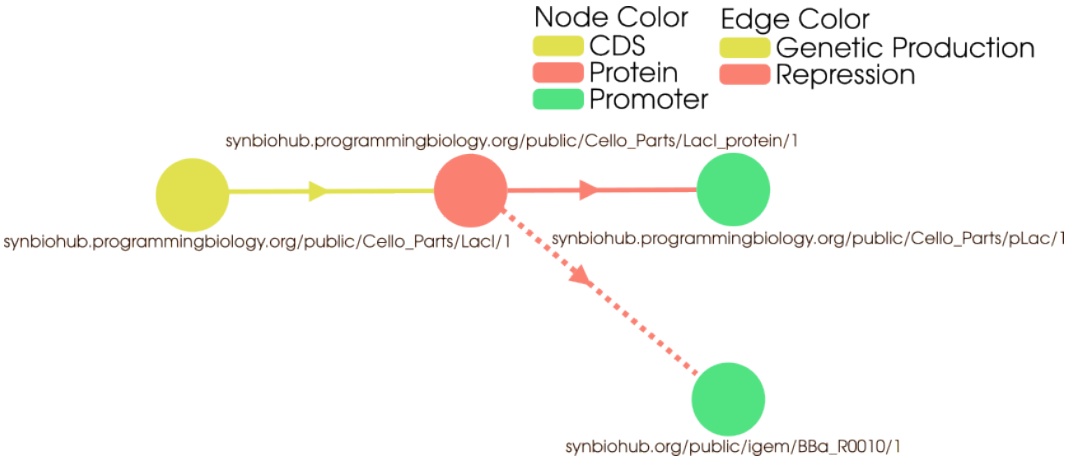

Figure 2: A network representation of the two interactions, the LacI protein's production and the pLac promoter's repression. The block lines represent interactions explicitly encoded. The dotted line represents a known interaction that is not encoded because the URIs are different despite the identical sequence data of the two promoters.

Quantifying the correctness of information Standards cannot ensure that the information encoded within is correct. If a composite is stored to be reused, there is no guarantee that the presented entity is functional or that the function is desired. Therefore, users must either trust the individual user uploading the data or thoroughly interrogate all information acquired. For example, One of the many instances of the LacI CDS within the iGEM parts repository (BBa_C0013) explicitly states "No Barcode", referring to a fragment in the sequence to identify the part. However, when analysed, this barcode is present within the sequence. While this example demonstrates an individual part and is relatively trivial to identify, the larger a construct, the more difficult it is to identify these discrepancies.

The change of entities over time Provenance within the context of the datasets refers to capturing the change of entities, such as genetic parts, over time. For example, if a genetic part is taken and modified within a dataset, the new part is classified as derived from the original. Cap-

turing provenance within this context allows the ability to track the history of an entity but also enables the clustering of very similar entities. For example, if a user is looking for an alternative part for a genetic design, tracking the connections of derived components will provide an easy method for suggesting alternatives. Standards may provide a mechanism to capture provenance, such as SBOL, by leveraging the existing PROV-O ontology[130] among others, but cannot enforce the capture of provenance. Furthermore, in practice, provenance is seldom captured.

Usage relationships between entities Generally, datasets exclusively encompass design-agnostic information (unless it is a collection of designs), ensuring that it does not rely on the specific context of any particular design. For instance, they exclude position-dependent regulation, where the functioning of genetic components (like genes, promoters, enhancers, and other regulatory elements) hinges on their relative positions within the genome. This omission of context-dependent information stems from the impracticality of encoding every potential combination. However, knowing usage relationships between entities may be valuable even if a dataset does not explicitly define combinations. Once again, standards such as SBOL do inherently enable the usage of entities to be defined, but seldom from a design-agnostic approach. For example, a standard may enable a hierarchical relationship, such as a promoter and CDS, to be contained within the same construct defined within a design but do not describe methods to describe the experiences of parts being used together. However, it will not define a relationship from a dataset perspective, describing that two genetic parts have some interplay from a design-agnostic perspective, such as two genetic parts being very commonly used together within designs.

1.4 The issues with the current landscape of data capture

As discussed, standard formats, such as SBOL, provide the mechanisms to promote rich databases by capturing the information in computer-readable formats that can capture information beyond DNA sequences. However, these features are rarely implemented for three primary reasons. Firstly, when design information is encoded within a standard format, it is often a translation of legacy information captured using flat file formats such as Genbank, potentially with extra free-text metadata encoded, often making up the most significant amount of database content. When translating this information to a standard format, the processes are usually direct translations, resulting in structures encoding the same flat information such as DNA sequence, annotations and handwritten descriptions. Therefore, much of this information is missing when translated into a standard format. Figure 3 displays the network representation of several genetic parts taken from the Synbiohub database (see database review for a further discussion on Synbiohub) that constitutes the LacI regulatory system. Here, all encoded are nodes with properties (sequence and descriptive text) and do not encode entity relationships, such as interactions with other entities within the dataset. Currently, specification methods backed by standards are often manual processes. For example, the SBOL visual[110] approach requires users to define the genetic design one glyph at a time. Therefore, all the extra information that can be encoded must be explicitly specified, which is not feasible, especially as designs become more complex, and undoubtedly shortcuts will be taken and information will be missing. Standards such as SBOL are designed for a specific design and do not consider the databases that may capture design data. Therefore, very commonly, there is no inherent mechanism to define meta-relationships which do not pertain to a specific de-

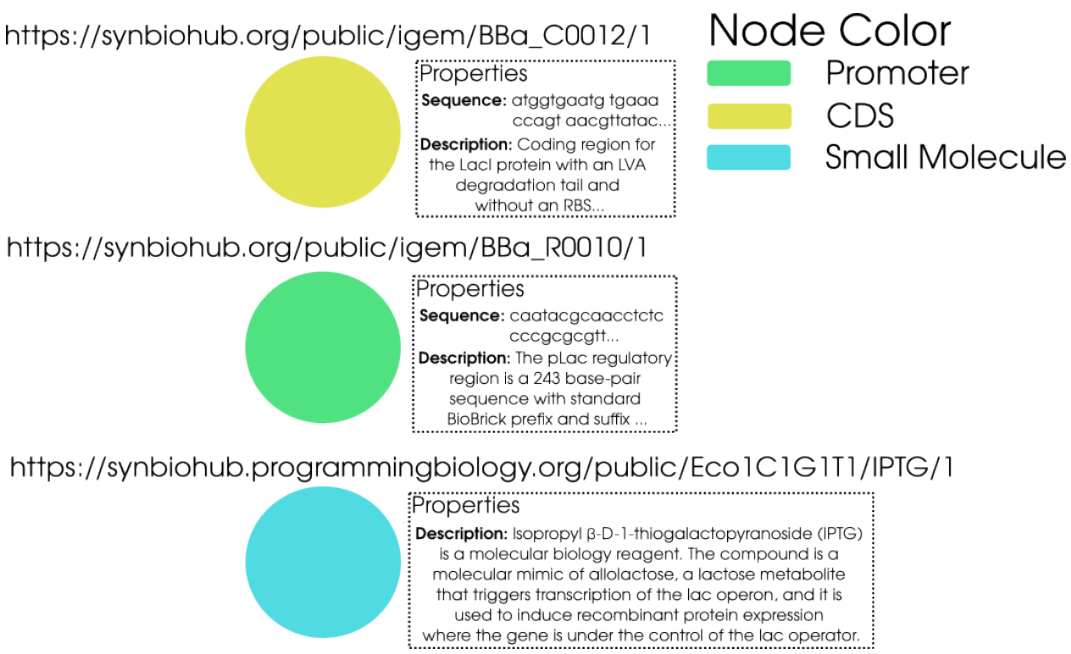

Figure 3: Representation of the LacI regulation system using nodes taken from instances of Synbiohub. The datasets do not encode interactions and are not visualised when represented as a network. Due to the parts being represented within a labeled graph, literals such as sequences are stored as properties on the nodes as opposed to nodes in themselves.

sign within the standard framework. What follows is a discussion of two prominent symptoms of the current state of data capturing within synthetic biology, namely the gaps of knowledge within the domain and the uncertainty within existing information.

Gaps in domain data

Because the definition and capture of designs is sequence-centric, current design data is overwhelmingly sequence data with written annotations. Take any current database or tool; the most common method to export data is via a flat-file format such as GenBank[131], which cannot formally encode any other information. While sequence data implicitly encodes more information, explicit definition of more abstract and broad types of information is required to reduce some perceived complexity. While standards such as the SBOL[132] enable the format specification of a much larger domain such as interaction, structural and functional hierarchies, experimental and build protocols or simulation models, the reality is that these are seldom defined. Figure 4 displays the count of each SBOL instance within the original Synbiohub[133], a database capturing design data in the SBOL data format[134]. Approximately 7% of the ComponentDefinition cases within the database encode Interactions (even more concerning is that all of these interactions come from the same collection encoding the VPR dataset reviewed previously). The data variance is even more limited with specific datasets within the database, such as the International Genetically Engineered Machine (IGEM)[135] parts repository. Despite being one of the most extensive datasets of genetic design information within synthetic biology, containing over 30,000 records

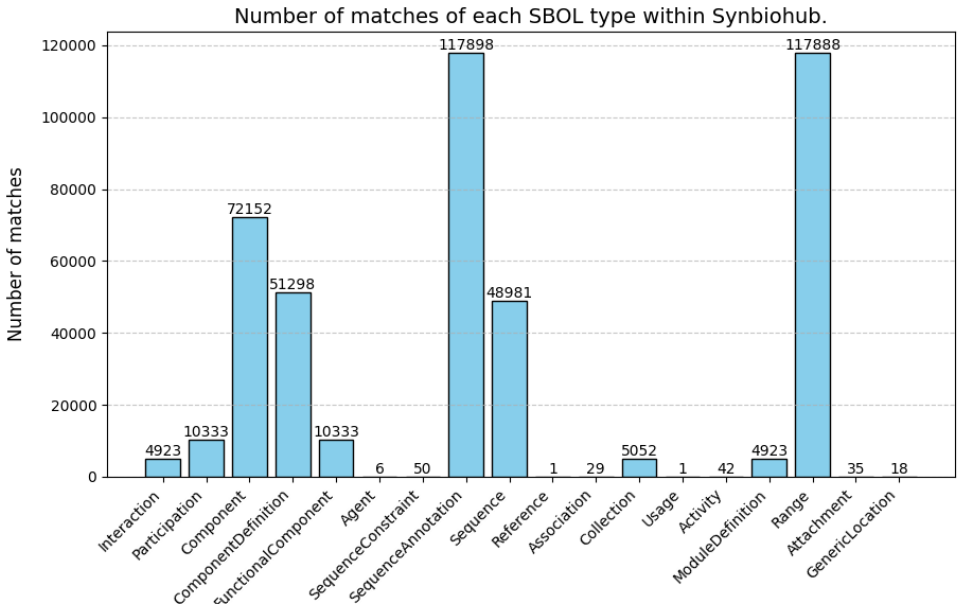

Figure 4: A Plot displaying the count of the instances of each SBOL datatype within the original Synbiohub instance. The number of sequence-centric data types (Component, CompondentDefinition, SequenceAnnotation and Range) is considerably higher than other data types.

in a structure with the mechanisms to capture more information, no records formally encode any information beyond sequence (Figure 5) data because they translate the written descriptions into SBOL properties. Therefore, despite the ability to define structured information, the bridge between the goal of synthetic biology and the current landscape necessitates a broader domain of knowledge to be explicitly encoded, such as a clear definition of the desired function within the design framework[136].

Information uncertainty

Reproducibility is a significant problem within synthetic biology[137]. Consistently, experiments cannot be replicated[138], and the acceptable error range is high[139]. While not the primary cause, permitting most open data sources to insert uncurated information introduces uncertainty where quantifying fidelity is challenging, which does not alleviate these problems. Even with a rich and broad domain of design information, there is no guarantee of quality or correctness. If a design reuses parts that another has deposited onto a database, there is no guarantee that the underlying entity performs the function stated[140]. For example, very commonly, a part will be advertised in a database as a specific part but will contain extra sequence data that makes the part non-functional when added to another context. Although likely due to simple human error, it can be costly to others who take this data at face value. Uncurated databases cause an issue of confidence within the underlying datasets. Most databases do not have mechanisms to quantify whether the information uploaded by users is correct or accurate. Reliance on data is exaggerated within open-source, such as Synbiohub, where anyone can upload information. While manual curation of

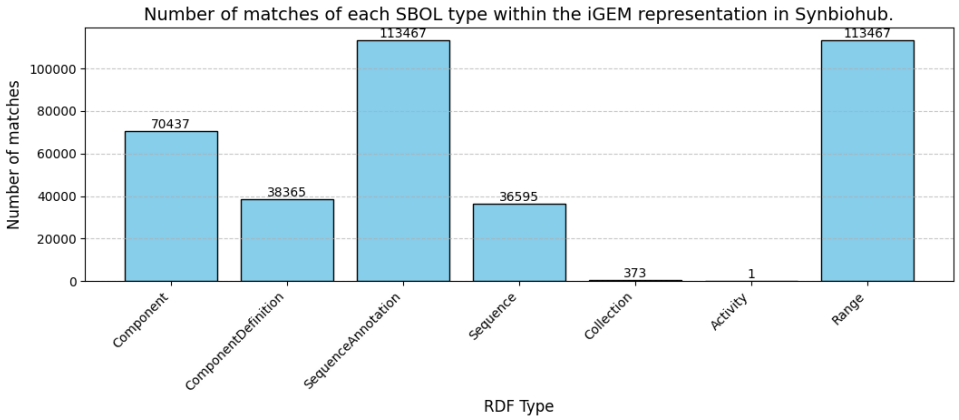

Figure 5: A plot displaying the count of each SBOL instance within the IGEM parts subgraph within the original Synbiohub instance. Functional information, such as Interactions and Modules, are absent in this dataset.

all data is infeasible, mechanisms to quantify the confidence (likelihood a piece of information is correct) would enable users to make more informed decisions on whether to use it. The IGEM parts repository[123] contains well over 40 records describing some YFP. With so many records categorised as the same or similar entity, it raises a key issue when the tracking of differences between records, the implications and reasoning are not captured; it is unclear which genetic part should be used within a design, which is exacerbated within computational approaches because software is unable to interpret written text without complex natural language approaches which in themselves are often prone to inaccuracies. Therefore, not only the uncertainty of correctness but also relationships is an issue. For example, provenance (the change of entities over time) is critical in identifying how the function of an entity may have changed and the editor's intent. However, unlike the uncertainty of correctness, the standards often enable provenance to be described (PROV-O), as previously mentioned. However, this type of information is seldom encoded within existing datasets, and while, of course, this information can be derived, it can be a challenging and laborious process.

1.5 A functional approach to synthetic biology

Concerning synthetic biology, a "function" refers to the specific role a biological component plays within a system[141, 142]. When it comes to design data, it can encompass the intended purpose of a particular aspect of the design – essentially, what the expected functionality of an entity should be before it undergoes building and testing[143]. A functional approach to design data means encoding interactions and other more abstract information[144]. This approach provides many benefits:

- **Reduce Perceived Complexity** - Explicitly capturing abstract functional information can reduce perceived complexity because practitioners with less understanding of the design will not be required to decipher the function from sequence data.

- **Met Expectations** - By defining functional modules with existing standard genetic parts, the design is more likely to perform as expected by validation within previous usage.

- **Flexibility** - Functional genetic designs may be initially defined without specific implementation details, which can be iteratively refined[15].

- **Rational Design** - Functional synthetic biology encourages a rational and systematic approach to designing biological systems. By understanding individual components' functions and interactions, informed design decisions can be made, leading to more effective and efficient constructs.

Designing and encoding designs from a functional perspective is not a new or impossible practice[92]. Practitioners already conceptualise functions at different levels of abstraction. For example, a genetic part such as a regulatory coding sequence is not usually considered at the sequence level but instead its function and impact within a biological system[143]. Despite this, a sequence-centric approach is taken when defining and capturing the design because the sequence data is required when the design comes to be built. For example, design tools such as Benchling, a sequence-based editor, are ubiquitous despite the inability to describe any level of functional information and are exported in flat file formats such as GenBank, which again have no mechanisms to capture abstract details formally. Figure 6 displays how, as a design moves from a concept to a concrete implementation, the level of abstraction decreases, and the design is conceptualised less functionally and more sequence-based. It may seem that choosing between function and sequence approaches is a crucial one[39], but a fully functional approach currently appears infeasible because designs are unable to be built from abstract parts and small nucleotide changes are common practise. However, if both the functional and abstract information and the sequence data are captured, the level of detail can be adjusted per some requirements [31]. Synthetic biology aims to

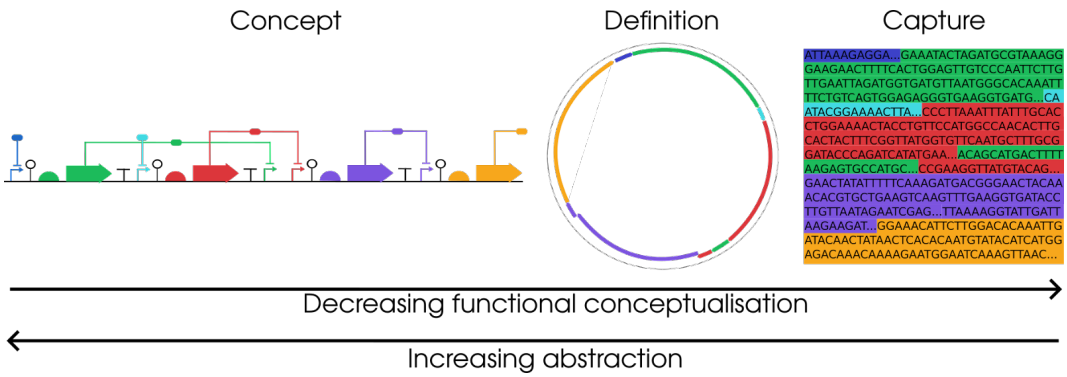

Figure 6: When a design is conceptualised, it is usually around function, such as how entities interact. However, when it is defined, the tooling is traditionally sequence-based. While function is still often considered, it is not explicitly encoded, so when the designs are captured (within a GenBank file, for example), they are purely sequence-based.

modularise sequence data into functional modules[145]. However, composing abstract reusable modules requires rich underlying data which can communicate and connect. The gap between

the aims and reality of design data within synthetic biology is more significant than simply modularising sequence data. To truly modularise genetic design data, they must explicitly encode the desired function because modularisation aims to reduce the perceived complexity. Conceptualising a design in terms of intent is a form of complexity reduction by abstracting sequence data into abstract interactions between genetic parts[146].

1.6 Dynamic knowledge graphs enable adaptation to changing conditions

More robust systems outside of synthetic biology[147] use a pseudo-feedback system that takes previous activity to drive future information provision. Search engines such as Google use similar approaches where results are based on the popularity of a website, implicitly ranked by other agents. Over time, the best results for a specific query rise to the top and inaccurate or useless information is filtered to the bottom, meaning a user rarely has to search far for desired resources. Weights are commonly used features within network science, referring to numerical values associated with entity relationships[148]. These weights measure the strength of a relationship between entities in a network given a specific metric[149]. Figure 7 displays an example of how weights can be encoded to describe the similarity of promoters, calculated given some metric. Therefore, a weighted knowledge graph (WKG) defines a knowledge graph where connections between data encode weights which capture some level of strength given some metric. Furthermore, these weights can be updated as more information is learned, for example, as evidence proving or disproving a piece of information is provided. A WKG could encode a broad range of information and swiftly adapt to new data. This adaptability empowers networks to respond effectively to changing conditions.

1.7 Aims and objectives

This analysis explored some existing datasets within the field of synthetic biology, finding their unique attributes, advantages, and drawbacks. Next, the possibilities and limitations of establishing data standards were studied. Following this, a critical evaluation of the prevailing issues surrounding synthetic biology data capture and storage was conducted. In short, the intricate landscape of synthetic biology databases unveils persistent challenges, including unstructured data, fragmented knowledge repositories, redundant information, and inherent uncertainties. Next, the exploration revealed the advantages of adopting a functional approach within synthetic biology. This strategic approach emphasises the design and construction of biological systems based on the functionality of individual components. Such a modular and standardised approach enables the customisation and tailoring of biological systems, ensuring a rational and predictable design process. In response, a promising avenue emerges in the form of networks that demonstrate remarkable efficacy in encoding and adapting to complex biological information. Networks, with their inherent capacity for dynamic interactions and knowledge representation, stand out as a potential solution to the challenges posed by the intricate nature of synthetic biology data. Moreover, integrating a dynamic knowledge system augments the potential of a functional approach. This dynamic system adapts to evolving information and facilitates iterative design and optimisation. In summary, the challenges faced by synthetic biology databases, including issues

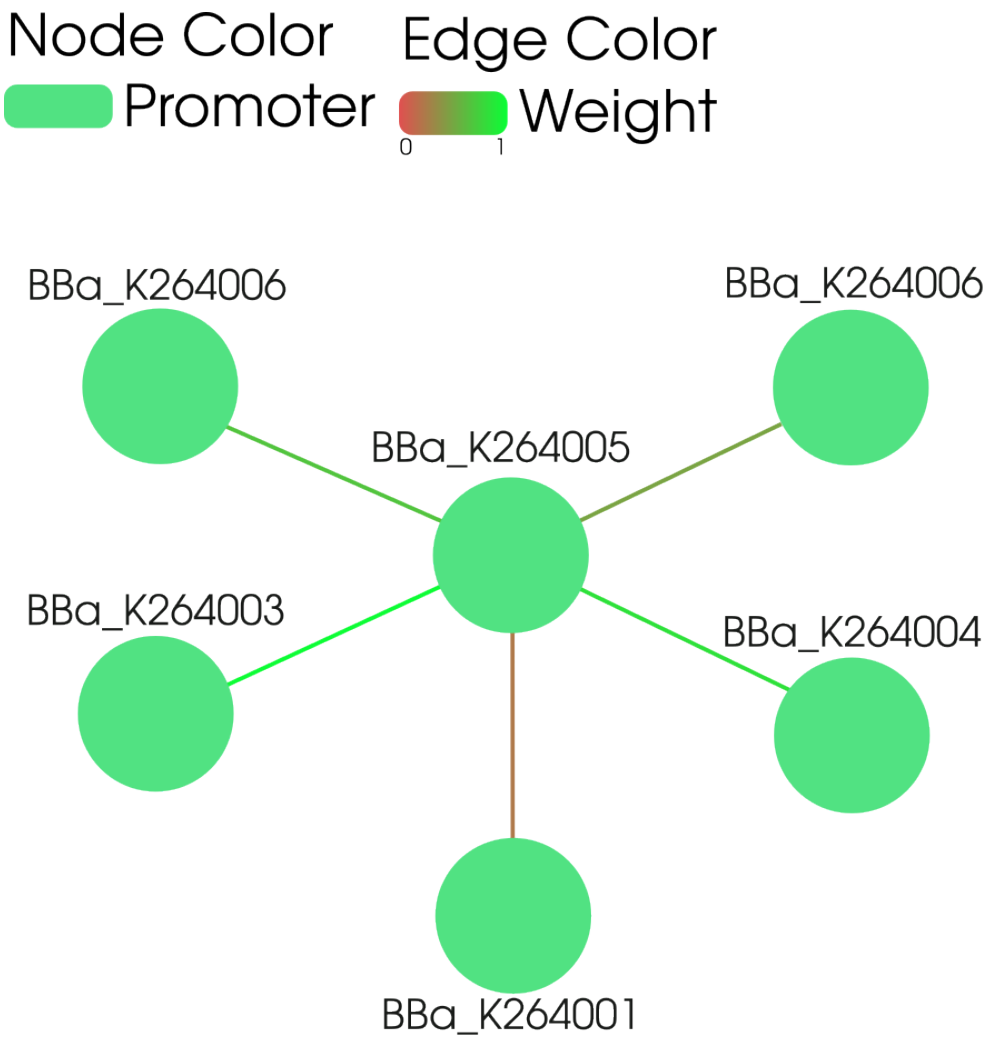

Figure 7: Network representation of similarity between promoters. Each edge contains a separate weight between 0 and 1, which could represent the strength of the connection by a given metric. This representation encodes the weight by colour, with green being closer to one and red closer to zero.

of data structure, knowledge fragmentation, redundancy, and uncertainty, demand innovative solutions. Networks and a functional approach emerge as promising methodologies to address these challenges. The combination of these approaches offers a transformative pathway forward, showcasing the adaptability and resilience required in the dynamic landscape of synthetic biology.

This proposal outlines integrating existing design data into a WKG, incorporating several key features for addressing uncertainty-related issues. The proposed network encodes dynamic metadata within the network. **Confidence** is the trust of the user base of the information[150], that is, the likelihood that the encoded information is believed to be correct. It impacts rankings, the order in which information is presented to the querying agent (human or software). **Usage** defines how commonly a piece of information is encountered. Usage may be how often the data is accessed within the network. However, it can also capture other usage scenarios, such as how frequently two genetic parts are encountered together within a design. Again, This metadata can impact how often information is presented to a user but could also be used to provide validation of entities being used together within a specific design, for example. **Provenance** captures the similarity between different entities within the network[151]. For example, provenance may capture the case when a genetic part is modified from an existing part, which may assist with reproducibility as the change in function can be tracked with sequence changes. However, provenance can also assist with a more query-based system, such as suggesting alternative parts. Provenance also encapsulates the idea of canonicity within databases, which ensures that each entity has a single definition. For genetic parts, it means that a canonical genetic part has a unique underlying sequence in the dataset, which guarantees an accurate comparison of designs using the same genetic parts. While useful alone, once initialised, the metadata can be updated given feedback (explored further in Chapter 5) from external sources, promoting correct and high-quality information and increasing the speed and accuracy of the synthetic biology community's access to design data. This approach aims to foster a more circular transfer of knowledge where the graph can learn from human interactions and tools interfacing with it[152]. The proposed system is a standardised weighted knowledge graph encoding a larger canonical domain of information (namely functional) with dynamic metadata to provide greater certainty of the presented information.

Aims

To effectively utilise data resources, various databases must be explored to identify potential datasets that can be incorporated into a sparsely connected network. Once suitable datasets are identified, the next step is to explore methods for normalising and integrating these datasets, which may be partially disparate. Finally, investigate strategies for enhancing the network's knowledge base, especially when starting with limited information by implementing network-based inference methodologies.

Objectives

The initial task involves constructing a foundational knowledge graph by unifying and normalising existing databases by integrating datasets into a singular, structured graph. After establishing the base, the goal is to expand the knowledge graph by incorporating newly inferred information by analysing patterns and relationships within the data to generate additional insights, including metadata, like confidence levels. Finally, the goal is to verify that the knowledge graph maintains

the desired features. It involves regularly assessing the graph to ensure it aligns with the predefined goals and requirements, such as accuracy, comprehensiveness, and relevance.

2 Results

2.1 Seeding an initial network by integrating networks

The natural growth of sparse and small networks can be slow, as described by Barabási in his study on scale-free networks, where a low degree often signals limited connections between nodes[153]. Such networks face significant challenges, notably in individual nodes' limited visibility and quantifiability. A practical issue arising from this characteristic is the difficulty in determining which nodes should receive new, partial information, such as data introduced into genetic circuits. Common regulatory elements like LacI-pLac, TetR-pTet, and araC-pBad, which are extensively characterised, can provide essential connection points that facilitate the initial formation of the network. When encoding with detailed functiona and structural information, these elements act as reliable hubs to which other data can be linked, effectively guiding the integration of new information and stabilising the network's expansion. To initiate the WKG, a high-quality dataset containing these common regulatory elements was employed to create a foundational core. This core allows for structured growth by ensuring that subsequent data additions enhance the network's robustness and functionality. The integration process starts with identifying and refining constituent datasets, which are then pruned for quality and redundancy, producing a subset that significantly improves the initial network. The methods and implications of this integration are demonstrated using three distinct datasets; each discussed in detail to illustrate their integration into the larger network framework.

- **International Genetically Engineered Machine (iGEM)**[135]: contains an extensive dataset with many genetic parts, much of the information encoded informally.
- **Cello**[154]: features genetic components similar to iGEM but defined under different names, showcasing the canonicalisation process in the WKG.
- **Virtual Parts Repository (VPR)**[128]: contains modular models of biological components, focusing on proteins and binding sites, explicitly encoding numerous interactions.

It must be noted that the process aims to be dataset-agnostic and does not exploit specific features of the datasets or individual records. These datasets are examples to display the process and how datasets specialising in different data types can supplement one another. However, combining data with low crossover creates an underlying network that will be disconnected at the points of the datasets.

International Genetically Engineered Machine (iGEM)

The International Genetically Engineered Machine (IGEM)[123] parts repository holds 50,000+ records with diverse information, including parts, composites, experiments, plasmids, cells, proteins, and project-specific details. However, two issues need attention. IGEM data is primarily in free text, making it challenging to extract information beyond sequences. Also, Synbiohub,

an SBOL-encoded repository, includes IGEM data but may have varying data quality and relevance because the records are directly translated from the free text IGEM parts. The process for preprocessing the IGEM parts repository involved removing inappropriate records and merging duplicates. When using Synbiohub with IGEM data, 38,365 records can be retained because this was the size of the data when initially translated into Synbiohub. However, data quality varies, necessitating a filtering process to remove irrelevant or low-quality records. Table 1 outlines each step of this pruning process, reducing the dataset's size and explaining the reasons for record removal.

Table 1: Overview of pruning process for the IGEM dataset before being integrated into the WKG.

Name	Removed number of records	Remaining number of records	Comment
Total	N/A	38365	The original Synbiohub dataset.
SBH Filter Type	24259	14106	IGEM categorises records into broad categories, and unwanted categories are removed.
Internal Components	472	13634	Records representing composites (multiple parts) are not required.
Invalid Sequence Data	792	12842	Remove entities with no sequence data.
Invalid Metadata	61	12781	The IGEM parts repository contains low-quality records which are uninterpretable and are removed based on tags.
Discontinued	667	12114	IGEM contains discontinued records which are removed.
Failures/Issues	256	11858	Parts that failed in tests are unused and removed.
Unused	4877	6981	Records are untested. Remove records which are not used in at least one other construct.
Duplicates	142	6839	Merge records with identical sequences.
Final	N/A	6839	The final number of records taken forward.

Cello

The Cello parts dataset[124] is smaller than IGEM but explicitly details interactions crucial for understanding genetic circuits in the Cello application, which converts design specifications into transcriptional logic circuits. Interactions are essential, as inferring this data from text descriptions can be challenging. Interactions, especially when represented as a network, can assist with

the abstraction of perceived complexity by enabling the consideration of abstract interactions instead of granular sequence data. However, the Cello dataset presents some new challenges. Firstly, It comprises two smaller datasets for different hosts, possibly with overlapping entities and interactions, requiring consolidation. Also, It defines non-DNA entities like proteins, complexes, and non-genetic molecules in interaction descriptions. Identifying duplicates for these entities is not straightforward. The process for preprocessing the Cello parts repository consists of merging the two smaller datasets. Table 2 displays the breakdown of omitted entities, where duplicates are merged or removed when redundancies (duplicates) are found within the datasets.

Table 2: Overview of the pruning process for the Cello dataset before integration.

Name	Removed number of records	Remaining number of records	Comment
Total	N/A	Components: 258 Interactions: 90	The original Cello dataset.
Internal Components	Components: 39 Interactions: 0	Components: 219 Interactions: 90	Records representing composites are removed because the dataset already presents constituent parts.
Sequence Duplicates	Components: 49 Interactions: 12	Components: 170 Interactions: 78	Merge records with identical sequences.
Inter-dataset redundancy	Components: 15 Interactions: 8	Components: 155 Interactions: 70	Datasets encode duplicate non-DNA entities. Merge duplicate entities and remove redundant interactions.
Final	N/A	Components: 155 Interactions: 70	The final number of records.

VPR

The Virtual Parts Repository (VPR)[128] is a model repository of virtual genetic parts, each representing a canonical entity. Genetic components in the VPR are represented using the Systems Biology Modeling Language (SBML)[71] and SBOL. The VPR dataset contains one primary issue that needs to be addressed. While the VPR dataset offers numerous genetic parts and interactions, many entities are unnecessary for this application. For instance, it includes low-level interactions like protein phosphorylation because they are constitutive interactions within abstract interactions. The process for preprocessing the VPR consists of removing inappropriate entities and interactions. However, it is free from duplicates, redundancy, or low-quality data. Table 3 outlines the pruning process, eliminating undesired or disconnected interactions to create a more focused dataset of components and interactions.

Table 3: Overview of the pruning process for the VPR Dataset before integration.

Name	Removed number of records	Remaining number of records	Comment
Total	N/A	Components: 12463 Interactions: 4583	The original VPR Dataset.
Unwanted Interactions	Components: 1070 Interactions: 266	Components: 11393 Interactions: 4317	The dataset contains low-level interactions, which are removed based on labels.
Disconnected Interactions	Components: 10882 Interactions: 4098	Components: 511 Interactions: 219	Remove protein production interactions when the produced protein does not partake in other interactions.
Final	N/A	Components: 511 Interactions: 219	The final number of records taken forward.

Integrating existing datasets

So far, the resultant datasets have not been unified and integrated into a network. Unifying disconnected datasets enables all information to be reasoned from a single point. For example, while the IGEM parts repository provides information on a large domain of biological parts, it does not capture more abstract information, such as formal interactions. However, the Cello and VPR datasets can begin to alleviate these knowledge gaps. Therefore, this section discusses how the final network is built, ensuring it is canonical, capturing provenance, encoding confidence in the information, and capturing contextual usage when integrating each dataset. The final stage (figure 8) combines all information within the built datasets, ensuring no duplicates, edges encode confidence, provenance between entities and common usage of parts are described. The process can be broadly categorised as extracting ComponentDefinitions (physical entities), Interactions, and all related information. For ComponentDefinitions, duplicates are checked based on the entity type (DNA or Protein, for example) to find if they are already present in the WKG. If this is true, then the name is added as a synonym on the existing entity, and if it is false, potentially similar entities are found, and a new edge is added to the most similar. Finally, the new node is added to the network if no duplicate exists. Interactions are also checked against the WKG for duplicates, and the confidence is increased if they are present, or a new interaction is added if not.

Canonical : Sequence alignment to gauge similarity between entities is the most comprehensive solution to match parts, as no user ambiguity is introduced. Therefore, between datasets (and within), two records are identified as synonymous if they contain the exact sequences (see

provenance for similar sequences). Confidence: Confidence captures the likeness that a piece of information is correct. It is encoded as weight and set based on the number of times the information was encountered during integration and can be dynamically updated (see feedback). While alone, this will not validate the information; quantifying confidence enables a user to gauge how much some data can be trusted.

Provenance : When a new physical entity containing sequence information is added to the network, existing entities are aligned with it, and the highest similarity is found. If the score exceeds a given threshold, a new edge is established between the nodes, denoting one being derived from the other. Because this new connection is not absolute, a higher sequence similarity implies a more likely relationship. The confidence of this edge is set based on the score. For example, a sequence similarity of 92% forms a confidence level of 0.92. Performing a sequence alignment for every sequence-based entity within the network upon adding a new node can be resource-intensive. As a result, a filtering process is implemented to mitigate this computational burden. This filtration involves identifying candidates with common attributes, such as being of the same type (e.g., promoter, operator, or terminator). Furthermore, candidates are selected based on the similarity in their sequence lengths, significantly narrowing down the pool of potential candidates.

Usage : Some datasets, such as the IGEM parts repository, contain records which define constructs, which are composed of other records representing genetic parts. While these records encoding composites are not added to the network, they can be used to identify entities commonly used together. The confidence of this edge is set based on the number of times the entities are found being used together within different records or designs. Also, the value is dynamic and, once set, can be updated based on new findings (see extracting designs).

Table 4 displays the breakdown of each dataset when integrated into a single network. From three datasets, IGEM, Cello, and VPR, 7,970 Nodes and 2,350 edges are present. Only 244 duplicates are present, 4 between databases, and 240 are internal duplicates. However, many more derived (similar) parts are found at 1,550, but only 17 between datasets. The small inter-set connections show that the datasets are largely unique but often contain similar internal entities. The result is that despite attempting to create a small connected network, most of the network is disconnected nodes or smaller components that only connect specific datatypes. Still, the network constitutes a small number of components which encode the required information. Figure 9 displays how a small sample of the final WKG appears, and three examples that combined display the desired network's features are achieved. The confidence requirement is described in the appropriate sections because the calculation depends on the datatype. Usage is described by an edge between entities (T7 and BBa_B0034), found together three times, resulting in a confidence of 0.15 (3*0.5 where 0.5 is the standard confidence step). Canonical entities are displayed with a node (BBa_K1897033) with an edge pointing to a synonym (BBa_K1893032) with a confidence of 1 because the sequences are identical and, therefore, conclusive. An example of provenance is displayed between two canonical nodes (BBa_K1897033 and BBa_K1893032) with an edge containing a confidence of 0.98 (98% sequence similarity). Several interactions are defined, which make up an instance of the LuxR repression system. The confidence is calculated based on the number of times found within the initial datasets (0.5 * 4, where 0.5 is the standard confidence step), resulting in a confidence of 0.20.

Table 4: Overview of the integrated datasets and their characteristics.

Integrated Dataset	Changes	Node Size	Edge Size
IGEM	Duplicates: 176 (All Internal) Derivatives: 1458 (All Internal)	Total: 6839 Parts: 6663 Synonyms: 176 Interactions: 0	Total: 1634 Synonym: 176 Derivative: 1458 Interactions: 0
Cello	Duplicates: 68 (4 IGEM 64 Internal) Derivatives: 53 (12 IGEM 41 Internal)	Total: 7240 Parts: 6926 Synonyms: 244 Interactions: 70	Total: 1876 Synonyms: 244 Derivative: 1511 Interactions: 121
VPR	Duplicates: 0 Derivatives: 39 (4 IGEM 1 Cello 34 Internal)	Total: 7970 Parts: 7437 Synonyms: 244 Interactions: 289	Total: 2350 Synonyms: 244 Derivative: 1550 Interactions: 559

2.2 Data expansion to introduce functional data

Three datasets were integrated during the seeding process. However, the resulting knowledge graph shows a notable disparity between the abundance of parts and the lack of interactions. Specifically, a substantial portion of the knowledge graph, comprising 7511 parts, lacks any encoding of interactions. The primary reason for this absence of interaction data is that the most extensive dataset (IGEM) does not provide any interaction data. Such data is only available when redundancies or duplicates are identified across different datasets. Consequently, many entities within the knowledge graph do not contain valuable functional information. As previously discussed, interaction data plays a crucial role in elevating design data analysis to a more abstract level. This abstraction, in turn, simplifies the comprehension of the dataset and enhances its potential for further development. When a substantial, well-structured dataset is represented as a knowledge graph, it becomes amenable to reasoning and inference, thus identifying connections and insights. Several methods are proposed to expand the WKG, alleviating this initial limitation.

Expanding data via domain knowledge

Designs and databases can hold information in abstract forms, namely interaction data, because practitioners often conceptualise abstract representations of interactions[155]. For example, within regulatory systems, proteins (and the interactions around them) are sometimes abstracted; instead, all interactions use the CDS entity. For example, CDS represses Promoter. This abstraction may be because designs are usually captured at the DNA level only, and proteins and non-genetic elements are implicit. However, capturing information more granularly is beneficial because it enables a more robust system that provides more significant insights and opportunities to gather information on more data types. Also, comparing designs via interactions encoded at different levels is challenging. As discussed, the WKG is encoded using semantic labels and within a standard format, allowing non-DNA elements to be captured and reasoned over using some level of

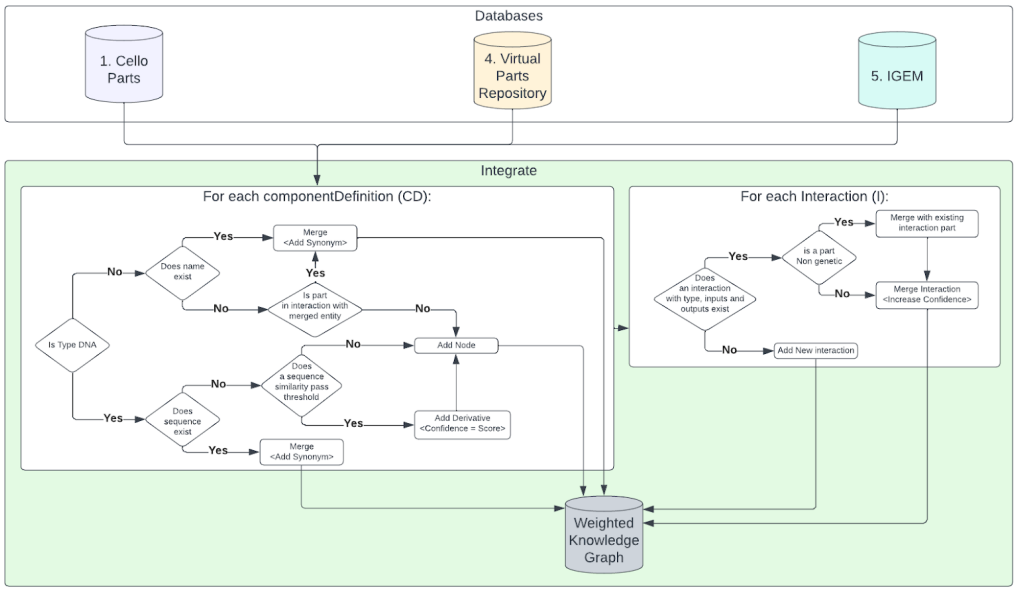

Figure 8: A overview of the integration steps when seeding the WKG. Each dataset is sequentially added into the final network while finding copies, derived entities and encoding confidence in relationships.

domain knowledge. Figure 10 displays how domain knowledge rules can automatically expand an abstract regulatory system.

- **Remove abstract interactions** - LacI - pTac repression and IPTG - pTac activation.
- **Add protein production interaction** - LacI-LacI.
- **Add repression interaction** - LacI - pTac.
- **Add complex** - IPTG-LacI.
- **Add binds interaction** - IPG - LacI - IPTG-LacI.
- **Add activation interaction** - IPTG - pTac.

While these changes appear trivial, they are required for more complex enhancements to take place because the level of detail is unified within the dataset, which allows more assumptions to be made in future.

Creating logical synonyms

Capturing synonyms enables matches between networks where the node labels are not identical, which is pertinent when parts often have multiple names. However, all synonyms within the WKG are currently created when duplicates are found when it was built. Genetic parts often

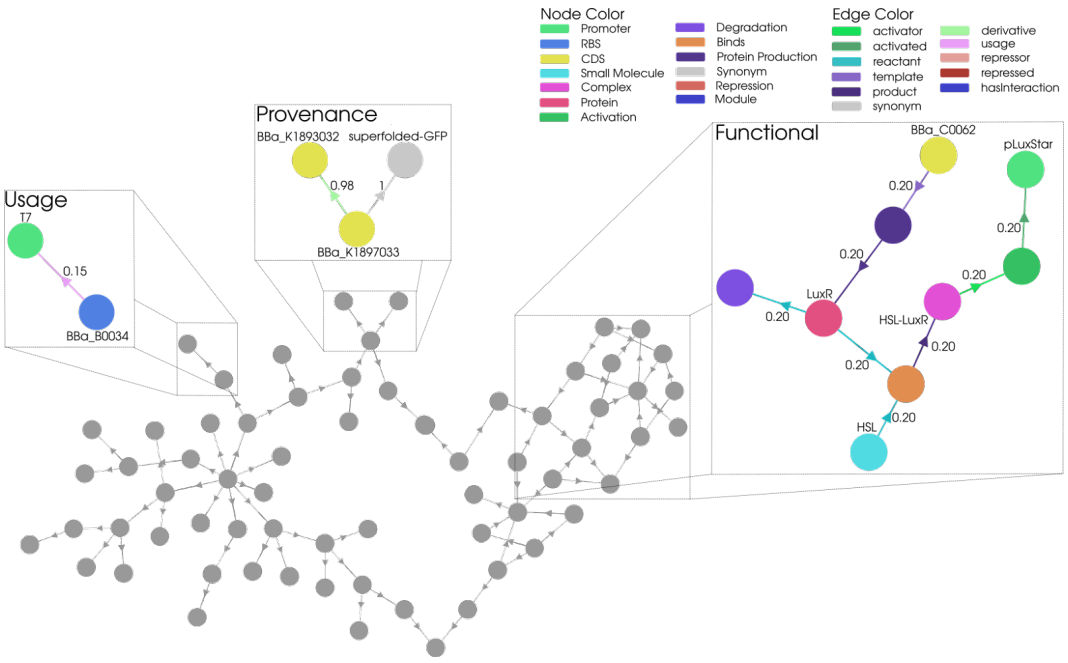

Figure 9: An example of features within the final integrated network. All instances contain weights calculated initially based on the integration of that datatype. Left - Usage is captured between the T7 promoter node and the BBa_B0034 RBS. The confidence is initialised based on the number of times these two entities are found within the same constructs. Middle - displays the provenance displayed by the sequence similarity between two instances of GFP and canonical shown by one version of GFP with a synonym node. The confidence is calculated based on sequence similarity. Right - Interaction data is integrated into the network, where confidence is calculated based on the number of times this interaction is described within all datasets.

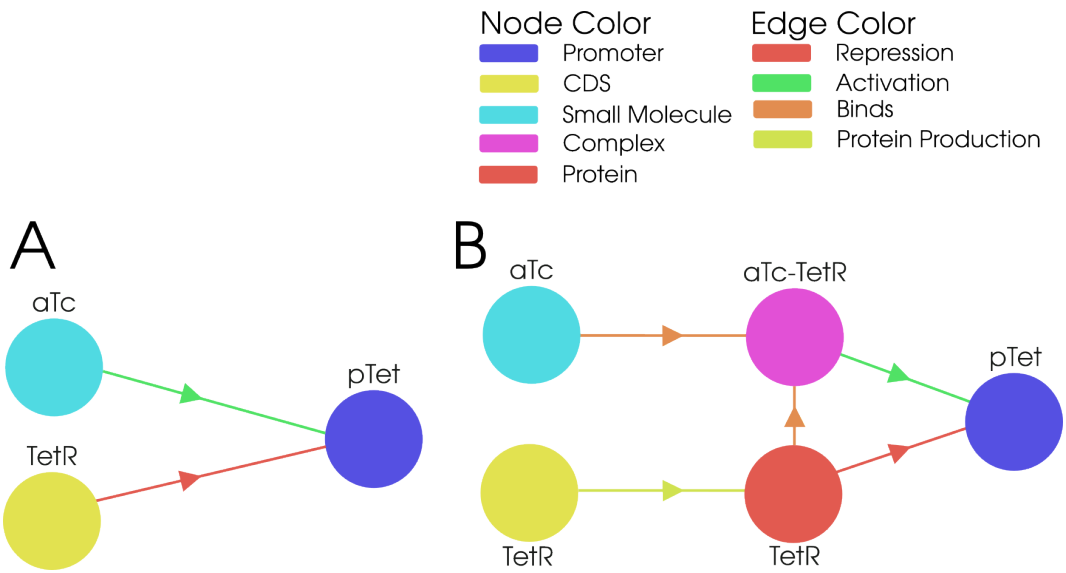

Figure 10: The expansion of abstract interaction data within the WKG. A) The TetR regulatory system is captured abstractly within a database. B) The same TetR regulatory system after expansion via domain knowledge, semantic querying, and manipulation.

have extra suffixes, such as genetic roles, added to the resource names. Therefore, a static set of synonyms can be added to the network based on the semantic labels denoting types and interactions attached to genetic parts. Figure 11 displays how synonyms can be automatically generated from information within the network, from simple additions such as adding the genetic type suffix to adding the role of a part on an interaction. Again, while these changes appear simple, they will enable easier identification of conceptually identical entities that do not reference the same resource. This enhancement is beneficial for comparing the WKG with external datasets such as existing design data.

Expanding interactions into derivative parts

As discussed, the weighted knowledge graph has two notable features: the ability to find similar genetic parts and the confidence that the information within the graph is correct. Currently, network interactions are only between the individual parts captured during integration. However, depending on the parts' similarity, the derivatives may encode the same interaction. Here, two features are used to add the interactions to the derived parts and set the confidence based on the similarity between the two parts. Figure 12 displays how new edges are added to connect derivatives (BBa_I732100 and BBa_I732103) of the "LacI" CDS to the interactions. The confidence of the derivatives and the interaction is known, and the confidence of the new edges can be calculated by interactionConfidence*similarity. Even though adding these new edges is a simple expansion, they enable identifying new potential interactions. While the information encoded by this edge could be implicitly derived, the confidence would be unknown. Only when the edge is inserted, and the confidence can be updated through feedback can the likelihood that the interaction is still functional between derivatives be quantified.

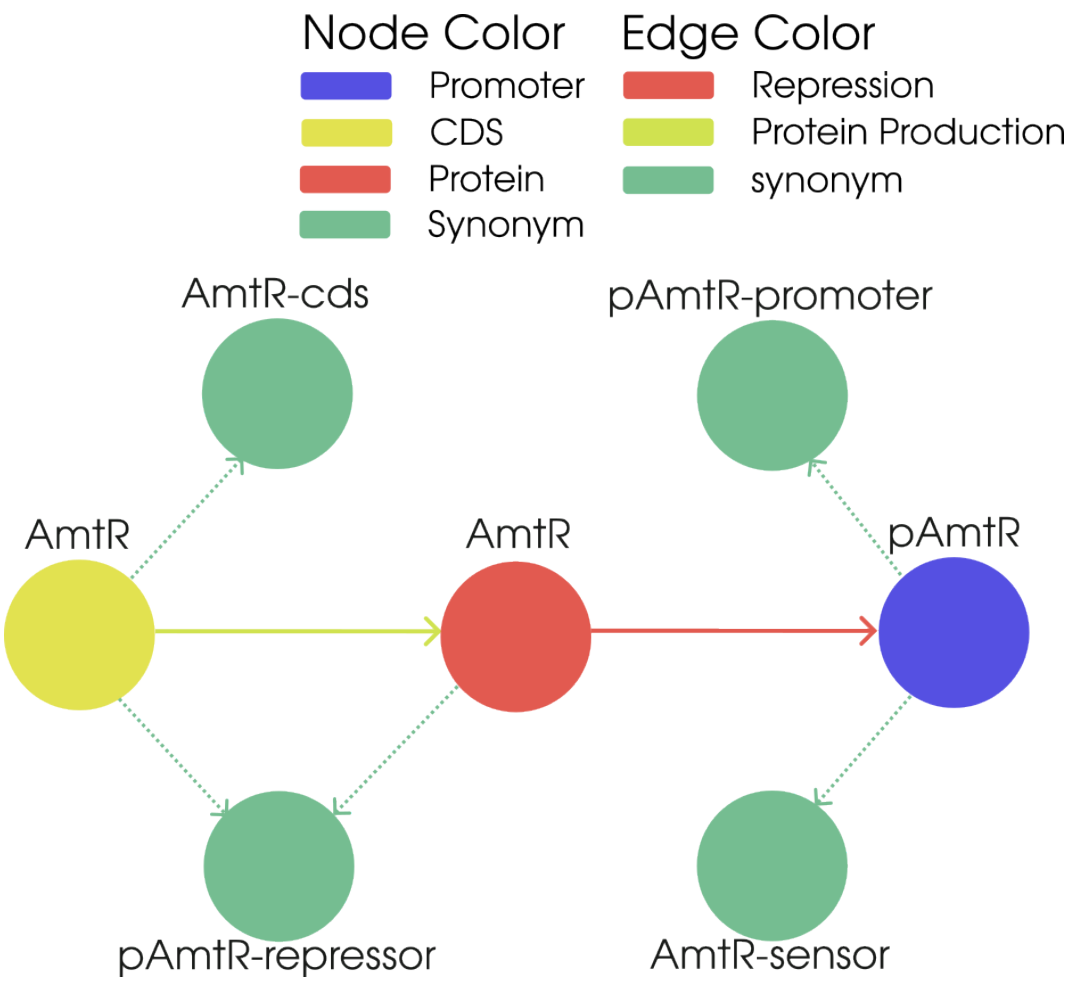

Figure 11: A network representing AmtR repression. New synonyms are created from features of nodes and edges within the design using simple static rules. Four new synonyms are added with new synonym edges to the relevant nodes.

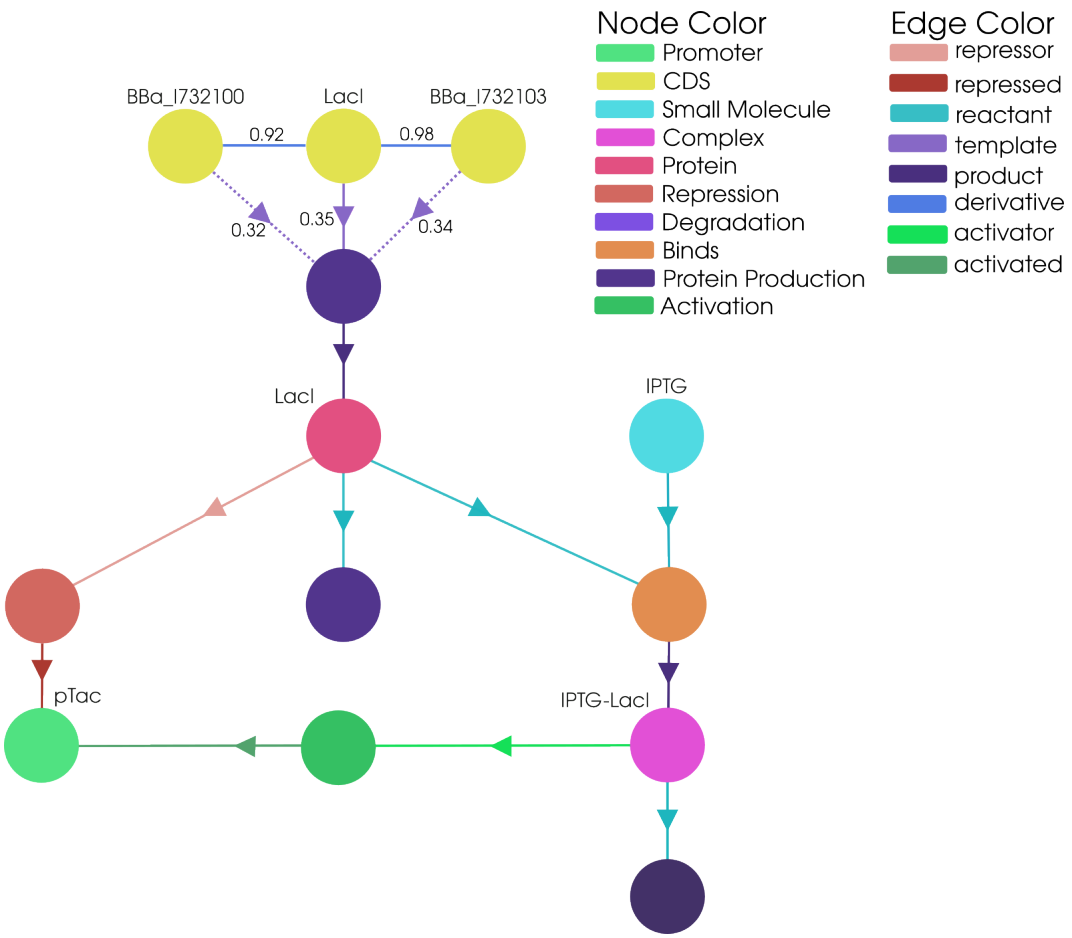

Figure 12: The derivatives expansion takes existing interactions and adds derivatives of participants. The derivatives (BBa_I732100 and BBa_I732103) of the "LacI" CDS have new edges (dotted lines) into an existing interaction component. The confidence is calculated by parentConfidence * similarity.

Extracting information from existing designs

Databases often play a pivotal role in refining and advancing genetic designs. Nevertheless, these designs may contain unknown or extra information, as well as supporting or conflicting evidence. Consequently, established designs can serve as valuable inputs into the WKG, contributing to its expansion and enhancing the accuracy of the existing knowledge. Integrating design information from diverse sources can be challenging when the underlying semantics are not unified, and conceptually synonymous entities may reference different resources. However, the same techniques employed in identifying synonymous or analogous entities during dataset integration can also be applied to identify canonical entities connecting the design with the WKG. The process of transferring data from the design graph to the WKG can be divided into two key steps: firstly, identifying matches between the two graphs and, secondly, extracting the relevant data. Figure 14 will be referenced throughout this discussion, which illustrates interactions and metadata integration into the WKG. It is important to note that this figure represents just a small sample of the WKG and the potential extractions that can be made.

Identifying matches

- A direct match is made where the resource URI in both networks is identical. Figure 13 displays a direct match between the "ATC" input chemical.
- A synonym match is made where the resource in the design graph matches a synonym within the WKG. In this case, the canonical entity is found and used instead. Figure 13 displays a synonym match between "TetR" and BBa_K1475003.
- No resource name matches are made, but a direct sequence match is made. Figure 13 displays a sequence match between "TetR_sensor" and "pTeT".
- No match is made using any metric. If the information can be extracted from the design graph, the design graph entities are inserted into the WKG as the canonical values.

Extracting data Once entity matches have been established between the design and the WKG, the next step involves extracting and integrating this information into the WKG. This extracted information may pertain to explicit design data, such as interaction data, as shown in Figure 13, which includes adding the TetR regulatory element. Additionally, it may encompass metadata, such as the identification of new relationships, like the synonymous relationship between "TetR_sensor" and "PTeT," or the common association of "mCherry" with the same promoter. However, some interactions may be context-specific and unsuitable for WKG inclusion. For instance, interactions predicated solely on the relative position of genetic entities within a sequence should be omitted from the integration process. Also, certain interactions may already exist within the WKG, albeit with different IRIs. In such cases, a conceptual match must be identified (as described in the methods section), meaning that the interaction type and the participating entities must align between the design and the WKG interaction.

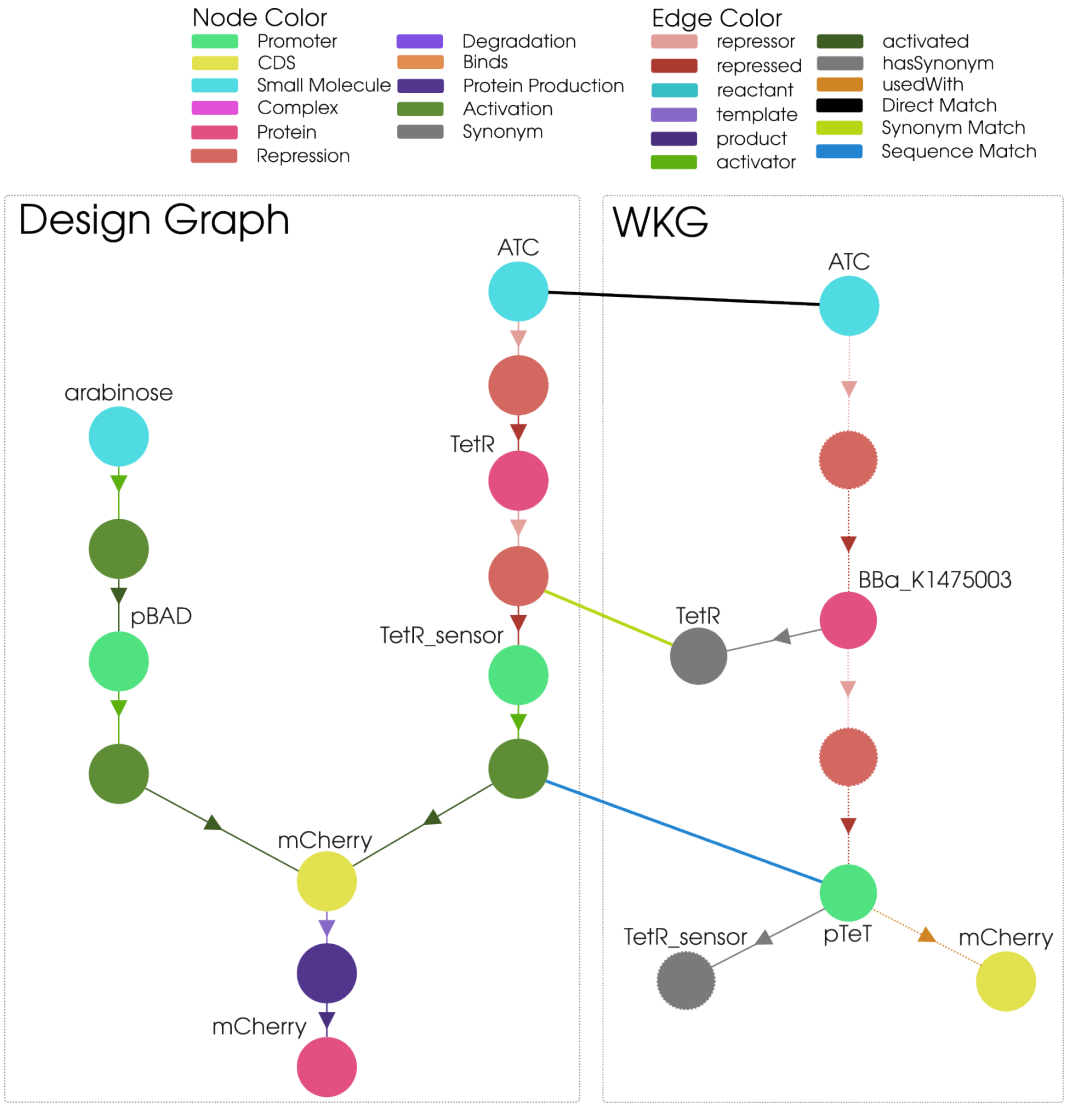

Figure 13: Sample example of extracting information from an existing design into the WKG. Dotted lines represent new information in the WKG based on the extraction. Three matches are made between the design and WKG: a direct, synonym and sequence match. When matches are made, the information is extracted. Metadata can also be extracted by matches (For example, the "TetR_sensor" is extracted as a synonym for the "pTet" entity within the WKG).

Inferring missing interactions by example

Incomplete interaction networks are common within existing designs. Missing interactions may occur for several reasons. Firstly, it is unknown to the person encoding the design due to an oversight or gaps in knowledge. Alternatively, it is implicit within the design and not explicitly encoded because the creator assumes a level of self-evidence. Partial interaction subgraphs may be adapted from existing designs because they were assumed to be complete. Partial interaction networks can cause many issues when captured within the WKG because they can cause half-truths to be perpetuated into future designs[156], introducing more uncertainty. A network-based methodology incorporates the concept of topology, which involves the examination of how nodes and edges are organised within a network. This approach aids in comprehending the fundamental structure and characteristics of complex systems[157]. When examining the topology of interaction networks, certain recurring patterns become evident among interaction components[158, 159]. As illustrated in Figure 14, this represents a subset of potential motifs that can manifest within the interaction networks (for details on motif identification, refer to the methods section). These motifs include small regulatory mechanisms like "negative feedback"[159] and "repression feedforward"[159] to more intricate systems like the NOR gate[160]. These predefined shapes can provide insights into potentially absent connections within an established interaction sub-network in conjunction with semantic labels. Figure 14 illustrates the process of identifying a missing edge based on the presence of existing motifs. When the "repressor feedforward" motif is detected as a partial match within the sub-network encoded in the WKG, the missing Activation displayed with the dotted nodes and edges can be introduced.

Defining low-level functional modules

A functional module in the design context refers to a discrete, self-contained design unit intended to perform a specific function or set of functions. These modules can range from small regulatory constructs to large logic-based systems and beyond. Abstract modules defining groups of interacting entities provide several benefits.[161]. Modules can reduce the perceived complexity of larger designs with many interactions by abstracting some unnecessary detail. Also, modules enable substituting design aspects which can perform the same function through different mechanisms. A modular approach can provide solutions for scenarios where the desired abstract function is known without a specific implementation. These modules are curated to contain entities often utilised together, increasing the likeliness of correctness when integrated into a design. The WKG, which encodes context-independent information, contains a subgraph encoding interactions between entities, consisting of many components. The projected network encoding interactions are composed of components because some interactions and the resultant physical entities (participants of an interaction) are not related to other genetic parts. For example, the LacI regulatory system does not affect the TetR regulatory system; therefore, no edges between them are present. Therefore, low-level modules can be derived that encapsulate these components into modules because all of the entities within the components are inherent to the system. Interaction components are not always present within the whole network because other data types, such as meta connections (derivatives, synonyms and usage), may connect nodes which do not have functional relationships. Therefore, a subnetwork is projected to identify the interaction components containing interactions and the physical constituent participants. Figure 15 displays a sample of the

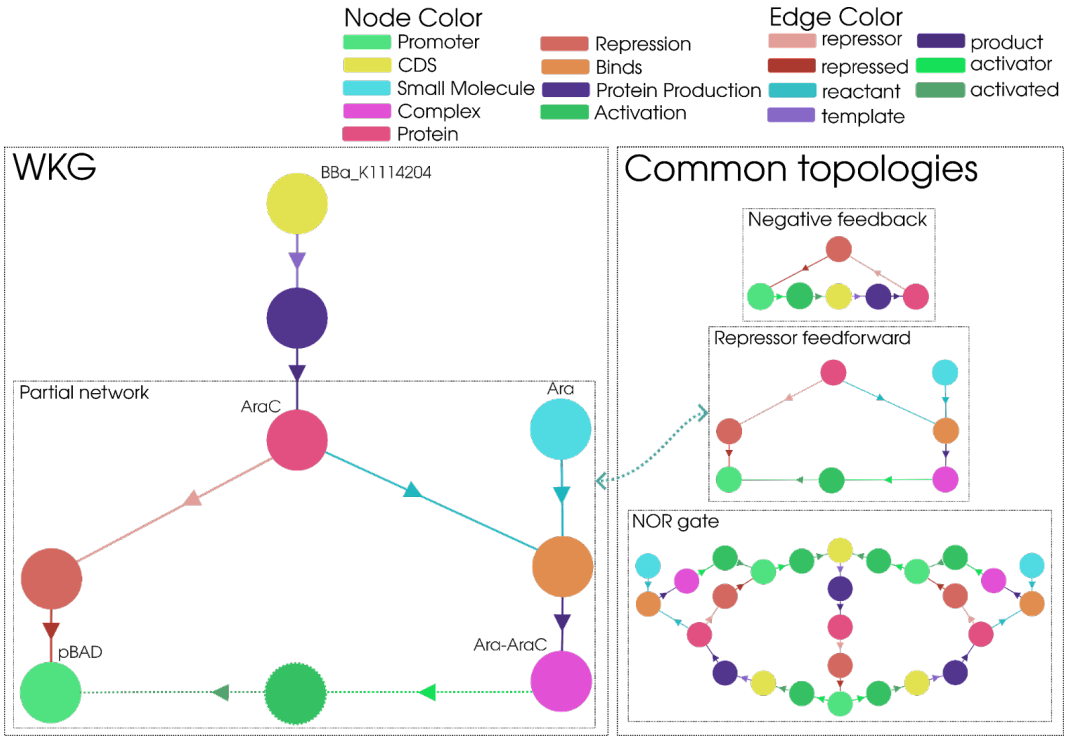

Figure 14: Matching a partially complete interaction network with complete generic motifs. Left: The partial interaction component encoded within the WKG. Right: A sample of generic motifs without specific implementation details. The motifs are matched with subgraphs within the interaction network until a close match is made. The new activation node and edges are inserted into the WKG.

underlying interaction network (HyllR and AraC regulatory systems). The interaction projection is a bi-partite, directed graph and is achieved by identifying each interaction within the network and connecting the inputs and outputs from each interaction. The Weakly Connected Compo-

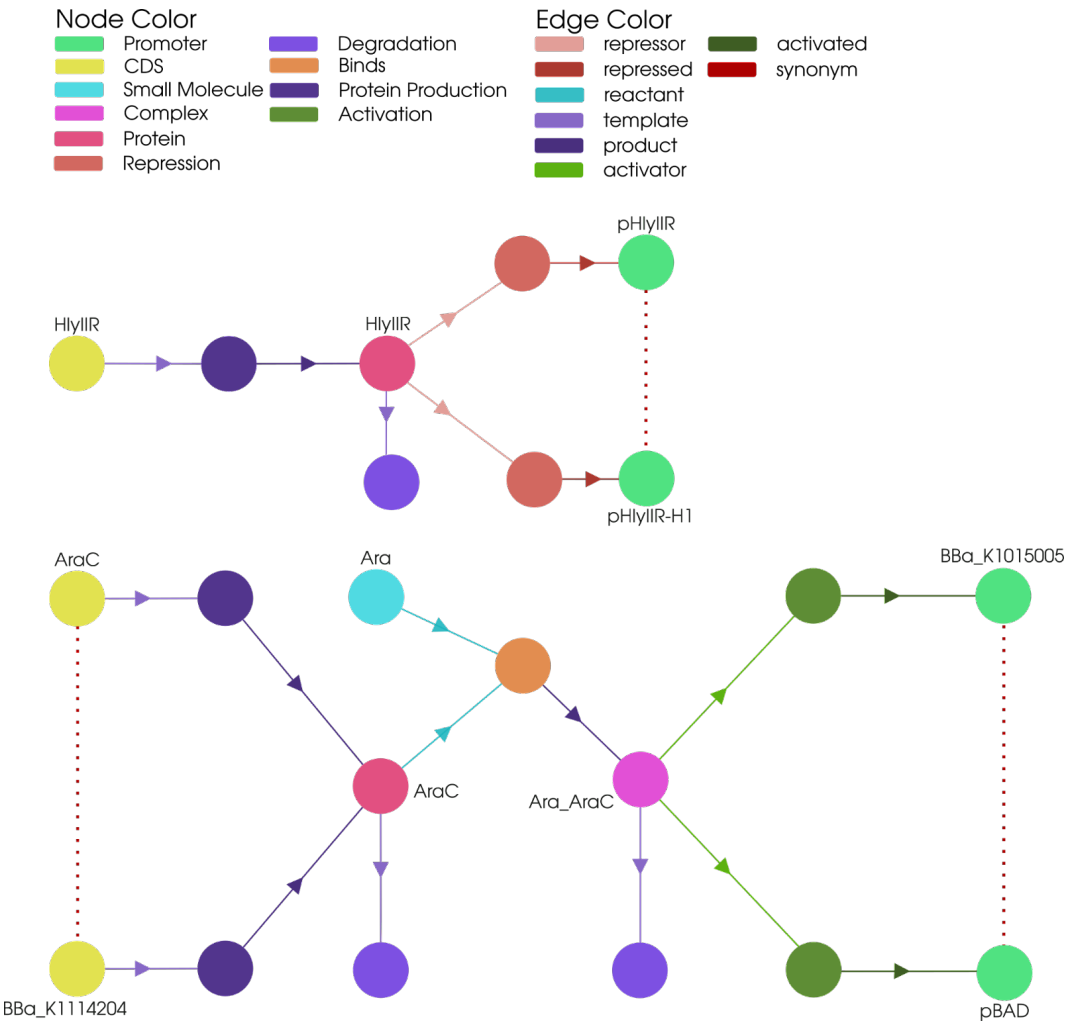

Figure 15: Sample of two components from the underlying interaction network projected from the WKG. Top) The HyllR repression system. Bottom) The Arac regulatory system. The interaction components may contain multiple versions of the same part, which are one another's derivatives. For example, the pBAD promoter has three versions.

nents (WCC) algorithm can quickly identify the interaction components within the projected network. The algorithm finds sets of connected nodes in directed and undirected graphs. Two nodes are connected if a path exists between them—the set of all connected nodes form a component. Derivatives within the network define similarities of genetic parts within the network. However, derivatives may commonly encode the same interaction that shares participants, meaning that network components may not directly map to a module. This feature is present within Figure 16; for example, the AraC and BBa_K1088005 nodes are derivatives connected within the

interaction projection. Therefore, the individual systems must be extracted based on derivatives to define each logical module from the topological components. From the two components within Figure 15, the inputs HlyIIR, AraC, and BBa_K1114204 and outputs pHlyIIR, pHlyIIR-H1, BBa_K1015005 and pBad are derived. To identify logical modules, inputs and outputs are identified based on each node's degree (number of edges). Inputs are categorised as nodes with a zero in-degree, and outputs as nodes with a zero out-degree. Next, inputs are grouped and duplicated where necessary, ensuring that each set of inputs and outputs does not contain derivates. Once the inputs and outputs are identified, the projection is traversed from each input to each reachable output. Six groups, displayed within Table 5, are derived from the initial inputs to ensure no overlap of derivatives.

Table 5: The logical groups taken from components within the interaction projection graph. Each group contains the input and outputs of the component without any derivatives.

Group Number	Inputs	Outputs
1	HlyIIR	pHlyIIR
2	HlyIIR	pHlyIIR-H1
3	BBa_K1114204, Ara	BBa_K1015005
4	BBa_K1114204, Ara	pBAD
5	Arac, Ara	BBa_K1015005
6	Arac, Ara	pBAD

For each grouping defined within Table 5, a traversal is performed from the inputs to the outputs displayed in Figure 16, where a different coloured line defines a group of paths. The combination of paths generated within each group is defined as logical components. Finally, once the logical components are identified, they must be added as new conceptual entities within the WKG. Each path is concatenated to produce the final module. Within the network, a module comprises a node denoting the module and new edges to each interaction which constitutes the module. The interaction edges are the only ones required because all other information (Inputs, outputs, and parts, for example) can be derived. Figure 17 represents the new modules when inserted into the interaction projection. Each container border is coloured to represent a specific path established within Figure 18 to display how these paths map to the newly established modules.

Genetta

During the studies conducted within chapters 4, 5 and 6, a tool was developed which implements the procedures and methods described. Here, the WKG (and the methods to normalise, clean, integrate and expand it) is enabled by Genetta. Furthermore, the use cases described within Chapter 5 (advanced query and design enhancement) are implemented, and the visualisation techniques from Chapter 6 can be applied to the WKG. Figure 18 displays an example of how the WKG can be visualised within the visualisation tool discussed in Chapter 6. Finally, the tools use cases discussed in Chapter 5 are implemented within Genetta utilising the WKG discussed here. Alongside the WKG and related methods, the application includes the following:

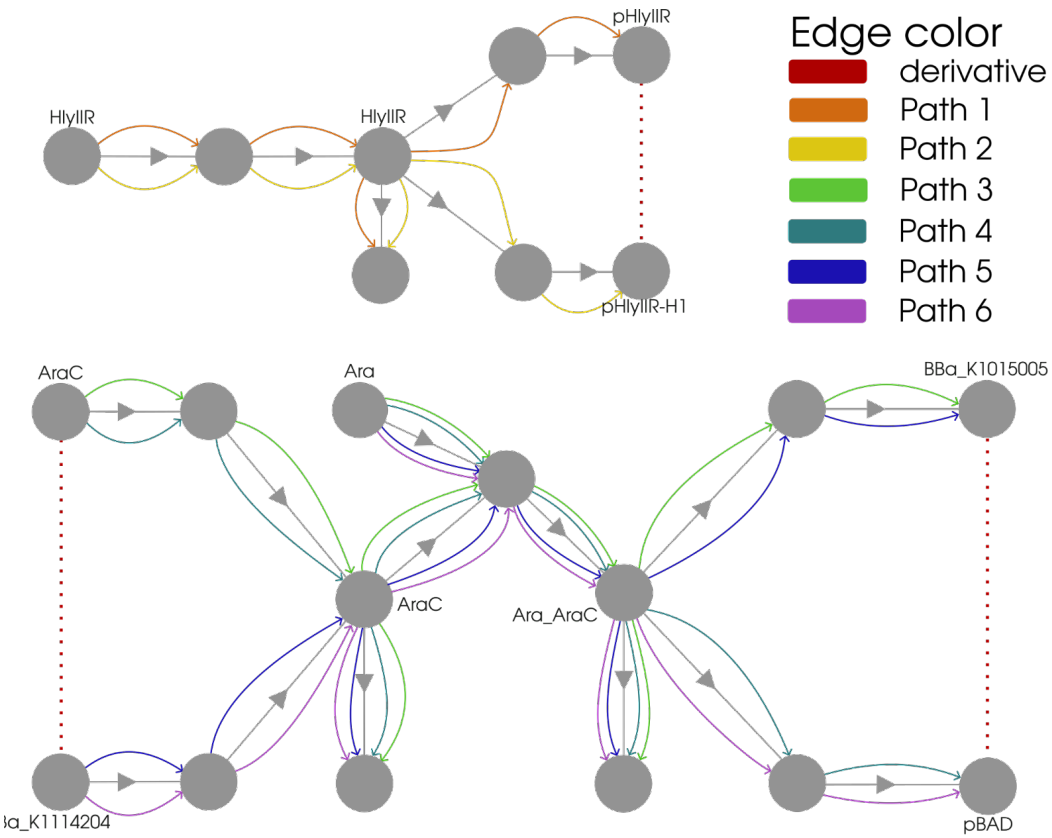

Figure 16: Identifying the logical modules from a sample of interaction components. The inputs and outputs are grouped and duplicated based on derivatives, so derivatives are never in the same path. The graph is then traversed from each input and each output, and the paths are saved.

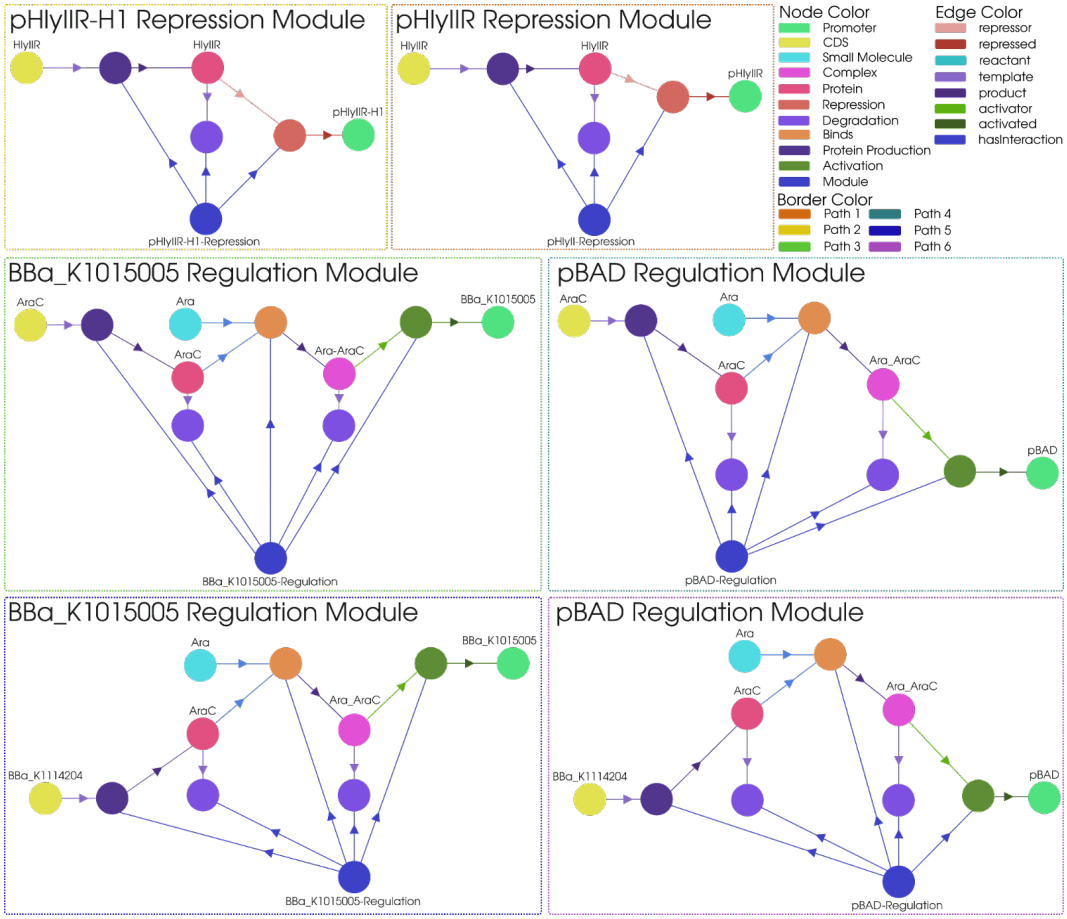

Figure 17: A representation of new module nodes within the interaction projection. The module nodes connect to the constituent interactions which make up the module using the "hasInteraction" predicate.

- Instructions on how to run all tools within the application.
- Documents describing use cases and examples.
- Independant access and manipulation (add, remove and export) of designs uniquely assigned to users.

The software application is accessible from the repository at Genetta-Github, and an online instance can be sampled here: Genetta-instance. Technically, the tool comprises two parts: the back-

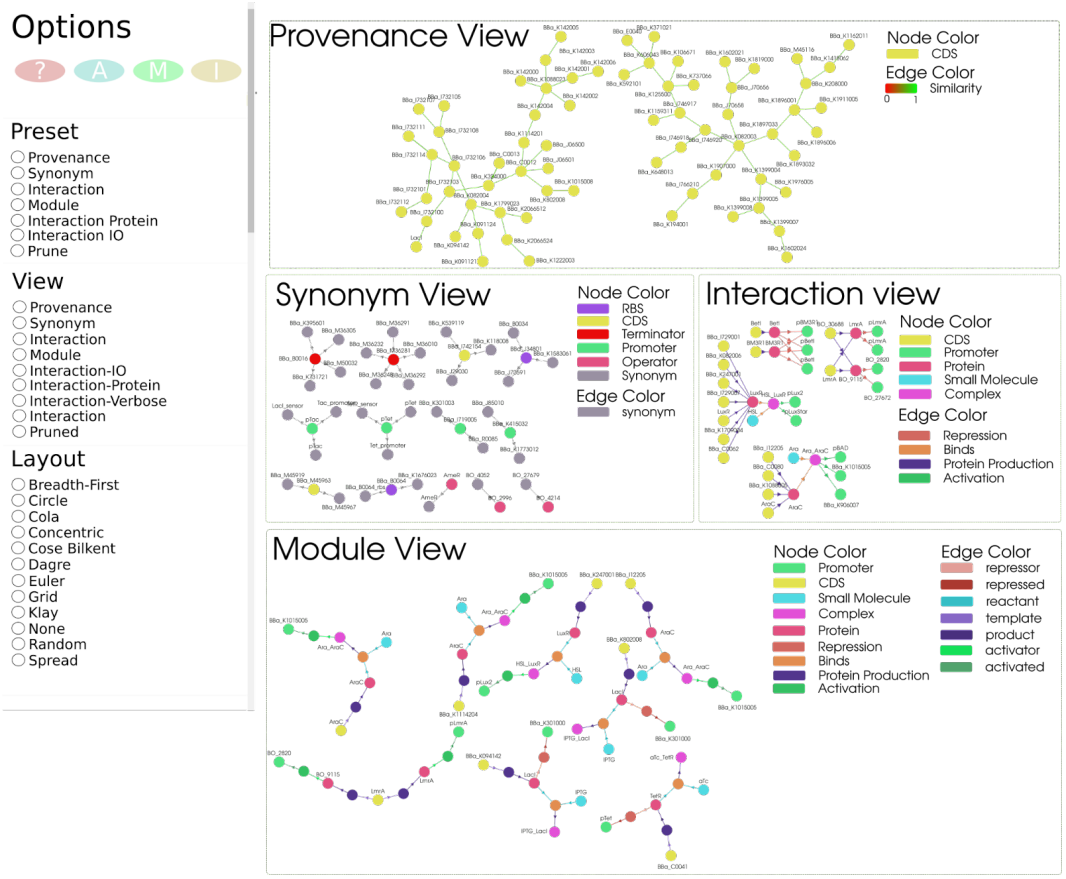

Figure 18: A representation of new module nodes within the interaction projection. The module nodes connect to the constituent interactions which make up the module using the "hasInteraction" predicate.

end, which stores the graph and the frontend, which is presented to a user for usage. The two parts are distinct such that different tools can interface with the neo4j environment, holding the WKG without adopting the Genetta frontend systems each time. The backend comprises a neo4j datastore that holds the weighted knowledge graph and any designs uploaded by users in a single graph. The underlying graph contains features that enable robust and safe access to the WKG and design subgraphs.

- Each node and edge contain a property describing the name of the containing graph.
- Nodes and edges of more than one design will have multiple graph names, ensuring the datastore is not cluttered with redundancies. However, entities within the WKG will only encode the WKG graph name, ensuring the graph is disconnected from any other data.
- A node within the datastore will always consist of two labels: the name URI and the type such as Promoter, Activation or Synonym.
- Edges within the datastore will always have a single label describing the type, for example, repressor, hasInteraction, or similarEntity.
- All entities within the WKG are structured given a set of semantic terms and structural rules described below.

The front end provides users access to the WKG and holds the tools implemented. Genetta ensures that all subgraphs coexist within the same neo4j environment by providing an abstraction layer over the service using the previously discussed network features.

- Genetta maps graph names to usernames when a new graph is introduced, and users can not access or manipulate graphs they do not own. When manipulation occurs on an entity with multiple graph names, the entity will always persist in its original form.
- Users cannot directly interface with the WKG and must use tools such as query, enhancement or visualisation systems. If a case occurs where a user can directly access the WKG, for example, from the fault of a tool developed in future, specific keywords to modify the graph are blacklisted within queries.
- When a user changes one of their designs explicitly or implicitly, these changes are made to the subgraph within the data store. However, they are also staged as changes to be made to the original data stored on the server. When the design is exported, the changes are applied sequentially to the design.

The WKG is powered by the FLASK micro web framework, which means that a user is not required to install software or perform any technical tasks and can access the services within the web. The frontend and backend communicate via the BOLT protocol, a neo4j-specific communication protocol carried via typical TCP connections.

3 Methods

3.1 Networks

This chapter focuses heavily on network approaches, especially integration and unification methods. While the background chapter has covered some general network concepts, this section expands on some processes used explicitly within this chapter.

Edge weights to represent strength

Edge weights in a graph are numerical values linked to the edges. These weights express different attributes or properties associated with the connections between nodes. In this context, the weights specifically indicate confidence levels, although they have the flexibility to represent any numerical value. Technically, edge weights are attributes assigned to an edge, characterized by a string key and a corresponding numerical value. In Figure 19, a segment of the WKG is showcased, illustrating various types of information. Notably, each edge in the network has a weight, serving as a measure of confidence.

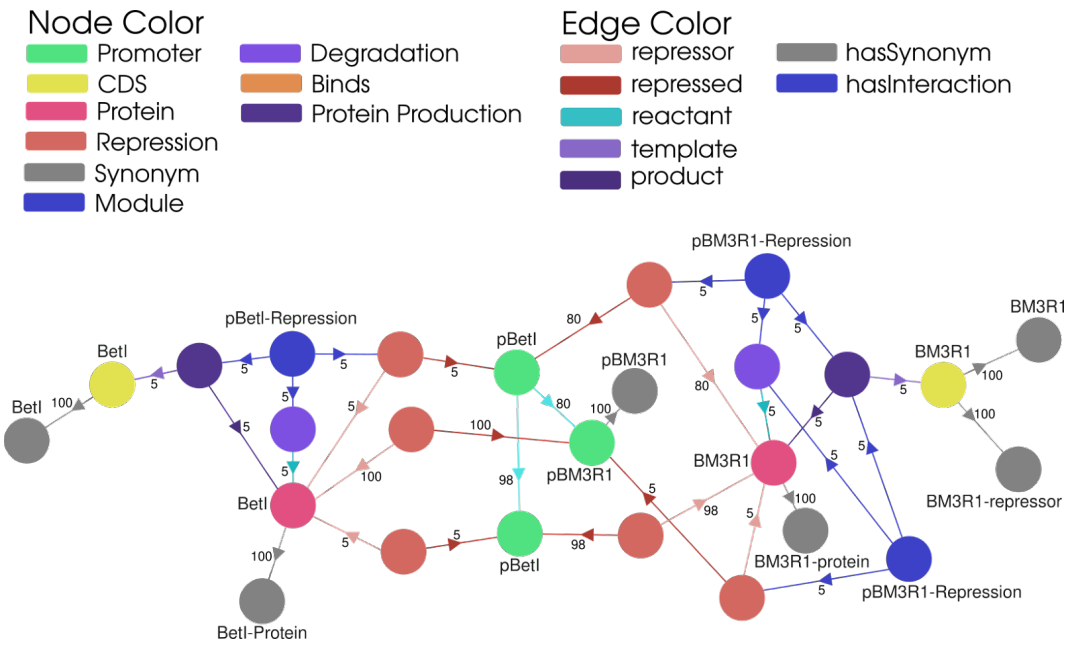

Figure 19: Example of a component from the WKG encoding multiple types of information (Interactions, Synonyms, Derivatives and Modules). The edges display the confidence weight, denoting the likeness that the relationship is valid.

Graph traversals to find routes in networks

Pathfinding in networks is critical for identifying efficient network routes or connections. It aims to determine the optimal sequence of nodes and edges between a network's source and target node. In this context, pathfinding is essential for connecting genetic entities and understanding how they relate. Within design data, it utilises established algorithms, such as Dijkstra's algorithm[162], which may consider the weights assigned to the edges within the network. These algorithms consider various factors, including the confidence and usage metadata associated with each connection, to determine the most suitable paths based on specific criteria. Pathfinding using weights has been used several times within this section, for example, to identify missing interactions and the confidence of the missing edge based on the confidence of the nearby interactions or, as displayed within Figure 20, to calculate the confidence of the edges constituting a new mod-

ule. Here, the initial confidence is set by traversing to the participating physical entities using the shortest path to each, where the cost to pass the edge is the confidence. Because higher confidence indicates an edge to be more likely valid, the highest cost is taken here. Once all paths are found, they are added and divided by the number of paths.

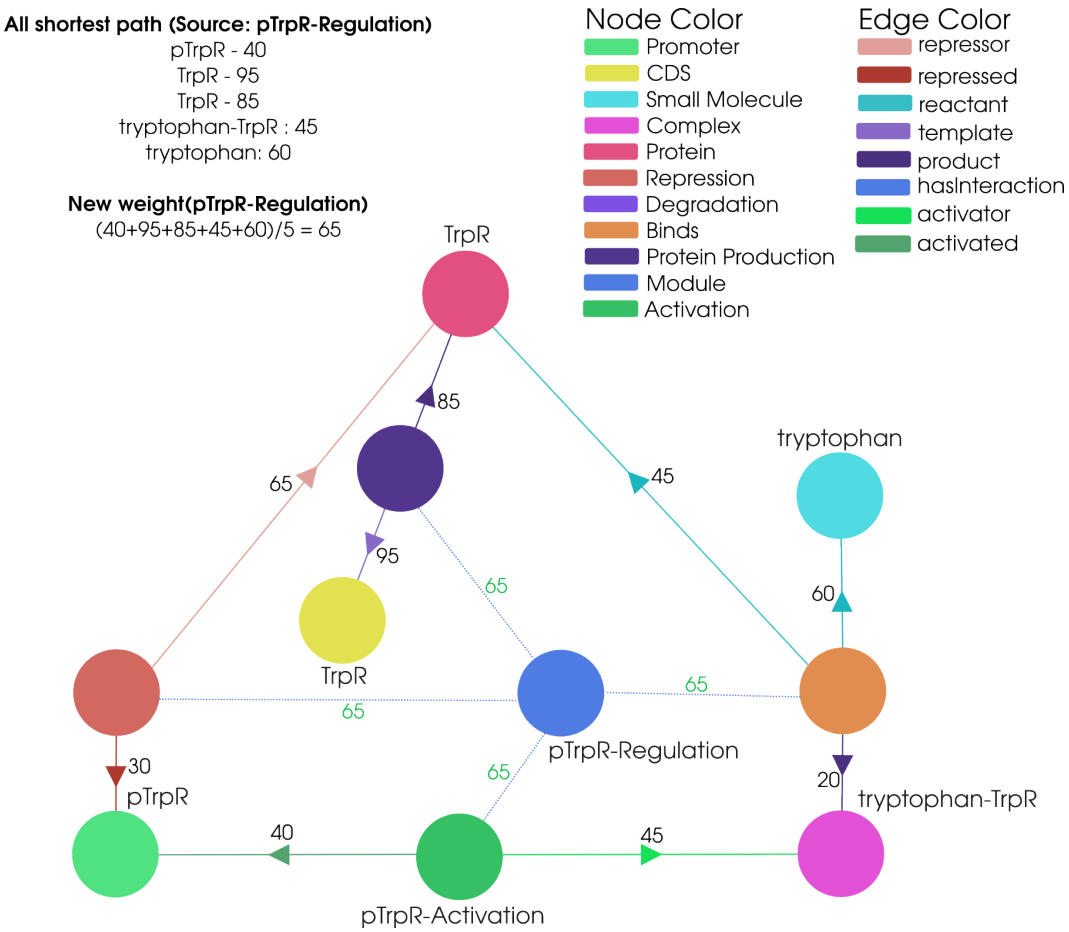

Figure 20: Example of how a new module's confidence can be set based on the confidence of its constituent interactions. From the source node (pTrpR-Regulation), the shortest path is found to each participating entity, the weight is the cost to traverse an edge, and the highest score is taken because higher confidence indicates more likeliness of being correct. All paths are combined and divided by the number of entities to create an initial confidence of 65.

Weakly connected components to identify modules

Community detection refers to identifying groups or communities of nodes within a network where nodes within the same group are more densely connected than nodes in other groups. These communities are often considered substructures or clusters within the more extensive network. Modules are subsets of nodes within the network that share a high degree of interconnectedness, which could represent coherent functional units[163]. Weakly connected components

(WCC)[164, 165] refer to sets of nodes within the network with a path between any two nodes and were used to identify modules. WCC is the simplest form of community detection. A WCC of a directed graph is a subgraph in which all vertices are connected by some path, ignoring the direction of the edges. The process begins with the projection of the interaction network. Next, begin traversing the graph from a given node. If two nodes are reachable, they are part of the same component. The algorithm continues its traversal until all nodes in the graph are visited and assigned to a weakly connected component. Figure 21 displays an example of identifying two weakly connected components. It simply identified the two components within the network because they are not connected.

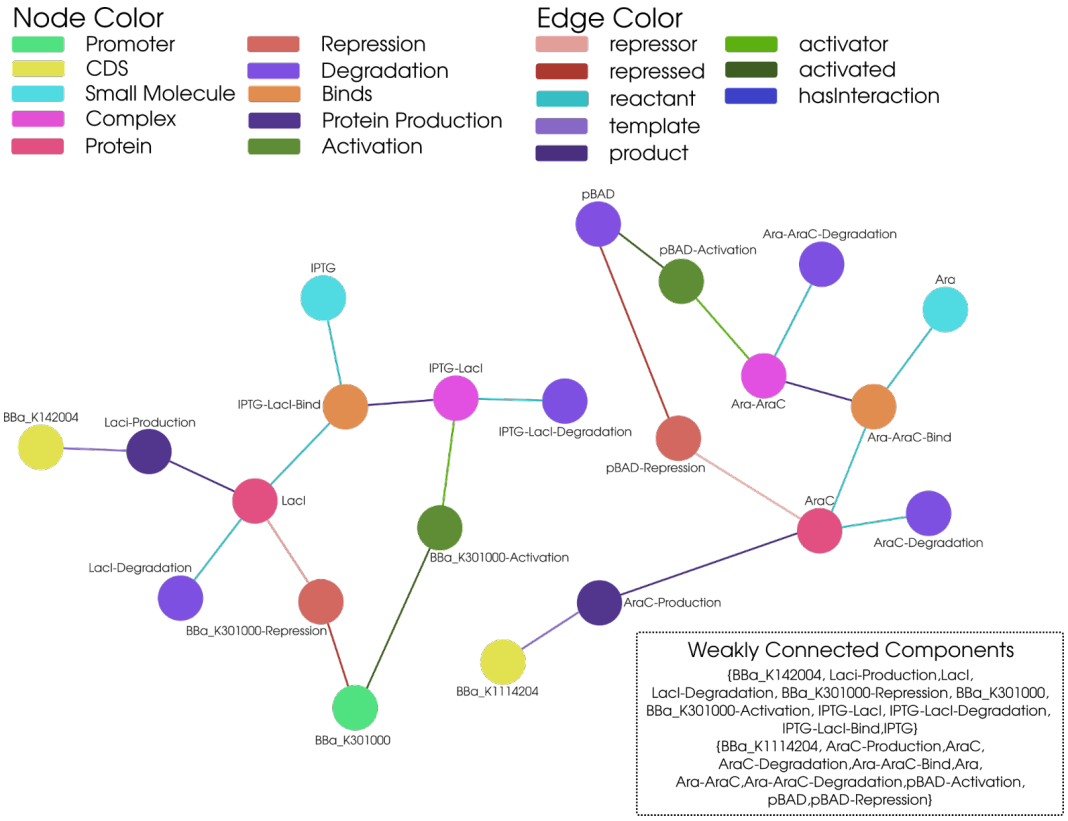

Figure 21: Example of identifying weakly connected components from a subnetwork. Each component constitutes a weakly connected component.

Conceptual equivalence

Examples throughout this section have displayed short names for any entities, enabling comprehension when graphs are presented visually in this work. However, knowledge graphs utilise Internationalized Resource Identifiers (IRIs) to identify resources, uniquely offering human-readable and computer-readable naming conventions. For instance, when "LacI" has been used as a label within a real network, it would correspond to an IRI such as `https://synbiohub.programmingbiology.org/public/Cello_Parts/LacI/1`. While IRIs provide unique identification, they do not guaran-

tee the encoded information's equivalence or conceptual uniqueness. Different IRIs may represent the same genetic part, such as encoding identical sequences. Equivalence is evaluated through two metrics: referential equivalence (when two entities directly point to the same resource) and conceptual equivalence (where IRIs, while not identical, contain equivalent information). Identifying referential equivalence is simple because the IRIs will be identical. Identifying conceptual matches has been critical for unifying datasets because different datasets seldom have referential equivalence. **Physical entities** are equivalent when the underlying sequences are equal. **Interactions** are equivalent when the interaction type, interacting entities and how they interact are equal.

Identifying Provenance

In this chapter, provenance is defined by a network with no duplicates, and edges are present when the underlying sequences of two nodes are similar. A simple direct comparison can be made to identify if two nodes have the same sequence. However, identifying the similarity between two sequences is more challenging, especially with many sequences to match, as in the case of the WKG. Here, sequence similarity is achieved using sequence alignment, a fundamental technique to compare and identify similarities between two or more biological sequences. However, sequence alignment on each sequence-based entity within the network each time a new sequence-based physical entity is added is not feasible from a computational perspective. Therefore, filtering can be applied to reduce the number of candidates. Firstly, entities not of the same genetic role as the input node can be removed. For example, a Promoter and CDS clearly can not be derivatives. Next, if an input is aligned with a node in the dataset and they are not a match (the score is below the threshold), then all of the derivatives of the existing node within the WKG can also be removed as candidates. When integrating a new node into the network, these filters reduce the number of alignments by approximately 80% on average.

Establishing motifs

Motifs were used to identify potentially missing interactions during expansion[159]. Initially, they were defined as a static set of networks. This initial set is required to seed the WKG because extracting motifs from a graph known to be incomplete will produce incomplete motifs. Later, the template motifs could be expanded using the WKG itself. If it can be guaranteed that areas of the WKG are complete, then motifs can be extracted from it. For instance, if a particular topology was frequently observed, its generic structure could be extracted and applied to complete missing interactions in different regions or when new subgraphs were introduced. Figure 22 displays an extended set of motifs defined beforehand or generated from extracting shapes within the WKG. It must be noted that two motifs may be topologically identical but contain different physical entities or interactions at different points. For example, there could be a repression bi-fan, which has the same topology as the activation bi-fan but has a different semantic makeup. With large networks, identifying the instance of subgraphs can be computationally intensive. Subgraph isomorphism, a key component of subgraph matching, is an NP-hard problem. Determining whether a given graph contains a subgraph that is isomorphic to another graph (the motif being searched for) falls into the class of NP-hard problems, which means no known algorithm can solve the subgraph isomorphism problem for arbitrary graphs in polynomial time. As the size of the graphs increases,

the computational complexity of finding subgraph isomorphism grows significantly. Usually, approximations are made because of this; however, the motifs here can be identified within the Interaction projection (a subgraph containing only interaction data), which is considerably smaller than the full network. Furthermore, semantic labels can help further reduce the search space. For example, regions of the network that do not have nodes that match the type labels of the motif in question can be removed from the candidates.

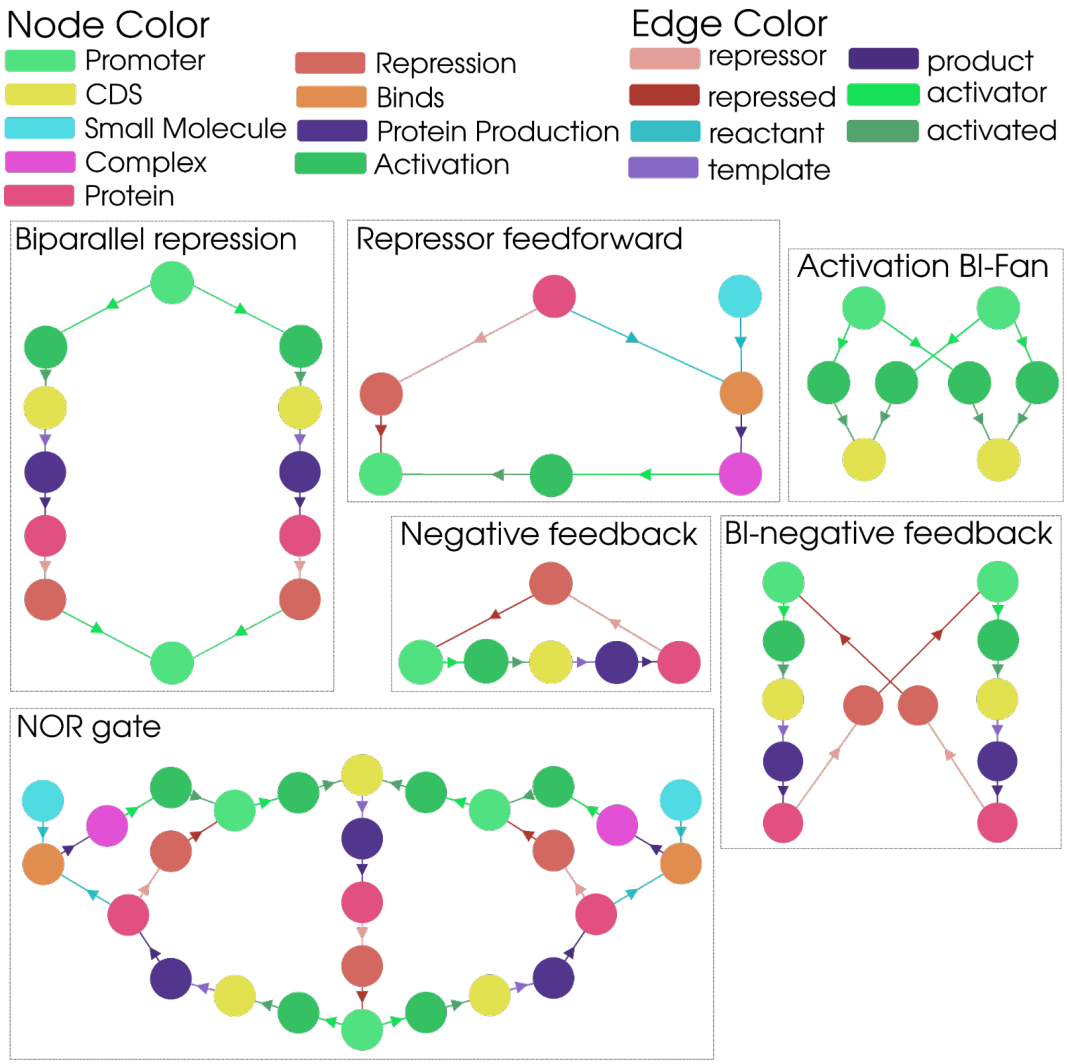

Figure 22: Extended example of generic motifs manually outlined or identified within the WKG. Each motif consists of generic nodes (nodes without references to virtual entities) and interaction edges.

Internal structure

To enable tractability between entities, either individual nodes or subgraphs, knowledge graphs must follow rules that dictate what nodes can connect and what labels are needed for an entity to be categorised as a specific real-world entity. In the context of RDF (Resource Description Framework) and OWL (Web Ontology Language), these rules play a crucial role in defining the structure and semantics of the knowledge graph. To unify several datasets into an initial WKG, the proposed knowledge graph must have unified semantics and structure to connect each database. RDF provides a standardised format for representing data and its relationships, while OWL offers a rich vocabulary for defining ontologies and specifying constraints. Without this unified base, even if the datasets describe the same or similar information, the underlying network will constitute separate components for each dataset. This implementation defines its internal model because of the need for easily extensible and unified semantics in RDF/OWL. Figure 23 displays a sample of the classes defined within the ontology for this exercise. The ontology is structured to transform information automatically by giving "requirements," a set of constraints that must be held within the data to be considered an instance of one of the classes. For example, the data is initially an "Entity" class, which can then either become a "Physical" or "Conceptual" Entity based on external and common ontological terms present within its data. This increase in specificity can then expand another level of detail if this data fulfils any other requirements. When added to the WKG, this approach enables each entity to be the most precise and aligns with the principles of RDF and OWL, facilitating semantic interoperability and knowledge integration. It must be noted that this is not a truly generic method for turning unstructured data into formalised structures. This system will only transform information which encodes different ontological terms within itself that are known to the internal ontology. General formalisation methods are an open challenge within semantic web approaches and are outside this report's scope.

4 Discussion

This section proposed a dynamic weighted knowledge graph to provide a more robust data control system by describing a broad domain of information, ensuring network features and encoding metadata. In this scenario, a functional approach[144] was prioritised by encoding interactions among network entities. This method simplifies design comprehension by reducing perceived complexity, especially for increasingly complex systems[14]. Additionally, interaction components were grouped to reduce complexity[89], promote reuse[166], and facilitate parts-agnostic design prototyping modules. Research began by exploring different compatible existing datasets within synthetic biology. From this, three datasets (iGEM, Cello, VPR) were identified, all with different features to exemplify this workflow. Next, the datasets were normalised by removing unwanted information given several metrics to create a manageable initial dataset. Once the candidate datasets were compatible, they were connected by an integration process, ensuring the network had certain features (canonical and containing functional information) and encoded specific metadata (confidence, usage, provenance). Once an initial network was established, several methods for expansion were developed to introduce new information and further connect the internal information. For example, expansions provided the ability to add missing interactions and create low-level modules from the components within the network. This section evaluates the outcomes

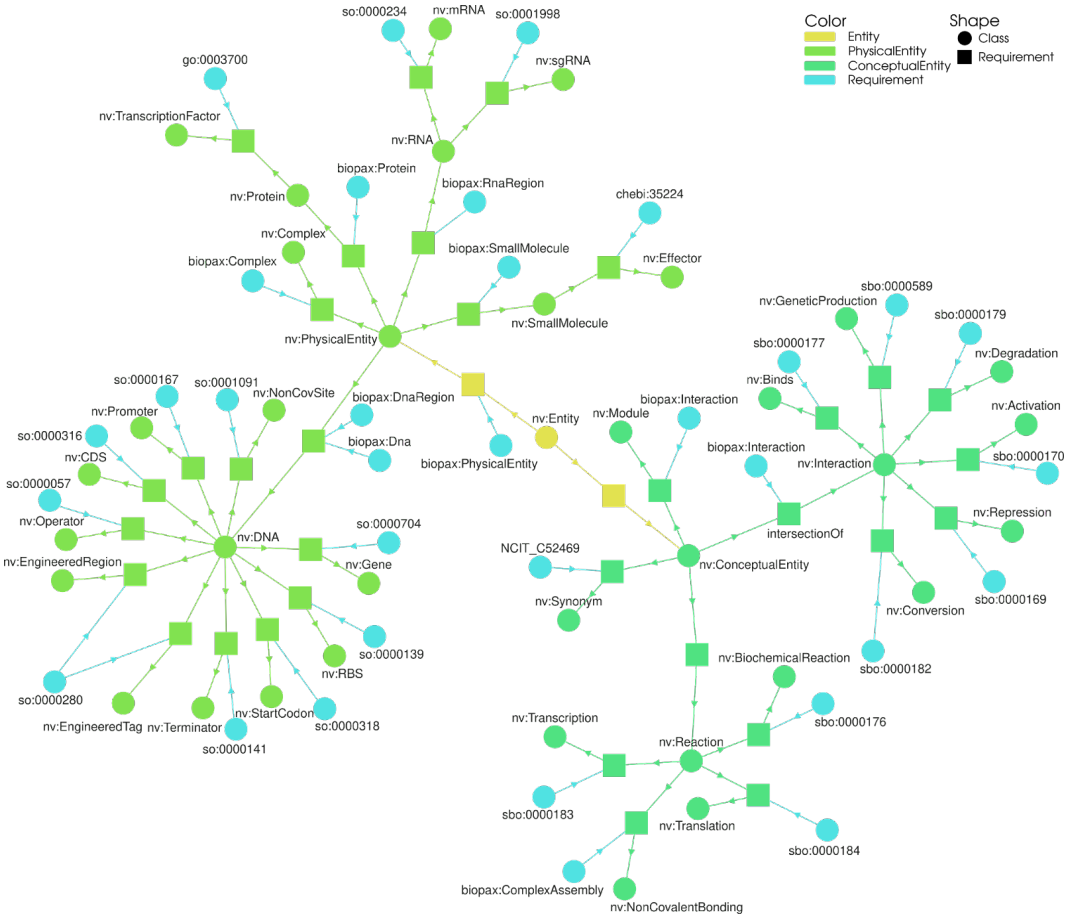

Figure 23: Requirements graph for the underlying ontology. Incoming information is used to traverse this network and checked against the Requirements of a Class node. Starting with Entity, given the ontology terms within the data, the node created within the network will capture an increasingly more specific until either another more specific Class does not exist or the information does not encode terms to make it more specific.

of this work. Furthermore, the potential future directions for research are discussed based on the challenges faced in this work.

4.1 Strengths

Dataset and domain agnostic enables the repurposing of processes and methods

Here, the process was displayed using three datasets. However, once the input datasets are normalised by removing unwanted entities to prevent flooding the initial WKG with useless information, the process is not inherently fixed to any specific information. For example, it is conceivable that non-synthetic datasets, such as natural metabolic networks, may be captured using the same system. Furthermore, using the integration techniques, providing some unifier exists, datasets from different computational and synthetic biology domains could be merged to provide unseen insight. Therefore, the creation, expansion and usage of the WKG system could take an arbitrary number of datasets to find information with different levels of affinity. Genetta enables an arbitrary number and types of information to be introduced within the dataset. It must be noted, however, that the quality of the outcome invariably hinges upon the quality of the information, and the input structure must be understood by Genetta, such as SBOL or Genbank.

Ability to quantify data via meta-characteristics

Within most synthetic biology-centric databases, there is a notable absence of information that comprehensively describes features from a user-centric viewpoint. By incorporating weights and specific features into the database architecture, users gain the capability to scrutinise information at a meta-level, particularly in terms of correctness. This enhancement is strategically designed to bolster the probability of software systems or human users accessing accurate information. As highlighted earlier, these characteristics are implemented with the overarching goal of facilitating correct information retrieval and streamlining the process of manual information interrogation, thereby minimising the time required for thorough examination and validation. When the initial WKG is built within Genetta, the metadata is automatically calculated, given the features of the larger datasets. For example, if a piece of information is encountered multiple times across the input datasets, the confidence will be higher to reflect this.

Improvement over time via community feedback

While the weights enable quantification of some metric regarding the underlying information, it will only be set relative to the information encoded within the input datasets. However, with the ability to accept evidence from external sources (user or software) as feedback, the weights can be continuously updated, further improving their accuracy. Updating information given conflicting or complementary information provides two main benefits. Firstly, the ability to suppress incorrect information and reward correct information will filter out incorrect information over time and leave a smaller and more accurate information core. Secondly, no individual is an expert on all entities and domains within biology. For example, one user may be an expert on a specific regulatory mechanism while another is knowledgeable on the performance of fluorescent proteins.

Currently, the literature must be combed for this expertise, which can be time-consuming. Existing uncurated datasets are largely shielded from the context of the information provider. Furthermore, an expert curating a database is a slow and arduous task, and the amount of existing curated data is minimal. The ability to provide feedback can bridge the gap between uncurated and curated datasets by providing experts on a specific aspect of a dataset to provide explicit feedback on a small section of the network, which spreads a burden which is traditionally put on a small group of volunteers. It must be noted that the specifics of feedback are discussed within Chapter 5, where use cases for the WKG are discussed. Genetta can take feedback from both the enhancer and query system (in the future, any tools that interface with the WKG will also be able to use this) and automatically update the weights or introduce new instances of features. Furthermore, in cases where custom behaviour is described for a specific weight or feature, such as redesignating the provenance edge based on the confidence of a similarity edge, Genetta can automatically make these changes.

A functional-centric approach moves towards abstraction

While a knowledge graph facilitates the encoding of functional and abstract information, existing databases in synthetic biology (not explicitly focused on interaction networks) often fall short of capturing this comprehensive range of data. The integration of datasets specifically prioritised the derivation of functional information. As discussed, encoding functional information can reduce the perceived complexity, increase the likeliness of a successful design and increase flexibility. Furthermore, a deliberate emphasis was placed on introducing abstract functional information, such as modules, during the expansion process. This abstraction has been a core principle since the conceptualisation of synthetic biology but has never come to fruition within the domain. Introducing high-level abstracting is not a simple process within synthetic biology because genetic parts and mechanisms, such as the environment, are highly context-dependent and do not function in isolation[18]. Context dependency makes defining functional modules challenging because it may not be clear what the information will represent, such as inputs and outputs[37]. However, here, defined low-level modules suffer less from the context of a design and do not need to be manually defined by a user, which may introduce uncertainty. It is crucial to acknowledge that inferring interactions solely through a network approach poses challenges, and this aspect has not been exhaustively explored in the current study, leaving room for future investigation (see future work).

Benefits of WKGs from the 'Learn' perspective in the DBTL cycle

From the perspective of the 'Learn' phase in the DBTL cycle, WKGs offer substantial benefits. The structured nature of knowledge graphs facilitates more effective knowledge, which is crucial for learning and refining synthetic biology processes. Specifically, Weighted Knowledge Graph Embedding, a technique where nodes and relationships are mapped into a continuous vector space, enables embedding complex, high-dimensional data in a form amenable to machine learning applications. This representation allows for applying advanced analytics, including predictive modelling and similarity searches, which are vital for the iterative learning processes in synthetic biology. By integrating these techniques, researchers can uncover hidden patterns and infer new knowledge, thus accelerating the learning phase of the DBTL cycle. This approach not only en-

riches the dataset with inferred relationships but also enhances decision-making processes by providing deeper insights into the biological systems being studied.

4.2 Limitations

Inability to move towards a fully functional approach

Shifting the focus solely to the function of a biological system would be a groundbreaking advancement within synthetic biology. However, a complete abstraction of sequence data is not feasible. Fully reusable parts remain unrealised because tailored changes to sequence data are common practice, mainly due to biological systems' context dependency. Therefore, making standardised parts that conveniently fit into all designs elusive and choosing from a range of existing entities based on function is a larger overarching goal of the field. However, this does not stop conceptualising and capturing functional information alongside the traditional sequence-based data capture, as this allows for a hybrid design comprehension and development approach.

Normalising constituent datasets

While this research aimed to enable a dataset-agnostic approach, some level of dataset-specific normalisation is required before it can be integrated into a WKG. The data type, how it is structured and relates to the other datasets are critical before integration. If the datasets are disparate, integration is futile as the resultant network constitutes disconnected components concerning the number of input datasets. However, even with complementary datasets, due to the broad domain of possible shapes in which the data could be structured, custom pre-processing must be provided beforehand. Assuming all input datasets followed a specific format (SBOL, for example) would be achievable.

Arbitrary primary source

When duplicates are found within datasets and one entity is labelled as the "primary source" and the rest as synonyms, this identification is arbitrary based on the first identification of the entity during computation. However, it may be the case that a particular name is more commonly known within the community than the one the WKG deems canonical. While this is not an issue of usage within the WKG because it has the mechanisms to identify the synonyms, if the canonical entity is exported, a less-used entity may be communicated. An improvement would be to label the primary source based on the most commonly used entity within existing designs.

A naive approach to identifying change of function

The expansion method, which transfers functional information into derivatives of a part, is naive in quantifying if the function would persist relative to the change in sequence. The relationship between change in sequence and change in function is not easily derivable and often requires experimental validation. With the confidence system, this issue would eventually be removed by negative user feedback, causing incorrect interactions to be removed. However, the expansion process itself may introduce incorrect information into the system.

4.3 Future work

Extend the expansions to infer and abstract functional information

Currently, the methods to identify missing interactions use a rule-based approach based on previously seen interaction subnetworks, which must be defined beforehand (or extracted from existing designs). However, this is only an initial effort that could be explored to infer interaction data from network topology. Related fields such as systems biology have made efforts to topological link prediction of partial networks and could be adapted to fulfil this gap within synthetic biology design data[167]. However, it must be noted that natural and engineered systems differ because the motifs and topology are often different. Therefore, the data used within systems biology may not be usable to predict missing interactions in engineered systems. The expansion to introduce modules within the WKG only introduced low-level modules describing small transcriptional mechanisms. However, an expansion method could be introduced, further expanding on modularising higher-level systems such as functional logic gates.

Language models for synthetic biology parts and interactions

An apparent issue with the formalisation of datasets largely consisting of written information is the challenges of interpreting the context computationally. For example, a dataset like the iGEM parts repository encodes much of its information, including interactions with genetic parts within other records. However, because this is an ambiguous written statement, it is unusable to form a structured network without the ability to comprehend natural language. Efforts are underway to fulfil these limitations already. The SBKS project[136] is exploring many topics, but the relevant one here is the machine learning methods to generate ontology annotations automatically. These methods will establish connections between data in different repositories and extract relevant information from publications. The ability to transform written information into semantic labels will provide a huge step forward in automatic formalisation.

Identifying provenance beyond sequence

The current implementation of our network leverages sequence similarity to infer initial provenance. However, many entities are functionally homologous and distant in primary structure, so they will not be caught using a sequence similarity approach. Predictive modelling could enable the network to infer complex biological relationships and evolutionary patterns not detectable through sequence similarity alone. For instance, machine learning models could be trained on a dataset of genetic sequences and their known histories to predict provenance with higher accuracy and efficiency. This approach would be increasingly beneficial as the WKG increases in size. The ability of machine learning to manage and interpret large-scale data would allow for a more nuanced representation of these variants in the provenance subgraph.

Representing large sets of variants

Within this instance of the WKG, the number of genetic parts was relatively small. However, some processes may produce datasets that contain thousands of parts alone. For example, thousands of mutant variants may be generated with directed evolution. The WKG could adopt a clustered

or hierarchical graph structure for future developments in these cases. This method would involve grouping mutants based on certain similarity thresholds or functional characteristics, thus reducing the complexity of the graph while preserving essential information about variant relationships. Such an approach would streamline the visual and computational management of the data and enhance the WKG's scalability to accommodate the exponential growth. Combined with the machine learning approach of identifying functional similarity, this representation strategy would provide a robust framework for tracking the provenance and evolution.

4.4 Conclusion

This chapter established a weighted knowledge graph (WKG), which integrates existing design datasets enriched with metadata and features, increasing users' accessibility, confidence and computational tractability. Furthermore, methods to expand the existing data were designed to increase the connectedness of the underlying datasets. The unique aspect of the WKG lies in its incorporation of weight values that describe metadata related to biological information, here defining confidence and usage. Quantifying such values in the data allows for assessing quality based on prior experiences, encompassing individual parts and utilising the information presented. Nevertheless, initial static values may not accurately reflect the actual values because they are estimated from the initial data, which is likely unvalidated. Thus, the ability to update these values (explored further in chapter 5), with input from external agents, enables a circular knowledge transfer from the user, increasing the metadata's accuracy. This process was illustrated with an example involving three datasets (IGEM[135], Cello[154], and VPR[128]), combining them to create a centralised data source. Starting with an initial dataset collection that shared similarities, they were integrated into an initial network, duplicates were removed, metadata was encoded, and all relevant information was extracted, thus establishing a robust foundational dataset. Next was a sample of the expansion process to the WKG, which enables the extraction and inference of new data from the existing network information. This process is a fundamental element of the described weighted knowledge graph approach, connecting potentially loosely coupled datasets by extracting and inferring new information from a centralised dataset. Furthermore, the expansion process demonstrates the network's capabilities by utilising network tasks to infer new data and seamlessly integrate it into the knowledge graph. In this section's introduction, some features that cannot be enforced by standards alone were identified, and the challenges with current methods for capturing design data within synthetic biology were highlighted. These challenges primarily revolve around uncertainties concerning the creator's intent, data accuracy, and contextual information, affecting both human users and computational applications. The functional approach tackled the uncertainty of intent by explicitly encoding abstract interaction information within the WKG. Explicit incorporation of functional information into design data facilitates abstraction, diminishing the perceived complexity[168] of designs, promoting a hierarchical approach to design data in synthetic biology[169] and eliminating the need to infer function solely from sequence data. Additionally, to mitigate concerns about data correctness, metadata encoding, including confidence values, was employed to quantify information quality and support informed decision-making. Introducing dynamic weights within a knowledge graph provides contextual information around captured data. This approach aids in mitigating the common uncertainty found in existing design databases through community-driven validation. Furthermore, weights

such as the confidences were employed to address concerns related to data accuracy. Incorporating dynamic weights within a knowledge graph adds valuable contextual information to the captured data, helping alleviate the pervasive uncertainties often encountered in current design databases through community-driven validation efforts. These dynamic weights enable the quantification of information quality and facilitate well-informed decision-making by establishing more reliable data sources, ensuring users can access desired information promptly. The challenge of contextual uncertainty, or how knowledge interconnects, was addressed by introducing new edges that depict relationships between elements, such as the provenance of genetic parts. This introduction allows for measuring related entities by some defined metric, enhancing our understanding of information context. One type of existing network approach may be seen as competitive to the WKG. Artificial Neural Networks (ANNs) are computational models that excel at learning from data and recognising complex patterns. Despite their proficiency in predictive modelling for synthetic biology, ANNs often operate as "black boxes," making their decision-making processes opaque and hard to trust in fields where interpretability is critical. In contrast, WKGs offer high interpretability with explicit logical mappings of entity relationships. Therefore, WKGs facilitate both computational and human inputs for integrating new data. Rather than being competitive, ANNs and WKGs are complementary; WKGs might utilise ANNs to enrich analysis rather than replace one with the other. In conclusion, the most important and valuable resources within synthetic biology are the practitioners and the invaluable knowledge they possess. However, with the current landscape of knowledge transfer, this wealth of resources is not exploited to the extent it deserves, resulting in a significant redundancy of effort. With the introduction of the WKG, specific information, such as newly designed genetic parts, can be shared, as well as the researchers' experience around the creation and usage of their and other's work. Encoding this knowledge can increase comprehension in cases where a user is unfamiliar with the domain or help them identify solutions to problems that others have already addressed, saving valuable time in both cases.

Chapter 5: Enhancing data access and designs by leveraging the weighted knowledge graph

Publications arising from this chapter

- Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "GENETTA: a Network-Based Tool for the Analysis of Complex Genetic Designs". *ACS Synthetic Biology*, 2023

Software arising from this chapter

- Genetta

1 Introduction

Standards are crucial in synthetic biology, deeply intertwined with its practices and evolution. Early discussions in synthetic biology at the beginning of the 21st century, though not explicitly labelled as data standards, already emphasised the importance of standardised genetic parts and abstraction hierarchies[170]. This discussion highlights a foundational recognition of the need for standardisation in the field[171]. A critical issue that has limited the adoption of standard data formats such as SBOL within synthetic biology is the complexity introduced by requiring a practitioner to learn a new skill, i.e. comprehending the data model and the tooling around it. For example, a typical design process involves sequence-based editing, outlining function intent using ad-hoc glyphs using literature to help guide decision-making. However, a data standard approach would require a practitioner to learn new, more technical tools, often requiring knowledge of the underlying data model. Furthermore, it would also require a user to explicitly encode a much broader range of information, such as functional information, which was previously encoded implicitly. From an experimentalist perspective, the benefits of adopting standards are unclear, and the challenges are too significant. Two goals must be fulfilled for the adoption of standards within synthetic biology to be genuinely successful within the community. Firstly, the barrier to entry of tooling and any requirements to learn any aspect of complex data models must be removed. Secondly, the tooling and benefits with standards must be superior to the current ones. When combined, a community would be naturally drawn to new working practices instead of being forced, which thus far has not been successful. Therefore, the effectiveness of data standards within synthetic biology has been somewhat limited, primarily when implemented for user-facing applications. For example, databases that contain datasets encoded within standard data structures, such as the iGEM parts, are encoded within SBOL and stored on the Synbiohub database. Despite conforming to the SBOL standard, the data's quality remains low and is often challenging to handle

computationally. This issue arises because much of the information is in written text, containing duplicate entries that hinder proper identification, and there are no assurances regarding the accuracy of the underlying data. These shortcomings are not inherent flaws in the data standard or the tools utilised; instead, they highlight the reality that existing user-facing tools do not entirely resolve all associated issues despite aligning with standards. Compounding this point, methods to present the data, such as querying tools, do not address these issues, and the small regions of quality information are not presented to the user. In Chapter 4, a weighted knowledge graph (WKG) was established to enable enhanced access to one or more datasets. The features of this new network can be used to implement systems that can target some challenges with accessing data from both a user and computational perspective.

1.1 Existing query methods and programmatic access

A database's methods to provide queries and results from a dataset vary from system to system. For the most part, however, within synthetic biology databases, these systems offer results using simple string-matching approaches, which are ordered arbitrarily. While many databases' specific implementations are unknown as they are hidden from the user, some techniques can be deciphered. This section explores how several databases query and return results given a specific input. Where known, the particular mechanisms will be analysed. Otherwise, the means will be interpreted. Furthermore, if these databases offer programmatic access, such as via API calls, the usefulness of these will also be quantified. Therefore, the evaluation of each database will be broken down into a manual and programmatic approach. Within the manual approach, how the input query is handled, how the database performs the search, how the results of the search are handled and finally, how these results are presented to the user. From a computational approach, how the programmatic approach interfaces with the database, how the database performs the search (if different to the manual method) and the structure of the results. Again, these points are predicated on them being available for evaluation.

Synbiohub

The database Synbiohub has been discussed in the background and in chapter 4. In short, SynBioHub, a web-based platform [122], is a repository for synthetic biology designs and houses biological constructs in SBOL format stored in a graph database. The functions of this database are known, and the specific methods can be disseminated.

Manual query The user's query is input directly into the search mechanisms without preprocessing or error correction, failing to address potential issues such as misspelt information. Additionally, although the advanced querying option allows for specifying the SBOL type (like ComponentDefinition) being sought, this feature is notably absent in the standard query system, limiting its functionality.

Synbiohub is an extension of the RDF data model, and the most common method for querying RDF triplepacks is via the SPARQL[172] query language, which is how Synbiohub performs the searching. SPARQL allows users to retrieve specific information, make connections between different pieces of data, and perform operations like filtering, joining, and aggregation on RDF

data. The language consists of various query types that enable users to retrieve information from RDF datasets. Figure 1 displays the exact SPARQL query to search the Synbiohub graph. The "PREFIX" lines define prefixes used as shortcuts for longer URIs within the query. For instance, "sbol2", "dcterms", "sbh", and "rdf" are aliases for longer URIs, making the query more readable and concise. This "SELECT" section specifies the variables the query will retrieve for each matching result. These variables include "?subject", "?displayId", "?version", "?name", "?description", and "?type". The "WHERE" is the main part of the query where the conditions for selecting data are specified. The FILTER clause filters results based on whether the entities' displayId, name, or description contain the case-insensitive string 'LacI'.

- ?subject a "?type" retrieves triples where "?subject" is of type "?type".
- ?subject" sbh:topLevel "?subject" selects entities where "?subject" is the top-level entity in SynBioHub.
- The OPTIONAL clauses retrieve additional information about the "?subject" entity, such as its displayId, version, name, and description. The OPTIONAL keyword means that if these properties are unavailable for an entity, it will not exclude the entity from the result set.
- The LIMIT clause limits the number of results returned by the query to 50. It ensures that only the first 50 matching entities are retrieved.

In short, this SPARQL query aims to find up to 50 distinct biological entities whose "displayId", "name", or "description" contains the case-insensitive string 'LacI'. It retrieves additional information about these entities, such as their identifiers, versions, names, descriptions, and types, from a dataset modelled using SBOL and SynBioHub ontologies.

While SPARQL query language is robust, this implementation is shallow because it simply searches for direct case-insensitive names. It does not consider misspelt or alternative names, nor enables a user to query via different types of information such as sequence, functional or abstract features such as contextual usage. Also, from a technical perspective, the query uses the FILTER clause, CONTAINS, and lcase functions to perform case-insensitive substring matching within displayId, name, and description. This approach can be computationally expensive, especially on large datasets, as it might not utilise indexing efficiently. Additionally, using multiple CONTAINS clauses in a FILTER might result in slower performance due to multiple string comparisons.

SPARQL is an excellent choice for searching within an RDF data store, but one potential issue may arise with the results. The default ordering behaviour might vary across different SPARQL implementations or datasets. Therefore, relying on the default order might result in unpredictable outcomes and for consistent ordering of results. From observation, Synbiohub does not change the order of the results from the initial search because Synbiohub uses a Virtuoso (a specific RDF triple store and graph database), which orders based on the internal indexing when presented to a user by default, are ordered arbitrarily giving no promotion to better results.

Once a desired result is identified within Synbiohub, this entity's information is used within several representation and visualisation techniques to display features. General information - Provides a textual overview of the most prominent information, such as names, sequences and de-

```
PREFIX sbol2: <http://sbols.org/v2#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX sbh: <http://wiki.synbiohub.org/wiki/Terms/synbiohub#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT
    ?subject
    ?displayId
    ?version
    ?name
    ?description
    ?type

WHERE {
    FILTER ((CONTAINS(lcase(?displayId), lcase('LacI'))||
      CONTAINS(lcase(?name), lcase('LacI'))||
      CONTAINS(lcase(?description), lcase('LacI'))))

    ?subject a ?type .
    ?subject sbh:topLevel ?subject
    OPTIONAL { ?subject sbol2:displayId ?displayId . }
    OPTIONAL { ?subject sbol2:version ?version . }
    OPTIONAL { ?subject dcterms:title ?name . }
    OPTIONAL { ?subject dcterms:description ?description . }
}

 LIMIT 50
```

Figure 1: The SPARQL query the Synbiohub database uses to query its data. The input query is “LacI”.

scriptive information. SBOL Visual - Provides a glyph representation of the data where appropriate. Sequence visualisation - Provides a circular plasmid representation of sequence data where appropriate. Metadata - Provides contextual information on the design when encoded. Synbiohub provides many different mechanisms for comprehending the information. However, two issues are present. Firstly, information is presented using SBOL-specific terms, confusing users and making exploring content manually challenging. Secondly, many options to represent the design are unused or broken, which complicates and reduces confidence from a user perspective.

Programmatic access SynBioHub offers an API that enables programmatic access to its functionalities, allowing users to interact with the repository, retrieve information, upload data and query designs, search via sequence and many administrative commands. The SynBioHub API provides endpoints that cover a range of functionalities and is very robust computationally. However, Synbiohub cannot return results based on partial sequence matches or via abstract or contextual questioning such as usage. The database will search the data using the same mechanisms as via a manual approach. The Synbiohub returns results via a standard JSON format that is consistent between different API calls. These standard and determinable results make accessing data computationally trivial.

Inventory of Composable Elements

Inventory of Composable Elements (ICE)[125] is a software platform designed to manage and facilitate tracking biological parts, plasmids and strains, and it provides a repository-style system to organise, store, and share genetic resources. This database is mainly sequence-centric and, unlike Synbiohub, is not based on a standard data standard.

Manual query Like Synbiohub, the user's query is directly used to find results without preprocessing. However, ICE allows adding sequence searches and other filters, such as partial names, to reduce the search space. Ice uses a traditional relational database and performs its searches primarily via full-text indexing, which searches for matches within large sets of textual data. While this indexing enables matches to be made on partial or indirect matches with very little structure, querying on specific datatypes is virtually impossible. ICE presents many similar features within the results list, such as IDS, name and other metadata. However, one unique feature of the ICE database is the relevance measurement based on the similarity between the search query and a record. This similarity is calculated based on how close the query string is to the match made within the record and the number of times a match is made. For example, if the search query "Lac" and a record containing "LacI" despite not having a direct match, the record will be taken with a lower relevance. With the ability to compare records to one another via relevance, the results can be ranked and present "better" results at the top of the list, reducing the burden on the user. While this is a valuable feature, the metric to quantify relevance does not consider other features away from simple close string matching and limits its applied usage. For example, it does not consider quality, context of datatypes or correctness. The presentation of results within ICE is more sequence-centric and is based on a plasmid (circular and linear) map. While these features overall are less, the quality and elements of this aspect are superior. For example, it enables a user

to perform operations such as identifying open reading frames or searching for regions on the sequence. In contrast, the Synbiohub implementation is a more static representation.

Programmatic access ICE offers an API that enables programmatic access similar to Synbiohub. However, it exposes different endpoints, such as one to modify existing data, which is notoriously challenging within Synbiohub[173]. However, overall, the API provides similar coverage to Synbiohub; therefore, all of the same issues are present here. The database will search the data using the same mechanisms as via a manual approach. Like Synbiohub, ICE returns results via a standard JSON format that is consistent between API calls. However, unlike Synbiohub, which returns only the requested information, ICE will return all known information, which can be problematic with large result sets.

Kyoto Encyclopedia of Genes and Genomes (KEGG)

The Kyoto Encyclopedia of Genes and Genomes (KEGG)[127] is a database and resource for understanding biological systems related to genes, proteins, pathways, and diseases. It provides a comprehensive collection of information on various aspects of molecular biology, genomics, bioinformatics, and systems biology. Unlike Synbiohub and ICE, which are general-purpose synthetic biology repositories, KEGG is a systems biology-centric database more specialised for genetic interactions. It has been chosen for review because the data is inherently interaction network-centric.

Manual query KEGG allows users to specify the query type, offering a tailored approach to data retrieval. Users can focus their search on specific categories like pathways, genes, diseases, drugs, or enzymes. However, it is noteworthy that KEGG, similar to Synbiohub, does not implement data manipulation features, such as autocorrecting spelling errors, meaning users must ensure the accuracy of their input queries. The inner workings of KEGG's search system are not fully disclosed, but certain inferences can be made. For instance, when a user selects a specific dataset for their query, KEGG excludes other datasets from the search scope, effectively narrowing the search effort and potentially speeding up the retrieval process. KEGG contains results based on the datasets but does not rank results based on relevance or quality metrics. In KEGG, the results are presented based on the datasets queried. However, the system does not use relevance or quality-based ranking to present these results, so users must manually filter the results to find the most pertinent information. KEGG excels in providing tools for pathway analysis, network visualisation, and data interpretation. These tools are handy for exploring biological pathways, offering an interactive and detailed way to analyse and understand complex biological data.

Programmatic access KEGG offers an Application Programming Interface (API) for programmatic access to its vast data collection and functionalities. Additionally, it provides a File Transfer Protocol (FTP) site for bulk data downloads, a feature especially beneficial for users needing large amounts of data without the limitations typically associated with repeated API calls. Programmatically, the database searches in KEGG are conducted using the same mechanisms as manual searches. Specific datasets can be targeted, and logical filters can be applied, such as retrieving only biological compounds within a particular molecular weight range, thereby allowing for

more refined and focused data retrieval. KEGG returns results in a structured format, using standard responses such as JSON or XML. This consistency in data formatting across different API calls simplifies the computational handling of the data. These results' predictable and structured nature makes it easier for computational tools to process and integrate KEGG data into various applications and analyses.

Reactome

Reactome[174] is a curated and comprehensive knowledge base focused on biological pathways and processes related to primarily human biology (model organisms are also included), namely cellular processes, molecular interactions, and signalling pathways. While Reactome is considerably distant from the other reviewed databases, it is an example of a fully curated dataset combined with a high-quality system. This database has many tools for querying, including pathway browsers, which visualise the pathways within the database as networks.

Manual query Reactome will suggest similar names and autofill partial names when inputting a query, enabling misspells to be fixed and providing a guide for the user. Also, the system will use fuzzy string matching like the ICE database, even if an incorrect spelling is inputted. Reactome also filters the search by several factors, such as species or reaction type. While this system does not enable contextual filtering, such as based on feedback from other users, the database is curated, so it may not be so relevant. Reactome is modelled as a knowledge graph of interconnected terms. Furthermore, it is captured as a labelled graph using the neo4j datastore. Because Reactome uses a neo4j[175] datastore to capture information, it uses the Cypher query language[176] and system to fetch results. It can perform various operations such as creating, updating, querying, and deleting nodes, relationships, properties, and graph patterns. Reactome uses a prepared query approach, which consists of template queries filled in based on what type of information is queried. This approach provides many benefits over the other databases explored here, namely, the ability to integrate not only the desired information but the context of the information, i.e. its neighbours.

When handling the results, Reactome considers many aggregated factors which impact the order in which results may be presented. It considers the relevance to the query much like ICE and other factors, such as the completeness of the pathway, which promotes results that are much more likely to be preferred when combined. Reactome does not consider user activity, such as how often a piece of information is accessed, which contrasts with more general-purpose search engine systems. However, this may not be relevant as assurances can be made that all information is high quality. Reactome's presentation of an individual record is based on the type of information that is accessed. For example, a record capturing an interaction will visualise the interaction network for that reaction or a chemical compound of the constituent chemicals. However, generally, it emphasises the network representation of the information by visualising the entities within the record and their neighbours or linking them to neighbour records via links. This approach lets users explore the database quickly by passing seamlessly between records, but it could confuse people without knowledge of the networks.

Programmatic access Reactome provides an API called the Reactome Content Service (RCS). The RCS offers programmatic access to the biological pathway data stored in the database.

The API enables querying and analysing pathway-related information in a structured and standardised manner. Much like manual querying, the programmatic approach varies based on the type of information being searched and accessed. However, the underlying data model must be understood because the API requests and responses reference the types of entities within the graph datastore. RCS returns most results via standard formats, such as JSON or XML, enabling consistency between API calls. These standard and determinable results make accessing data computationally trivial. However, RCS also enables the retrieval of graphical representations of pathways and molecular events, which could still be serialised as JSON/XML or graph-specific formats such as GraphML or XGMML.

Genbank/NCBI

GenBank[177] is a comprehensive, general-purpose repository that captures curated and uncurated information. Within all fields of biology, this database is one of the largest and most accessed. The more extensive database contains information about DNA, RNA, proteins, taxonomy, and many other types of information. Therefore, this review will only cover the datasets that capture DNA-centric information, such as the Nucleotide subset.

Manual query This database enables different datasets to be chosen depending on the user's desires, such as the nucleotide, genes or genome datasets. Furthermore, it enables further filters, such as based on sequence length or the species related to a sequence. Also, it contains a mechanism like Reactome to autofill or correct misspelt inputs, reducing incorrect inputs, which will likely require optimised text indexing due to the sheer size of this database. Searching database NCBI's process to implement query handling is unknown, so some assumptions based on observation will be made here. Once an input query is submitted, no close string matching is performed, so a single character difference will either return different or no results. Furthermore, the search process will match against all of the written textual information often encoded within Genbank file formats. Without complete standardisation of the underlying information, such as in the case of Synbiohub or Reactome, more comprehensive approaches would be impossible without complex algorithms to analyse the sequence or text. While the process of handling results again is unknown, some observations can be made. The database enables the user to rank the results based on metrics such as accession number, date modified, release date, organism name, taxonomy ID or sequence length. Presenting information is based on the type of information; however, for sequence-based results, the information presented is relatively consistent. The presentation is sequence-centric, displaying the underlying Genbank structure and a linear sequence which can be manually explored. With such a sequence-centric approach, the more abstract and higher-level representations established with the functional approach to synthetic biology are impossible to represent.

Programmatic access NCBI offers programmatic access via the Entrez Programming Utilities (EPU). These utilities can perform searches, retrieve specific data records, fetch sequences, access metadata, and more through a series of HTTP requests. Much like the manual approach, EPU's process to search the data is unknown. However, observations show the results are the same when using the same input query and parameters. Like the other databases, EPU returns

a structured and standardised format, ensuring that data retrieved through programmatic interfaces is easily parsable by software applications. EPU most commonly returns information in XML, JSON, Tabular Formats or Custom Formats for specialised information.

1.2 Time-consuming validation and costly preprocessing

The type and quality of information and the methods to return it vary, given the source[178]. However, the results are often poor if a database does not curate its information because there is no context to the quality. Even in cases where a database may return reasonable results given the input query, this does not mean that the underlying information can be trusted. This section discusses the analysis and assessments that must be performed when information is manually extracted and the preprocessing and normalisation that must be performed during a computational approach when extracting data from existing databases. Leveraging information from databases is common practice, yet designers must critically analyse this data due to varying quality. Databases gather genetic sequences and biological components from various sources. However, these databases often vary in accuracy, completeness, and reliability. Inaccuracies and outdated or conflicting data can arise due to differences in experimental methods or annotation techniques[179]. Before integrating information into designs, designers must conduct quality assessments. It includes verifying source credibility, cross-referencing across multiple databases, and evaluating supporting experimental data[180]. Furthermore, designers validate this data using in-silico analysis, experimental tests, or computational simulations to validate the functionality and reliability of genetic components before their integration into designs[181]. Also, it is common for practitioners to reach out to the community, which provides designers access to shared experiences. While databases are valuable resources, designers must critically assess and validate the information obtained from these sources, which can be time-consuming and financially costly. Furthermore, even the evaluation is open to human error, which can further introduce potential points of failure.

When programmatically accessing existing design information from a database for use in a computational or programmatic approach, several preprocessing steps are crucial to ensure the data's reliability and usability. Before utilising the accessed data, addressing inconsistencies, errors, and missing values is essential[136]. This process includes resolving any evident errors in the retrieved information. Furthermore, even with the backing of a data standard, ensuring uniformity across different datasets or entries and making them consistent and compatible is required because dataset-specific features may be present[182]. Also, checks are required to evaluate the data quality obtained and flag data that does not meet certain quality criteria, such as unreliable sources, incomplete records, or conflicting information. These checks require methods to validate the accuracy and reliability of the retrieved data, which may involve cross-referencing information with other sources and performing consistency checks to ensure the data's correctness. Much like the manual approach, a computational approach must evaluate and check the information obtained from these sources, which can require a lot of manual intervention or complex checking algorithms. These requirements have been seen within Chapter 4 as the example datasets needed to be cleaned and normalised before integration, which was not a trivial process.

1.3 Functional approach to genetic designs

Synthetic biology's essence lies in engineering intricate logic in living systems. While progress has been made, larger and more complex genetic circuits require a departure from a purely sequence-focused approach. Synthetic biology aims to integrate sequence data into functional modules to reduce complexity and facilitate the forward engineering of complex systems. At its core, synthetic biology is driven by function. It centres on orchestrating functional modules to simplify complexity by prioritising function over low-level sequence data[9]. The functional approach in synthetic biology design was established within Chapter 4 and emphasised defining biological components based on their roles within a system. It was explained that this offers advantages: reducing complexity by capturing abstract functional information, meeting expectations through standardised genetic parts, enabling flexible design iterations, and encouraging systematic decision-making. Furthermore, it was also established that functional information is seldom encoded within datasets of design data. When looking at the designs themselves, it can be seen that this is also missing information.

1.4 Legacy design data

As described in the previous section, functional information is rarely explicitly encoded within a design. This point can be expanded to describe all data types that cannot be derived from the small amount of structured data in a Genbank file (sequence and positional annotations). The lack of information is because most existing designs are taken from these sparse datasets discussed in Chapter 4 or are translations of designs encoded within other formats, such as Genbank files. Quantifying the general composition of existing designs is more challenging than the datasets. However, some collections of designs exist which can provide an estimate of the trends. The collections have been taken from the URLs in Table 1 with a description of the collection for context. It must be noted that the SBOL community has a chronic shortage of available developed designs encoded within SBOL, making this review not entirely representative of how designs may appear. However, this in itself further proves the limited adoption of this standard. Once these designs have been evaluated, the constituent datatypes are categorised in the chart displayed in Figure 2. Despite SBOL enabling modules, structural and functional hierarchies, interactions, and experimental and simulation data, to name a few, these features usage remains limited and underspecified, and designs still mainly consist of sequence and annotation data. If so much existing information is SBOL compatible but does not capture the extra information it supports, it raises the question of how it can be retroactively introduced. While it is true that several methods and tools exist to edit designs manually, such as SBOL Visual[132], ShortBOL[2], and many programming language packages[103, 116], processes to enhance existing data retroactively or automate the addition of this information during development, have been explored much less. Currently, the most common approach for bringing existing non-standard designs into a standardised workflow and introducing extra information is converting the data into a basic standard representation, for example, translating Genbank into SBOL and using the specification tools described above to enhance the data manually. However, the number of records and adhoc structure of the existing information make retrospectively updating manually unfeasible.

Table 1: Several SBOL designs and collections and descriptions regarding their contents.

Collection URL	Description
Digitalizer	A genetic device to digitalise gene expression into a sharp on/off signal[183].
Devices from the iGEM 2016 interlab	A collection of devices used in the 2016 iGEM inter-lab study.[184]
RepressionModel	Cas9, guide RNA CRISPR repression system.
Toggle_Switch	TetR-LacI genetic toggle switch.
Capturing Multicellular System Designs Using Synthetic Biology Open Language (SBOL)	Several multicellular designs of decreasing abstraction[185].
GitHub - SynBioDex/SBOLTestSuite	Many small SBOL examples are used for testing within the SBOL development community.

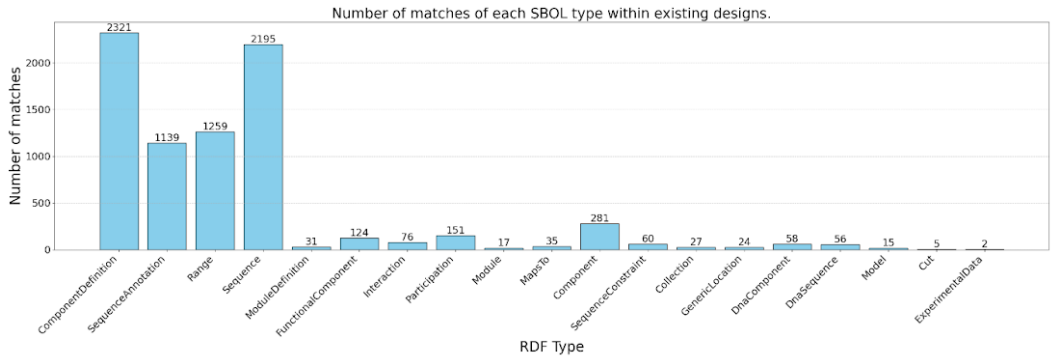

Figure 2: Types of objects within several existing SBOL designs and collections.

1.5 Weighted knowledge graph

Chapter 4 explored the weighted knowledge graph (WKG), designed to provide a more robust data source that could enable a more consistent interface with the ability to query different types of data away from the current sequence-centric databases. The WKG has specific features and interrogation methods that could help with some of the challenges discussed.

Standardised knowledge graph - The graph is standardised from a semantic and structural perspective. The semantic standardisation attaches labels from a defined domain (an ontology) to all nodes and edges in the network, ensuring unambiguous comparisons of individual entities like genetic parts or interactions. Meanwhile, structural standardisation permits only specific nodes to connect via edges with designated labels, which ensures predictable traversals based on the topology of the data. For instance, if a node exists within a network, it can only have a predefined type of connection with other labels. Overall, this standardised approach enables data

to be easily merged and integrated into the WKG, providing that the incoming information has the same structure or a mapping can be made. Utilising a network-centric approach for storing synthetic biology design data in a graph format is highly beneficial due to the graph-like nature of such data. This method removes the need for data normalisation, allowing for more natural and intuitive storage, as it mirrors the interconnectedness of biological systems. The graph-based structure also enables quick traversal and retrieval of data, particularly for nodes near a selected starting point, which facilitates speedy and efficient analysis of connected network segments, offering insights that might be challenging to obtain with traditional relational databases.

Canonical - The graph is canonical, meaning no duplicate exists within the network. For example, no two genetic parts with the same sequence exist. Furthermore, the processes around the WKG can identify a conceptually identical match where the underlying data is equal, even if the URIs are not identical. Also, the WKG can encode multiple names and URIs for one entity. In practice, this enables matches to be made within and outside of the WKG even when the URI is not identical, which is a valuable mechanism from a computational approach where a human is not available to judge conceptual matches.

Provenance - The network contains relationships between entities predicted to be derived from one another or homogenous. For example, if two genetic parts are thought to be functionally similar because one is a modified version of the other. Capturing provenance enables similar entities to be grouped, alternative entities to be swapped, and incorrect information to be identified and cluster functionally similar entities.

Functional - The methodologies associated with the WKG, particularly those related to expansion techniques, are primarily aimed at incorporating and enriching functional information. This emphasis on encoding functional information is significant as it allows for integrating this crucial data into existing designs where it is often absent. From a user's standpoint, this capability facilitates more abstract and function-focused queries. For instance, users can search for a desired functionality to be included in their design without being limited to a specific implementation. By enabling the encoding of functional information, the WKG thus supports a more flexible and intuitive approach to design queries and modifications.

Confidence - The WKG encodes confidence as a weight, measuring the perceived accuracy of the information it contains. This encoding of confidence is crucial in quantifying the likelihood of correctness, thus reducing uncertainty. From a computational viewpoint, this feature helps prevent the integration of incorrect information into designs. For users, it assists in discerning more reliable data, as higher confidence values can highlight more trustworthy information, guiding decisions and analyses. This dual benefit enhances the overall reliability and usability of the WKG in both computational and user-focused applications.

Usage - The WKG encodes entities not derived from one another but related from a contextual perspective, such as some experience a practitioner has had previously using two parts together. From a user perspective, usage can provide some context of the success or failure of the practical

implementation of two entities together. For example, if two genetic parts have some level of interplay that is not immediately apparent, this may be highlighted within usage.

The reviews undertaken within Chapter 4 show that these meta characteristics and weights are seldom encoded within databases. However, combined, they may alleviate issues around accessing and using uncurated databases from a manual and computation approach within synthetic biology.

1.6 Aims and Objectives

This introduction began by reviewing the methods to handle input queries, the process to identify results, how results are presented and computational access to the contents of several databases. The result was that databases vary not just in quality established within Chapter 4 but also in the robustness of the systems to present information to a user both manually and computationally. However, uncurated open databases generally present data poorly because of the inherent uncertainty in the underlying data without any mechanisms to regulate or alleviate it.

Next, the issues and hurdles arising from this type of access were discussed. It emphasised that databases can vary in accuracy, completeness, and reliability due to differences in sources, experimental methods, or annotation techniques. Additionally, it touched upon the essential manual analysis that must be performed when using information from databases and the preprocessing steps required when programmatically accessing data from databases. The conclusion was that manual and computational approaches require thorough evaluation, checking and filtering of information obtained from these databases, which can involve extensive manual intervention, complex checking algorithms or testing.

From this, the functional approach to synthetic biology idea, which highlights a shift from a purely sequence-focused approach to defining functional modules, was expanded from a purely design context. It emphasised establishing a functional approach in synthetic biology design, defining biological components based on their roles within a system. This approach offers advantages such as reducing complexity, using standardised genetic parts, enabling flexible design iterations, and encouraging systematic decision-making. However, a limitation was defined: the absence of functional information encoded within datasets of design data and the designs themselves, indicating that functional information is missing despite its critical role in synthetic biology design. Because of this discovery, an evaluation was extended to include various data types beyond what is typically found in a Genbank file (which primarily contains sequence and positional annotations) and functional information. It was found that despite the capabilities of SBOL to encompass various features, their usage remains limited, likely due to their reliance on sparse datasets discussed earlier or designs translated from other formats, such as Genbank files, which can be seen as the original composition from integrating datasets in chapter 4 was sequence centric.

Finally, the WKG from Chapter 4 was reintroduced, designed to provide a more robust data source that could enable a more consistent interface with the ability to query different types of data away from the current sequence-centric databases. Because of this, the WKG was proposed as a backbone to assist with the issues established.

In summary, the introduction covered database variability, emphasising the challenges posed by data quality, manual analysis, and the absence of functional information in design datasets,

proposing the WKG as a potential solution. Therefore, this chapter explores how the WKG represented in chapter 4 can be used to provide solutions to data access from both a user-facing perspective, namely the querying process to identify information for a design manually, and a computational approach which can be used to introduce the information commonly missing within a design automatically.

As discussed, generally, querying synthetic biology-centric databases is achieved by matching the search query with record names, metadata or sequence data[121]. This simplified search contrasts strategies with established search engines, which consider additional information when identifying matches[186] that often produces answers more accurately. With the knowledge graph previously established, more comprehensive methods can be established. The user-facing use case aims to describe methods for superior querying to provide better results to a practitioner quicker by exploiting the features of the WKG. Furthermore, Chapter 4 briefly explored how the WKG can access feedback, which can update the weights and features of the network by user-provided feedback on the quality or correctness of this information. This will explore this feature further by giving a specific example of how this information may be integrated into the network.

As highlighted before, most existing designs (and design information collections) contain minimal or no extra information that SBOL enables. Designs and datasets could be enhanced by transferring the extra information encoded within the WKG. Therefore, the second computational-centric use case explores how the missing information not commonly encoded within design data can be automatically enhanced with a focus on adding missing interactions and identifying potential functional modules to add a hierarchical structure to a design.

Aims

Explore the practical application of the established WKG to address data access challenges in synthetic biology from user-facing and computational perspectives. The user-facing aim is to develop and implement superior querying methods leveraging the features of the WKG to enhance the efficiency and accuracy of information retrieval for practitioners engaged in synthetic biology design processes. Furthermore, it will investigate and illustrate how the WKG can facilitate the integration of user-provided feedback to update network weights and features, thereby improving the quality and correctness of information retrieval mechanisms. For the computational use case, develop and implement approaches leveraging the WKG to supplement design data automatically by incorporating missing interactions and identifying potential functional modules. Also, this aim is to introduce hierarchical structures within designs and improve the comprehensiveness of design information through automated data enrichment methodologies.

Objectives

Develop methods for querying, focusing on providing quicker and more accurate results for practitioners in synthetic biology by using the features of the WKG to provide improved access to data. Furthermore, the goal is to provide a detailed example of how the WKG can incorporate user-provided feedback to update network attributes, emphasising enhancing information quality. Next, from a computational application, demonstrate the potential enhancements of existing designs and design information collections by incorporating additional SBOL-enabled information within the WKG. This goal focuses on developing computational-centric approaches to

automatically supplement design data by adding missing interactions and identifying potential functional modules, utilising information from the WKG to introduce hierarchical structures within designs.

2 Results

This section comprises two parts, each focusing on distinct cases. The first part explores a user-centric approach, examining how the manual querying process can be significantly improved by applying the Weighted Knowledge Graph (WKG) and its features, as outlined in Chapter 4. The second part shifts focus to an automated context, demonstrating how the WKG can seamlessly integrate into computational pipelines to enrich existing designs with previously missing information. Collectively, these illustrate the versatility and effectiveness of the WKG, showcasing its utility from both manual and automated perspectives.

2.1 Superior Interfacing using network features

As seen when reviewing database query systems, interfacing with synthetic biology-centric databases via queries is commonly achieved by matching the search query with each record's name, metadata or sequence data. The results are usually ranked arbitrarily or by the number of direct matches. This simplified search contrasts strategies with established search engines. These search engines take into account more information when identifying matches, such as the intent of a query and ranking results based on several factors, such as how much a resource has been used in the past[147]. The holistic approach often produces answers to questions more accurately relative to intent, and the desired results can be identified faster and more accurately. However, most existing databases cannot implement superior querying because neither the underlying data, structure, nor metadata capture the required information. Using the knowledge graph previously established, more comprehensive methods can be established using the new metadata (confidence), meta characteristics (provenance and usage) and extra information (interactions) encoded within the network and methods (feedback). The encoded **confidence** can be used to rank results to highlight information likely of higher quality. **Provenance** within the network can provide a user with similar entities by function or sequence given an original entity. Users can provide abstract requirements such as **interaction** data instead of names or sequence data as a query. Other parts commonly used with a genetic part can be provided based on previous **usage**. When query results are provided, **feedback** on these results may be given, resulting in new information added to the dataset or updated values within the network.

Non-sequence-based queries

Genetic parts are commonly searched using sequence or direct name matching. However, a user may want to search for records using several different metrics, for example, based on the function of a genetic part or similarity with other parts. As discussed, unlike many databases, which usually encode a small number of datatypes (usually sequence), the WKG we have built encodes several, each providing a unique insight into the data. The different search mechanisms will be displayed

using the same query input to show that the search method and results change based on the desired information. This example is illustrated in Figure 3.

Derivatives and partial sequences : Partial sequence matching over large datasets is not feasible when performing queries because the computational requirements are too great. However, as discussed, the WKG contains Derivatives edges, which refer to genetic parts believed to have some homogeneity, perhaps because one is a modification of the other. Therefore, these edges enable the fast identification of similar entities, which may provide similar functionality. A graph traversal is performed from all derivatives edges. This process will take the derivative component (The subgraph of derivatives of the subject node), providing an exhaustive list. Figure 3 pink traversals from the source (LacI) to BBa_I732100 and then BBa_I732103, as the latter is still implicitly a derivative of the query node.

Interactions : Interactions between entities enable a more function-focused approach to query for genetic entities. With the WKG, it is possible to specify functional requirements for a part instead of identifying parts and then function. Figure 3 grey shows where the query node (LacI) interactions are found by finding direct neighbours with interaction edges. Like the derivative search, the interaction search could be expanded to cover the entire interaction compery. onent in the graph and return the whole regulatory system that controls the source node.

Usage : Parts used within a design can have an unknown and undesired interplay even when seemingly independent. Due to biology's inherent complexity, experimental approaches are the only way to identify this interplay reliably. The WKG captures a relationship between entities, which describes cases when parts are commonly found together within designs. While not an absolute metric, it could provide a higher guarantee level from previous designs to validate the feasibility of future designs. Figure 3 red displays the usage, where the specific edge is queried (usage), and the resultant nodes are returned (BBa_B0034).

Modules : Previously, abstract functional modules were defined within the WKG. Modules may be valuable for several reasons, such as identifying implementation from function, increasing the likeliness of correctness, abstracting biological complexity and fast prototyping. Once the modules are described within the WKG, the modules can be identified by traversals. Unlike the previous example, this query will return a subgraph instead of an individual node within the network (See network for creating module subgraphs). Figure 3 blue displays an example of returning the pLacregulatory module given an individual genetic part source (LacI). It must be noted that an entity may be part of multiple modules so that these traversals may find multiple modules.

Metadata : Although not particularly useful from a network approach, the written textual information may be helpful for users requiring a more natural description of an entity. Figure 3 green also displays that metadata can be returned from the same search query.

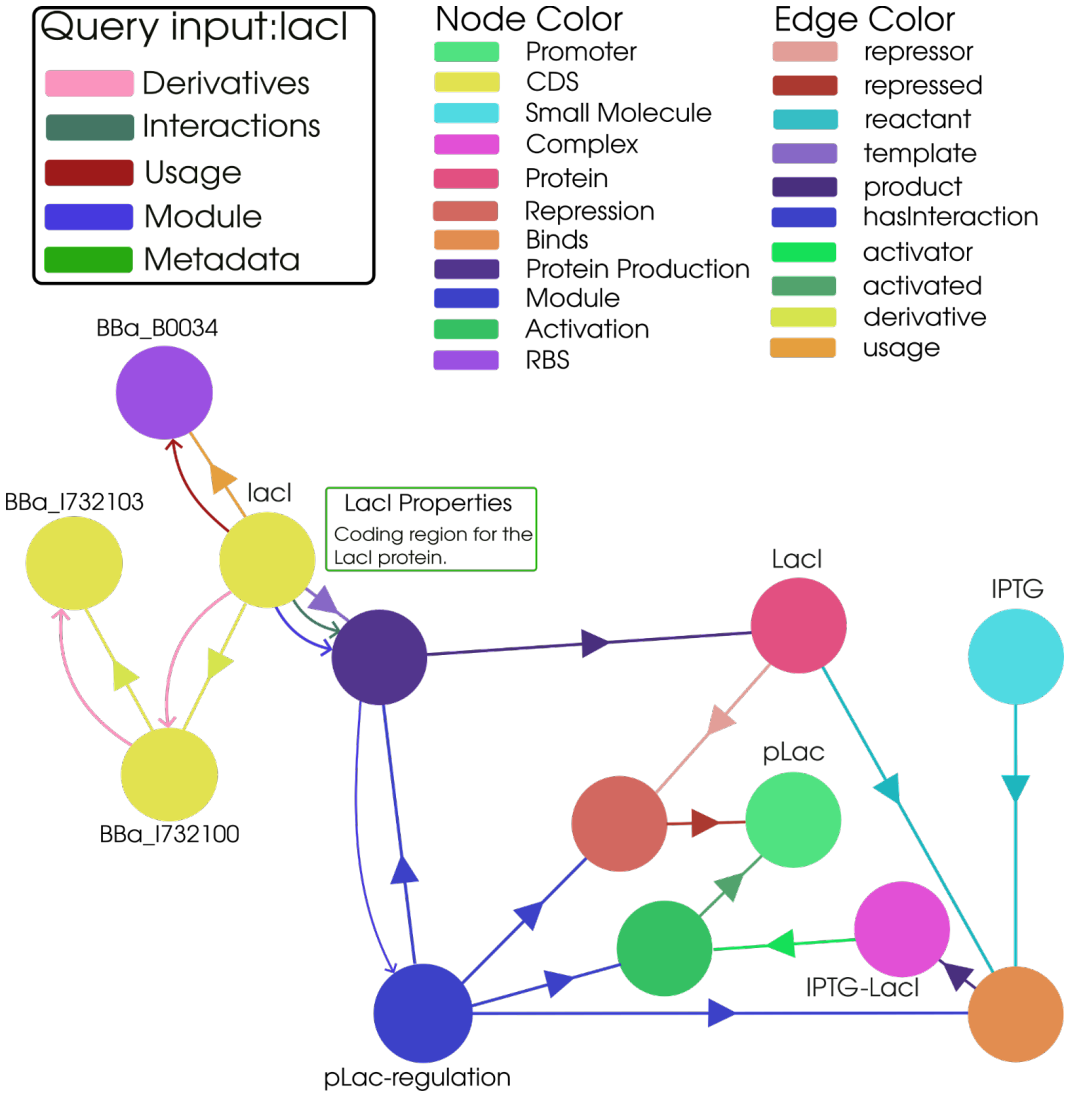

Figure 3: The same input query ("LacI") can return different information based on the search type. **Pink**: Walks the network for similarity using the synonym label. **Grey**: walks the network for interactions occurring using the interaction labels. **Red**: Traversal to find entities commonly used together. **Blue**: Walks the network to find modules within which the entity is implicitly contained. From the module, walk the network to find all interactions and parts that make up the module. **Green**: returns the metadata of the search query.

Indirect entity matching (Synonyms/Canonical & Fuzzy)

Issues arise when information querying moves away from sequence matching because one input may match multiple records. For example, multiple records from different databases may use the same name to define the part (in many cases, collisions occur within the same dataset). While this is not an issue within the WKG, as it ensures that each record is canonical, the opposite may arise where a single genetic part has many names. While it is impossible to know all potentially colloquial names, when found, the duplicate can be transformed into a "Synonym" node when duplicates are found instead of removing the duplicate. Figure 4 shows how this is structured within the network; it is simply a "Synonym" node (pTac and LacI) connecting to the canonical node (BBa_K864400). Although a genetic part or entity may have extra information capturing common aliases, it is impossible to capture every possible derivation. For example, retake the synonym node "pTac"; instead of querying the exact word, a user searches for "pTac promoter". No results will be returned if this query is directly searched within the network. In this case, approximate string matching may be employed to find "close" matches. Approximate string matching searches for strings similar or equal to a given pattern, even if they are not an exact match. For example, in Figure 4, if the user now searches "pTac promoter" and no matches are found directly, approximate string matching (see methods) can be employed to show that this input may be similar to the "pTac" synonym and, therefore, the conceptual match is found despite not being syntactically identical. It must be noted that this is a limited solution, but a more robust solution is complex and is outside this work's scope (see discussion).

Ranking results

When executing abstract queries, multiple results are often presented. For instance, tasks like deriving genetic part derivatives or conducting metadata searches may yield numerous outcomes. Traditional databases equipped with simplistic search systems tend to rank results arbitrarily, for example, by their order of discovery or alphabetically. However, this approach disregards crucial factors like data quality, reliability, and utility, burdening users to filter through the information. Advanced search engines, on the other hand, consider diverse external elements such as popularity, context relevance, and resource quality. These considerations establish a refined system wherein end-users seldom need to navigate beyond the first subset of results to discover an appropriate answer. Therefore, by embedding confidence metrics within the network and incorporating capabilities for precise matching (including partial sequence and fuzzy string matching), a query system emerges that takes strides toward presenting more pertinent outcomes. Using a breadth-first-search method, figure5 shows how derivatives are ranked when the graph is traversed from a source node (the query input BBa_K082003). Each node is assigned a score calculated by the formula: score = (parentScore * confidence) / 100. This scoring system is applicable because the provenance subgraph forms a forest structure, meaning each node has a degree of either 0 or 1 and thus can have only one score. After scoring, the nodes are ranked, with the highest scores placed at the top.

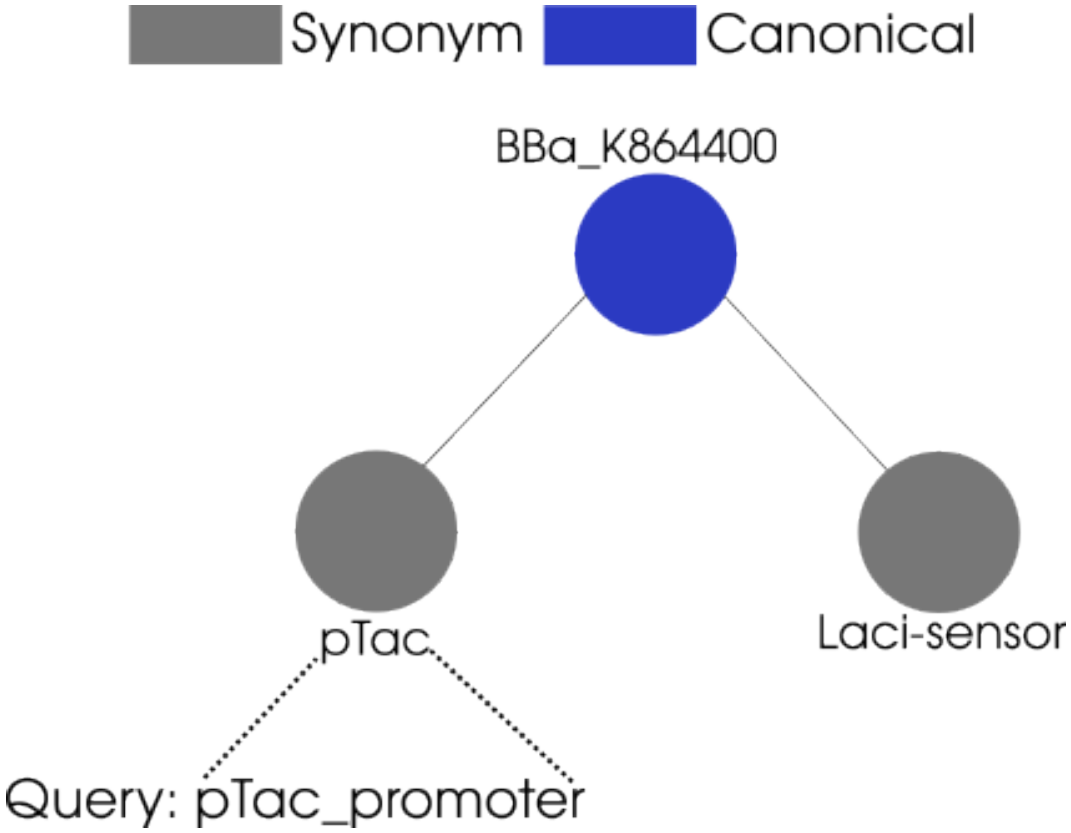

Figure 4: Representation of identifying a canonical node from a query. The first stage involves finding the synonym node "pTac" via a fuzzy string match between the input and node name, which is then connected to the canonical node "BBa_K864400".

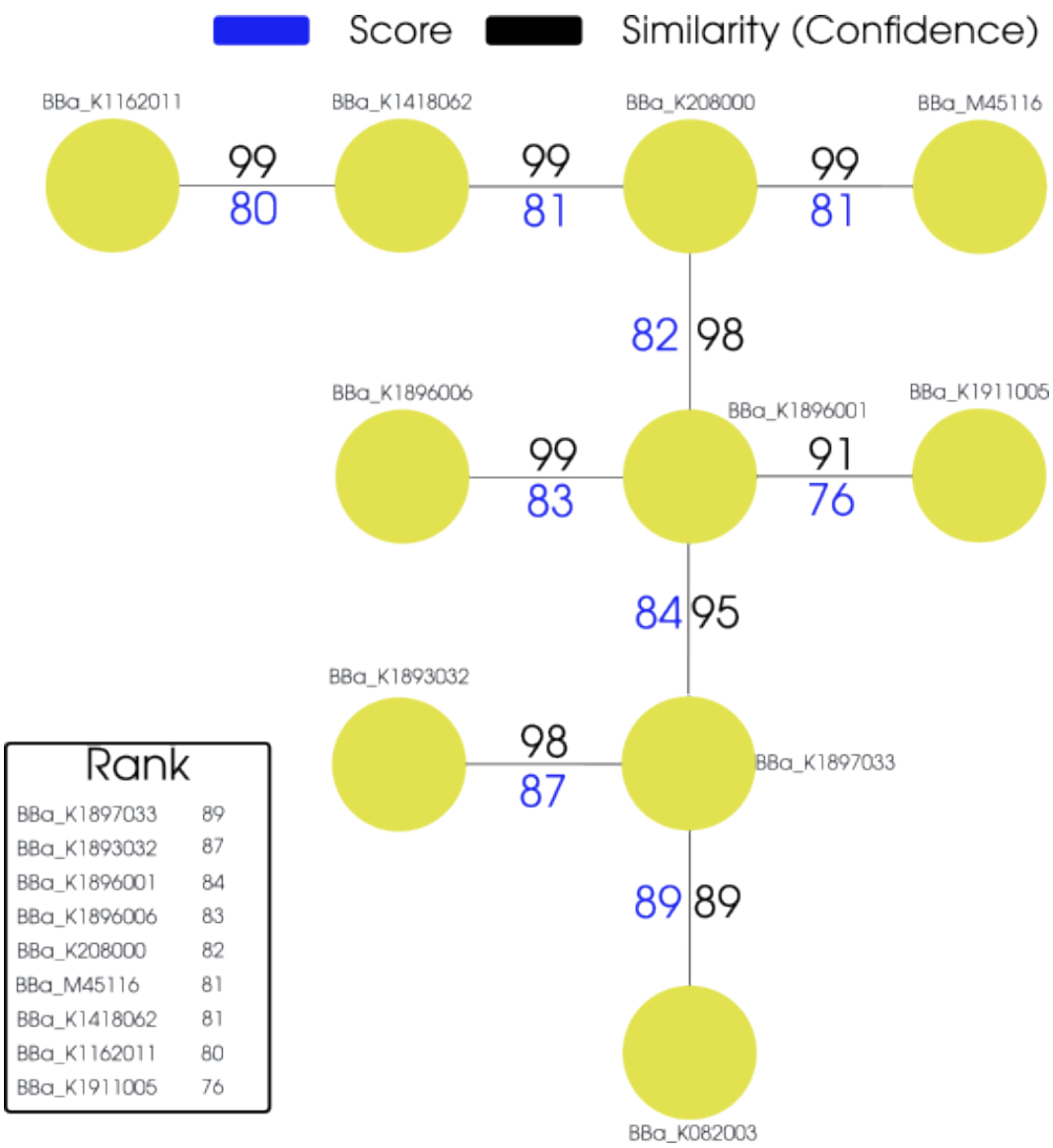

Figure 5: The ranking of derivatives of part BBa_K082003. The graph is traversed from the source node (BBa_K082003) to each derivative. The score of a node is calculated based on the score of the previous node and the confidence of itself; the higher rank indicates an entity that is thought to be closer to the source.

Integrating feedback

A key feature of the WKG is to accept feedback from agents, whether human users or automatically extracting information from sources of information. Feedback can be positive and negative, updating the confidence encoded inside all concerned edges. Confidence measures the likelihood that a piece of information is correct, thereby reducing uncertainty. Figure 6 displays three examples of how statements can be fed back into the WKG, where the integration method is specific to the datatype and if the information already exists in the WKG. When providing information from the knowledge graph directly to a human, the information may be reformed and not in the same structure. The restructuring is often because the underlying format is designed to capture large and complex structures, which means the connections between data are not always intuitive.

End-End query examples

This section covers some practical end-end use-case examples of how the WKG can enable superior querying with a technical explanation of how it is achieved using several scenarios, including normalising free text inputs, ranking results and integrating feedback into the network. See the Genetta section for a tool which implements this query system and can be used to replicate these results.

Scenario - Return similar genetic parts to BBa_E0040 This example 7 accepts the input BBa_E0040, which is already a canonical entity, identifies similar entities via the datatype, ranks these results based on similarity to the input, and accepts a feedback statement. The following steps are undertaken to execute a query for BBa_E0040, which is identified as the canonical entity:

- Initiate a breadth-first search on the derivative projection graph starting from the node representing BBa_E0040. It is important to note that this derivative graph consists of multiple disconnected components because not all parts are derivatives of one another; therefore, edges do not exist, which makes a connected graph. This search aims to traverse the component containing BBa_E0040, and the search begins from it.
- Each record encountered is returned during the search, which means that as the search progresses through the derivative graph, each node connected directly or indirectly to BBa_E0040 is identified and listed.
- Upon retrieving these records, they are ranked based on similarity to the initial query entity, BBa_E0040. This ranking process evaluates how closely each record matches or relates to BBa_E0040.
- For each record, a scoring mechanism is applied. The score is calculated using the formula: Score = (previous_confidence * current_confidence) / 100. This formula considers the confidence levels of the previous and current nodes, providing a normalised score that reflects the cumulative confidence of the pathway leading to each record.

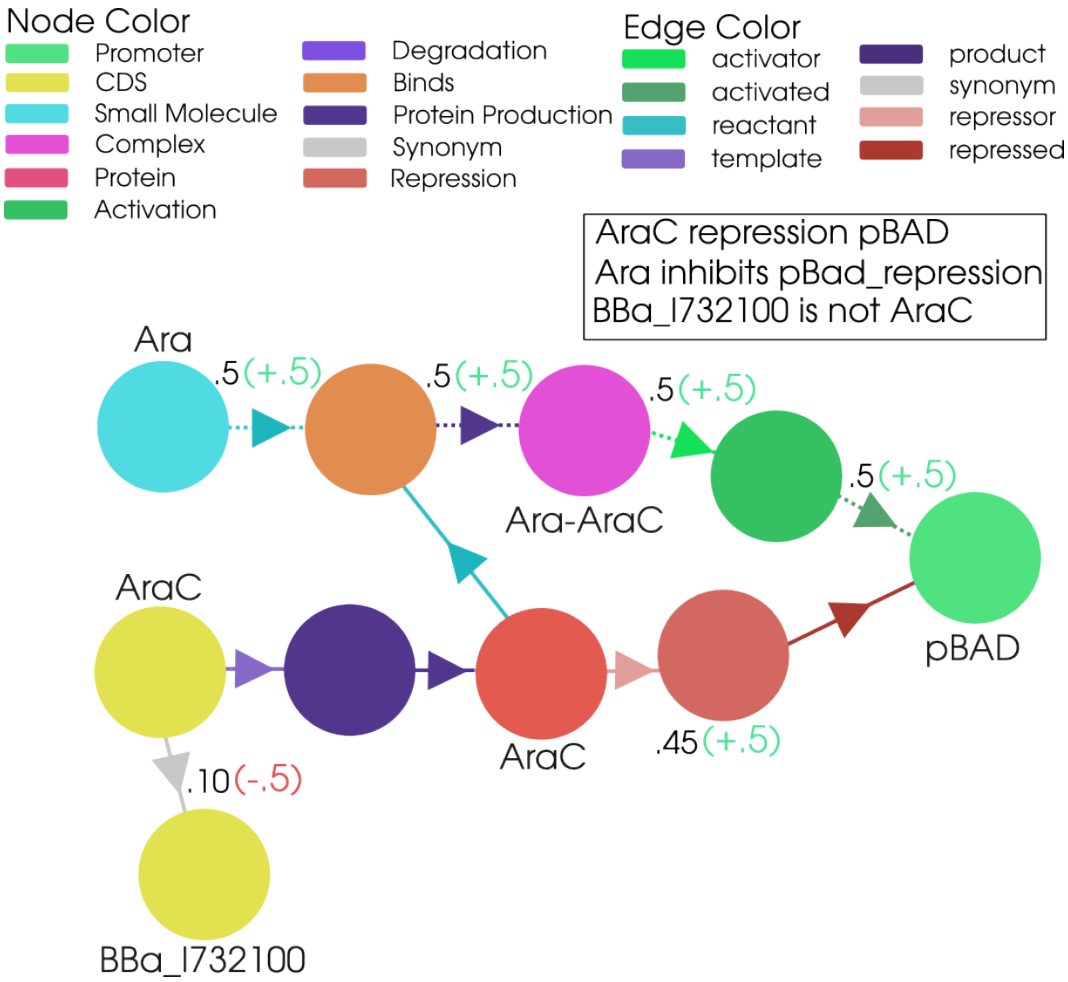

Figure 6: Example of three feedback terms used to update existing confidence or create new edges and nodes. In this case, the confidence increase is set at 5. "AraC repression pBad" maps directly to an existing edge; therefore, the confidence is increased. "Ara inhibits pBad_repression"; this case creates new interactions because the edges do not currently exist and sets the confidence to the initial value. "BBa_I732100 is not Arac" explains that these two values are not synonymous, reducing the confidence of the existing edge.

- If the user provides feedback on the search results, this is used to adjust the confidence levels. Specifically, positive feedback increases the confidence score of the presented records, thereby refining the search process based on user input. If the revised confidence score for a record is higher than the confidence level of the derivative node it is connected to, the existing edge is removed. Subsequently, a new edge is created that connects the node with the highest confidence score. This step ensures that the graph continually adapts to the most reliable and relevant connections based on the dynamic confidence scores influenced by user feedback.

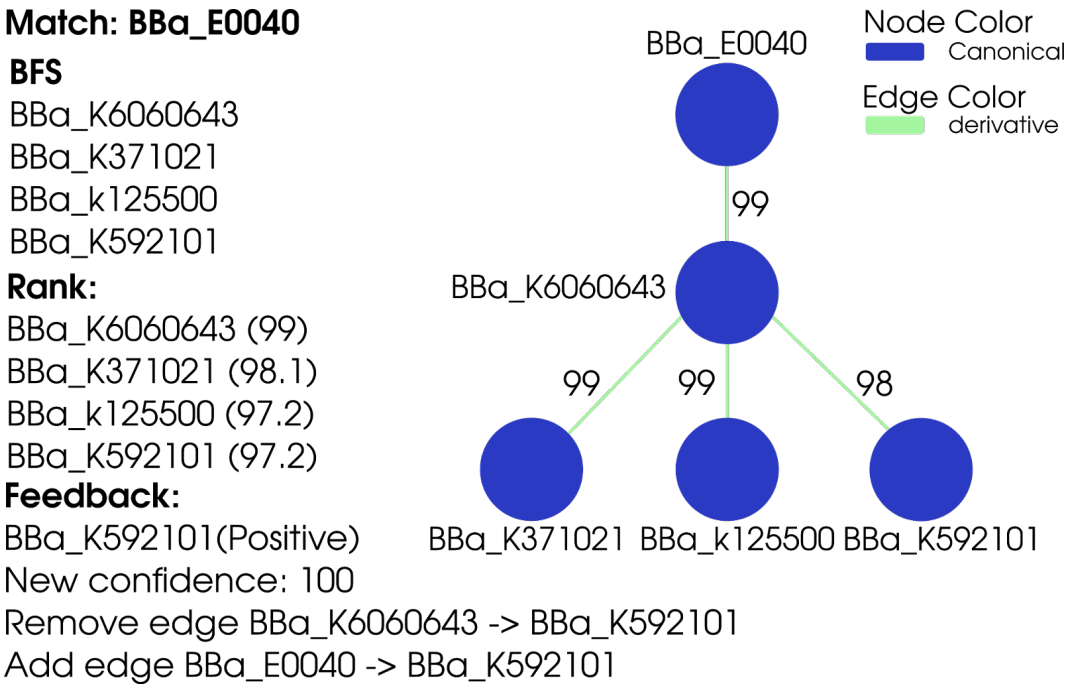

Figure 7: Visualisation of identifying similar entities to genetic part BBa_E0040 from the IGEM dataset. A breadth-first search is performed from the source (BBa_E0040). Because all derivatives graphs are disconnected, all related entities will be considered, and non-related ones will never be considered. Finally, the position of a node during ranking is calculated by $(parentNodeScore * nodeConfidence)/100$.

Scenario - Return regulators of LacI This example identifies 8 the interactions that the LacI entity has. The input is the "LacI-sensor", which is not a canonical entity within the network and must be resolved. In this case, only one interaction is found, so ranking is arbitrary. When feedback is provided, all physical entities involved in the interaction have their confidence increased. Initiate the query specifically for the "LacI-sensor", which is not a canonical entity:

- Identify the canonical edge from the "LacI-sensor" node designated as BBa_C0012 in the database.

- Begin walking the network, starting from the BBa_C0012 node. This process involves traversing the network to explore each interaction edge originating from BBa_C0012.
- As the network is traversed, focus on identifying specific types of interactions, in this case, the repression node. If an interaction edge from BBa_C0012 is identified as one of these regulatory types, that interaction is selected and returned.
- After identifying the relevant interactions, the next step is to rank each. This ranking is based on the confidence level associated with the interaction between BBa_C0012 and the interacting entity.
- If user feedback is provided during or after the query process, it updates all participants' confidence levels in each interaction. This change means that the confidence scores of both BBa_C0012 and the entities it interacts with are adjusted based on user input.

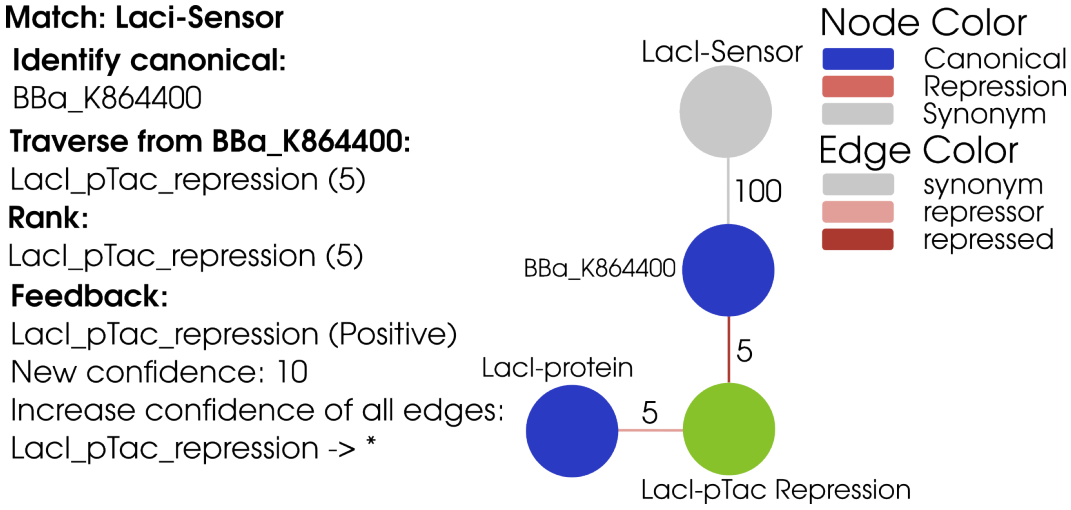

Figure 8: Identifying the interactions of results from the "LacI sensor" query. A canonical node (BBa_K864400) is found, a participant in a repression interaction.

Scenario - Return all wild-type t7 promoters. This scenario displays accessing metadata within the nodes of the network. The input query "wild-type t7 promoter" is queried where a fuzzy string search is performed within the metadata fields of each node. This displays the issues with written information; without complex language models (see future work), basic approaches to matching textual information will nearly always produce superfluous results alongside the desired results. A fuzzy text index query on all canonical nodes by the following procedure:

- Initiate a fuzzy text index query across all canonical nodes in the database. The query identifies BBa_K2084000 and BBa_K2150031 as the relevant nodes in this specific instance. These nodes are selected based on their similarity to the search terms used, as determined by the fuzzy text search algorithm.

- Once these nodes are identified, they are ranked according to a fuzzy string score. This score quantifies the closeness of the match between each node's name and the search query. Nodes with a higher fuzzy string score are deemed more relevant to the search terms and are thus ranked higher.

- If feedback is received from a user, it is used to refine synonym connections. Specifically, a synonym node is created for the canonical node to which the feedback pertains.

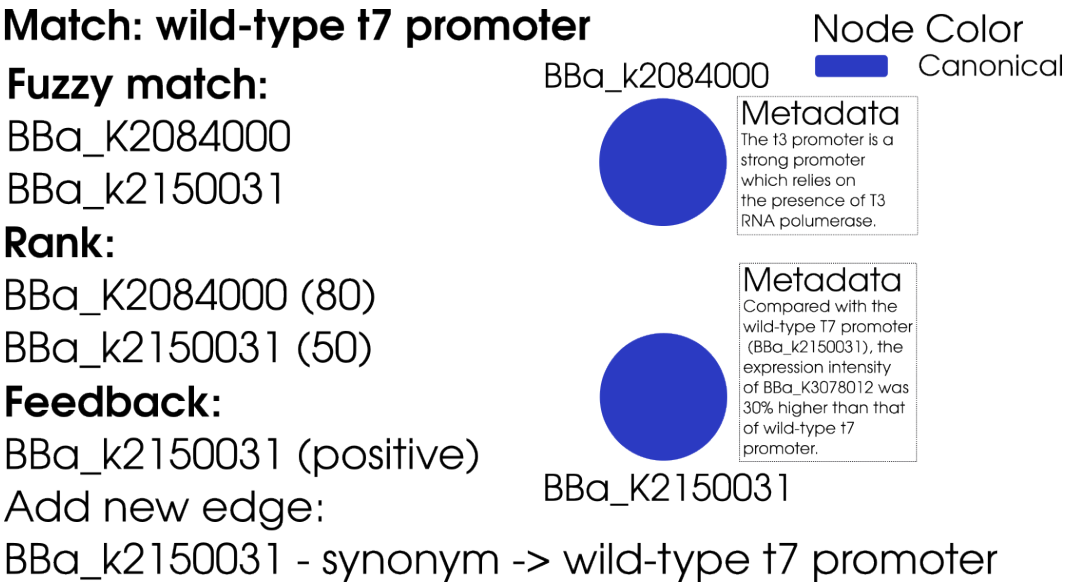

Figure 9: Fuzzy string matching on nodes within the network. The ranking is based on the fuzzy string score. Metadata searches often produce incorrect results. For example, BBa_K2084000 is the T3 promoter when the user requested the T7 promoter.

Scenario - Return all entities commonly used with BBa_J107113 This query returns all entities with relationships of the usage type with the input query (BBa_J107113), which is the canonical entity within the WKG. Rank is based on the confidence of each usage, and when feedback is applied, these values are updated. The following steps are taken to query for entities known to be used with the canonical entity BBa_J107113:

- The system identifies the canonical entity BBa_J107113, which involves locating the node corresponding to this unique identifier. Once the canonical entity is identified, the system then queries for all usage edges connected to it.

- These usage edges are then ranked based on the confidence level associated with each edge.

- If user feedback is provided, it is used to update the weights of these edges. A specific action is taken if an edge's confidence drops to zero. In this scenario, if the edge connected to node BBa_K10114 is found to have zero confidence, it is removed from the network. This

removal is based on the understanding that an edge with zero confidence no longer reliably represents a meaningful or accurate connection in the network.

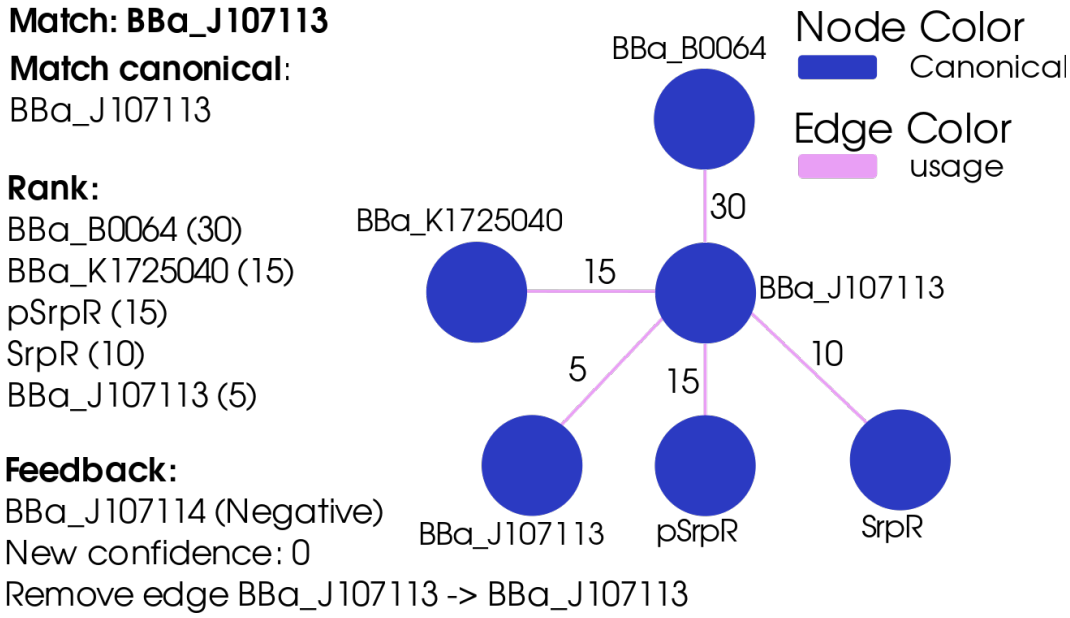

Figure 10: Usage search performed from the source node (BBa_J107113). Returns all entities within the WKG that are thought to be compatible with the source. Rank is based on confidence; feedback may remove edges if the confidence is zero.

Scenario - Return pTet regulatory system This example identifies modules containing the input query (pTet). In this specific case, a module encodes the regulatory mechanism. Broadly, a module is identified by searching for interactions and then the modules which contain the interactions. A module search will return a subgraph, which identifies all interactions and constituent parts (see methods for an extended explanation of decoding modules). The confidence of a module for ranking is based on the average confidence of the constituent edges. When feedback is provided, all the edges are inputted. The following steps are undertaken to query for the pTet regulatory module:

- Begin by querying for the canonical entity, in this case, pTet, by identifying the specific node that corresponds to it.
- The next step is to identify all the interactions in which pTet is a participant, in this case pTet-Activation and pTet-Repression. For each identified interaction, the system queries for any modules that contain this interaction, in this case, the pTet-Regulation module that encompasses the pTet-related interactions.
- The process involves traversing each identified module to find all constituent interactions. This step aims to map out the complete network of interactions that form the basis of the module.

- The next step for each interaction within the module is to identify all the participant entities, which involves mapping out all the nodes (entities) involved in each interaction, providing a complete view of the entities that constitute the network. This process results in a subgraph that includes all the nodes and edges related to the pTet regulatory module.

- If feedback is provided, it is used to update the confidence levels of all the edges within the module.

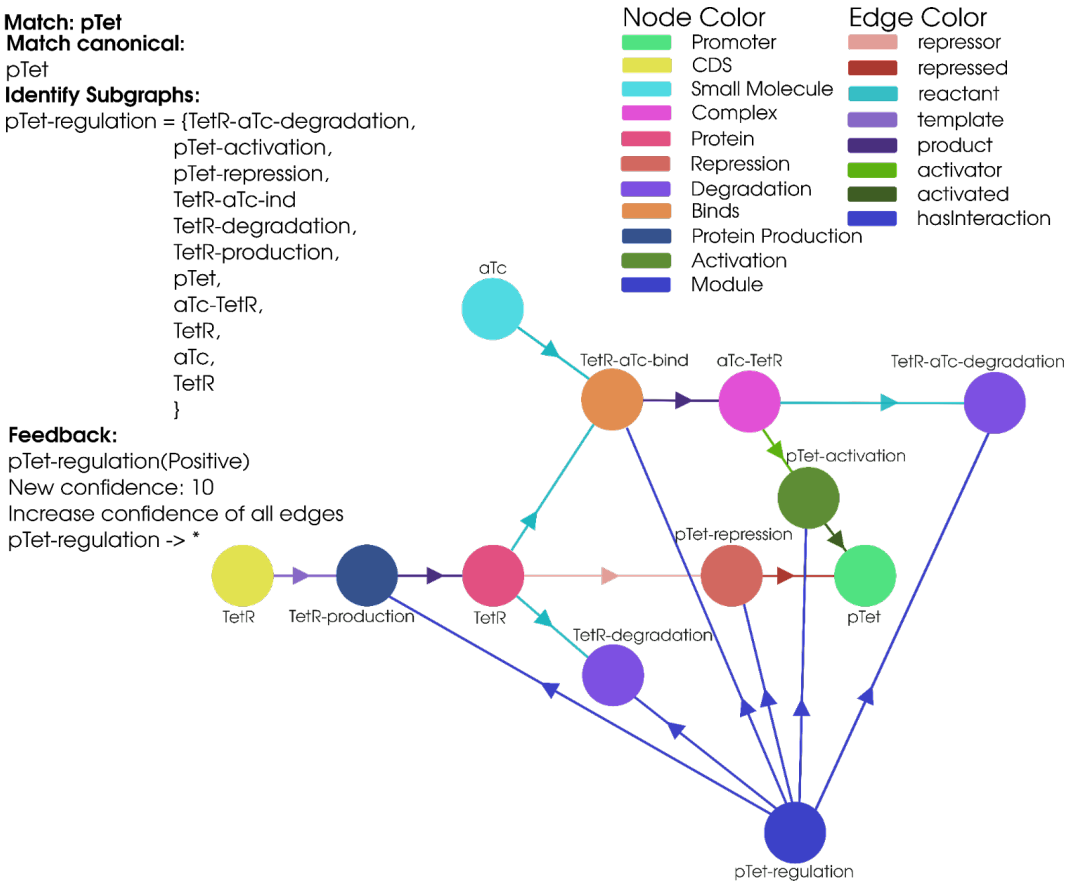

Figure 11: Identifying the functional module (pTet-regulation) given the search query pTet. When the pTet node is found, all connected modules (modules implicitly containing pTet) are returned. A subgraph is returned, denoting all the constituent nodes and edges. On feedback, all edges connecting the module to the interactions are updated.

2.2 Automatic enhancement of existing designs

Incorporating additional information beyond sequence data into a genetic design is beneficial because it enriches the design with a broader context and deeper insights, which can significantly improve the understanding and manipulation of genetic systems[181]. A more comprehensive

view of the biological system is achieved by integrating data such as functional annotations, regulatory elements, interaction networks, and environmental response factors[187]. This enriched perspective facilitates more informed decision-making in design processes, enables the prediction of system behaviour under different conditions, and enhances the ability to engineer more robust and efficient genetic constructs. However, as discussed, this information is seldom defined within databases or individual designs. This gap arises because most existing standardised designs are direct translations of designs originally encoded within formats that do not support including such information. Furthermore, the burden of manually inputting this extra information is often impractical for practitioners due to limited resources, time constraints, and a lack of incentives or clear guidelines on how to do so. This situation significantly underutilises potentially valuable data, limiting the depth and applicability of genetic designs.

The second use case proposes the computational application of the WKG to address this issue by automatically enriching design data with this often-missing information. This approach aims to streamline the process of incorporating diverse data types into designs, thereby overcoming the limitations of current practices and significantly enhancing the richness and utility of genetic design data. The design is a modified form of an AND gate by Jones and colleagues[154]. This example will help demonstrate how these enhancements can be practically implemented. This AND gate consists of two inverters whose outputs are fed into a NOR gate, ensuring the only output is when both inputs are high. The desired representation can be seen in Figure 12, where the interaction data is encoded. However, only sequence data with annotations are defined

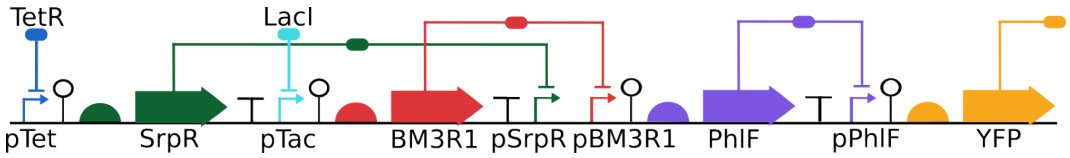

Figure 12: Visualisation represents genetic parts, proteins, and chemicals as glyphs. Interactions between physical entities are represented via lines. This AND gate [154] design consists of two inverters flanking a NOR gate, resulting in an AND gate design.

within the initial design data because this information would be extracted from a Genbank file. The names of annotations (genetic parts) are mixed, i.e. entities containing the canonical name, synonymous name, exact or close sequence, names similar to WKG entities or the name encoded within the metadata to explain multiple possible outcomes. The initial design network can be seen in Figure 13, which is a simple network where nodes denote each genetic part that constitutes the network. Currently, the network contains no edges because a flat design format does not encode any information about relationships between entities. However, each node will contain properties, represented in figure 13, by displaying the properties for BBa_B0064. Properties are encoded within nodes instead of individual nodes with edges to the subject because they do not play a role in the network's topology. The names and properties of parts are mixed, such as some containing sequence data and some not. This large domain of types and quality of information is designed to display each possible use case.

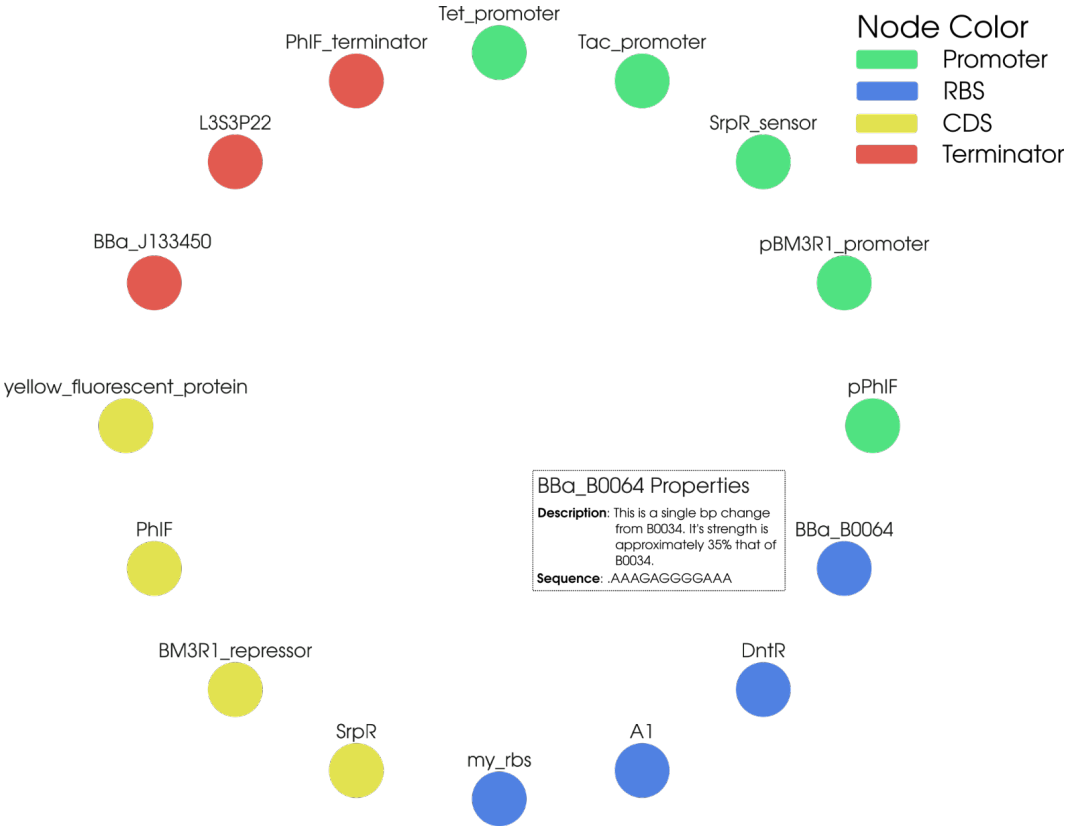

Figure 13: Unenhanced AND gate design represents each genetic part as an individual note. Node labels are truncated URIs for a more comprehensible visualisation, and each node contains properties. Visualised here are the properties for the BBa_B0064 node.

Canonicalise

An expansion is only possible if the existing data is tractable between designs. For example, if two designs use the part BBa_R0040 aka pTetR and the two designs use the alternative names (one BBa_R0040 and one pTetR). While humans understand they are synonymous, it is challenging to identify this computationally. Therefore, before a design can be enhanced, it must be first canonicalised relative to the WKG or resources on external databases, i.e. mapped to the exact virtual analogue.

WKG replacements The first step to canonicalising a design is to find matches between the design graph and the WKG. Matches with the WKG are the most desirable outcome because this graph will likely encode more information than an external database. Figure 14 displays the four potential match cases explained below.

Canonical entity - The entity is already canonical within the WKG. Figure 14 A displays the case when the entity (BBa_B0064) within the design graph is already canonical. In this case, no changes are made. Canonical name - The entity is not canonical but has the same name as a canonical entity within the WKG. A name usually pertains to the suffix of a resource but can also be any shortcode or name. Figure 14 B displays when the name (encoded within the properties of a node) matches the suffix of a WKG entity (A1). In this case, the WKG entity replaces the design graph entity.

Identical sequence - The entity has no matches within the resource name, but instead, the sequence is identical to a canonical entity within the WKG. Figure 14 C displays when the sequence of a design entity (encoded within the properties of a node) matches exactly an entity within the WKG (pTet). In this case, the WKG entity replaces the design graph entity.

Synonym of canonical entity - The entity is a synonym of a canonical entity within the WKG. Figure 14 D displays the case when the name of an entity within the design graph matches a synonym node within the WKG (SrpR_sensor). In this case, the graph is traversed to find the canonical entity (BBa_K1899004) and replaces the resource within the design graph. If all of these checks have been made without a successful match, then the entity does not directly reference any entity in the WKG by name or sequence.

External replacements The ideal outcome is that each entity within a design maps to a canonical entity within the WKG. However, it is impossible to encode each entity due to an incalculable number of potential genetic parts. Therefore, cases may occur when no matches occur. Therefore, because an entity does not exist within the WKG, this does not mean there is not a virtual analogue. An analogue may exist on external databases not contained within the WKG. Figure 15 displays the three potential match cases similar to identifying on the WKG but using external datasets, which are explained below.

External canonical entity - The entity already references an accessible external resource. Figure 15 A displays when the entity within the design graph references an actual resource. In this case, no changes are made.

External canonical name - The entity resource does not resolve to an accessible external resource but has the same name as an external resource. Figure 15 B displays when the name (en-

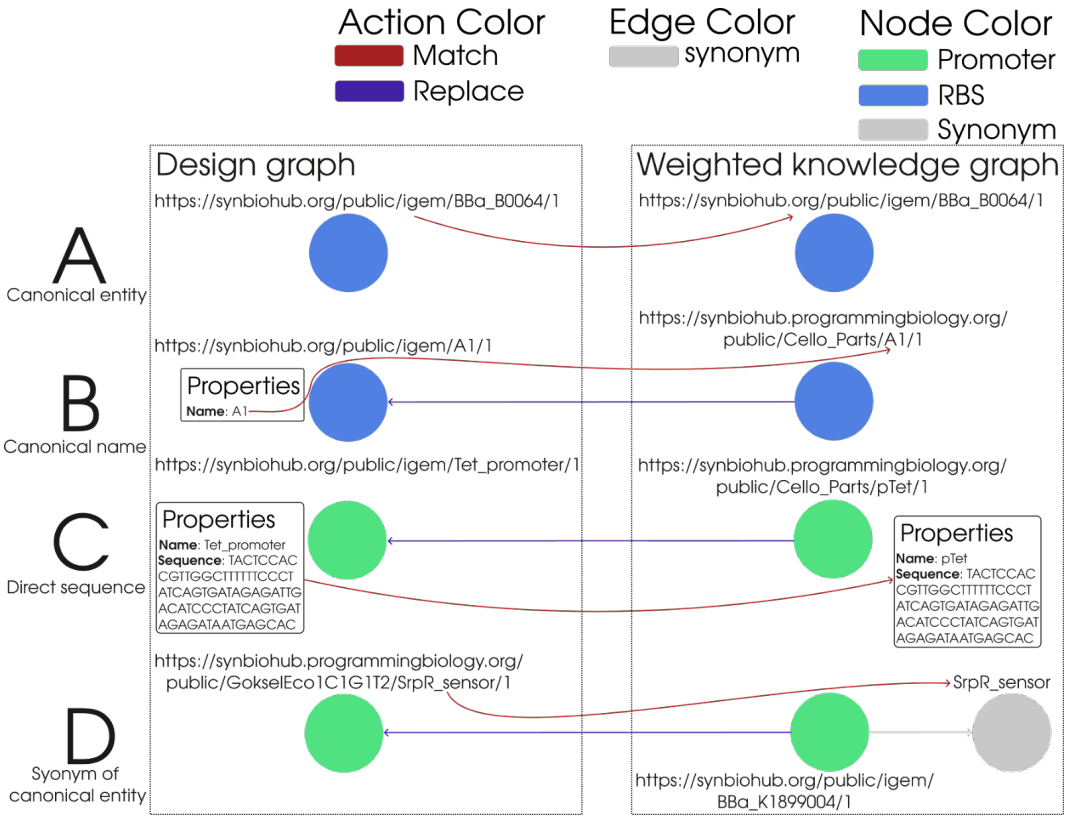

Figure 14: The four cases can occur during WKG definitive canonical match and replacement. A) When the design entity is identical to the canonical WKG entity. B) When the name of the design entity is contained within a canonical WKG entity but does not refer directly to the same resource, i.e., the final path in both URIs is the same. C) When the sequence of the design entity directly matches an entity within the WKG. D) When the design entity is captured as a synonym of a canonical entity in the WKG. The design entity is replaced with the WKG entity in cases B, C and D.

coded within a node's properties) matches an external entity's suffix. In this case, the external entity replaces the design graph entity.

External identical sequence - The entity has no name matches; instead, the sequence matches an external resource. Figure 15 C displays when the sequence of a design entity (encoded within the properties of a node) matches exactly an external entity. In this case, the external entity replaces the design graph entity.

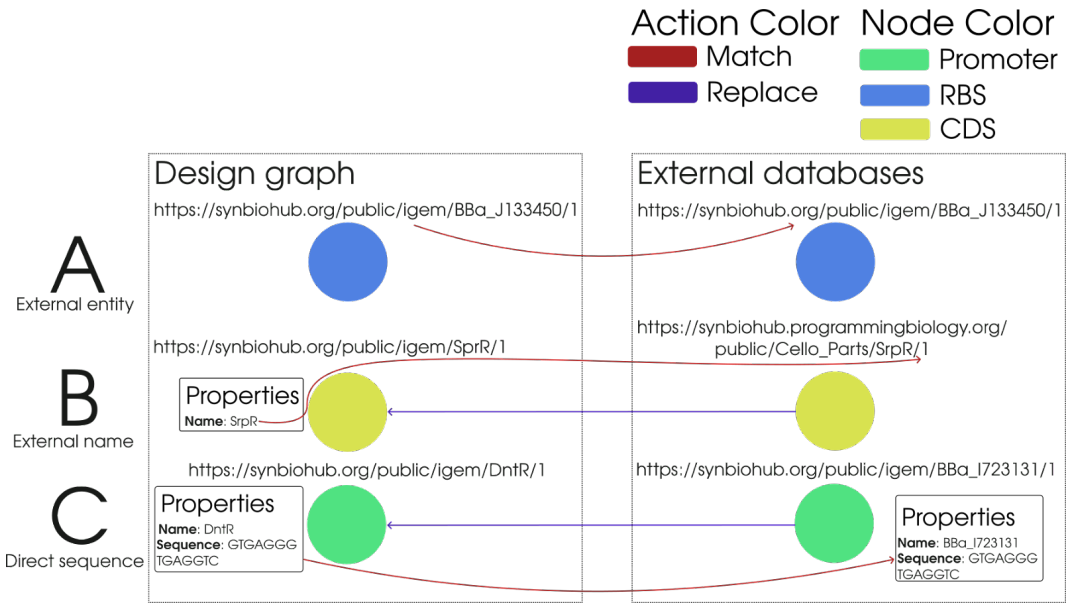

Figure 15: The three cases can occur when searching for direct external matches and replacements. A) When the design entity directly references a real external resource. B) When the name of the design entity is contained within an external entity but does not refer directly to the same resource. C) When the sequence of the design entity directly matches an external resource.

POTENTIAL REPLACEMENTS Despite searching both the WKG and external resources for canonical versions of entities, the case may still occur where no match is made because no absolute evidence can be found, likely because the URI is a purely local term and does not route to an available resource. In this case, potential replacements can be explored. These are replacements in which some evidence of a relationship exists, but it is not absolute, such as in the case of a direct sequence match. Figure 16 displays the three potential match cases, which are explained below. With a fully automated approach, it is unlikely to be integrated (see semi-automated enhancements for further discussion) because it is highly likely to introduce errors in the design. For example, if the sequence is not a direct match, the function may not be the same and may break the design somehow.

Partial sequence matching - A common approach to finding related entities is via sequence alignment algorithms to measure similarity. The scores of partial matches here are based on sequence similarity, where a higher score equates to a more similar primary structure. Figure 16 A displays a case where the sequence of the design entity partially matches two entities from the

WKG. In this case, the highest score is taken. However, it is not always the case that the entity with the highest score (or any entity) is genuinely the canonical version.

Fuzzy string matching - Names of entities can often be conceptually identical but syntactically different. For example, if there is a spelling mistake, extra information or even lower or upper case differences, the entities are syntactically different despite conceptually referring to the same entity. While fuzzy string matching can help alleviate this issue, it is a massive challenge within computational approaches (such as enhancing designs) because it is unclear when entities become conceptually different relative to syntactic changes. This simple approach implements fuzzy string name matches for cases where entity names are similar but not identical. While this is not an infallible solution, no other quickly deployable processes that focus on the language within synthetic biology, such as part names (see future work), are available. Therefore, the score is based on the Levenshtein distance, the minimum number of single-character edits required to change one word into another. Figure 16 B displays an interesting occurrence where the design and WKG entities appear identical. However, the URI does not resolve to a valid resource due to a small spelling mistake (i is used instead of l, which looks identical when upper case). In this case, the incorrect entity is replaced with the canonical version.

Metadata matching - Another case is when matches occur within an entity's metadata. Even when a direct match is found within metadata, without context, it is unclear if the match refers to the containing entity. For example, within the IGEM part: BBa_C0012, the written information states, "LacI binds to the pLac regulator BBa_R0010". Humans can understand this information, but it is challenging to understand context computationally because it is impossible to account for all the possible combinations in which information can be written. Therefore, taking the code "BBa_R0010" as synonymous with BBa_C0012 is incorrect and thus why matching within metadata are only taken as potential replacements (see future work). Due to limitations, direct matches within written text metadata are taken as a potential replacement. Therefore, metadata matches are based on fuzzy string scores over the description sentence. Figure 16 C displays the occurrence where the design entity's name is contained within the description of an external entity. Despite a direct string match within the description, the score is relatively low because neighbouring words are also considered. In this case, the external entity replaces the design entity. Once all entities have been considered potentially replaced (providing a replacement is found), the current AND gate network can be represented within Figure 17, which is identical to Figure 13. However, each entity is now labelled with the canonical URIs either from the WKG or via an external resource.

Transferring knowledge

So far, the design contains a set of physical entities denoting genetic parts, which refers to virtual analogues within several existing datasets. However, this process aims to provide a standard framework to introduce functional information, namely interaction data. Therefore, the first step to introduce new data is to transfer any information from the WKG into the design directly. Extracting interaction data from the WKG is simple, providing that the physical entities exist within both networks. An assumption is made here that if an entity is contained within the design network, it is valid and canonical. However, this does not ensure that an entity that should be within the network to satisfy defining an interaction is present, i.e. none-genetic elements that may play a role in an interaction which can not be defined within a Genbank file. Furthermore, this assumption

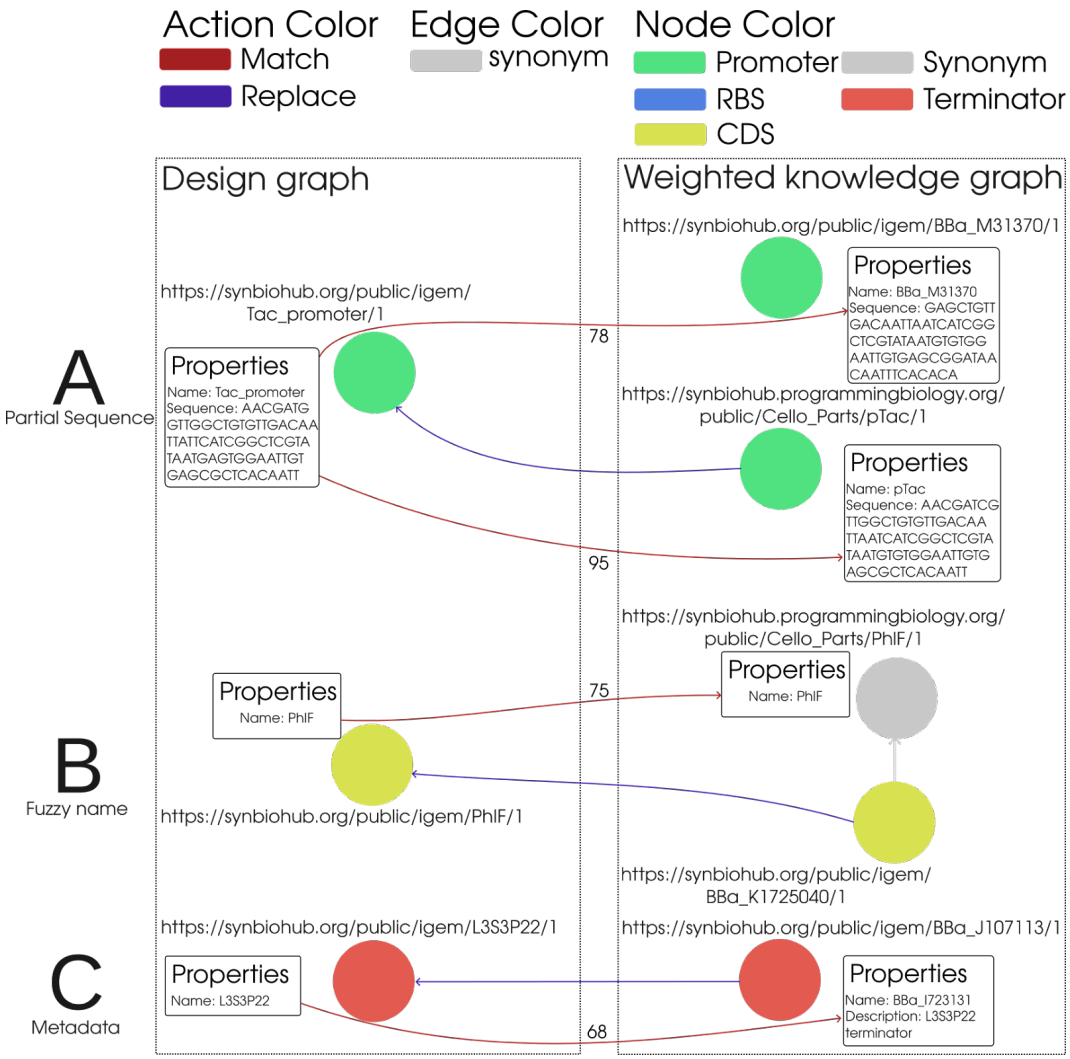

Figure 16: The three cases can occur when searching for potential replacements. Because matches are not absolute, matches are scored with a metric based on the match type. A) When the design entity partially matches the sequence of two other entities. The match score is based on sequence similarity; the highest score is replaced in this case. B) When the name of the design entity fuzzily matches an entity from the WKG. In this case, the match is a synonym of another node, so the canonical node is swapped. The score is based on the fuzzy string match score. Note that the names appear identical; however, one contains the character l and the other I. C) When the name of the design entity is contained within the metadata of an external entity. The score is based on the closeness of the two strings (in this case, the surrounding words are also accounted for).

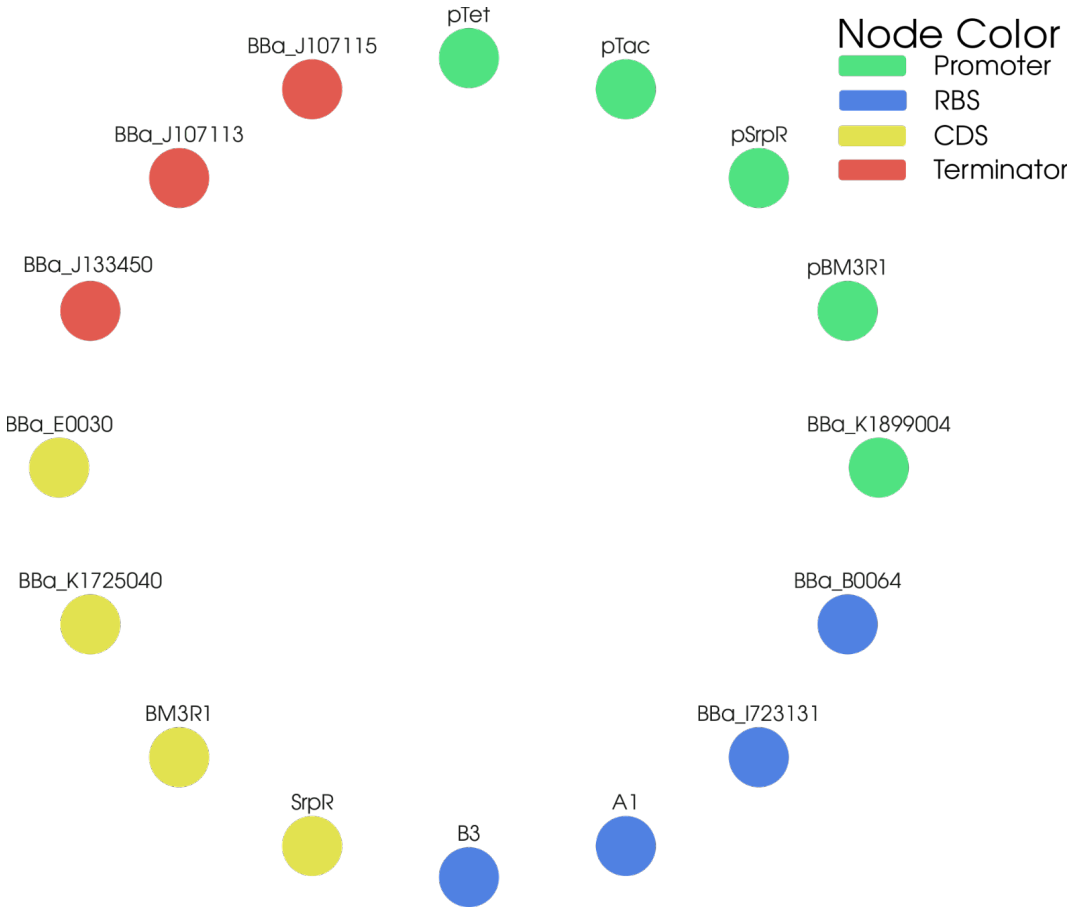

Figure 17: The canonical version of the AND logic gate. Each part refers to an actual resource that can be accessed within the WKG or external database. Node labels are truncated URIs for a more comprehensible visualisation. No edges exist because no relational information is added during canonicalisation.

of canonical entities does not extend towards interactions. Identifying a specific resource that references a specific interaction is often impossible because the exact interaction and combination of participants will unlikely be defined within existing datasets. Figure 18 shows how matches are made between the design and WKG, and data is transferred into the network. The process is simple: find matches of physical entities denoted by nodes within the design graph and the WKG. In this case, a match on the pTet and BBa_K1899004 promoters is found between the design graph and WKG. Here, the network will transfer all known and relevant functional information into the network. Because the WKG contains irrelevant information to be inserted into a design, such as synonyms, the whole subnetwork is not merged. Instead, the interaction subgraph is projected from the WKG and merged. Two minor considerations must be met. Firstly, interaction participants must also be transferred if not present within the design graph. For example, with Figure 18, the original design does not encode proteins and other non-genetic elements, which are incorporated. Secondly, if the design already contains a semantically and structurally identical interaction, the WKG interaction is not added. The interaction does not have to be referentially identical because, as discussed, it is highly uncommon for interactions to be encoded within datasets. It is even less common for inter-designed references of the same interaction. A conceptual interaction match is made by checking that the participants (physical entities involved) are equal and how they interact, i.e., the labels on the edges are identical (see methods for an extended explanation). At the end of this transfer, 21 new interactions are added, as displayed when the design graph is visualised within Figure 19, where each interaction within the network references a virtual analogue. This process displays the power of having data encoded semantically, structurally and primarily referentially because no complex and potentially stochastic methods are required to reason within and between datasets. However, all information extracted so far from the WKG is agnostic of the design, meaning that the information is not specific to the features of the design, such as the structural positioning of genetic elements. Context-dependent information is not encoded within the WKG because it is highly coupled to a specific design, meaning it would be useless outside its context. The network representation (figure 19) shows the effects of missing contextual interactions, where several independent components are present, each defining a regulatory system. With a complete network, these should be connected based on the design composed from a positional perspective.

Context dependant expansion Integrating and expanding existing datasets into the WKG creates a dataset of design-agnostic information, which means that no information that depends on the context of a specific design is contained. For example, position-dependent regulation where the effects of genetic parts (such as genes, promoters, enhancers, and other regulatory elements) may rely on their position relative to other parts within the genome. Context-dependent information is not captured within the WKG for this exact reason; it may be the case that the specific makeup of a design does not apply to any other design in existence, and therefore, the requirement to encode every potential combination is impossible. The only exception to this rule is in the case of functional modules, where the overall function may depend on the module's context, for example, the relative position of transcription factors. Context-dependent information is essential to build a more complete design network. As seen in Figure 19, the current representation of the design is incomplete; it consists of several small components and some individual nodes with no edges. This section explores methods to derive context-specific information based on the existing

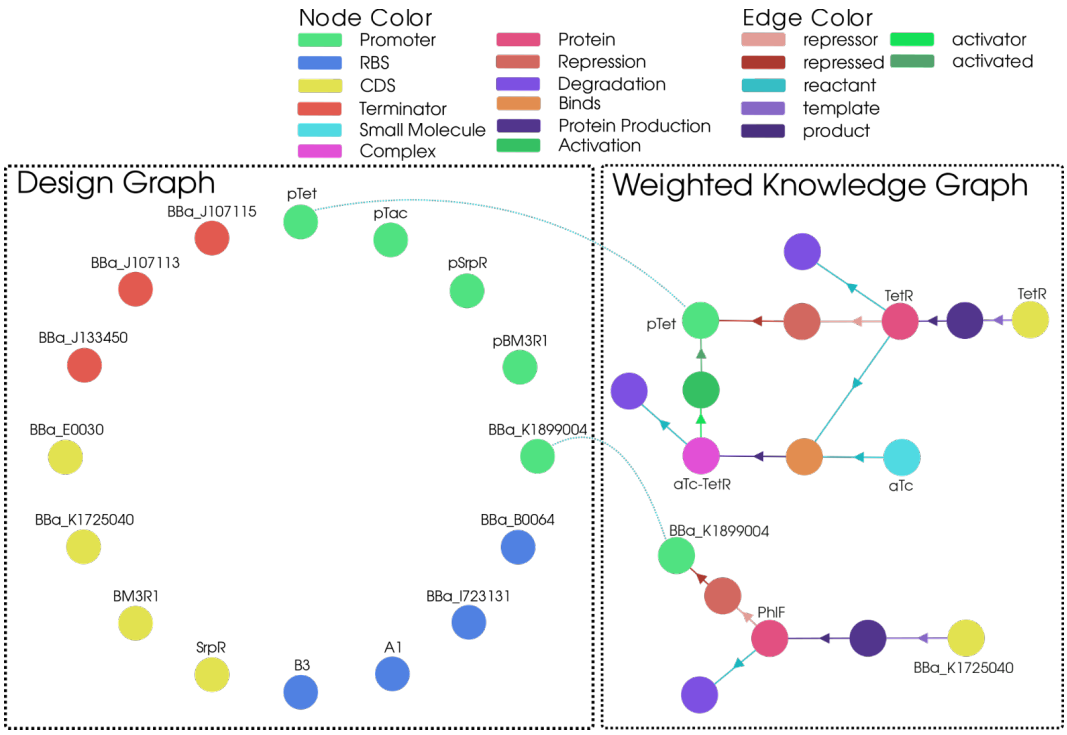

Figure 18: The process to transfer interaction data (in this case, the TetR and PhlF regulatory system) into the design graph. The match is made via the pTet and BBa_K1899004 nodes. Left - The design graph without any edges between nodes. Right - The interaction information known for these parts is encoded within the WKG. Once the match is made, these subgraphs can be merged into the design graph.

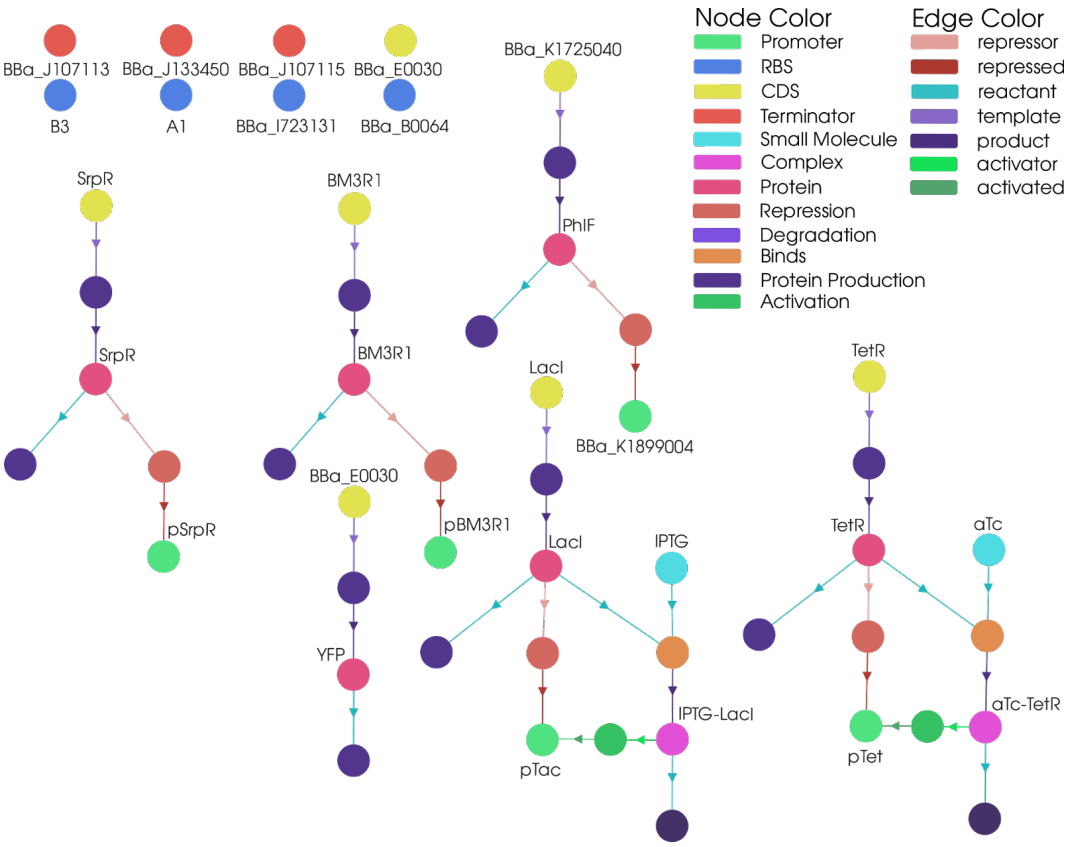

Figure 19: A network has been created that incorporates new interaction data extracted from the WKG and integrated into the AND logic gate design. The newly added nodes in this network represent the interactions between various physical entities. Correspondingly, the newly introduced edges in the network depict how these physical entities participate in these interactions.

information encoded within the design. Positional information (the location of genetic parts on the sequence) is always encoded within design information, even with semi or not-structured formats such as Genbank. Some interactions can be inferred purely based on the relative position of a genetic part on the primary sequence. Therefore, an addition to the functional side of the design is to encode the interaction between upstream promoters of CDS, which initiate their translation.

Positional interactions can be inferred through context-dependent expansion, as illustrated in Figure 20. Firstly, a network is projected based on the original design's relative positioning of genetic entities (see methods for the projection of the positional network). This network is traversed starting from each promoter, and interactions are added as the traversal encounters a CDS. When a terminator is reached, the traversal concludes. Figure 20 illustrates the absolute positions of entities with nucleotide positions and demonstrates the initiation of BBa_E0040 by the BBa_k189904 promoter, thereby connecting the PhIF regulatory system to BBa_E0040 protein production. Four additional connections are not depicted in this context for comprehension. However, they are applied, resulting in the final design represented in Figure 20, which demonstrates the integration of regulatory mechanisms from the WKG into a single-component network based on interactions derived from promoter-CDS positions.

Modularising groups Functional modules that are interchangeable components that perform specific aggregate biological functions were established within Chapter 4. These modules are often standardised units of genetic information or biological parts that can be combined and recombined to create novel biological systems or organisms with desired functionalities[9]. Within the context of enhancement, modules can be added with designs in two different states: a design with interactions and one without interactions present. However, because the previous enhancement would add all of the interactions within the modules, the assumption is that the design either already contains interactions or has recently applied the previous enhancement. Module matches are made in the design graph using a much simpler approach than establishing the modules in Chapter 4. Because the specific implementations of these modules are now known, the computational cost and complexity of subgraph matching are not required. If the design graph is canonicalised relative to WKG, it is as simple as checking for matches between the two graphs on individual nodes.

Figure 21 displays how modules are introduced via direct node matching between the design graph and WKG. Each node within a module (this check requires that each node has the same edges) is searched for within the design graph, and when all nodes of a module are found, the module is integrated into the design graph. The example given is the BBa_K1899004 regulatory system, where each match is made between design and WKG between the physical entities (BBa_K1899004, PhlF and BBa_K1725040) and the interactions (GeneticProduction, Repression and Degradation). However, as discussed, it must be noted that the interactions are not checked for referential matches because the reality of standards is that the URIs seldom refer to the same virtual analogue (see methods). The module is integrated into the design graph when all matches are made. This figure presents the module nodes within the design graph as boxes around the constituent parts for easier comprehension, where, in reality, the module node and edges would be transferred. One consideration that must be made, however, is to identify what parts of the module must be added to the design graph and which are already present. For example, even if a design is canonical relative to the WKG, some entities may not be present in the

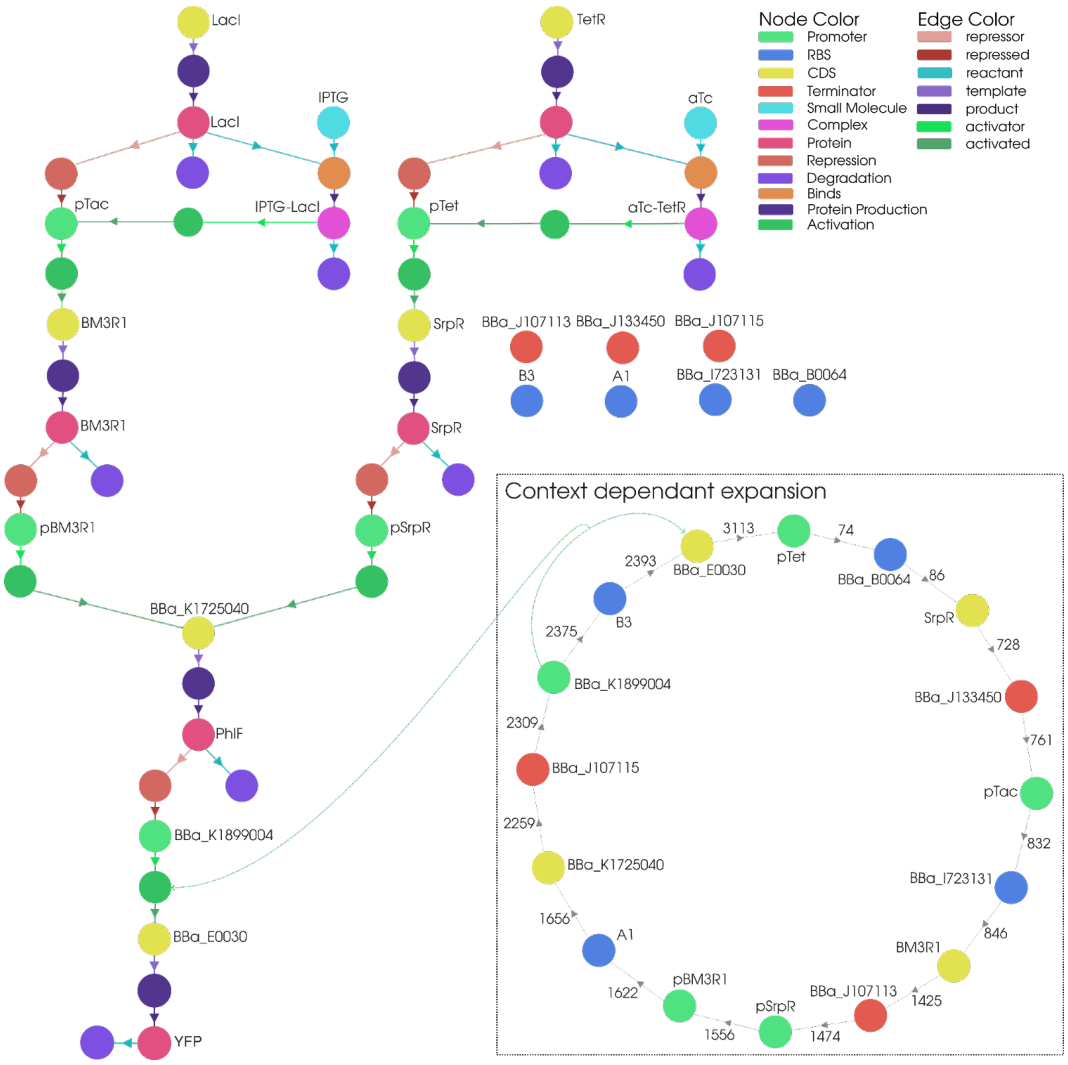

Figure 20: Network containing new interaction data extracted from the WKG and inserted into the AND logic gate design. Also, the network is connected using the positions of genetic parts and the functional impact of their positioning. The context-dependent expansion displays the projection of the relative positions of genetic parts within the AND gate with one example, which attaches two regulatory systems.

design graph, such as protein, non-genetic elements or even nucleotide-based parts, which could be unknown to the designer.

A question may be raised about why these modules cannot be added when the regulatory systems are introduced in Figure 19. While this is true, in this case, with low-level modules introduced into a system which only contains interactions taken from the WKG, these modules could have been inferred. However, different cases would make this impossible. Firstly, if the design that is being enhanced already contains the interactions, then the interaction enhancer discussed would never be applied to the design. Also, if the design already has a partial set of interactions, the constituent interactions of these modules would not be fully transferred. To expound on these points, the modules discussed in this section have been context-independent and relatively small. However, in the future, modules could be more extensive and context-dependent and even contain modules within themselves (a module of modules). Therefore, the low-level interaction enhancer would not be concerned with these modules. In short, if all interactions from modules are transferred, the module could also be defined in the design simultaneously. However, this is not done because this cannot be guaranteed, and enhancements are designed to be isolated changes to the network (see Genetta for an implementation of an enhancer pipeline).

2.3 Genetta

Genetta is a tool established within Chapter 4 and contains an implementation of the WKG. Genetta also implements the mechanisms to realise the two use cases discussed within this chapter, namely a query and enhancer (both automatic and semi-automatic). The software application is accessible from the repository at Genetta-Github and an online instance can be sampled here: Genetta-instance. Genetta implements a query system which interfaces with the WKG given human inputs. An example is displayed in Figure 22 and has several features. Firstly, it enables the type of query to be specified, as described in figure 22 A. For example, the query can be tailored to search for functional modules, usage, derivatives, metadata or sequences. The next feature is the ability to visualise the results (Figure 22 B). Each returned entity can be visualised using the techniques established within Chapter 6 to gain insight into the primary data, its neighbours, and the context of the information in the larger dataset. The results will be ranked based on the weights within the network (Figure 22 C). For example, information with higher confidence will be placed higher. The query system can take the user's feedback (Figure 22 D) for each result provided. The confidence and other weights are updated based on the relationship of the input query with the result being examined. Genetta implements an enhancer system which can enhance designs. All enhancements have been presented in this section as atomic changes made directly to a design document. However, with information within a dataset which is not guaranteed, incorrect information may be passed into a design. The implementation of the enhancer within Genetta can run automated or semi-automated. With the automated version, a design will be provided, and each enhancement within Genetta will perform operations on the network and add, remove or modify a specific type of information (seen in all examples so far). The semi-automated version is similar, but after each enhancer identifies improvements, Genetta will ask the user to validate them before they are added. A semi-automated approach also provides a second benefit, the ability to gauge user feedback, which is impossible with the automated approach. Figure 23 displays a sample of the enhancements described within the enhancements use case, which introduced a

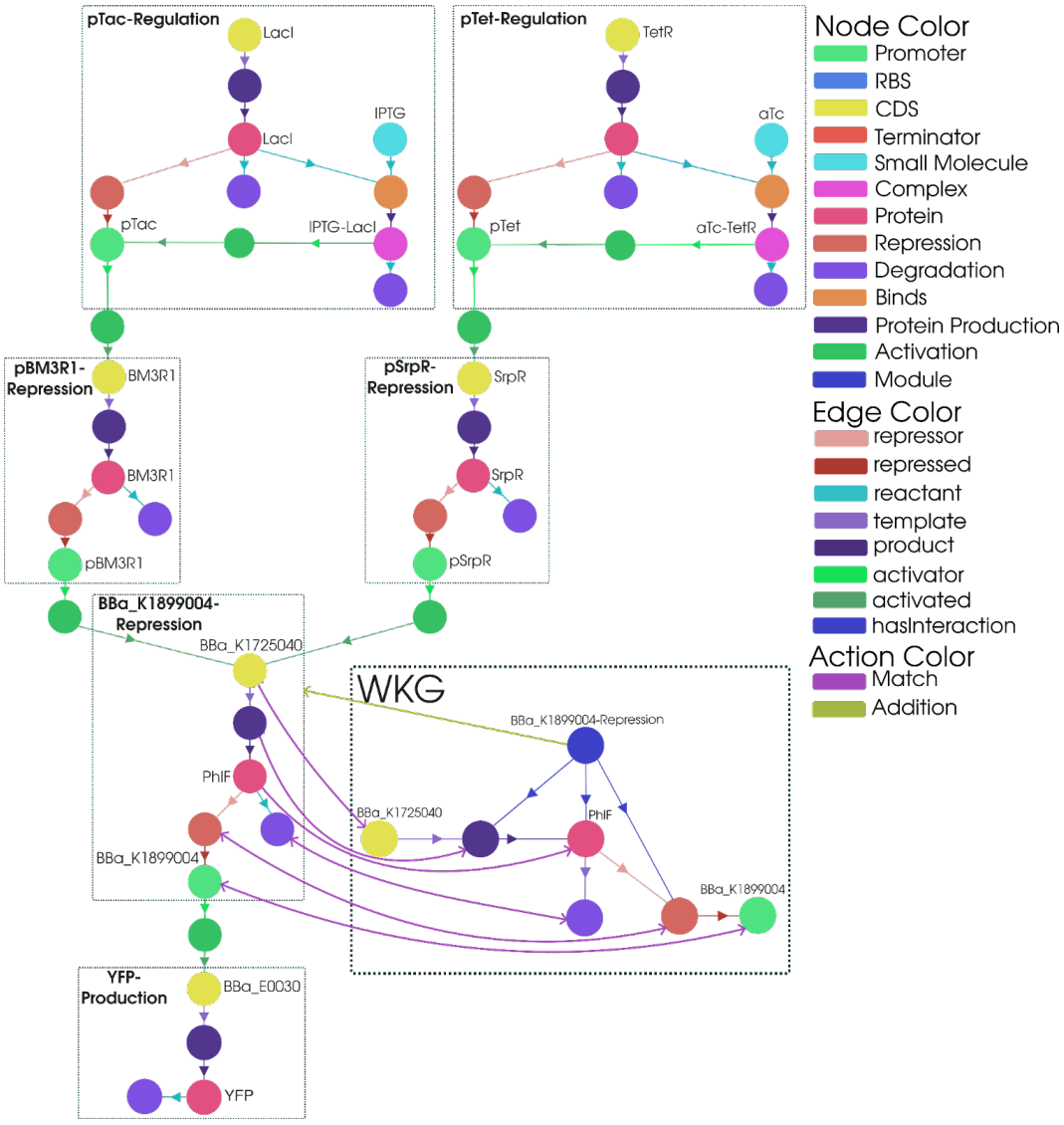

Figure 21: Network containing AND gate with all interaction data. These interactions are modularised relative to their regulatory function by matching each entity within a WKG module with an element in the network. An example shows the BBa_K1899004 regulatory system in the design graph and WKG. When all matches (purple lines) in the WKG are satisfied, the module is encoded within the design graph (yellow line).

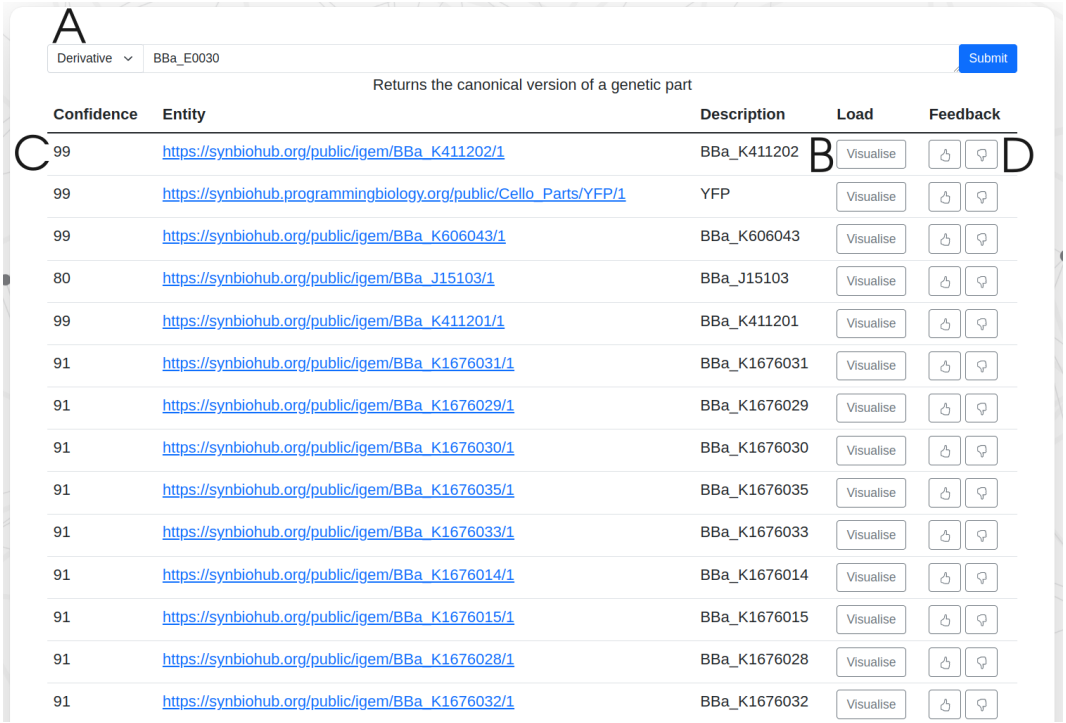

Figure 22: Genetta's query system. Takes written information and results information from the WKG. A) The user can change the type of information to search. B) The results and neighbours can be visualised to understand the context. C) An aggregated score ranks information depending on the search type. D) Feedback can be given on which weights are updated in the WKG.

set of interactions from the WKG on the AND gate example. Here, a simple box-check system offers both the WKG and contextual enhancements. If an enhancement is accepted, it is added to the design using the same methods the enhancer uses in the automatic approach. However, the enabled enhancements will also have their confidence values increased. It must also be noted that some enhancers do not directly use the WKG to identify improvements. For example, Figure 23 displays the positional enhancement discussed previously and doesn't use the WKG because the information is derived from a contextual aspect of the design. In this case, feedback would not be applied to the WKG because it would be meaningless. Genetta can export the network in the

Figure 23: Genetta provides a user with a set of potential enhancements to a design. If the user enables the integration of an enhancement, the confidence values are updated within the WKG.

SBOL format for usage within different tools when all enhancements have been made to a design. Exporting is achieved by making all changes applied to the network to the design, which is stored locally within the tool.

3 Methods

3.1 Fuzzy string matching

Despite the attempt to formalise all data within a dataset into a structured and standardised knowledge graph, this is infeasible due to the inability to formalise written text consistently and the large domain of potential names for entities. The challenge of formalising written ad hoc information into a formal structure is a complex one that has yet to be solved[136] and is outside this work's scope. However, several times during this chapter, the requirement to match names and small sentences which may not match directly is required when querying the WKG. Text indexes are data

structures used in database systems to accelerate the retrieval of text-based data. These indexes enable efficient searching, matching, and retrieval of textual content within large collections of documents or datasets. Fuzzy string matching, within the context of text indexes, refers to finding approximate or similar matches to a given string or query in a text-based index. Unlike exact string matching, which seeks exact matches, fuzzy string matching considers variations, errors, and similarities in the compared strings. The Levenshtein distance algorithm, crucial in fuzzy string matching, is a mathematical formula to quantify the difference between two strings. It calculates this difference by determining the minimum number of single-character edits required to change one string into another. These edits include insertions, deletions, and substitutions of characters. For example, to transform the string "apple" into "aple," one deletion is required (removing the second 'p'), giving a Levenshtein distance of 1. Similarly, changing "test" to "tent" involves a single substitution (replacing 's' with 'n'), also resulting in a Levenshtein distance of 1. The algorithm systematically counts these operations to give a clear numerical value representing the distance, or difference, between any two given strings.

3.2 Traversing modules

During both use cases, modules returned results or enhanced designs. However, within the WKG, a module is captured via a single node and "hasInteraction" edges. This structure was decided because all other information, such as inputs, outputs and constituent parts, can be derived from this. Therefore, a traversal must take place to identify the component interactions and physical entities. Figure 24 displays how the subgraph of a module can be derived:

- **Pink**: The module is found to be given input (the LacI node) via a reverse traversal via the "hasInteraction" edge.
- **Green**: A traversal to all interactions via the "hasInteraction" edges is performed.
- **Red**: Another traversal is performed from each interaction using the interaction edges (dependent on the interaction node type) to all the participating physical entities.

The outcome is a subgraph containing all interactions and physical entities.

3.3 Projecting positional information

During the expansion process, context-dependent interactions were added to the design to unify the interaction network within the design. However, to derive these interactions, a positional network was projected. The result is a simple network consisting of nodes (genetic parts) and edges (the relative positions of the parts to one another). This positional information is also described when the design is first converted into a network. Figure 25 displays a subgraph of the AND gate used during the enhancement section and how the relative position is encoded within the topology. The "Position" nodes always contain two edges: the position of a specific entity and the position of the next "Position" on the sequence. When encoded in this way, the contextual interactions can be derived.

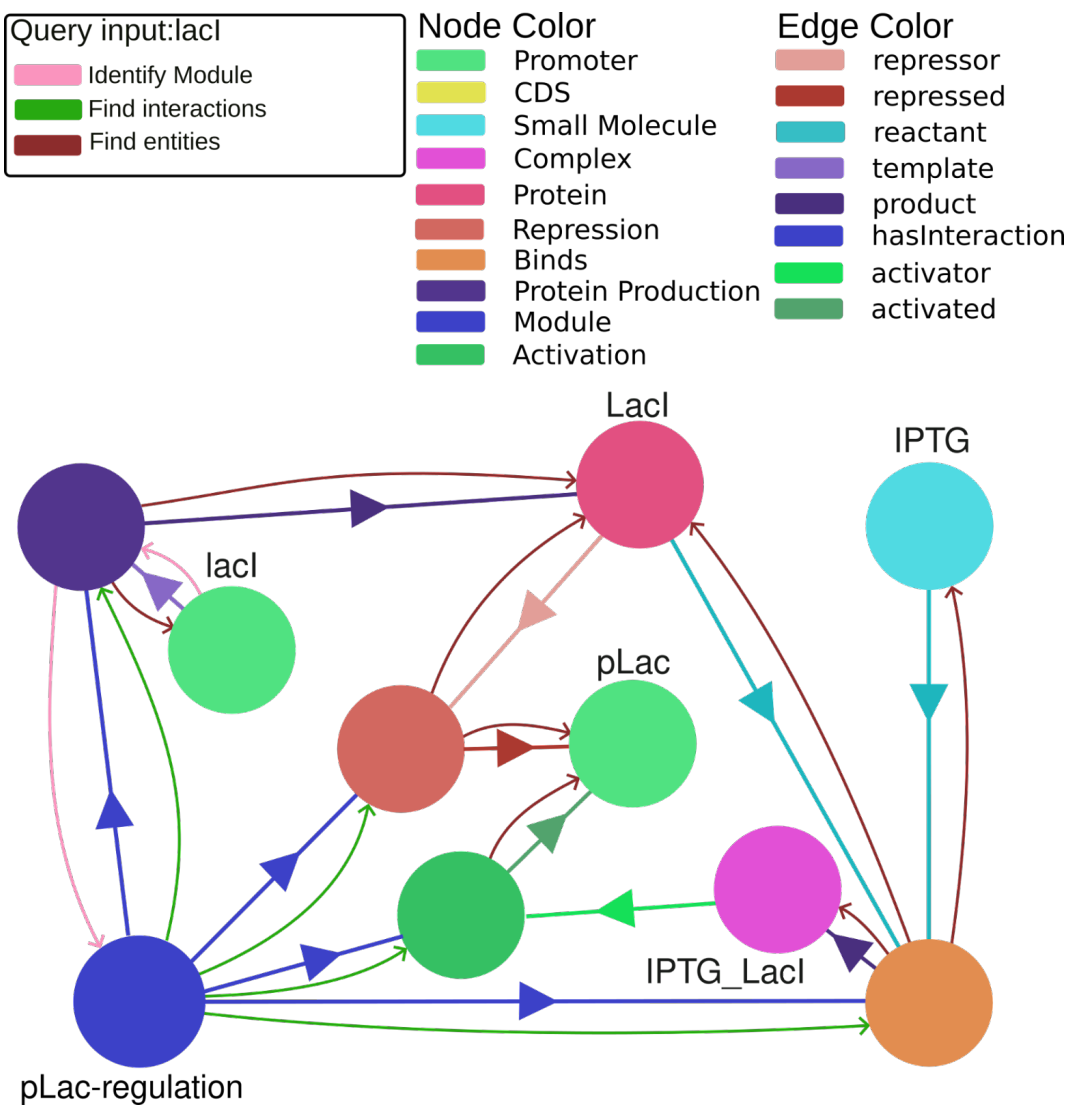

Figure 24: Example of how module subgraphs are derived from the module node. Pink: Given the input (LacI), walks the network to find modules the entities that are implicitly contained. Green: walks the network from the module to find all interactions that make up the module. Red: finds all entities which participate in the interaction.

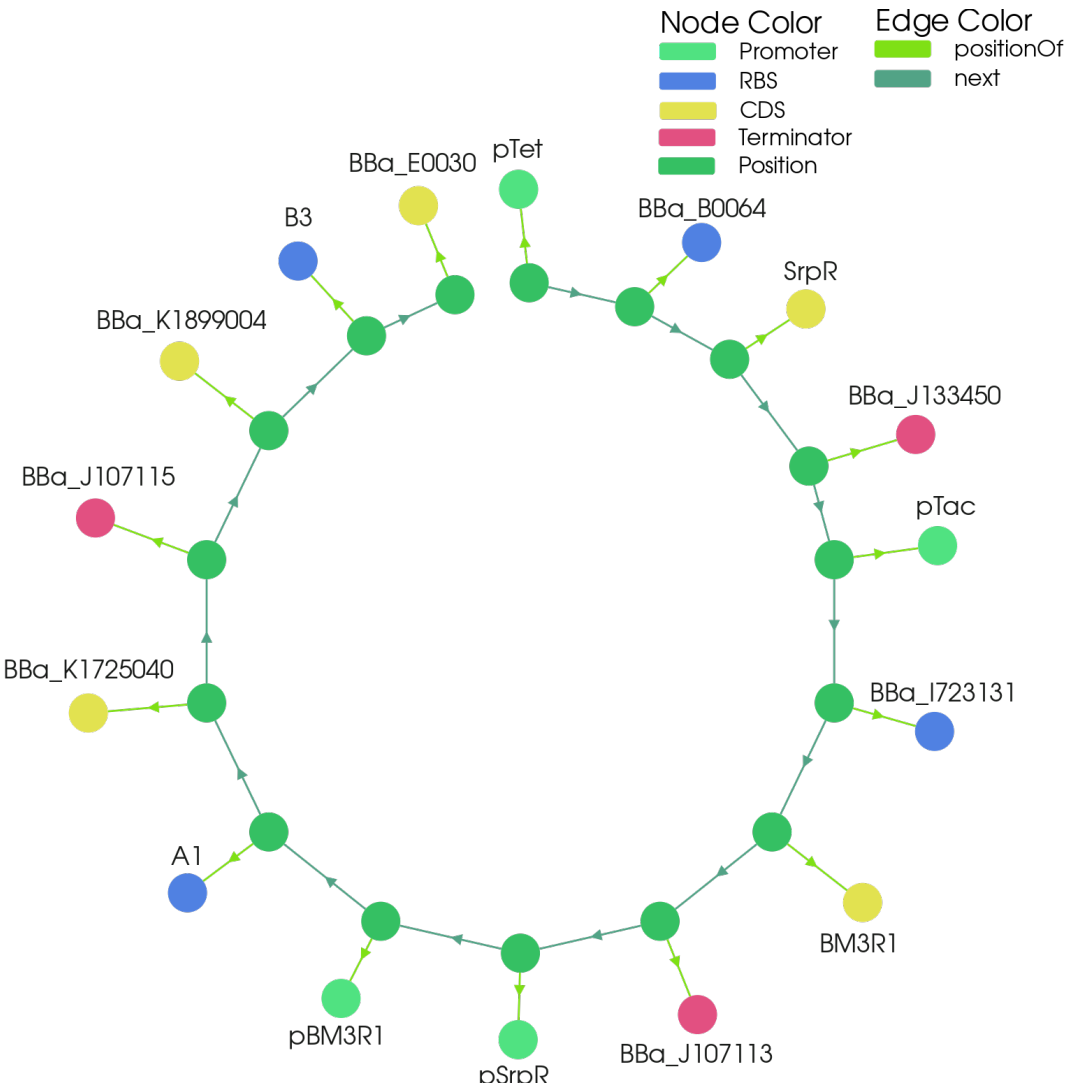

Figure 25: Example of how module subgraphs are derived from the module node. Pink: Given the input (LacI), walks the network to find modules the entities that are implicitly contained. Green: walks the network from the module to find all interactions that make up the module. Red: finds all entities which participate in the interaction.

4 Discussion

During this work, two use cases explore how the WKG established within Chapter 4 can be used to reduce the issues with accessing and using data from uncertain data sources without time and resource-heavy manual analysis. This section will discuss a review of the two use cases and the performance of the WKG in these two applications, along with some opportunities for future work.

4.1 The weighted knowledge graph as an interface to knowledge

During the introduction, a review of existing databases and their systems to provide access to information was reviewed. The outcome was databases of mixed quality and approaches. For example, Reactome[174] provides excellent access to information by providing features to handle query inputs and present results from a user-centric perspective, implements a structured network approach which can explore the context around the search query and ranks the results based on several factors to reduce manual filtering. However, some other databases act as thin barriers between the user and the data, often requiring the user to interpret the data and its quality. From this, it was established that database-derived information, though valuable, demands critical assessment due to varied reliability, necessitating users to rigorously validate, cross-reference, and ensure data quality before integration, a process consuming time and resources whether approached manually or computationally.

The primary advantage of the approach described in this section is that the search method can be specified depending on what aspect of the dataset is being interrogated. This approach contrasts with many previously reviewed systems in which a user may be able to specify the datatype, but this will not change the search method. For example, the user can search for specific datatypes such as Interactions with some of the databases reviewed within the introduction. However, the method to search will be the same as any other data type. The method used to search within the WKG depends on the type of information that is being searched. For example, searching for modules will yield a subgraph of that module rather than solely the module node, whereas searching for derivatives of parts will return a list of nodes because the method employed is different.

This query process also has some other key advantages. Firstly, the quality of results can be quantified, enabling them to be ranked relative to one another. The ability to promote certain records above others can reduce the requirement of manual filtering, which is very common when using lower-quality systems. Another benefit is introducing context to the results due to the structured network approach. A network approach enables the results to be presented by displaying the individual result, its neighbours, and the relationships between them. For example, a user may be interested in seeing how a node denoting a physical entity interacts with other physical entities or identifying other commonly used parts. With other systems, this information may be written in metadata or, if encoded structurally, deriving it from mining the context. In a network approach, the context is encoded within the topology and can be traversed. The final main benefit of this querying is that feedback can be presented, thereby updating the WKG where applicable. Feedback is critical for the success of a system such as the WKG because, without it, the weights and features encoded when the network was built will remain static, removing one of the main reasons to implement a network approach: their dynamic nature. As discussed, the ability to up-

date the network over time with the expertise of practitioners will create a more trustworthy and robust system, reducing wasted time and redundant effort.

4.2 Automating processes using the weighted knowledge graph

Also, during the introduction, a proposal highlighting the functional side of design was established. This functional approach was an extension of a discussion beginning in Chapter 4. It proposed how synthetic biology emphasises engineering intricate logic in living systems, integrating functional modules over sequence-focused approaches to simplify complexity yet acknowledging the scarcity of encoded functional information within design datasets, posing challenges in capturing and leveraging functional details in genetic designs. Extending this, a review of the number of information standards like SBOL can capture within related databases (chapter 4) and in existing designs was extremely low. Two main reasons for this were highlighted: most SBOL-encoded designs and datasets are direct translations of information encoded within older formats such as Genbank, which cannot capture this information, and there is a lack of tooling to introduce this information retroactively.

From this, the second use case displayed how the recently established WKG can enhance existing designs automatically, removing users' requirement to define this information manually. The use case covered several "enhancers", which were methods to introduce new information into the design mainly by using the information encoded within the WKG. The main advantage of this enhancement system combined with the WKG is that once the design graph is canonical relative to the WKG, enhancements which extract information from it are a trivial integration because the nodes and edges in both graphs are comparable. Therefore, the focus can be on the data and creating increasingly valuable enhancers as opposed to requiring complex and potentially stochastic integration techniques.

This enhancement process also has some other key advantages. Firstly, with the confidence metric encoded across the WKG, the quality of individual enhancements to a design can be quantified. For example, a rule could specify that only enhancements over a specific confidence value can be accepted into the design. Another benefit is that each enhancement is atomic, deterministic, observable and isolated. Furthermore, the enhancement is transparent and can be quantified regarding its impact on the network. This approach contrasts with a "black-box" approach where the system's inner workings are unknown, such as some machine learning approaches such as deep neural networks[188] and random forests[189]. The WKG's features combined provide a much higher control over the system and can benefit from human manipulation of the information. Genetta implements each of the "enhancers" discussed within this section, which are sequentially applied to the design. Furthermore, a feature of Genetta is the ability to define and apply new enhancers easily because each method is applied in isolation.

4.3 Future work

Generating designs from abstract specifications

One use case explored was taking existing designs and introducing some commonly missing information. Another similar use case could be applied using features of the WKG, which is to automatically generate new designs given a set of abstract requirements from a user. These re-

quirements could be from an abstract perspective, such as defining inputs and outputs or some conceptual procedure that must occur within the system to specific implementation details, such as specific genetic parts that must be used. The network structure would enable a pathfinding approach in which traversals are performed within the WKG, and each resultant path constitutes a design. Within these traversals, they must fulfil the requirements, such as containing specific genetic parts, or the start and end nodes must indicate specific inputs and output. Furthermore, the quality of a potential design could be based on the aggregated confidence score to provide a potential accuracy metric. As discussed, the WKG consists of design-agnostic information, and therefore, these designs will not be stored persistently within it but instead generated at a time of need.

Introducing different datatypes

Fundamentally, the weighted knowledge graph and the enhancement application are not coupled specifically to functional information. While a focus here, functional information is only one of the many different datatypes rarely encoded within designs. The knowledge graph could be adapted to different datatypes as discussed within Chapter 4 - future work. Therefore, different design aspects could be enhanced. For example, a design could be enhanced from a structural hierarchy approach. Structural entities (organisation of biological components based on physical levels of complexity) could be contained within the WKG based on usage within previously encountered designs. The hierarchy could be added to the design based on matches between itself and the WKG.

WKG as a unifier for DBTL

Within synthetic biology, the Design, Build, Test, and Learn cycle is an effort to design, implement, and perform workflows in which future iterations learn from previous executions. However, a persistent challenge is how the distantly related information from each stage is captured and integrated. A true solution would require connections between design, protocol, experimental and performance information. The WKG could be used as a unifier (centralised source) of all of this information and to make connections between it. As discussed, networks are inherently able to connect information provided that a mapping exists between them, but the WKG could use some of its designed features to enhance these capabilities. For example, modularising subgraphs within the DBTL could provide different perspectives of a DBTL workflow where the information could be relayed at different levels of detail. One level of detail may capture iterations as an individual node, which could be used to analyse the workflow as changes to designs after the conclusion of each cycle or another to represent the mapping of information, providing insights into the flow of information between stages of a cycle. Even within functional information, the scope could be expanded. For example, the modules introduced into the design were relatively low-level (abstract regulatory systems, for example). However, as the process to identify modules during WKG expansion discussed in Chapter 4 future work was implemented, these can be further used to enhance designs from a hierarchical perspective.

Parsing user-ambiguous information

A significant issue was identified while establishing the WKG and the superior querying technique sections. This problem is the difficulty of handling written ambiguous information, i.e., being able to transfer written information into a format structure. Specifically, this section established it as an issue within the query approach where a user inputs written information to be processed before querying the system. With the current simple close string matching approach established, two issues can occur. Either the user inputs an input that is just dissimilar enough that the matching algorithm does not pick up on a match, or the input is similar to many entities and matches with many spurious entities within the WKG. Both of these are caused by the approach's inability to account for context. Therefore, this challenge must be overcome for a query system comparable to established general-purpose search engines. Generative language models[190], which can structure written text by understanding the contextual relationships between words, sentences, and paragraphs, could be a significant breakthrough to this ongoing challenge within standardisation efforts in general.

4.4 Conclusion

Chapter 4 established a weighted knowledge graph (WKG) with integration and expansion techniques to keep a structured and rich dataset with meta characteristics and weight to provide more significant insights into the data. This section identified several issues with data access within synthetic biology and, more specifically, open-source databases with partial or no standardisation. These issues include poor-quality information, a narrow domain of knowledge, substandard query systems and lack of context. Therefore, in this section, two examples demonstrate its utility from both human and computational perspectives, aiming to address some of these issues.

The first use case leveraged the WKG integrated with enriched metadata to create superior querying methodologies. This approach encompassed several essential methods, including incorporating confidence metrics to promote higher-quality information, harnessing provenance data to identify analogous or identical entities, accommodating abstract query requirements, and continually integrating user feedback mechanisms to update datasets and refine result accuracy. Users can benefit substantially from the WKG's capabilities through this multifaceted approach. This initiative provides great potential for increasing search accuracy and speed within synthetic biology databases. Leveraging enriched metadata and a robust knowledge graph framework provides more sophisticated querying techniques. This use case exemplifies the transformative potential of integrating advanced knowledge graphs and metadata enrichment strategies in synthetic biology databases and across various domains. One limitation of this section is the challenge of comprehending natural language, i.e. the query inputs. Automating the formalisation of natural language presents several challenges. Firstly, natural language is inherently ambiguous; the same text can convey different meanings in different contexts. To accurately resolve this ambiguity, a system must possess an in-depth understanding of language and the capability to contextualise and infer meaning. Additionally, the significance of words or phrases often shifts depending on their contextual usage, making it difficult to represent this information accurately in a formalised system. In this research, addressing these complexities was not the primary focus. Instead, where written information was required, it was processed using established, simpler methods such as fuzzy string matching, bypassing the more intricate aspects of natural language formalisation.

Integrating additional information beyond sequence data is pivotal in advancing design methodologies. Beyond the raw sequence, functional data serves as a crucial layer, offering a higher level of abstraction that streamlines the construction of designs. Nevertheless, a prominent absence of this critical information prevails within numerous databases and individual design frameworks. This absence is primarily rooted in standardised designs from formats incapable of accommodating such specific details. Therefore, The second use case utilised the WKG, showcasing computational approaches to enhance existing designs. This approach aimed to supplement and incorporate the absent information, which conventionally remains unencoded within design datasets. Central to this approach is the concept of canonicity within the WKG, facilitating the normalisation of entities within the design network. This normalisation enables seamless data transfer and exchange, introducing functional information into pre-existing designs. As previously discussed, the pivotal focus lies in rectifying the persistent absence of such crucial data, yielding various benefits derived from explicit and comprehensive descriptions[27]. The main result is introducing a system that can remove significant entry barriers to implementing standards. These barriers typically encompass the time-consuming manual definition of intricate information and the inherent complexity of the underlying data structures necessary for defining and incorporating this critical data. This system addresses the current limitations and provides an approach toward more streamlined and enriched design methodologies, positioning them for greater adaptability and efficacy within diverse applications. The primary limitation of the current enhancement system lies in its limited scope, particularly in the types and complexity of the modules it can introduce. For instance, the system can add basic modules, but there is potential for incorporating more substantial and intricate modules. Future developments should broaden the system's scope to encompass the integration of these larger and more complex modules. Such an expansion would facilitate the creation of more sophisticated genetic designs. This enhancement would entail diversifying the range of modules in the knowledge graph and improving the algorithms responsible for matching and integrating these modules into existing designs. Enhancing the system this way would likely lead to a more automated and streamlined design process capable of handling more complex designs.

Chapter 6: Dynamic networks to present multiple design aspects and scale levels of complexity.

Publications arising from this thesis

- Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "A Network Approach to Genetic Circuit Designs". *ACS Synthetic Biology* 11:9, 2022. PMID: 36044984, pp. 3058–3066. DOI: 10.1021/acssynbio.2c00255. eprint: https://doi.org/10.1021/acssynbio.2c00255. URL: https://doi.org/10.1021/acssynbio.2c00255

Software arising from this chapter

- Genetta

1 Introduction

Designs are often the first step in each iteration of a synthetic biology project, and an implementation can be challenging without a well-conceived design and a solid understanding of the design and its constraints. Mathematical and computational tools[191], automation methods[83, 192], knowledge-based systems[136, 187] and repositories[133] can assist circuit design to minimise the iterations within the design-build-test-learn research cycle. These processes generate a diverse set of information beyond DNA sequences, such as functional modules, abstraction hierarchies, implementation instructions, dynamical predictions and validation strategies. This multivariate nature of the data, which encodes much more than sequence data, cannot be represented entirely as a sequence-level representation. Furthermore, complex and large genetic designs are infeasible to conceptualise at the sequence level simply due to the inability to comprehend a complex system at a high level of detail. Therefore, there is a need for a more generic approach that can visualise arbitrary datatypes with the ability to expand and retract detail.

1.1 Existing specification methods

Various tools and methods for representing and visualising genetic designs are available in diverse forms. This discussion will provide a brief overview of several existing approaches and tools, highlighting their advantages, shortcomings, and missing features. In this review, we will use the existing design of an AND gate (comprising two inverters connected to a NOR gate) created by Nielsen and colleagues[19] to illustrate the visualisation techniques.

Design representation with plasmid maps

The most common visualisation of genetic-centric data is the plasmid map, where the sequence is circularly represented relative to its position on the plasmid, and genetic annotations are displayed sequentially. Figure 1 illustrates how the abstract AND gate may appear within a plasmid map, with parts or annotations depicted as coloured regions. Benchling[111] is a widely used tool for sequence-focused editing, offering both linear and circular plasmid representations. Many databases and services implement a plasmid map representation, including AddGene[193], Benchling[111], NCBI[194], and Seva vector[195]. Most existing tools and databases offer several formats to export data, but exporting Genbank or European Nucleotide Archive (ENA) files is overwhelmingly the most popular.

Advantages This representation implicitly displays all the information required to comprehend the design at a genetic level because all features can be derived from the sequence data. It is also the most comfortable method for most users, as it is the most commonly expressed representation of a design.

Disadvantages Sequence visualisation cannot encode interactions and becomes impractical when designs become larger and more complex. For example, in a system with numerous interactions between thousands of entities, deciphering the function would be almost impossible. Additionally, the fundamental principle of modularity in synthetic biology cannot be adequately represented with this approach. Therefore, while visualising the plasmid map helps make minor sequence-level modifications, it becomes incompatible with the dynamic and core principles of synthetic biology. Furthermore, representing design features at other layers of molecular biology, such as transcription, translation, protein modification, protein interactions, pathways, metabolism, multi-cellular systems, phenotype, and physiology, is also unfeasible using this method.

Design representation using standardised part glyphs

Another common visualisation method is abstracting sequence data into glyphs representing genetic parts positioned linearly based on the relative position of the part, adding non-genetic entities such as proteins, regulatory chemicals, and interactions. Figure 2 displays this concept where each genetic part is represented as an icon on a linear sequence and may include abstract interactions illustrated by lines between glyphs. Many tools implement this specification, such as IGEM parts brown2007igem, SBOL Canvas [110] and VisBol [109]. Many glyph representations are backed by the SBOL visual standard[132], which refers to graphical symbols and glyphs used to represent genetic and biological parts and their interactions. SBOL Visual provides a standardised way to visually represent DNA components, such as genes, promoters, terminators, and other genetic elements, using easily understandable symbols that can map directly to the SBOL data standard.

Advantages The mapping of genetic parts to glyphs is a well-known practice within the synthetic biology community. Consequently, like other established methods, it demands relatively little adaptation from the broader community. Also, the glyph approach abstracts all SBOL classes and necessitates no prior knowledge of the SBOL data model, software, or language. These simplified representations can be derived from a standard data model. This mapping to a standard allows for the creation of formal representations from a more informal mechanism, eliminating

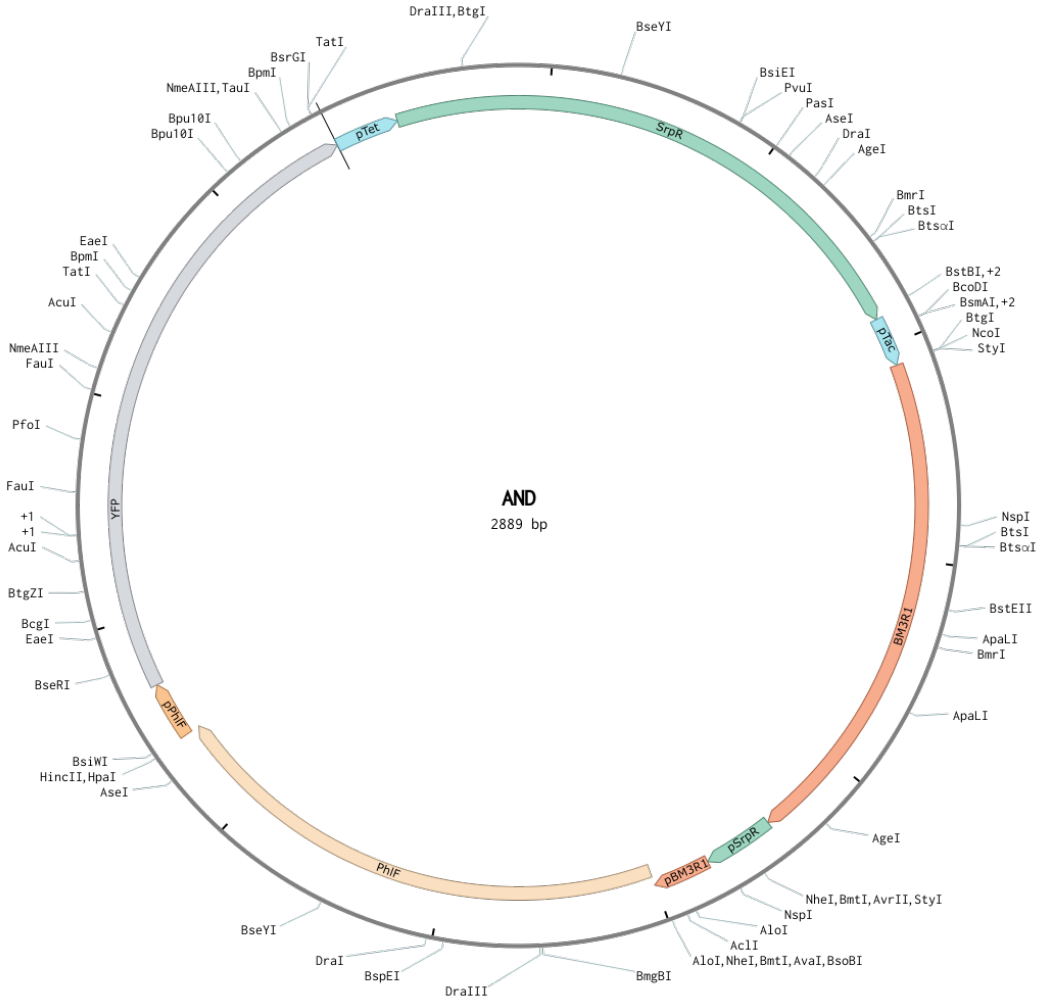

Figure 1: An example of a Plasmid Map which represents the AND[19] gate construct. The coloured lines represent sequence annotations (Promoter or CDS, for example) with restriction sites displayed on the outer edge. The visualisation was created using Benchling.

unnecessary complexity while retaining the advantages of the standard data. Moreover, the glyphs approach can scale detail down by abstracting proteins and regulatory chemicals participating in an interaction.

Disadvantages Understanding large and complex designs can remain challenging with this visualisation, mainly when they encode numerous interactions, making it difficult to discern the intended function. Also, this visual representation is tied to specific data types, including sequence, parts, proteins, chemicals, complexes, and interactions, making it incapable of representing diverse data types. For instance, hierarchical information, experimental data, or metadata cannot be effectively visualised. Furthermore, static representations have limitations in scaling the level of detail. For instance, glyphs are inadequate for representing increasingly abstract systems and their interactions.

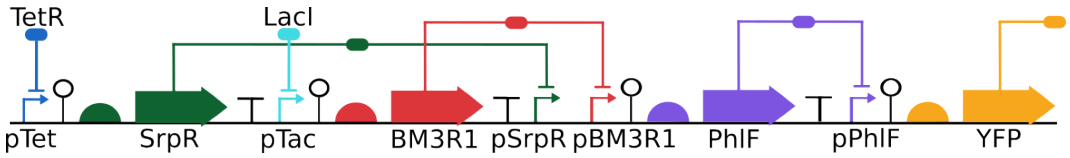

Figure 2: Visualisation where genetic parts, proteins and chemicals are represented as glyphs. Interactions between entities, whether physical or conceptual, are represented via lines. This AND gate [19] design consists of two inverters flanking a NOR gate resulting in an AND gate design.

Design representation using logic gate symbols

The logic gate visualisation represents a design most abstractly. It visualises modules of a design as boolean logic gates used within electronics design. Figure 3 displays three abstract logic gates, which constitute an implementation of an AND gate. The figure shows the same circuit as figure 2 but uses a different mechanism at a different level of abstraction.

Advantages This approach abstracts a significant portion of the implementation details, concentrating primarily on conveying intent. This simplification enables the comprehension of very complex designs. The analogy employed in synthetic biology, likening genetic circuits to functional modules, directly parallels their basis in electronic logic gates and circuits. **Disadvantages** While effectively conveying the intended design, this representation lacks specificity regarding implementation details. For example, Figure 3 does not provide insights into how the inverters or NOR gates are implemented. Furthermore, it primarily visualises a narrow domain of design information, limiting its ability to represent interactions that resemble analogue systems and cannot be quantified within the confines of boolean logic. For example, complex processes like internal degradation or those using evolutionary mechanisms are challenging to visualise within this framework. The three visualisation methods discussed, while each unique with specific advantages and disadvantages, all share a fundamental flaw, the lack of dynamism and, therefore, the inability to be tailored to focus on the interests of a specific individual. Furthermore, no individual tool can present a scaling level of detail as each offers a narrow range or contains too much detail, which is problematic when the scale of a genetic design's size and complexity can vary considerably.

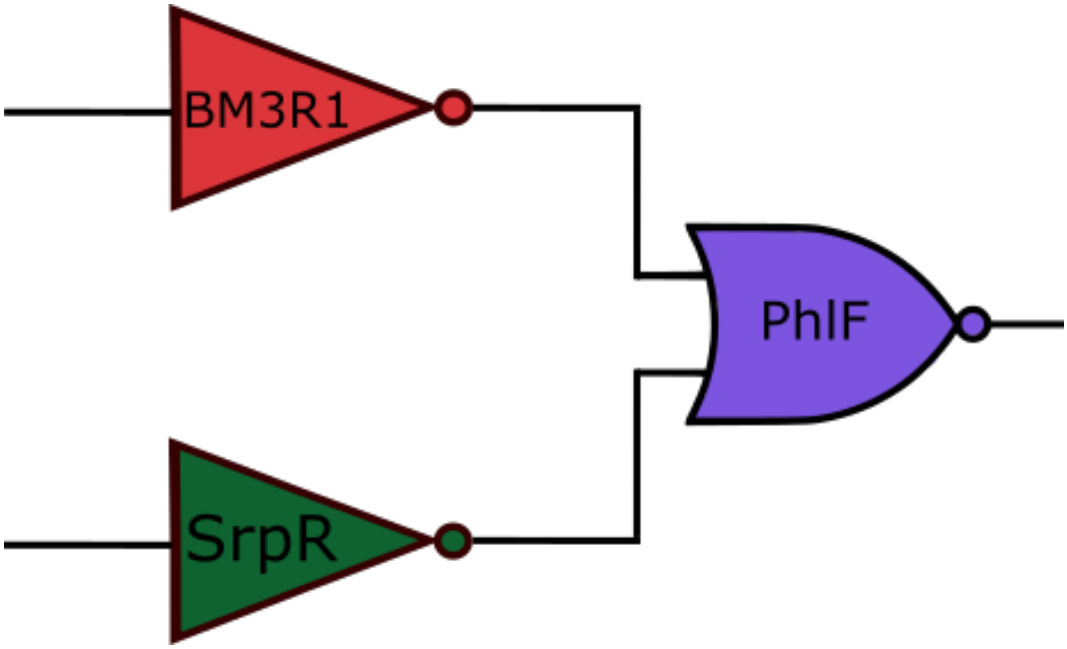

Figure 3: Visualisation mirroring the electronic logic gate symbols. A more abstract representation based on boolean interactions between entities. This representation consists of two inverters flanking a NOR gate resulting in an AND gate design [19].

Challenges and opportunities in representing multi-dimensional genetic design data

As genetic circuits become increasingly sophisticated, the size and complexity of data about their designs increase. The data captured goes beyond genetic sequences alone; information about circuit modularity and functional details improves comprehension, performance analysis, and design automation techniques. For example, within figure 4, a single design document can hold multiple datatypes. However, new data types expose new challenges around the accessibility, visualisation and usability of design data (and metadata). As discussed, understanding multi-dimensional design data with a single representation is challenging.

Data standards such as SBOL contain the mechanisms for users to explicitly capture multiple datatypes, which allows the design to be presented in various perspectives depending on an individual's requirements. For example, a design containing sequence, functional, structural, meta and experimental data can be represented from the perspective of each of these data types. Therefore, as potential datatypes encoded within a genetic design expand, the opportunity to target the representation to the user's requirements becomes feasible. However, this benefit of standard data has not been exploited; instead, each tool focuses on a specific datatype.

Specialised Representation

Design documents may encode multiple instances of many types of information. However, each can be expressed at different levels of abstraction or further specialised. For example, in figure 4,

the datatypes discussed in the previous section can be expanded to present several more specialised representations. These representations could be specialised subsets explicitly defined within the data, such as genetic parts or where new information is inferred from existing where it is not expressly described in the case of abstract interaction networks. Further specialisation expands the issues of current methods to represent and visualise genetic designs because the tools only represent a single data type and cannot scale or specialise in detail. It is unfeasible to create a new suite of mechanisms to represent arbitrary levels of detail that may occur as more data is encoded within designs.

Network Visualisation

Network visualisation presents networks of connected entities where nodes represent data points, and edges are relationships between data points[196]. A significant advantage of a network approach over the discussed visualisation techniques is the ability to represent multiple aspects of the original data and scale abstraction, which is especially attractive when visualising complex systems. Data is often captured in a multi-dimensional format, where many types of information are aggregated, and relationships between data can represent different relationship types. For example, social networks often contain multiple data types, such as friends, locations and interests, as seen in figure 5. When represented as a network, the graph is called **multipartite**. However, when networks encode many data types, or the size of the underlying dataset is large, visualising comprehensible multipartite graphs becomes infeasible. Several reasons make complex multivariant networks challenging to comprehend. For example, the significance of connections between nodes is lost if the intent behind visualisation is not established. Also, as humans, we cannot comprehend the entire dataset simultaneously when the size of the information passes a threshold. Figure 6 displays an extended example of a social network where many interconnected data points are visualised. The network is a randomly generated network to display a larger dataset. The nodes represent people, cities and activities, with edges representing their relationships. The network is the complete dataset visualised in its native structure. Making visual inferences around this dataset, such as potential friend groups within the network, is challenging because other datatypes can obscure this information. Furthermore, even this network is negligible when considering real social networks, which could contain billions of nodes, further displaying the infeasibility of raw network visualisation of complex multivariate data.

However, unlike the representation methods discussed previously, networks can represent arbitrary datatypes. Therefore, the network can be modified and tailored for a more suitable representation. As displayed, new representations can be created from the original dataset, which produces a more comprehensive visualisation by focusing on a single datatype and reducing size. For example, figure 7 displays a version of figure 6 that only shows the relationship between people. The result is a disconnected graph of components. An inference could be made that each component represents a potential friend group within the dataset, which is more hidden within the previous figure as the activities and locations of people mask this.

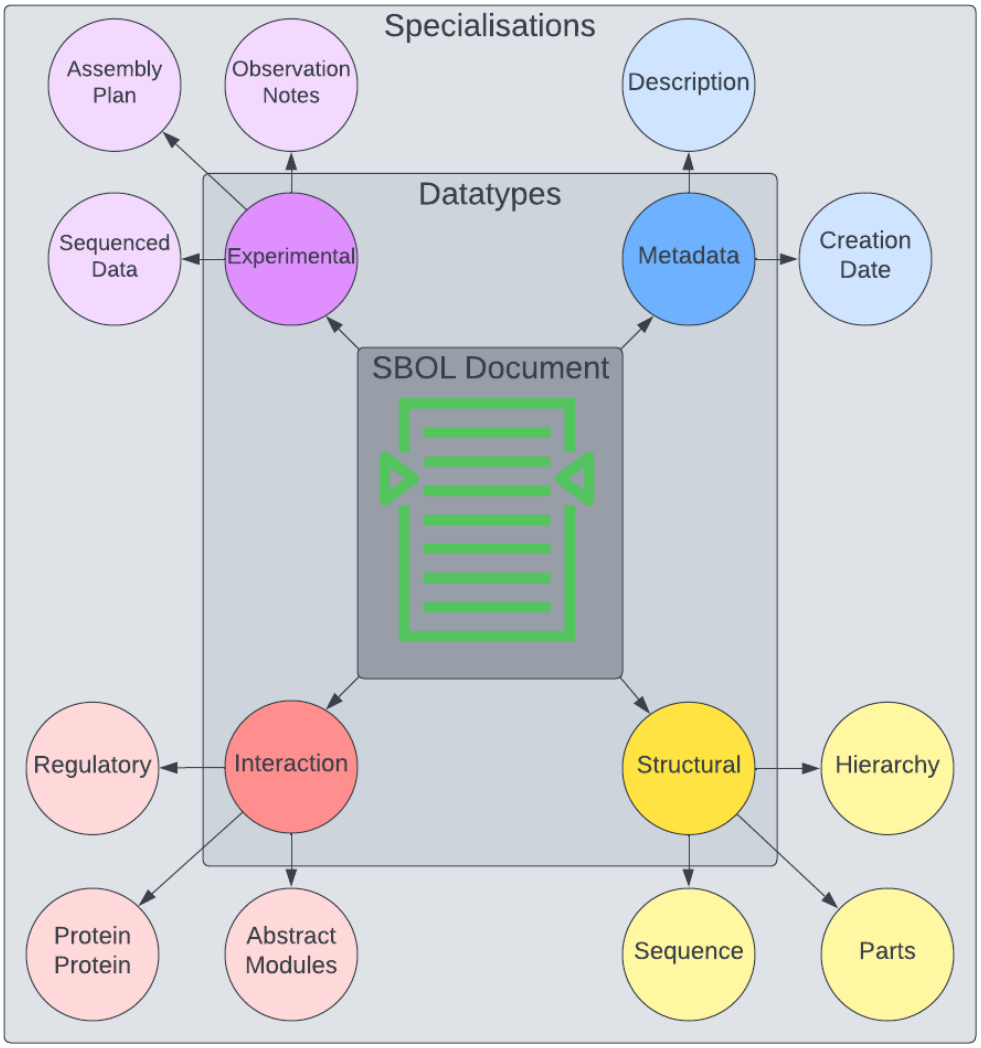

Figure 4: Conceptualisation of a potential SBOL document that explicitly encodes experimental, metadata, structural and interaction data. Furthermore, each data type also contains several more specialised types of data such as sequence, genetic parts or the structural hierarchy in the case of structural data.

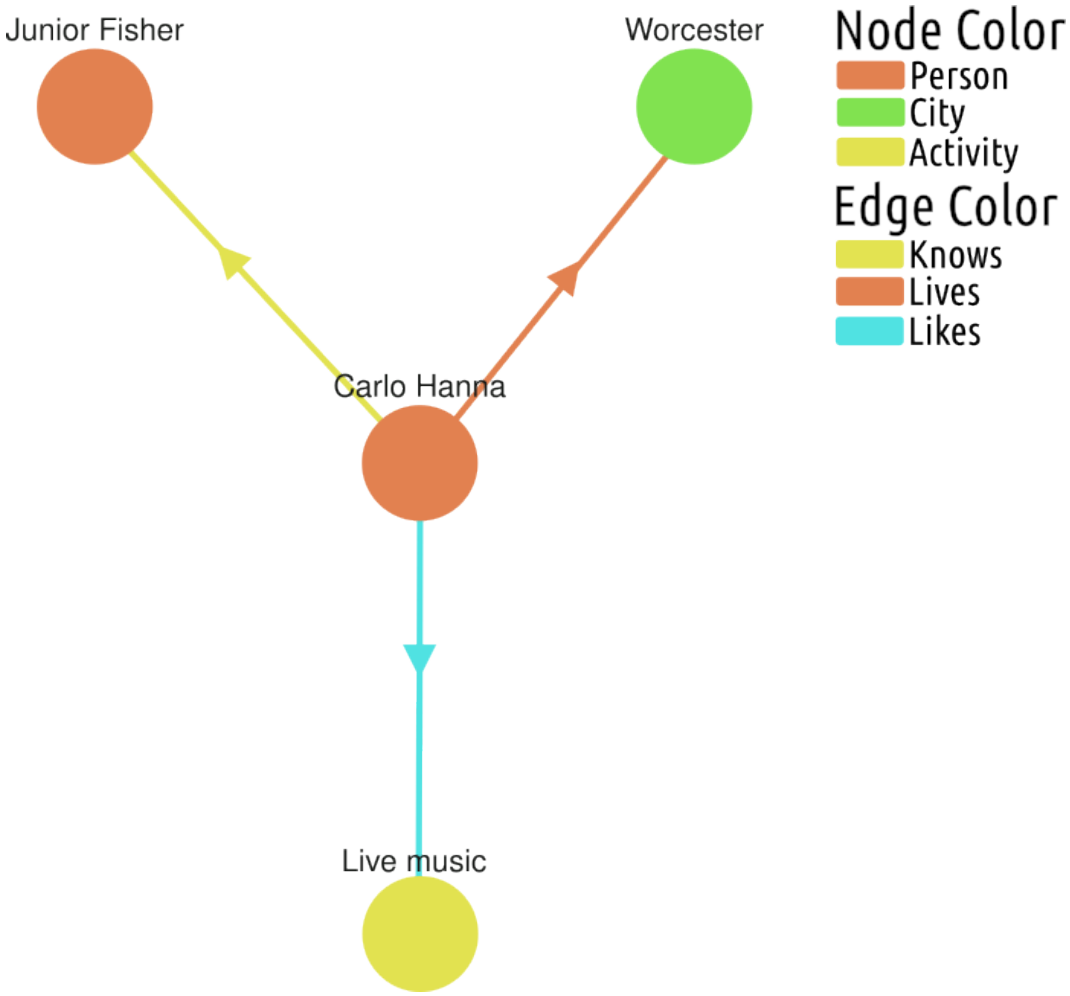

Figure 5: Example trivial social network, a multipartite graph containing multiple data of data (people, places and activities), connected by various relationship types (knows, likes and lives).

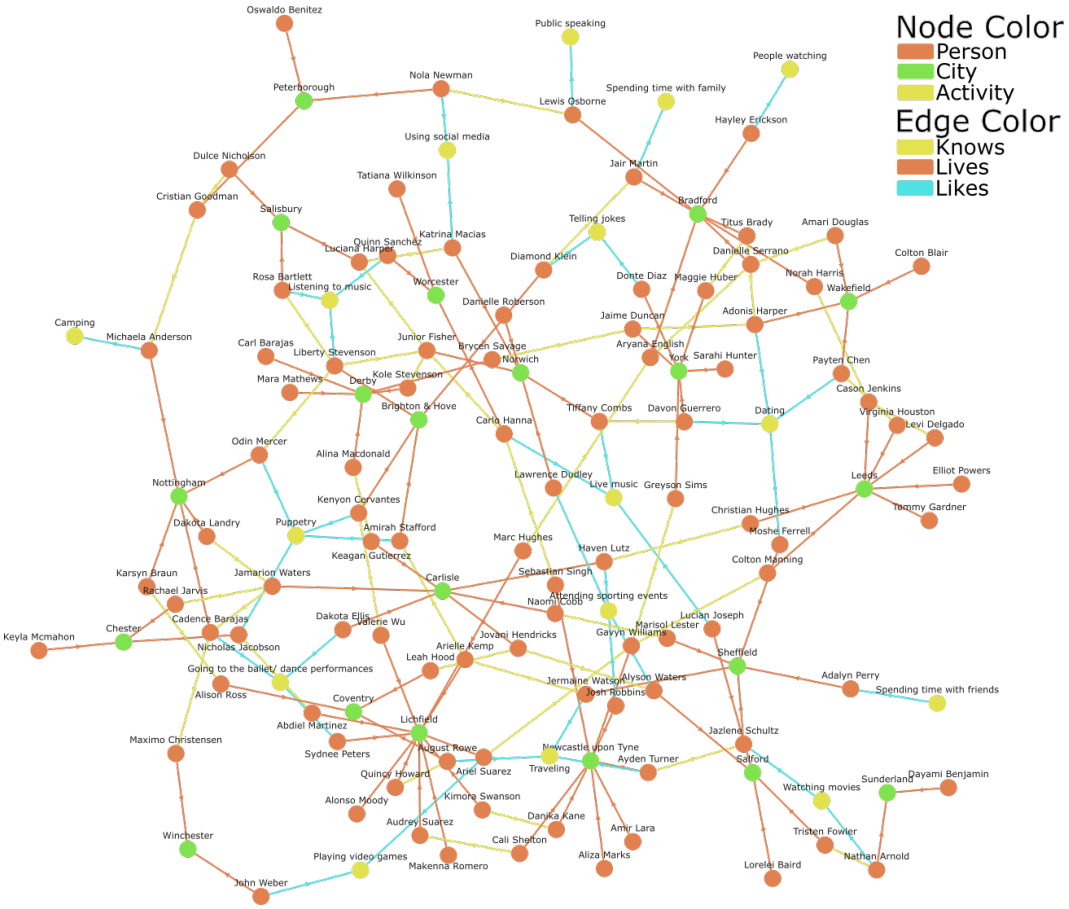

Figure 6: Example of a social network that encodes relationships between people, cities where they live, and their interests. n=136, e=96. The network diagram is a randomly generated visualisation that illustrates a complete dataset.

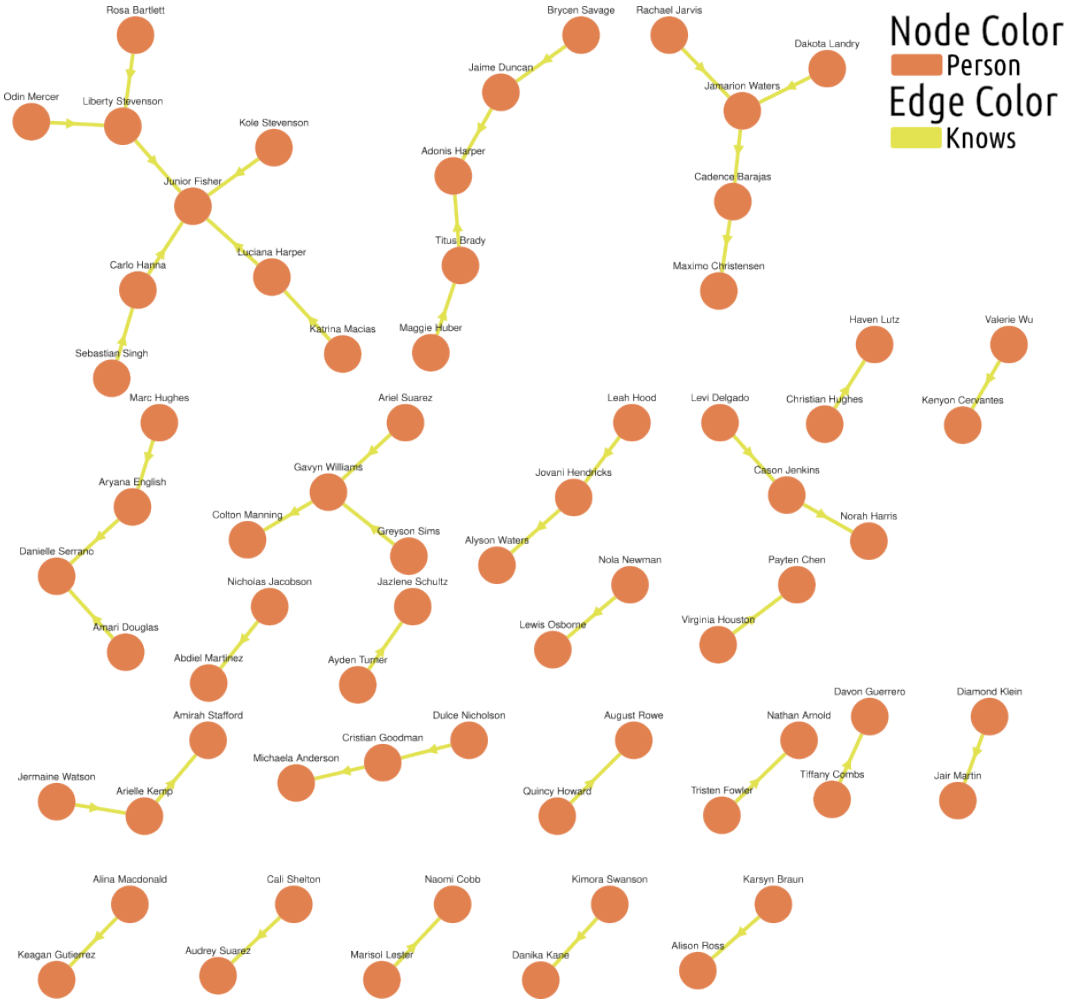

Figure 7: Interaction network encoding relationships between people only. Here, the network is disconnected, and many isolated components exist. n=73, e=45

1.2 Network analysis and representation systems and synthetic biology

Network visualisation is a foundational tool within systems biology[197]. It provides insights into various interaction networks, including gene regulatory networks, protein-protein interaction networks, and metabolic networks. Furthermore, network visualisation enables the exploration of the structure and function of complex biological systems and aids in discovering key regulatory mechanisms, disease markers, and potential therapeutic targets[198]. In this context, biological data is typically portrayed as a network comprising nodes (representing biological entities such as genes, proteins, or metabolites) and edges (representing interactions such as physical interactions, regulatory connections, and metabolic reactions)[199]. This representation allows for understanding how different elements within a biological system communicate and influence one another. Furthermore, systems biology often deals with dynamic data, such as how interactions change over time. Network visualisation accommodates this by enabling the depiction of temporal changes in interactions, providing insights into the dynamic aspects of biological processes[200]. Network visualisation also seamlessly integrates with omics data, such as transcriptomics, proteomics, and metabolomics. This integration enables experimental data to be superimposed onto the network, enriching the visualisation further[201]. In summary, network visualisation is a versatile tool in systems biology, offering insight into the complex interactions that underlie biological systems with the ability to depict dynamic changes and overlay different data types. Because network analysis and visualisation are already established in systems biology, the question may be posed as to why these mechanisms cannot be reused within synthetic biology. While it is true that some overlap will occur, several reasons make the analysis, representation and visualisation of a design different from systems biology. Within systems biology, networks mainly describe interaction names such as gene regulatory, protein-protein interaction, metabolic and co-expression networks. Within synthetic biology, these interaction networks are only one area of potential representation because of the different information captured, such as structural hierarchy, functional modules, and build protocols, which can all be represented as a network and then visualised. Secondly, within systems biology, the networks are designed to convey the actual or predicted systems. In contrast, when representing a design in synthetic biology, the system portrayed is a desired outcome, as it encapsulates the intended function. Consequently, even when interaction-based networks are employed, the representations may diverge, reflecting the distinctive approach of synthetic biology, emphasising presenting information from a developmental perspective, making it readily adaptable for modifications.

1.3 Aims and Objectives

This research explores some aspects of visualising genetic designs within synthetic biology. During the review of existing visualisation techniques, the represented data is often closely coupled with the representation and visualisation mechanisms. For example, the glyph approach cannot be used to describe any other type of data encoded within a genetic design because each glyph and the linear structure are inherent to the representation of the parts. Therefore, while several methods exist that can represent specific data types, users must use various tools to inspect multiple data types. Furthermore, the level of detail is often fixed because it is inherent to the underlying

representation. Here, we explore how networks can be used to represent genetic designs. The focus is tailoring the resultant visualisation to the user's data types and complexity requirements.

Aims

This study investigates the common aspects of genetic designs that users are interested in to identify patterns and preferences within these designs. Additionally, the research explores methodologies for generating network representations directly from existing data sources, focusing on assessing the efficiency and accuracy of these projection methods. A key objective is to evaluate the scalability of detail in network representations. Another aim is to associate discrete values with specific projections to establish the optimal level of detail representation based on varying data values. The final aim of this study is to analyse strategies for reducing complexity in visual representations. This aim will be achieved by mapping complex data to simpler visual features to enhance comprehensibility while preserving essential information.

Objectives

The main objective is establishing procedures for projecting diverse views within a design dataset, highlighting specific, desired data features to ensure a comprehensive dataset representation. Also, to develop techniques that adjust the complexity level of these representations, allowing for a fine-tuned approach that caters to varying degrees of detail as per user-defined requirements. Another key objective is to introduce mechanisms that mitigate the continuous complexity inherent in the dataset, effectively streamlining and simplifying representations to enhance user comprehension. Additionally, the study seeks to develop integrated methods that collectively provide a complete representation of complex genetic designs, ensuring that the portrayal of the data is both holistic and thorough.

2 Results

Here, we present a method to transform circuit designs into networks and showcase its potential to enhance the utility of design data. Since networks are dynamic structures[202], initial graphs can be interactively shaped into sub-networks of relevant information based on requirements such as the hierarchy of biological parts or interactions between entities. A significant advantage of a network approach is the ability to scale abstraction, providing an automatic sliding level of detail that further tailors the visualisation to a given situation. Additionally, several visual changes can be applied, such as colouring or clustering nodes based on types (e.g., genes or promoters), resulting in easier comprehension from a user perspective. This approach allows circuit designs to be coupled to other networks, such as metabolic pathways or implementation protocols captured in graph-like formats. Here, we discuss each step, from representing design data as networks to representing specific design features and visualising a comprehensible network.

Representing designs as networks

The first stage is to take a design file and represent this information as a network. As discussed in previous chapters, a network is composed of nodes, individual points or collections of data and

edges, connections between points of data. Synthetic biology design data can be conceptualised as this, for example, interactions between genetic and non-genetic parts, structural usage, or metadata relating to some entity within a design. The NOR circuit is frequently built since assembling these gates can achieve any logic function. We used the design of the NOR logic gate built by Tamsir and colleagues as an example. This gate outputs 1 (i.e., target gene expressed) if both inputs are 0 and outputs 0 (i.e., target gene not expressed) in any other case. Figure 8 displays the circuit and glyph representations for context. As discussed and addressed in Chapters 4 and 5, the quality and variance of the input data are fundamental to meaningful representation. Expressive visualisation is not feasible without rich data, irrespective of the visualisation method. Here, we used a high-quality design of the NOR logic gate in SBOL format, which captur Figure 9 shows the results

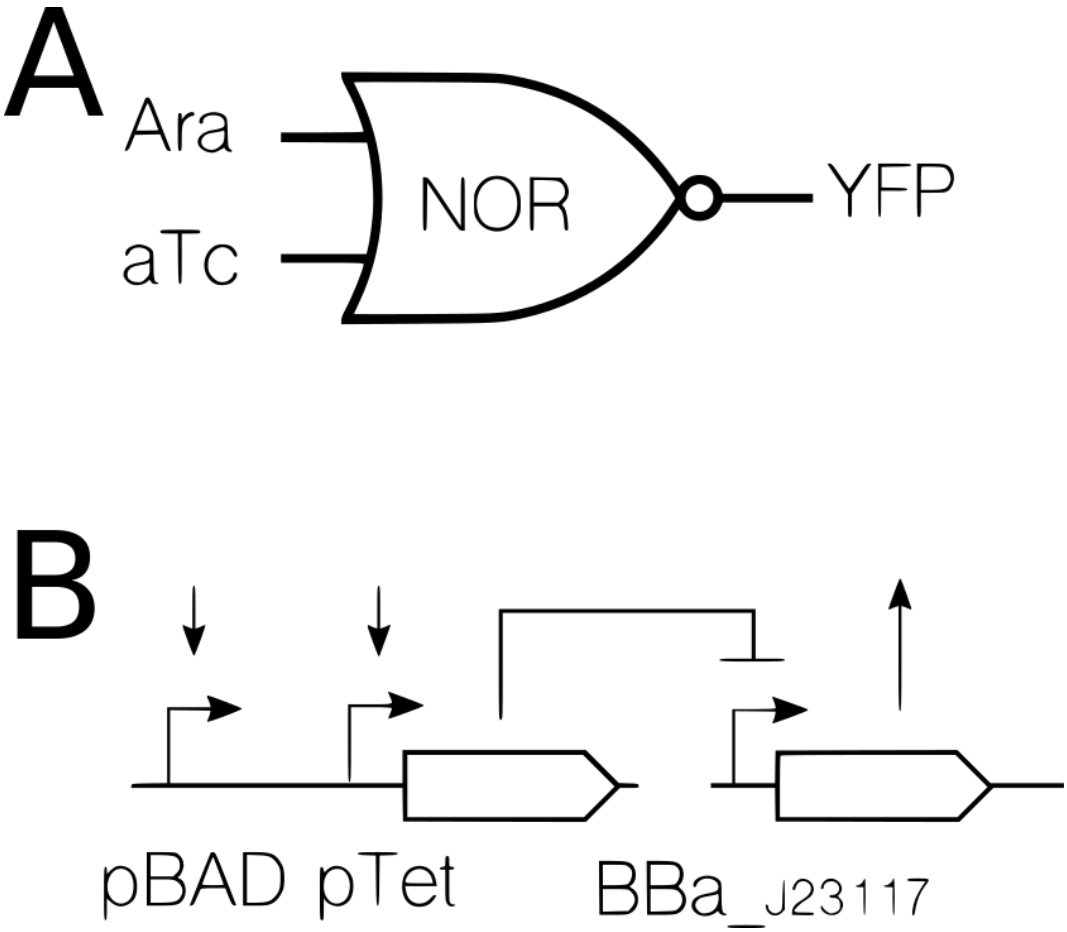

Figure 8: The logic gate and glyph representation of the NOR logic gate. A) The logic gate with inputs (arabinose; aTc) and output (YFP). B) The glyph representation displays two promoters (pBAD and pTet) regulating a CDS, which represses the expression of a second CDS.

of building a network with all data and metadata in the original design file. The complete design graph is a multipartite containing numerous data types, such as parts, interactions, sequences and metadata (e.g. entity role, free text descriptions or data type). Before building this graph, design

data was converted into an intermediate data structure that is the same regardless of input format; the resulting graphs are compatible and not format-dependent (see Methods).

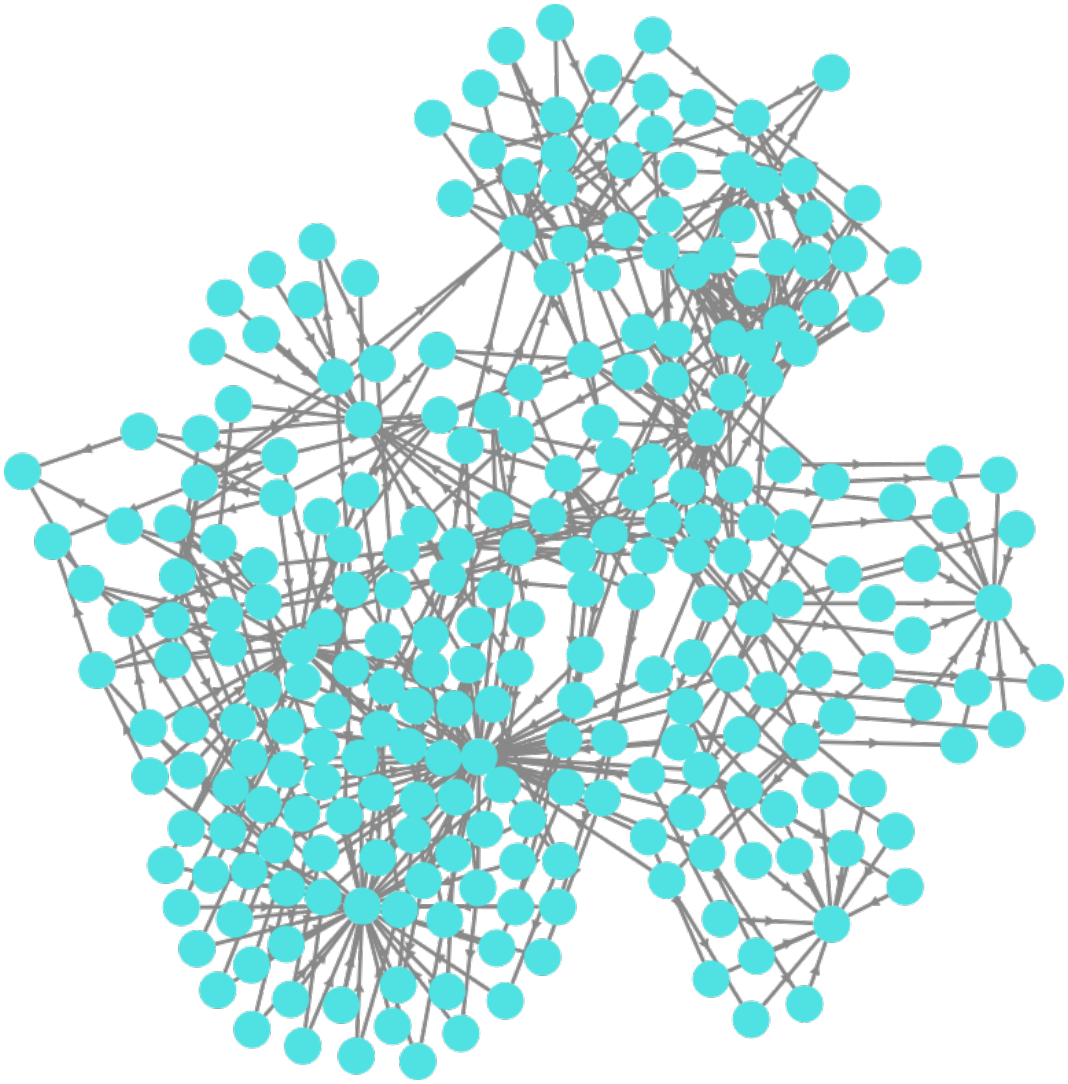

Figure 9: The NOR logic gate when directly represented as a network. The network is unreadable but computationally tractable. Note that the figure omits labels due to the size and cluttered visualisation.

Projecting Networks

As discussed (and represented within Figure 9), complex multipartite graphs, when visualised, are highly challenging to comprehend for several reasons.

- A human cannot simultaneously understand each connection's meaning and, therefore, cannot form a more abstract understanding of the whole system.

- Visualisations are presented within a finite space (usually a computer screen). Past a given threshold of nodes and edges, the real estate will be exhausted, and sections of the information will be hidden.

- The underlying data structure may not be understandable by the user because the model generally aims to capture data. For example, the SBOL data model encodes interactions as nodes, with participation nodes denoting the conceptual instances of physical entities within an interaction. However, humans would more likely conceptualise interactions of edges between two nodes indicating physical entities.

Projections are specific views of the data which focus on a more compact representation, more likely of a commonly understood representation that a user can easily conceptualise. Thus, the data's projections are often visualised to produce meaningful visualisation instead of the entire network. Projections can be achieved by aggregating, removing or inferring new nodes and edges and usually, the final network representation will be monopartite or bipartite. A simple projection produces the network of Figure 10, an example of how to query the initial structure. It is a specific sub-graph that focuses on physical elements only (e.g., DNA and molecular entities) and omits metadata details according to user requirements. By presenting a single perspective, overall cognition is increased. However, projections alone do not ensure a comprehensible visualisation and, in some cases, may obfuscate necessary information. For example, in retaking Figure 10, two key issues are present. Firstly, the roles of nodes (such as genetic roles) are abstracted. Secondly, the nodes are presented randomly, meaning that nodes often overlap, resulting in a confusing visualisation.

Visual Additions

As discussed, visualising a design in network form does not inherently ensure a comprehensible representation. Transformations to the represented data must simplify the graph and tailor the data before visualisation. However, nodes and edges encode many continuous and categorical values, which, in some cases, are inherent to the design and cannot be removed or modified—for example, genetic roles, names or interaction types. Therefore, we will present another set of purely visual modifications which can be applied to reduce perceived complexity further. These visual additions aim to remove clutter, such as textual labels, replace them with a visual medium, or manipulate existing visual features.

Graph information (Color) The nodes and edges which encode information from the design may contain properties, which are attributes of the entity. One choice to represent this information is adding another node with an edge connecting the property to the entity, which is impractical for non-trivial designs as the number of nodes could grow exponentially, resulting in a verbose graph which is less comprehensible. Colour can be used to represent any categorical information (biological or network features)—for example, the type of interaction or the clustering coefficient to display the connectedness of a node. Mapping categorical attributes to colour can increase detail without increasing perceived complexity. However, the number of categories should be no more than 26 due to the finite number of colours a human can differentiate[203].

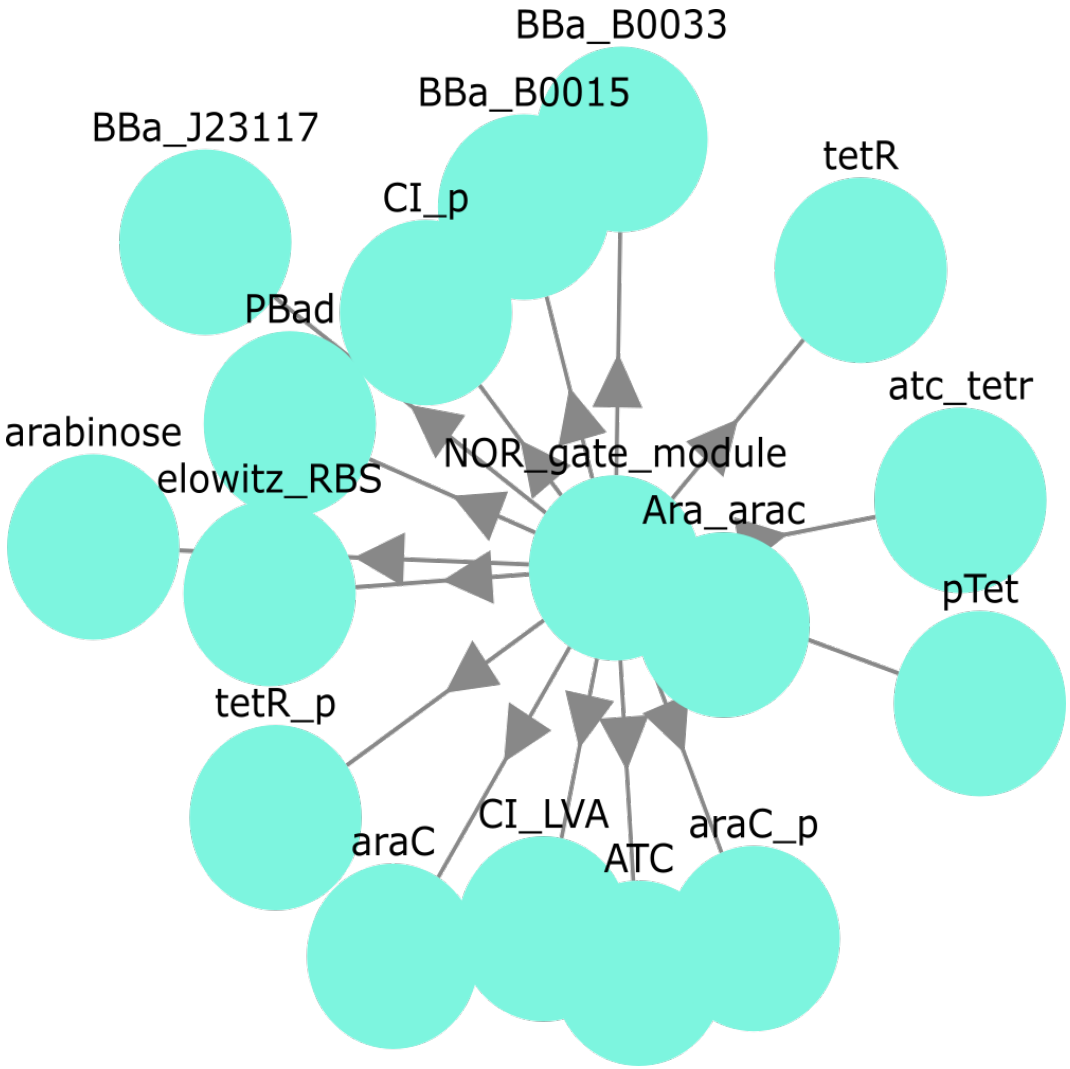

Figure 10: A network is generated from the same design where only the physical elements (i.e. DNA and molecular entities described) are shown. The nodes represent physical entities (DNA, proteins, chemicals and complexes), while the edges represent that the entities are constituent entities of the NOR gate.

Different means can be used for quantitive data, such as size. In Figure 11, the genetic roles are represented as distinct colours to differentiate them.

Spatial layout and positioning (Layout) If many nodes with many connections are visualised, edges will inevitably overlap within a finite canvas. While this factor is unavoidable with data of non-trivial size, the layout of nodes relative to one another, when visualised, can help to reduce the number of overlaps and, therefore, make the final visualisation more evident. Layouts can be taken a step further by matching the type of layout with the data projected. However, we will discuss this further with design projections. Algorithms to calculate layout can be broadly broken into three categories. **Geometric layouts** organise the graph into common geometric shapes and are usually the simplest. **Hierarchical layouts** contain the graph in a hierarchical structure based on the direction of edges. **Force-directed layouts** are a collection of algorithms which emulate a physical approach where nodes are positioned based on a simulated repelling or attracting of one another, which can be based on a given predicate such as node degree. For this example, in Figure 11, the concentric layout organises the nodes into a concentric circle based on a given metric, in this case, the relationship to the NOR gate. However, in the upcoming use cases, more complex layouts are displayed.

Network Features (Size) Quantitative data is more challenging to visualise because it cannot be categorised into manageable groups, which are then mapped to a visual medium. Instead, scalable visual changes must be employed, for example, by changing the size of nodes and edges based on some predicate. It is not represented in Figure 11 due to the simplicity; size examples can be seen within the user cases discussed next. In what follows, we explore three use cases which convey several complex features of network designs that showcase the benefits (and limitations) of having data captured by dynamic structures. The focus is on a specific aspect of the data and thus reduces complexity.

Use Case: Hierarchical Trees

Here, hierarchical trees refer to structuring genetic parts into conceptual entities, which are used to constitute more abstract conceptual entities. For example, Figure 12 displays the analogy of biological systems where smaller components are structured into higher-order components that fulfil a specific function. When implemented, this analogy produces a hierarchical tree because the physical parts are leaves, and the roots and branches are abstract systems and modules. Hierarchically representing genetic parts is a core principle to promote engineering in biology [204] because this provides structure to increasingly complex circuits. A *tree* data structure is a fundamental network topology commonly used and represents data hierarchically and at an arbitrary depth. Figure 13 shows the hierarchical network corresponding to the *digitalizer* genetic circuit[183]. Its hierarchical tree (Figure 13B) displays the conceptual modules into which single parts are structured. This information is often particular for each circuit- even similar or identical circuits- since it follows the author's conceptual framework. In this case, the top module represents the whole device and is broken down into four modules, which are, in turn, leading either to the final parts (e.g., promoters *Pm* and *P_A1/04S*) or to smaller sub-modules (e.g., GFP cassette). Specific structural details that refer to implementation strategies are essential in those genetic circuits whose goal is

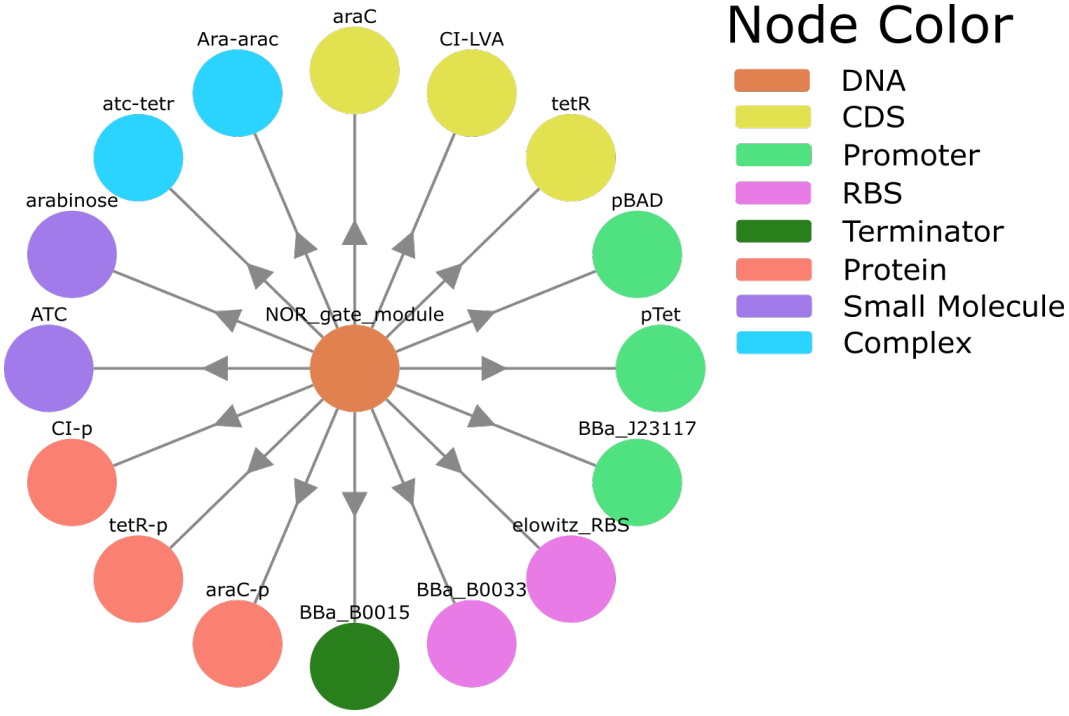

Figure 11: NOR gate visualised with a concentric layout and colour denoting genetic type or role within the design.

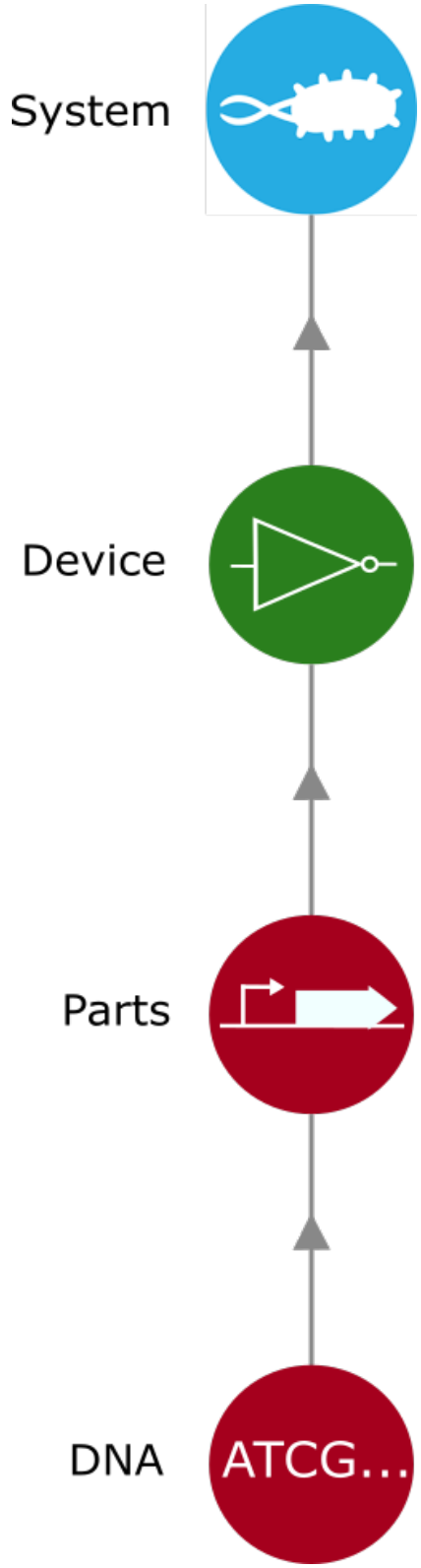

Figure 12: A potential abstraction hierarchy defining high-order cellular systems constituting several boolean devices of genetic parts defined within DNA fragments.

to let users modify parts of them. The *digitalizer* circuit is an example where the user is meant to switch the reporter gene to their gene of choice. By browsing through the network in Figure 13B, the user can find a module where the reporter is included (named GFP cassette) and the hard-coded procedure for cutting out the gene (restriction sites *NheI* and *EcoRI*) without looking at the genetic sequence of the design.

Hierarchical representations are formed by finding "parent-child" relationships (ownership labels attached to edges to denote a node "owns" another node) encoded within semantic tags. The graphs displayed in 13 biological roles are mapped to colour to encode information without an increase in perceived complexity. Also, the pyramid shape of the nodes (layout) helps transmit information concerning the hierarchical nature, with each level of the hierarchy decreasing in abstraction from the top-bottom. Finally, another small change is made with size, where the further down the tree a specific node is, the smaller it becomes to convey further the hierarchal structure of increasingly complex and abstract entities.

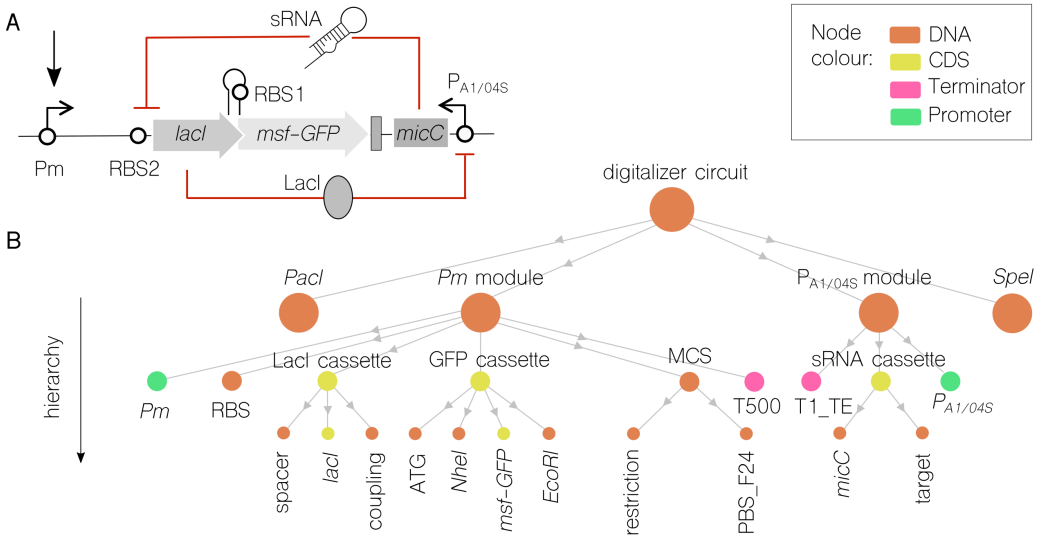

Figure 13: A hierarchical network of increasing abstraction, from modules to parts. **A** Glyph representation of the *digitalizer* [183] synthetic circuit. The circuit is based on two negative interactions between the regulatory protein LacI and a small RNA. It offers the ability to plug and play any gene of interest the user wants to *digitalise*—the reporter *gfp* gene is used for characterisation. The goal of the *digitalizer* circuit is to minimise the leakage expression of a specific gene of interest while maximising the full production. That is to say, to enlarge its dynamic range. **B** is the hierarchical network, nodes represent biological and conceptual entities, i.e., nodes at the bottom represent DNA parts, nodes at higher levels represent modules (the top node is the entire circuit), and edges represent hierarchical direction. Circuit building details are highlighted within the network, e.g. restriction sites or sequence to couple *lacI* to *msf-GFP*.

2.1 Use Case: Interactions Networks

The information captured by designs can be broadly split into two groups (discounting metadata). Firstly, constructional details, i.e., the DNA sequence and related data. Secondly, functional infor-

mation, i.e., non-genetic elements and their relationships once built. While constructional details are commonly communicated, functional elements are often less clear and can provide insight that the former cannot offer. Therefore it is essential to complement sequence-based designs and visualisations with regulatory information. Specific interaction networks can provide a higher-level understanding by visualising regulatory proteins and abstracting relationships. For example, the mechanisms that allow a regulator to bind its cognate promoter and repress a downstream gene's expression into another regulator can be abstracted into a simple network with two nodes (one per regulatory protein) with an edge-denoting effect. While this network lacks structural information at the sequence level (e.g., implementation details), it maximises the functional aspect of the circuit. Figure 14 displays the boolean logic circuit and interaction network for the 0x87 design described in "Genetic circuit design automation"[19], constituting several logic gates. The interaction network displays all physical entities interacting with other entities as nodes and the interactions between the entities as edges.

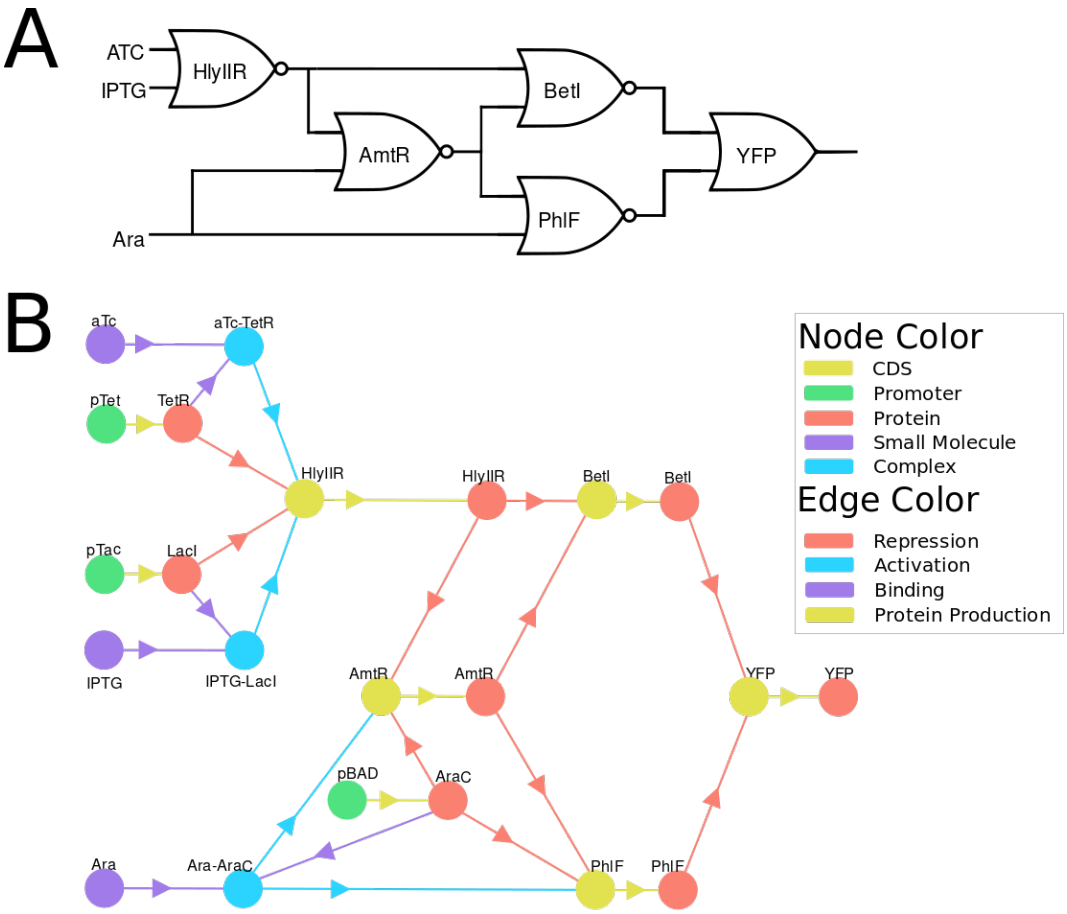

Figure 14: A) Boolean gene circuit 0x87. The circuit couples four NOR logic gates and one OR logic gate (top diagram) and uses three molecular reagents, five regulatory proteins, five genes and ten promoters (bottom diagram). B) Network representation of the 0x87 design, where physical entities (nodes) and interactions (edges) are presented.

2.2 Use Case: Scaling Complexity

So far, we have discussed how multiple projections can be created from a genetic design. However, numerous potential projections may exist within a specific data type (or collection of data types). For example, retake interaction networks where interactions between all entities explicitly encoded within the dataset are projected. The ontology that a design implements can be used to define data types for nodes and edges and can represent incrementally more abstract terms. Therefore, that interaction projection is at a specific level of detail based on the information encoded within the original data. However, the level of detail can be further modified to present a perspective which is not explicitly encoded within the original data. For example, an interaction network may omit details to simplify a representation, increase details for a more in-depth view, or infer new data from existing information. Here, we explore how projections can be further modified by increasing or decreasing the level of detail to tailor a representation further. Interaction networks provide a high-level metric with the potential to scale, and this view of the data has been chosen to display dynamic abstraction.

The network in Figure 15A displays all molecular, genetic, element types and relational information, i.e. an interaction network. An interaction network is generated by querying the data to find nodes and edges with semantic labels denoting interactions and transforming the following structure (See methods for a more in-depth description). Physical entity (node) -> interaction (edge) -> physical entity (node) Until now, the process has been the same as explained within the Use Case: Interactions Networks.

One example is to abstract all non-genetic elements (Figure 15B), limiting the information to only the indirect effects of DNA-based features (e.g., promoters and genes) on one another. In this case, relational information remains; for instance, when expressing, the coding sequence *araC* represses the promoter node BBa_J23117. Therefore, the visualisation is simplified by abstracting mechanistic details such as the production of regulatory proteins. The automatic level of detail can be taken to a conclusion by displaying input and output elements for the whole system ((Figure 15C). The highest abstraction level allows for quick circuit performance communication while abstracting all implementation details and internal workings. This over-simplification may be excessive for a relatively simple design but could benefit more extensive and complex structures. Scaling abstraction is achieved using transitive closure, i.e. the reachability matrix to reach node n from node v. It estimates the costs of different paths across the network to merge nodes along a path [205] (see methods for an expanded explanation). All graphs displayed in 15 biological roles and conceptual processes (interaction types) are mapped to colour to encode information without increasing perceived complexity. For example, two nodes: *yfp* (yellow) and *YFP* (red), are connected by an edge denoting *protein production* (yellow). As the projection moves away from the original data's structure, calculating the value of edges is likely more challenging despite knowing an edge exists. In this case, the type of interaction is calculated by tracking how the regulatory edges affect one another during the collapsing of a path (see methods for further explanation). Finally, the arrangement of the nodes (layout) helps transmit the flow of information. In this case, data flows from inputs (upper nodes) to outputs (lower nodes).

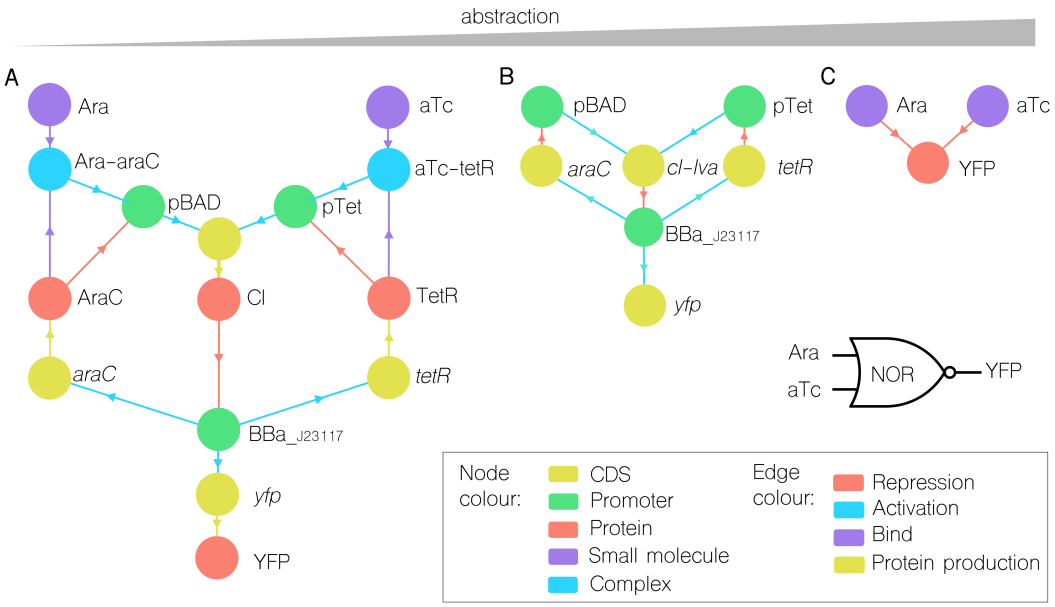

Figure 15: Adjusting network abstraction levels using a NOR gate design [206] modelled in SBOL (see methods). **A**. The NOR gate design is turned into a network with all molecular and genetic elements (nodes); and interactions between entities (edges). **B**. Non-genetic elements, i.e., non-DNA-based elements, are merged into the appropriate genetic elements. For instance, Ara and Ara-araC are merged into the pBAD node. **C**. Maximum abstraction into input-output data. The colour scheme is constant regardless of abstraction levels.

2.3 Comparing Designs

The requirement to compare multiple designs is common in iterative synthetic biology projects, such as finding shared sub-systems between designs or tracking how a design has changed over several iterations. However, current visualisation methods cannot represent differences and similarities within multiple designs and often rely on manual identification and representation, such as comparing designs visualised using the glyph approach. In the work described above, each network maps to one design. However, a single network may encode multiple designs, which can be used to identify similarities or differences between these designs. When multiple designs are merged into a single network, nodes within both designs are combined, so connecting edges from both designs will be present. It must be noted that for the unification of two designs, the labels of conceptually equivalent nodes must be referentially equivalent. That is, they reference the same resource. Otherwise, they will be considered different and will not be merged. We have discussed these issues within Chapter 4, and here we will assume that all nodes contain the same labels when conceptually identical. The ability to project networks and then apply a comparison makes network comparisons even more powerful because it enables comparison with any data or level of detail. Figure 16 displays the intersection (similarities) between two circuits where a protein interaction network is projected (Network with protein (nodes) and interaction (edges) representing negative regulation) and only where interaction processes are utilised across both designs. The process to generate the intersection of two graphs is simple: each edge from graph one is queried in graph two, and if the edge is found, it is added to the intersection graph. The comparison could also be applied to find differences within the design. Alternatively, node-only comparison where connections between nodes are not considered only the elements, for example, with interaction graphs, the discovery of cross-talk between systems when coupled. Graph comparison will be especially beneficial with designs of higher-order complexity or when comparing a consortium of designs, i.e. when the manual inspection is not feasible due to scale.

2.4 BioDesign beyond genetic designs

An advantage of a network approach is that data integration is easier when the underlying information is represented as graphs with unified semantics. While this is useful when representing designs, as many types of data constitute a design, this characteristic can excel in unifying more disparate or loosely coupled data. We briefly cover two such elements, namely metabolic pathways and experimental protocols, and discuss the potential of networks to provide a general framework for biodesign efforts.

Metabolic networks

Genetic circuits *run* inside a cellular host (except cell-free systems[207]), and the host context, particularly its metabolism, impact circuit performance. Synthetic biology aims to enhance genetic circuits by exploiting metabolic mechanisms that offer dynamics beyond the existing genetic toolkits[33]. A question that needs to be answered regarding the design process is whether metabolic systems can be merged into a genetic design[208]. To this end, we show in Figure 17 that the descriptions of a NOR logic gate and a metabolic pathway can dynamically interact if encoded into compatible data structures. Specifically, the NOR logic gate uses arabinose as input,

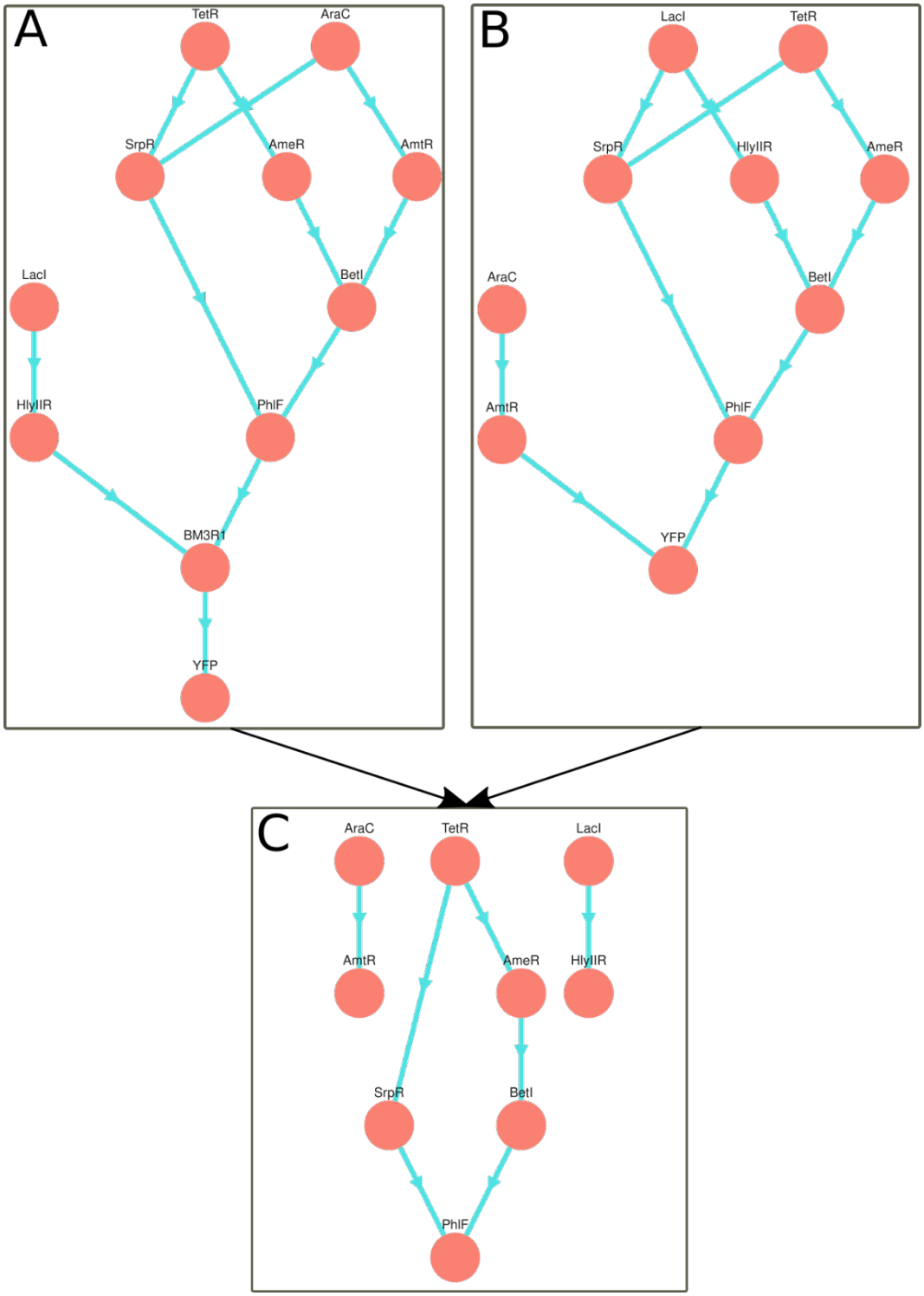

Figure 16: The intersection of two protein networks. In this case, protein network refers to the abstraction of sequence and non-genetic nodes into the resultant proteins and how the presence of this node affects the state of the design. A) Protein network representation of Boolean gene circuit *0x41*[19]. B) Protein network representation of Boolean gene circuit *0xF6*[19]. C) Intersection (joint edges) of A and B protein networks.

which interacts with the same node of the arabinose degradation pathway. Having this information within the same network allows for formalising the impact of metabolic dynamics on one of the inputs of the target genetic circuit. Metabolic pathways are added by simply merging the new data into the graph. However, ensure the data is encoded within the graph as nodes are not duplicated and new edges are attached to old nodes. Unlike genetic designs, which describe a system's intent, metabolic pathways capture dynamics such as reaction rates. Within existing metabolic networks, kinetic parameters or dynamic behaviours of metabolic reactions are encoded within the nodes or edges of the graph. For example, consider a metabolic pathway involving the conversion of substrate S to product P catalysed by enzyme E. Within the graph; nodes represent each physical entity (S, P, and E) and an edge from S to P through E represents the catalytic reaction. This edge stores dynamic information, such as the current reaction rate, which updates based on substrate availability and enzyme activity. With values encoded within the edges, which may change when information changes propagate through the network, visualisation methods must be applied to address this information. Firstly, projections can be generated based on the current values within the network. This approach enables the network to be compared in different states. Furthermore, parameters could be set from a user perspective, and the visualisation could be updated to highlight proposed feature changes in distant network regions. Also, the network layout can be adjusted to emphasise pathways or make significant changes. For example, pathways with higher activity could be more centrally located or displayed more prominently. Finally, edges can change colour based on the reaction rate they represent. For example, a gradient from blue (low activity) to red (high activity) could indicate the current reaction rate.

Protocol visualisation

The goal of all circuit designs is to be built and validated experimentally. However, formalising implementation protocols into well-characterised steps and their representation in standard data structures is still a significant challenge[209, 152, 179] that deserves more attention. Therefore, finally, we showcase the use of networks for representing experimental protocols. Figure 18 shows the network corresponding to the protocol for building and testing the NOR gate used as an example. Here, we chose (from the many options available) to represent materials and methods as nodes and information flow as edges. As in other examples, protocol graphs can also be adjusted at different levels of abstraction. For instance, the assembly node (18) includes processes such as restriction, purification and ligation—which are conveniently clustered to provide an overview of the inputs (i.e., what the assembly process gets) and outputs (i.e., what it returns). This network can be linked to the NOR graph at the top node of the hierarchy, having genetic circuits and protocols within the same data structure. The integration and visualisation of protocol data are similar to handling design data but with a critical difference. Primarily, the protocol data was encoded using the autoprotocol standard, while all design data displayed was modelled using the SBOL standard. However, when the input data has been transformed into a formalised knowledge graph, the same processes of querying semantic labels to produce focused sub-graphs can be applied (See methods for further discussion).

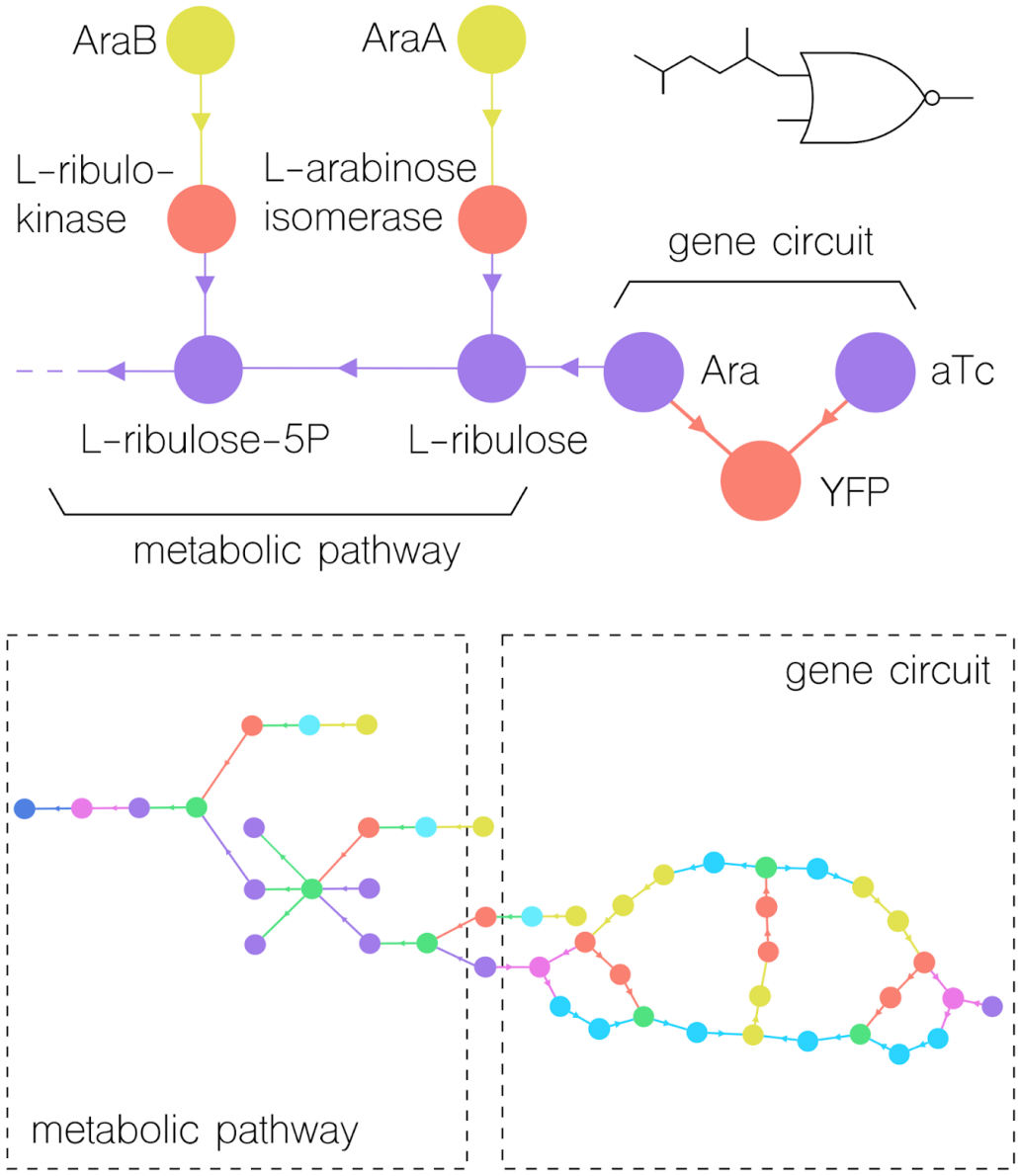

Figure 17: The network of a gene circuit that uses arabinose as input can interact with the arabinose degradation pathway. The top figure is an abstract network displaying a NOR gate's critical components and the arabinose pathway's initial steps—the bottom figure: links the corresponding extended networks.

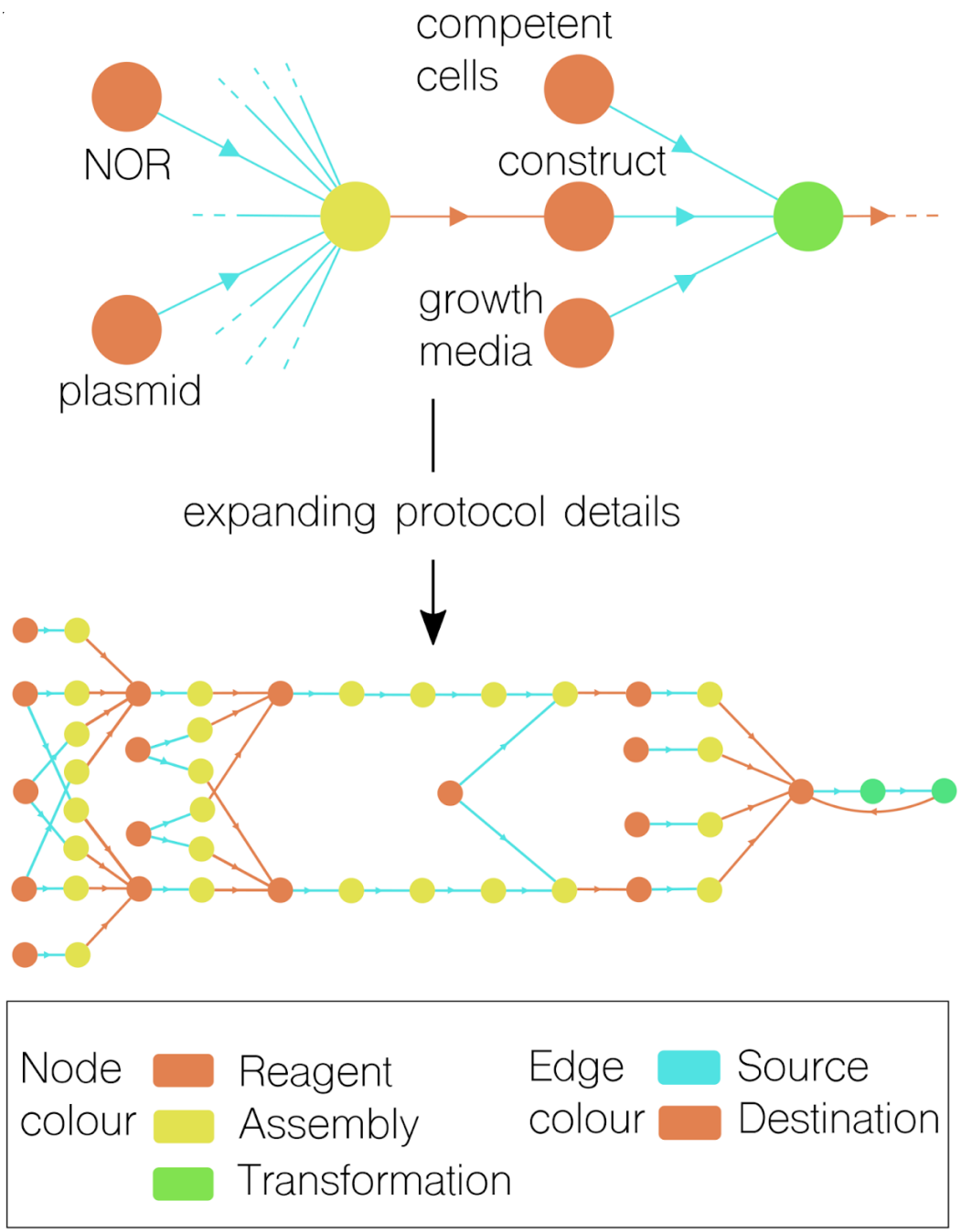

Figure 18: NOR-gate experimental protocol formalised as a network structure. The network can be interactively adjusted to show different levels of abstraction. Nodes represent reagents or sub-protocols, and edges imply input/output relationships.

2.5 Genetta

Genetta is a tool developed during this network-based research. This tool implements the new methods discussed previously such that a user can quickly load and visualise a design without knowledge of graphs. Genetta's code can be accessed here: Genetta-frontend and as a service here: Genetta-application Below, we discuss the usage of Genetta, such as features and user-facing applications. From this, we will discuss how visualisation is actualised within Genetta by explaining the pipeline that takes genetic design data and produces a tailored visualisation.

Genetta UI

This section briefly covers the usage of Genetta, namely how users upload, load, visualise, customise and edit designs.

Loading designs Before any designs can be visualised, they must be provided. Genetta currently supports two primary data formats, SBOL and Genbank. However, SBOL is recommended over Genbank because it allows for more rich data capture, including functional and structural data. While traditional data formats such as Genbank are ubiquitous, visualisation will be worse than if an SBOL file is provided in nearly all cases without adding more information. Within Genetta, these files can either be uploaded or pasted or the address of an external resource, such as a Synbiohub ID, can be provided. When uploaded, the user will have exclusive access to this design, which will persist through multiple visits. Furthermore, the design can be exported, including any changes made.

Visualisation The visualiser provides the projection and visualisation design data discussed earlier. The focus is hiding the complexity behind a series of button presses, which will change a specific visualisation feature. This section will refer to figure 19, which is Genetta's UI for the visualisation tool.

The first step is to load designs from the previously uploaded designs **(1)**. The visualisation tool can load multiple designs and integrate them into a single graph where the load predicate dictates how multiple graphs are connected with options of Union (All data in all graphs are loaded), Difference (elements that only exist within one graph are loaded) and Intersection (elements that exist within all graphs are loaded). If a single design is loaded, the type of load predicate is redundant.

Once the graph is loaded, it is visualised within the dynamic graph panel, allowing manual inspection. Depending on the chosen view, nodes **(2.1)** represent single data points within a design, such as genetic parts, interactions or metadata. Edges **(2.2)** represent connections between data in a design, such as interactions between entities or to describe structure. Edges will take the shortest path between the two connected nodes.

The options panel **(3)** primarily takes inputs and renders the output from the visualiser using asynchronous communication between the client and server, allowing users to view the graph during computation.

Four buttons provide specialised features.

- Documentation **(3.1)** - Provides information on how each option affects the projected graph.

- Layout customisation **(3.2)** - Provides further customisation of the layout algorithm used to calculate the positions of nodes on the canvas.
- Canvas Changes **(3.3)** - The button allows manual manipulations of the canvas and graph, such as colour changes and node removal.
- Network information **(3.4)** - The information button provides graph-centric technical information from the node and edge count to more specific information, such as the number of components.

The rest of the options are groups of buttons that make specific projection or visualisation changes.

- Presets **(3.5)** are collections of options (modes, views, layouts and other visual changes) that, when combined, provide a preset insight into the data.
- Modes **(3.6)** are similar to the load predicate (discussed in the loading designs section) and allow further data manipulation. For example, set operations directly applied to the node or edge labels.
- Views **(3.7)** are data projections that focus on a specific aspect by rearranging, aggregating or making inferences on the complete network—for example, Interaction between entities or a parts hierarchy.
- Layouts **(3.8)** are algorithms to calculate the coordinate positions of nodes on the screen and can be broken into three types force-directed, hierarchical or geometric.
- Several remaining options (3.9) are not displayed here, such as colour, text, size and shape based on features and datatypes within the network. For example, colours that map to genetic roles. Furthermore, the options to export the visual and data representations of the current projection.
- Legend **(4)** maps categorical values to colour and is automatically generated from changes made within the options panel.

Editing designs An extension of the visualisation tool is the editor. The editor tool has the same functionality but allows users to add new nodes and edges to a design. A key feature of the editor is adding new information to the discussed projections, which is disseminated and expanded to keep the original data's structure. For example, if a user wants to add a new interaction between entities, the interaction graph can be projected, and additions changed will be propagated backwards. This section will refer to figure 20, Genetta's UI for the editor tool.

Firstly, nodes can be added to the network directly by taking several required and optional values. The primary requirement is a node key **(1.1)**, a unique name for adding the node. The following required value is the node type **(1.2)**, which classifies the category of information inserted into the design. Metadata **(1.3)** are optional additions such as sequence data, which will also be added as new node properties depending on the information type. When submitting **(1.4)** nodes to be added to a network, the node key will be checked to ensure it resolves to a valid resource, for

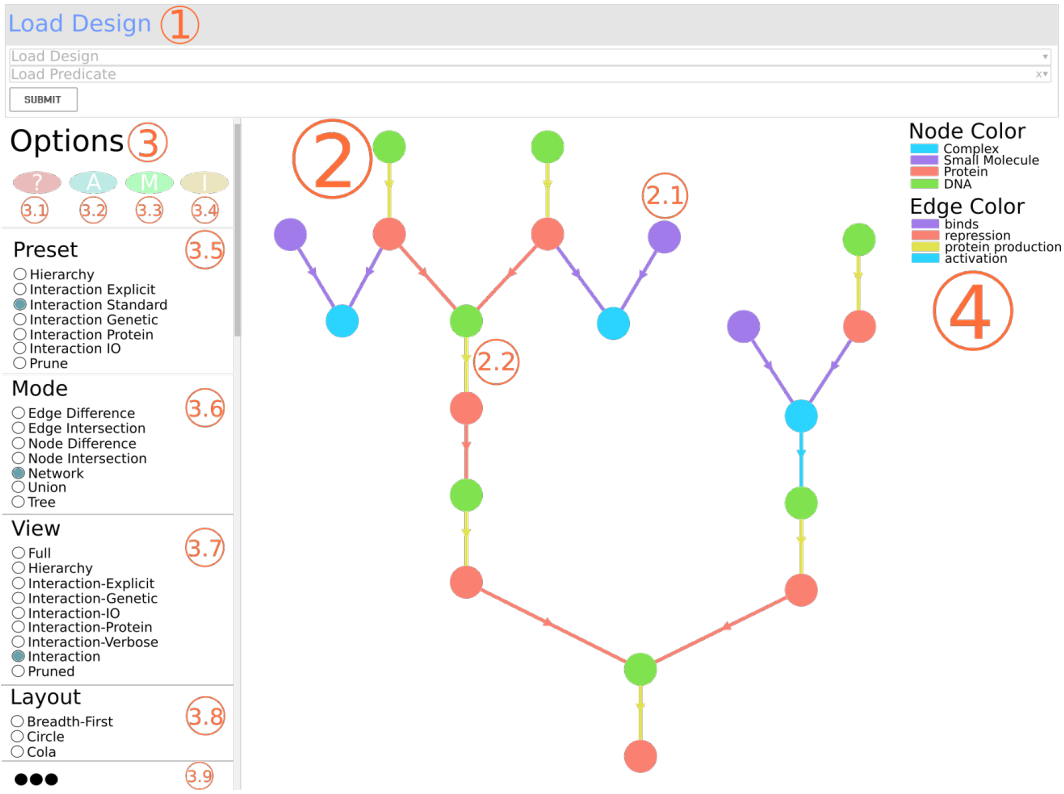

Figure 19: The user interface for the Genetta visualisation tool.

example, synbiohub.org/public/igem/BBa_R0010/1. Suppose the IRI provided does not resolve to an accessible resource. In that case, Genetta will attempt to find a set of candidates which could be used instead, using methods to identify canonical entities discussed within Chapter 4. Finally, when the final value of the node is chosen, this node will be automatically added to the design, ensuring the information is in line with the underlying structure.

Edges representing relationships between nodes can also be integrated into the design. The predicate **(2.1)** is the first choice made and represents the type of connection between two data points. The predicate options are dependent on the current projection. For example, if the interaction graph is projected, predicates such as repression, activation and genetic production will be options. The subjects **(2.2)** in a relationship are filtered and presented based on their types and whether they are valid with the predicate. For example, if the user chooses the predicate "protein production", the input node must be a coding region and the output subject a protein node. Sequential filtering means invalid information is not accidentally added to the network via a projection that does not support it and makes integration easier. Once the abstract edge between two data points within the connection is defined, the data can be submitted **(2.3)**, and the new edge will be added to the design. When working on the projection of the data, the new edge and add all the information required for this new piece of information to align with the complete data structure.

Genetta pipelines

This section describes how the underlying methods of Genetta visual and editor are achieved at a high level. Each aspect is described relative to figure 21.

Modifying graphs and managing data Genetta stores the input data as a labelled graph where labels are semantic tags. An initial step is to convert **(B2)** the input data into a shape valid for the knowledge graph. Conversion to a knowledge graph provides several benefits, such as a structure tailored for a network approach, and new input data types can be quickly introduced. Genetta contains an internal ontology containing several classes, each defining a set of requirements which must be fulfilled to be eligible to become an instance of that class. Furthermore, the data model defines an inheritance hierarchy where more specific classes inherit information from more general classes. For example, the DNA class is derived from a physical entity class, and then an RBS class is derived from the DNA class, and when combined, these features mean that all converted data is always the most specific object possible.

All data within Genetta is stored within a single "world graph". This neo4j[210] graph database management system allows storing and accessing graphs whose sizes are infeasible to hold in memory. Therefore, behind the scenes, all designs currently stored in Genetta are stored within a single graph where nodes and edges shared between designs will be connected. To identify ownership, i.e. what pieces of data belong to which design, nodes, and edges contain a property defining what graphs own them. Storing all information within a single graph helps reduce the application's size because duplicates are not stored within multiple graphs. However, it also allows reasoning on a larger dataset instead of single designs, which is beneficial for identifying any general patterns that may occur. Furthermore, because the data has been converted in line with an ontology, the persistent graph is a knowledge graph with semantic labels on nodes and edges and specific constraints

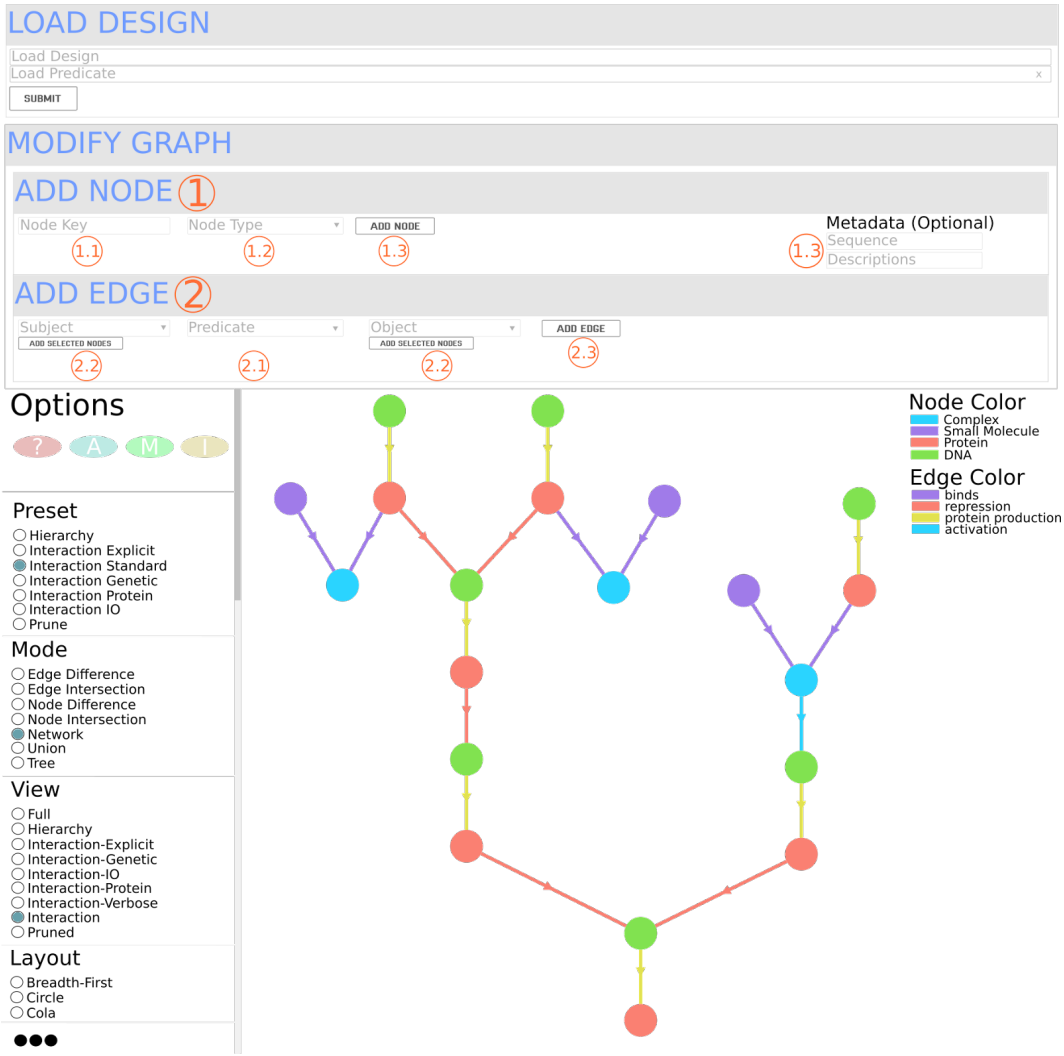

Figure 20: The editor tool, an extension of the visualisation tool that enables modifications to be made to a design.

specifying what nodes can connect. The resultant knowledge graph is one where data will automatically connect when introduced, and the semantic labels and structure allow for more abstract questions to be asked.

VISUALISATION The visualisation tool uses a pipeline system, where the currently loaded graph is provided, and the output is all the information required to visualise a comprehensible representation. Multipartite graphs **(B3)**, such as the full graph described above, can rarely be successfully visualised in their entirety, mainly because the presented data domain is too large. Therefore, specific views of this data are projected (usually bipartite or monopartite). These views focus on a particular aspect of the data, such as the interactions between physical entities or the conceptual hierarchy common within synthetic biology. Building **(B4)** a specific projection is unique (building an interaction network differs from a parts hierarchy), but some high-level processes persist. A projection will likely only represent one or two classes. Therefore, we query the data using the semantic labels for the classes in question, and this step may be required to produce an understandable representation. An optional step to scale the level of detail within the graph to further fit the requirements of a user. Scaling abstraction is achieved by transitive closure, which involves traversing the graph from a node looking for a target node and collapsing the traversed path.

The second module group is responsible for presenting the projected graph visually according to user requirements. The modules use dash-Cytoscape, a Python library for creating interactive network visualisations. It is built on top of Dash[211], a web application framework which enables graph visualisations to be modified in real-time, and Cytoscape.js[212], a JavaScript graph visualisation library. The visualiser module **(B5)** computes visual elements such as layout, colour, shape, and size to help the user further comprehend the data. The module receives the projection and a set of requirements (user inputs) from the dashboard. For each requirement, each node and edge is provided with a value (size of a node, for example). Finally, the visual elements are combined with the projection graph (nodes are given screen positions, RGB colours, for example) and sent back to the dashboard to be presented to the user. The dashboard **(B6)** is the user interface of Genetta and takes inputs, begins the pipeline of processes previously discussed and renders the output from the visualiser. A significant advantage of the dashboard and the technology it is built on is that user inputs are sent via callbacks to enable asynchronous communication between client and server, allowing a user to view the graph during computation.

EDITOR The editor is an extension of the visualiser tool that can take information from the user and integrate it into the design. The key feature is the ability to take information that is not specified within the structure of the underlying data model and transform it into a valid format which is then integrated into the design data. The editor dashboard **(B7)** enables adding new nodes and edges. When a network is projected (as is done within the visualiser), a set of valid node types and edge types are provided to the user. These types are specific to the current projection. They are calculated within the expansion builder **(B8)** module, which queries the underlying ontology for restrictions encoded between classes (rules defining what entities can connect using specific edge types). For example, if the structural hierarchy is projected, interaction nodes and edges will not be offered as new information to be added because it is not the type of information being displayed here. If a genetic part node is chosen to be added, the optional metadata will present specific information, such as sequence data. The dynamic system, which constrains user input

based on previous inputs, ensures that only valid information is returned to the builder. Finally, the editor automatically updates the figure when the expansion builder returns an updated view.

The expansion builder takes information from projected representations, which may not align with the underlying data structure. It then expands this information and integrates it into the underlying network while ensuring the same form is retained and connections between old and new information are established as needed. While it is true that each projection module integrates data differently (adding interactions is not the same as genetic parts, for example), specific processes are worth discussing. In most cases, a user will add a new edge to signify a connection between entities. For example, a new interaction between two physical entities is created using a new edge. However, the underlying data structure likely does not encode an interaction with a single edge. For instance, in SBOL, interactions are represented using several classes, such as Interaction, Participation, FunctionalComponent, and ModuleDefinition classes, with multiple edges connecting instances of these classes. Therefore, each projection includes a method for mapping this abstract piece of information to expand it into all the required information to align with the underlying structure.

A significant challenge with the expansion arises when the level of detail in a design is highly abstracted, as seen in the protein interaction network. This abstraction illustrates the impact of a protein's presence on the production of other proteins within the design. These projections are achieved using transitive closure to collapse paths and create a simplified representation. However, when a user specifies a new interaction between two proteins, it becomes unclear how this interaction affects the larger design. Consider our protein interaction abstraction: when a user defines that p1 represses p2, they are not specifying direct repression; instead, p1 represses a transcription factor upstream of the p2 coding region. Therefore, the implicit reference encoded within the newly projected data must be identified. Therefore, the protein example is reexamined to calculate the likely transcription factor. The complete interaction network is projected, and a traversal is performed from p1 to p2. It is assumed that the promoter element nearest to p2 on this projection is the subject of this interaction. If a promoter is found, a new repression interaction is established between p1 and the identified promoter.

3 Methods

3.1 Projecting networks

The ability to project networks that will be visualised is at the core of the visualisation process. As discussed, projections are subgraphs with different structures (likely more familiar to natural networks, so the comprehension of the underlying data model is not required) to provide insights into the data, which may not be apparent when looking at the whole graph. Here, we will describe how an intermediate projection of the interaction network is made from the entire graph and how projections can be chained, thus changing the final representation (in this case, to produce more abstract output). The complete network (figure22 A) contains all information within the native structure. Note that due to the network size, the labels are only presented on the areas of the networks that are being discussed. The interaction network (figure22 B) is projected by querying for all interactions. When an interaction is found, a transformation step is applied where instead of projecting the interaction node, directed edges between the inputs and outputs are added. In this

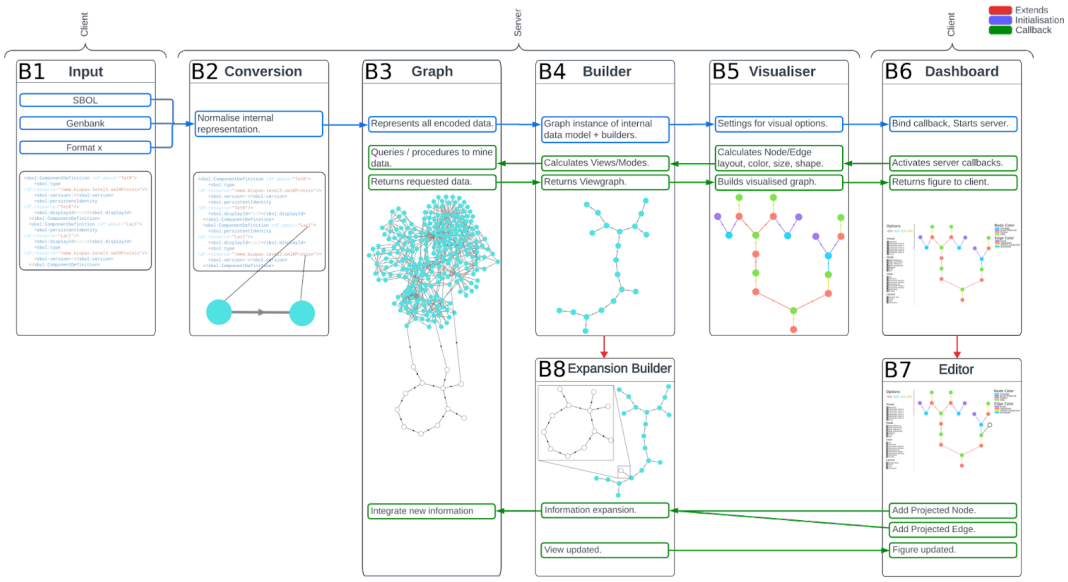

Figure 21: The workflow for transforming designs into dynamic network structures. **B1**) The input design should be formalised using existing formats. We advocate for using SBOL for genetic designs since it allows for capturing complex information. **B2**) Input data is normalised into an internal structure by mapping semantic labels or keywords to a pre-defined network data model. **B3**) The graph with all design information is represented and ready for algorithmic analysis. **B4**). The builder module of the software produces specific sub-networks based on user requirements and the resulting analysis of the original structure. **B5**). The visualiser calculates all visual-specific elements (layout, colour, shape, size) and renders the graph accordingly. **B6**). The dashboard is the user aspect of the application. It handles the graph rendering and user inputs by returning callback requests to the server.**B7**). The editor dashboard provides the ability to add new nodes and edges. When a network is projected (as is done within the visualiser), a set of valid node types and edge types are provided to the user. **B8**.) The expansion builder takes information from projected representations and expands it to the underlying network, ensuring the same form is kept and that connections between old and new information are made where necessary.

case, the "Binds" node is added as "Binds" edges from "ATC" and "tetR" to "ATC-tetR". Once the interaction network is projected, the protein network (a network displaying the effects of the presence of one protein on the rest of the network) can be projected by finding transitive connections between proteins. figure22 C displays this final network, which displays how the presence of a specific protein affects the circuit. The network is established by transitive closure by performing a Dijkstra shortest path traversal from each protein node and terminates when all reachable protein nodes are found. When these paths have been found, the route is collapsed into a single edge. For example, a path between "BM3R1" and "SrpR" is found, which is collapsed into a single edge. However, despite identifying an existing edge, it is unclear what it means (in this case, what type of interaction is taking place). In the case of interaction projections, the regulatory edges (activation and repression) flip the regulation. For example, with figure22 C, because "BM3R1" represses the promoter, which initiates transcription of "SrpR", the resultant edge has a "Repression" type. However, this process is dependent on projections and becomes more challenging to derive as the representation structure moves further away from the initial configuration.

3.2 Graph comparison

Graph comparison, as discussed previously, is a relatively trivial process once the semantics of the graphs are unified. Table 1 displays the stages of producing a third interaction network. Furthermore, the table contains an example using two protein projection networks and is visualised in Figure 23. In short, it involves identifying common nodes and then common edges between these nodes.

Table 1: The process to perform the intersection of two graphs with an example of the intersection of the graphs of 0xc7 and 0xF6 circuit as described in Nielsen *et al.*[19] that have been abstracted into protein interactions.

Action	Description	Change
Identify Common Nodes	The common nodes between the two input graphs by comparing their node sets.	V ={AraC, LacI, AmtR, PhIF, TetR, YFP }
Create Intersection Graph	A new graph to represent the intersection. The new graph is created with the common nodes in the previous step.	G = (V, {})
Iterate over Edges	Iterate over the edges of one of the input graphs and check if the corresponding edges exist in the other graph. If an edge is found in both graphs, it is added to the intersection graph.	G=(V,{ {AraC,AmtR},{AmtR,PhIF}, {AmtR,YFP},{PhIF,AmtR}, {PhIF,YFP},{LacI,PhIF}, {TetR,PhIF},{TetR,YFP} })

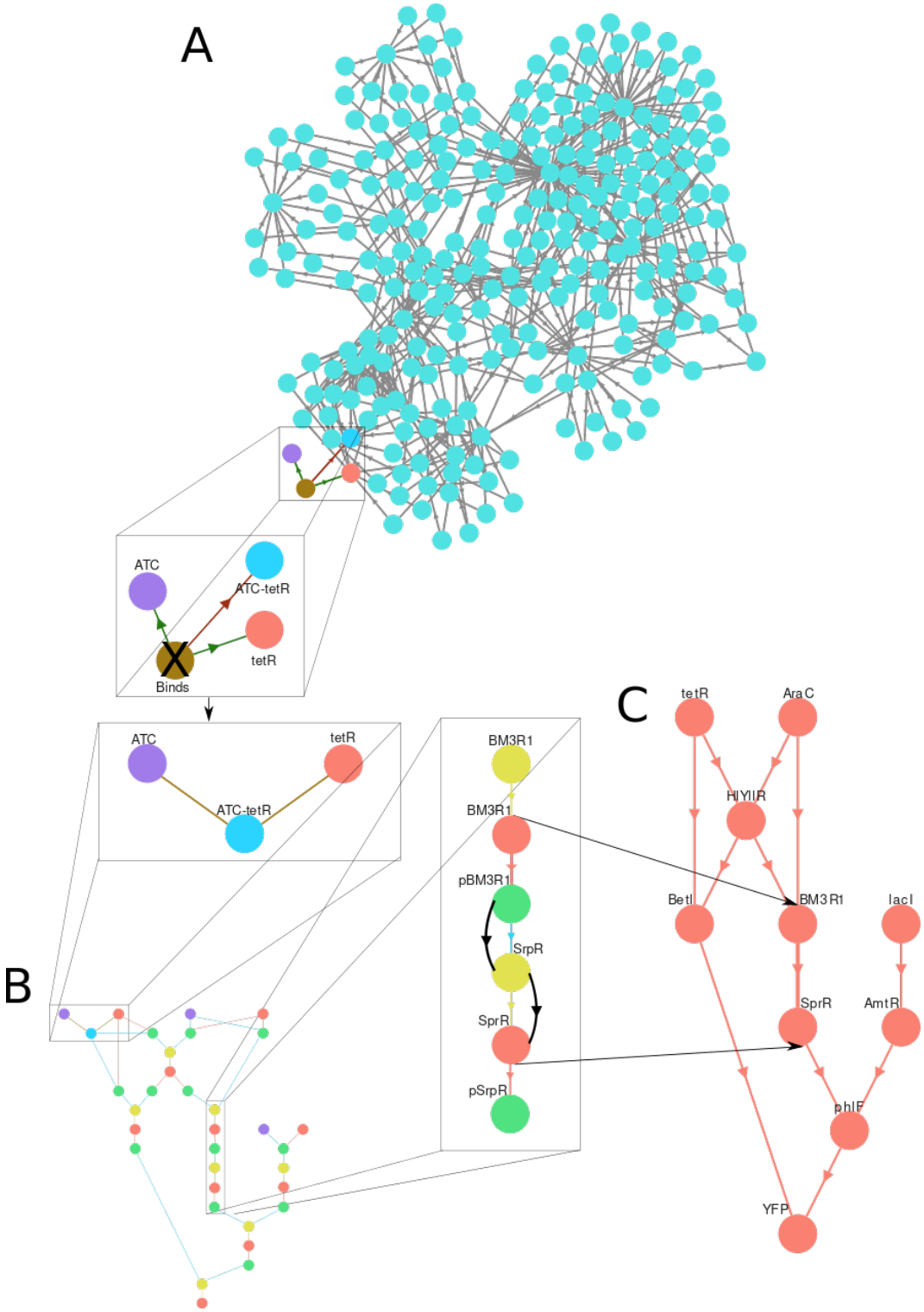

Figure 22: Displays the process to produce the protein effect network (abstract interaction network displaying the effects of protein presence on the circuit). **A)** All design information is visualised. Interactions are structured as nodes with interactant edges connecting to the participating physical entities. **B)** The interaction network is visualised by transforming information from the complete network. Interactions are formed by collapsing the node into new edges between the participating physical entities. **C)** The protein effect network is visualised from the interaction network. Edges between proteins are formed by traversing from the source protein until a different protein is found.

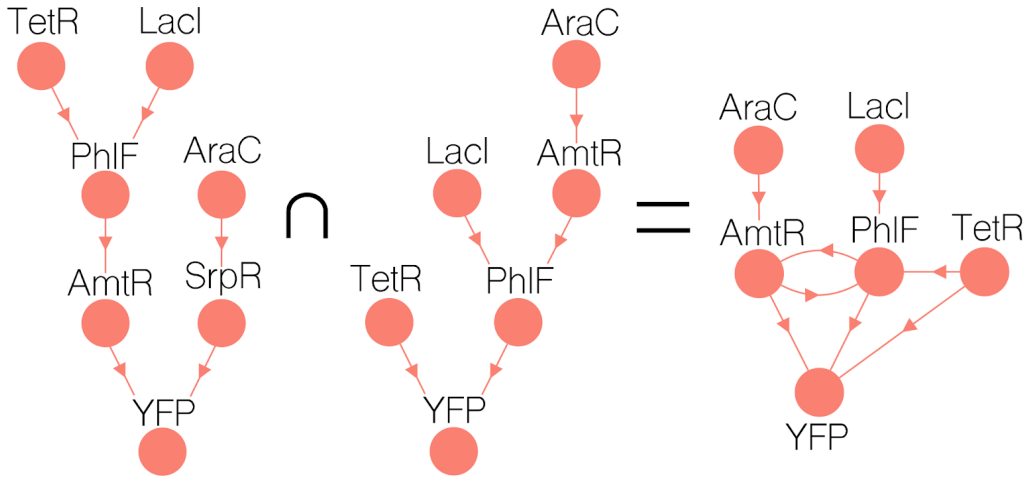

Figure 23: The graphs of 0xc7 and 0xF6 circuit as described in Nielsen *et al.*[19] are abstracted into protein interactions by transitive closure via depth-first-searches (left) and intersected with another network (middle) to identify common nodes and sub-graphs between the two (right).

3.3 Protocol representation

For conveying purely biological systems (genetic designs and metabolic pathways), SBOL has the mechanisms to capture all the required information. However, when representing protocols as networks, a new format must be found because SBOL does not have an established solution for describing build protocols. Here, initially, the raw opentrons log files were used by presenting individual steps taken during execution as a node and edges representing the order. These files are unstructured post-execution log files that capture a specific robot's granular activities, precisely the movements and actions of a liquid handler over time. Figure 24 shows the outcome of visualising this information as a network where the result is a linear set of liquid transfers. However, these visualisation types are of minimal value because they are simply a linear connection of nodes and edges with abstract information that does not utilise network features. Furthermore, these transfers have no context, such as what equipment the transfers are occurring within or temporal information, such as how long an activity would take. The primary issue with the previous approach was that the visualisation was too specific to the opentrons process and resulted in an incomprehensible visualisation. Furthermore, the creation of the representation cannot leverage any processes that made visualising designs valuable because the underlying log data is a flat structure that does not encode any context or abstract information, similar to many of the pitfalls of representing and visualising genetic designs encoded within Genbank discussed previously. Therefore, instead of using data from a specific robot within a particular process, the building protocol has been defined more robustly with more detail within a standard experimental protocol language named Autoprotocol. Autoprotocol is a standardised language designed for laboratory automation and experiment control and allows the definition and execution of complex experimental protocols. Much like SBOL, autoprotocol uses a standard specification, ensuring all protocols

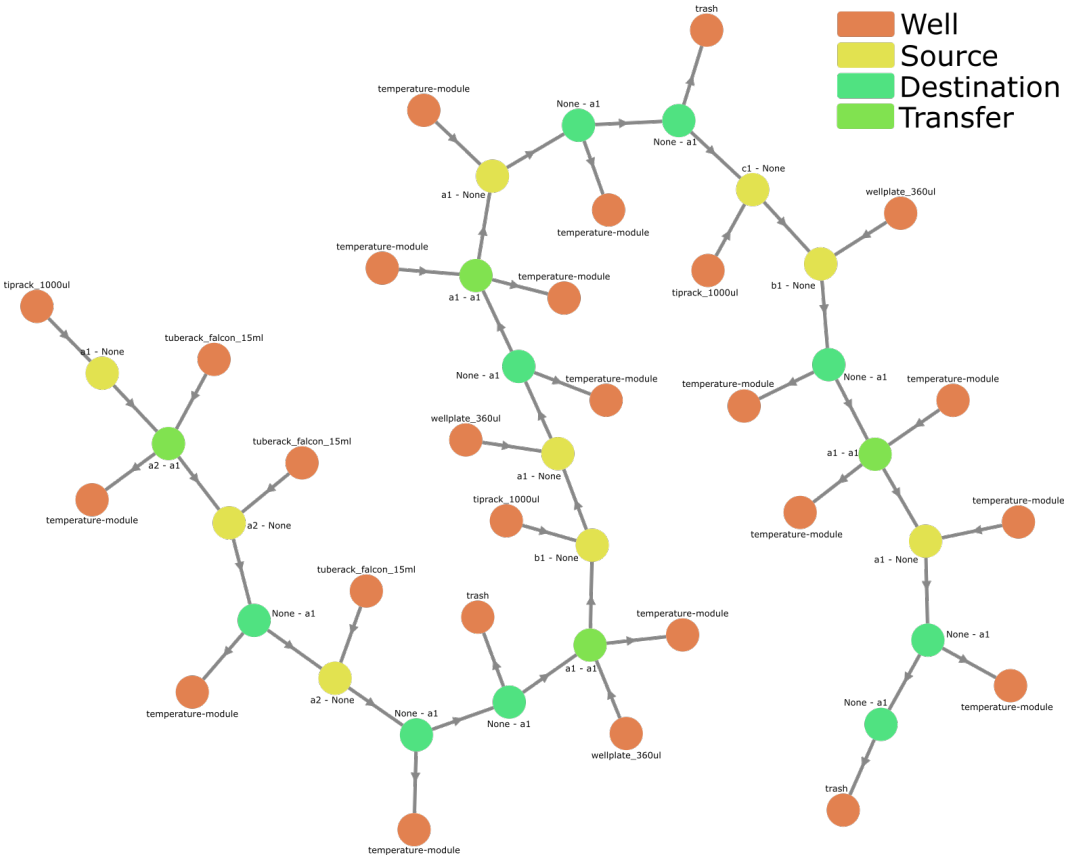

Figure 24: Network of potential build instructions for NOR gate constituting opentrons (OT2) movements (transfers from different wells).

are defined using these terms and interchangeable, providing a standard interface convenient for creating networks. For this process, Autoprotocol enables two specific features to abstract specific actions into modules and to name and provide information on particular actions. Figure 25 displays the example NOR gate protocol defined within Autoprotocol. The protocol constitutes three major modules, with further levels of abstraction and then specific actions. Within each module, less abstract but still non-specific actions are defined, such as DNA purification. These abstract actions list specific, actionable instructions, such as the instructions to wash said DNA. This more detailed and structured representation enables the same visualisation techniques to be applied to visualising genetic designs. Figure 26 displays a similar representation as figure 25, which displays the inputs and output of each step but contains much more detail. For example, unlike the previous representation, the liquids transferred between containers are labelled. Furthermore, when visualised, it displays that the protocol is not linear and specific steps can be executed in parallel. When described within a robust standard format, it is also possible to represent the protocol differently. Figure 27 displays the same protocol but with a different structure. Instead of an input/output approach, edges represent the actions, and nodes represent physical entities such as reagents. Like design visualisation, these two examples are possible representations of many that can be projected when data is formatted in a standard format, and the underlying data is rich. In conjunction with changing the representation, it is also possible to modify the level of detail because it is explicitly encoded within the original data (see Figure 25). Figure 28 displays two representations with the same structure as figure 27 but with increasingly abstract representations. Both metabolic and protocol network visualisation displayed here are initial efforts and could be expanded to provide more robust representations and insightful visualisation (see conclusion for further discussion). However, it further displays the power of network visualisation as many of the same processes used to generate design representations can be applied to metabolic and protocol visualisation. Furthermore, as discussed when displaying how metabolic networks can be represented, any information can be visualised in conjunction, providing that the underlying structure is unified.

4 Discussion

This work explored how genetic designs can be visualised to provide multiple insights, making complex systems more comprehensible. Some methods are already established to represent and visualise genetic designs but commonly produce static sequence-based representations which only provide a single representation and scale poorly. Network visualisation is an established method for understanding complex systems by revealing valuable insights and facilitating the discovery of hidden patterns. Furthermore, networks can represent any arbitrary datatype so that the representation can focus on any information (or interpretation) defined within the dataset and is not explicitly bound to an individual type of information, such as many existing visualisation techniques. Also, networks are malleable, so the structure can be changed to fit specific requirements.

Here, we developed new methods that employ the existing network visualisation techniques for genetic design visualisation. The key to this work is to project new representations from the whole design to produce several insights at differing levels of detail. Furthermore, the projections are designed to be comprehensible even when the underlying data model is not understood. Here, we

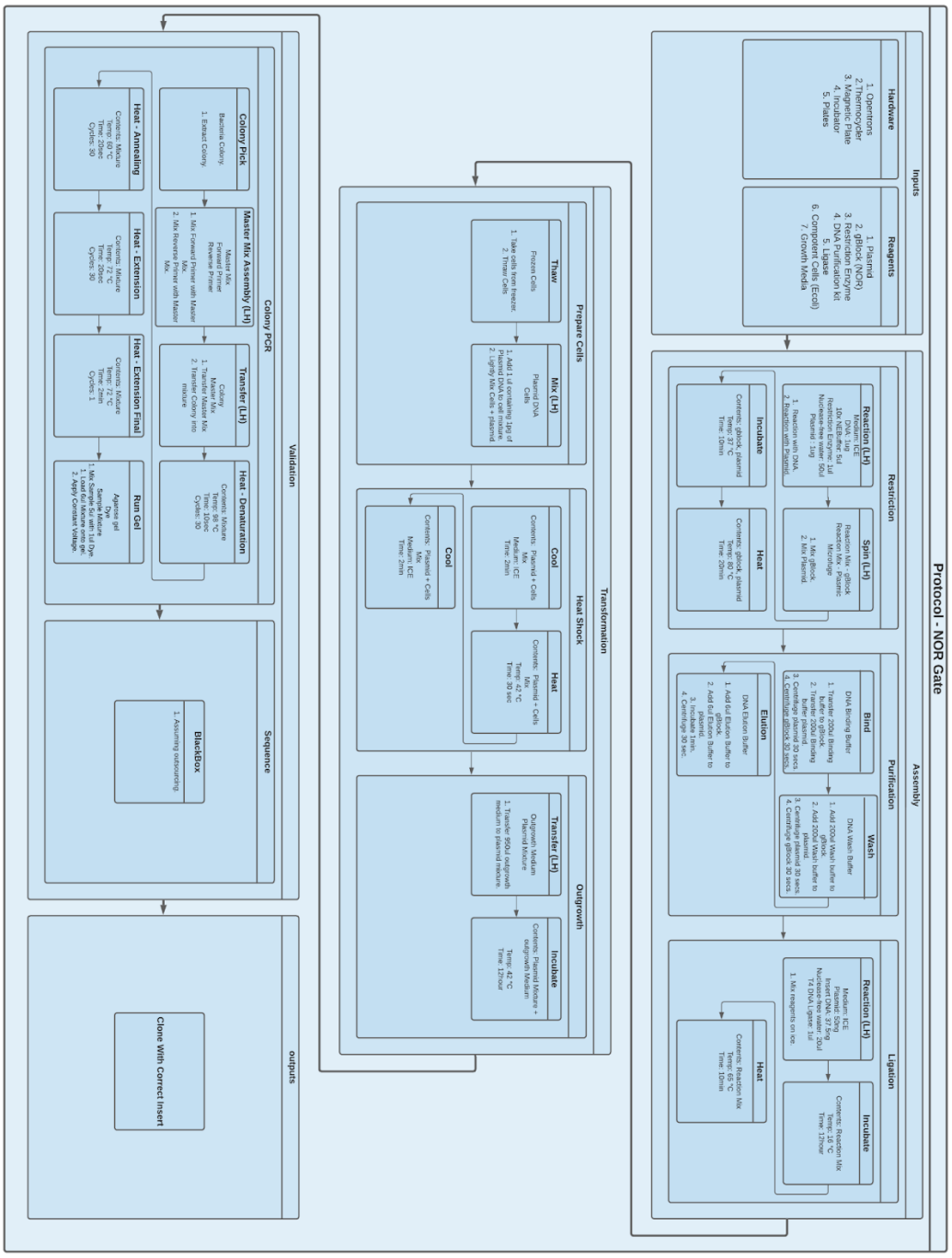

Figure 25: Modular build plan for the NOR gate design. Constitutes three abstract steps (Assembly, Transformation and Validation).

Figure 26: The NOR gate build protocol's input/output representation. Nodes represent physical entities such as reagents, and abstract actions such as transferring liquids and edges represent connections to sources and destinations.

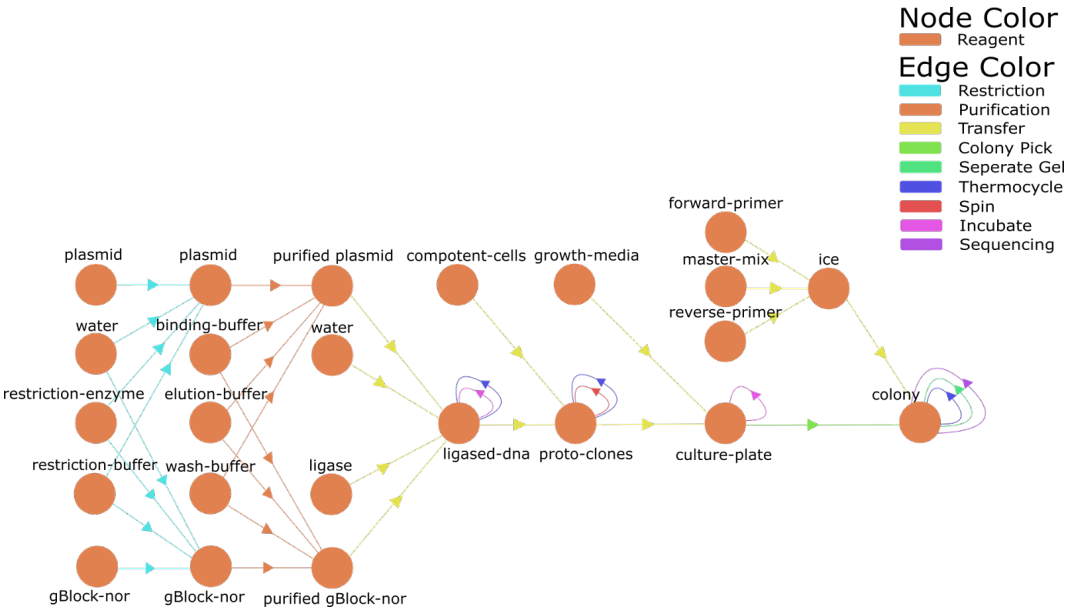

Figure 27: The NOR gate build protocol's process representation. Nodes represent reagents (the input and outputs of processes), and edges represent specific and abstract interactions.

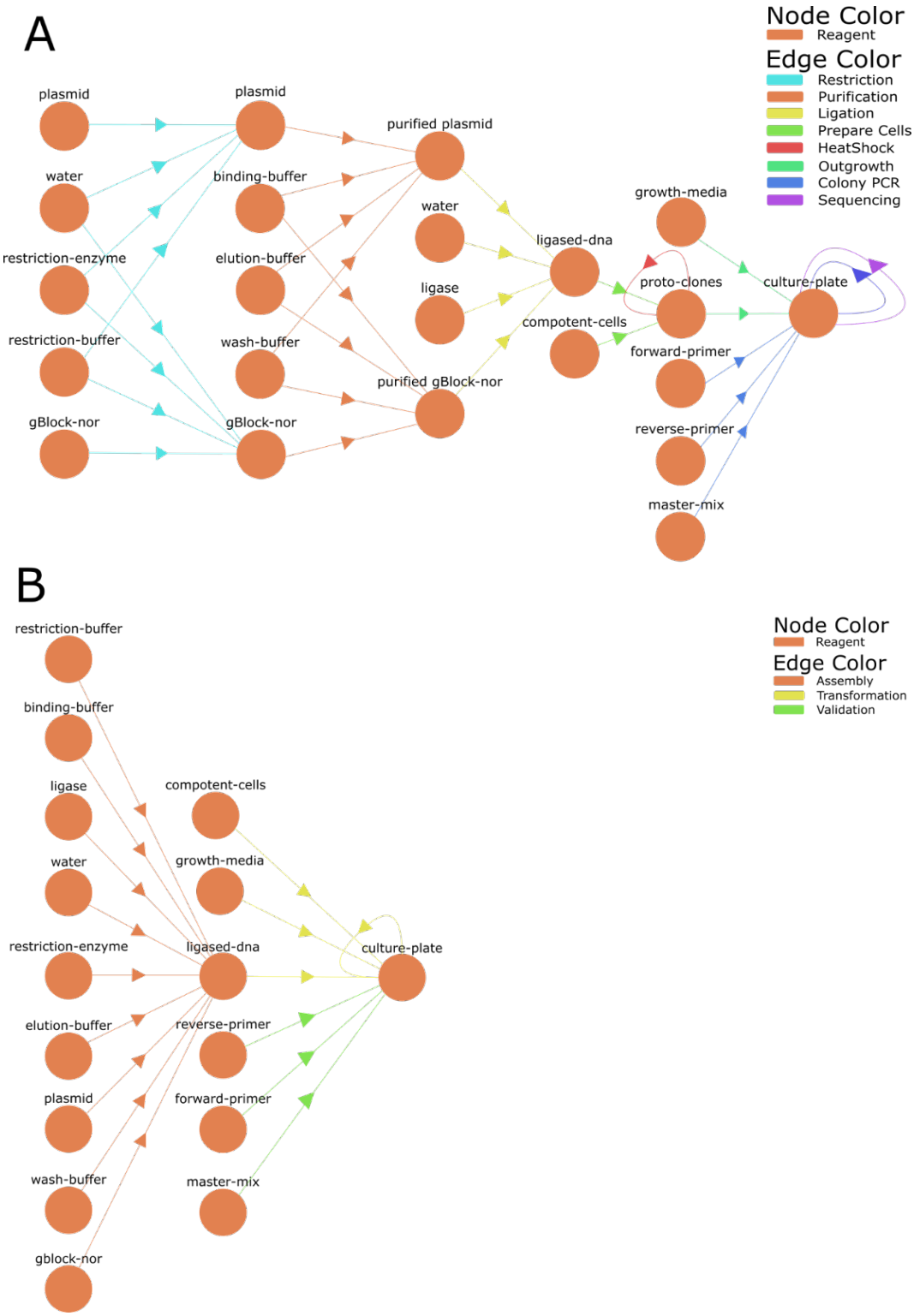

Figure 28: Two abstract representations of the protocol process to build the NOR gate design. A) Abstracts specific actions into higher-level processes, for example, the ligation step. B) Abstracts all processes into three modules (assembly, transformation and validation).

also explored how mapping categorical and continuous information to visual features can further reduce complexity. Finally, some other applications of applying network visualisation, namely, the ability to compare designs, coupling metabolic networks to genetic designs and picturing experimental build protocols. This section evaluates the outcomes of this work. Furthermore, the potential future directions for research are discussed based on the challenges faced in this work.

4.1 Strengths of a network approach to genetic design visualisation

Tailored network representation for multivariate genetic design data

Existing methods within synthetic biology can only represent a specific subset of the design mainly because the visualisation mechanism is coupled to the types of information. For example, the glyph approach can only represent abstract sequence-based information. However, genetic design data is not only sequence data. For example, it includes functional, metadata, experimental and design constraints. With network visualisation, any information can be presented because network visualisation is decoupled from a specific datatype. The outcome is that any information within a genetic design can be visualised. Presented previously (interaction networks, structural hierarchies, protocol flows and merging design and metabolic information) were a sample of the potential information that can be presented. If new information that is not currently considered relevant needs to be presented, this would be inherently impossible with existing methods. However, with a network approach, a new projection strategy would be defined, and this new information could be visualised without a requirement for the user to understand the structure of the original dataset or comprehend a domain-specific representation. Furthermore, another dimension is enabled because designs can also be merged, compared or contrasted within the projections. For example, two designs could be compared to how designs are similar from an interaction perspective but are similar from a structural. In short, a vital advantage of this approach is the options provided to a user by exploiting the dynamic nature of networks.

Genetta, the accompanying tool, enables users to upload an existing design file, which is transformed into a network. The user is then able to interrogate the different aspects of the design quickly with no requirement for them to understand the technicalities of how networks work but also the underlying data structure, such as SBOL. Furthermore, the user can further develop the design within these projections views, which is automatically expanded and integrated, ensuring the underlying data structure is maintained.

Scaling genetic design complexity

The complexity and size of genetic designs can vary massively. Because of this, a single level of detail can never be optimal for all designs. For example, a high level of detail may be desired for more moderate designs but will become incomprehensible with large designs because of the unmanageable amount of information. A network approach can scale complexity because the visualisation method is decoupled from the underlying data. Furthermore, given some requirements, networks have the inherent ability to retract and expand. Here, the ability to expand and merge nodes and edges was exploited in line with how practitioners within synthetic biology may consider a design. For example, with an interaction-based network, sometimes the interactions explicitly encoded within a design are not at the desired level of detail. Take a classic regulatory mechanism where

a coding sequence transcribes a protein which binds to an operator, inhibiting a promoter and stopping the production of another protein. A network could merge nodes and edges to present alternative representations, such as how the presence of proteins affects the production of another protein by merging all genetic elements (promoter and CDS in this case).

In conclusion, a key feature of networks is the ability to create a more or less detailed version of a specific representation, further enabling users to tailor the visualisation to their requirements. Combined with the ability to present multiple representations of the same design, Genetta can also scale the complexity (where applicable) to provide further specialised insight to a user by increasing or decreasing the detail of the network.

4.2 Limitations of networks in the current landscape

User unfamiliarity with representing designs as networks

Currently, sequence-level visualisation and considering genetic designs at the sequence level is still ubiquitous within synthetic biology. Therefore, such a drastic change in visualisation would be a challenge for the user to use, especially if this user is unfamiliar with networks. This issue is not a limitation of network visualisation but a social challenge within the community. It is a challenge that is valid for all efforts to introduce standards into synthetic biology, that is, the challenges of changing the everyday working practises of established groups.

Reliance on rich input data

Strong network visualisation requires rich underlying data. For example, network visualisation cannot visualise sequence data without a pre-processing method to mine explicit features (see enhancing designs) from the sequence. Therefore, for a network approach to representation and visualisation within synthetic biology to become more widely accepted, standardisation and fostering the capture of rich data must be advanced.

Limited usage with trivial designs

Network visualisation excels in reducing and representing complexity. However, the advantages of these methods are less apparent with smaller-sized datasets because they are less complex. Therefore, it is likely that the difficulties that a network approach introduces, such as unfamiliarity with the visual output, outweigh the advantages.

4.3 Future work

Connecting network visualisation to design specific methods

As discussed, the advantages of network visualisation are less clear with small datasets, such as small genetic designs, and existing visualisation methods, such as the SBOL glyphs approach; in future, it may be advantageous to explore how these two methods could be coupled. For example, if a user is exploring a design at a macro level, network visualisation is applied. However, when inspecting the specifics of an area within the design, the representation could be converted to a glyph representation. It provides a user with the best representation of the data they inspect.

Automatic scaling of detail

These network manipulations are only an initial effort into the many possibilities available[168] since network science[213] is an active field with applications in many disciplines, including life sciences[169, 149]. Currently, the processes to scale abstraction are designed for a specific aspect of the data genetic interactions or protein networks, for example. However, future efforts could explore how any aspect of the data could be expanded or retracted, creating a method for scaling abstraction to any point desired. For example, community detection [163] is partitioning the network into multiple communities, which may present a specific level of abstraction within certain projections.

4.4 Extending non-design visualisation

A key feature of network visualisation is the ability to capture and connect multiple data types. We described how the protocol and metabolic information could be visualised. However, these efforts were only a short example of how this data could be connected and visualised. In future, this work could be extended to:

- Integrate these data types fully such that the underlying dataset is captured in a standardised format.
- Create comprehensible representations that display how the different data types connect.
- Create visualisations specifically targeted to reduce complexity, especially when different types of information connect.

5 Conclusion

Here, we present a graph-based methodology for representing, analysing, and visualising circuit design information. Our approach transforms design files into networks, dynamic structures that can be automatically modified on demand according to user specifications. When molecular entities, relationships, and other information (e.g. types and roles) are encoded into nodes and edges using semantic labels, a network representation of a genetic design can be established. We have showcased the benefits of this approach by converting into networks the structural and functional data available within several genetic circuits. Specifically, we showed that design networks could be automatically adjusted to display different levels of detail, from full molecular representation to input/output information only and single-type graphs (e.g., protein interactions). The selection of abstraction as a metric to showcase the potential of networks is rooted in the intrinsic complexity of designs and the need to separate high-value information from superfluous details for a given purpose—thus improving understanding.

From a user perspective, this network visualisation approach provides many benefits beyond merely another method for visualising designs. Firstly, due to the complexity inherent in standard data formats, what a designer intends a genetic circuit to be versus what is captured can differ markedly. Network visualisation is a powerful tool for quickly conveying these discrepancies, enabling designers to verify and adjust their designs more effectively as they iterate on them.

Secondly, changes made during the design process can often be subtle and not immediately apparent; network visualisation provides a clear means to compare designs and highlight differences or modifications, ensuring that all changes are intentional and well-understood. Moreover, while individual parts of genetic designs are frequently reused, the overall connections and interactions within full designs are less commonly visualised. Displaying these connections through graphs can be incredibly beneficial, illustrating how commonly used components interact in various configurations. This is particularly useful for the original designers and others in the community who may be utilising shared designs. For someone who has not been involved in the initial development, network visualisation offers an intuitive way to comprehend the function and structure of a design, facilitating more straightforward modification, replication, or enhancement of existing circuits.

The robust nature of networks enables any information to be described. We displayed that designs can be compared to one another using a network approach by performing graph operations (union, intersection and difference) on design projections so these operations can even be applied to the human-friendly representations. A network approach to design representation and visualisation could be a key to fast comparisons of libraries of designs, tracking the lineage, that is, the change of a design over several iterations or identifying small changes to designs of high complexity.

The intrinsic modularity of networks allows for coupling genetic circuit designs to other data types, providing these are also represented in graphs. We have demonstrated this in two different ways. We showed that a genetic circuit that uses arabinose as input could be automatically coupled to the arabinose degradation pathway graph. By doing this, circuit designs can be extended to include information from their host context, improving the functional description of the device. Secondly, we have represented an implementation protocol in network format. While this is just a preliminary effort, which deserves further attention, it shows that protocol networks can also interact with circuit designs for the sake of building a data structure that can be shared along the design-build-test-learn[214] (DBTL) research cycle. In short, when data is represented as a graph, merging and clustering potentially disparate entities becomes a far less challenging task, and the graph could be the key to unifying data.

Designers should capture as much information as possible to generate high-quality and information-rich networks. Indeed, networks can only work with the provided data—networks cannot fabricate entirely new data, only derived from existing sources. While commonly used formats, such as GenBank, still capture information beyond genetic sequences, this information can be challenging to manage computationally due to the inherent informality and loose connections. Therefore, we advocate using knowledge graphs because more abstract questions can be formed, and more in-depth analyses can be made with the data. More specifically, a knowledge graph that implements the Synthetic Biology Open Language (SBOL) standard since it represents formal information, such as modularity or hierarchy, that cannot be captured otherwise.

As the complexity of genetic circuits increases, we advocate for networks to manipulate, analyse and communicate design information. We hope networks can maximise the efficiency of design automation procedures and help unification by providing standard[215] data structures for merged mathematical, genetic, protocol, and other prominent datasets established during synthetic biology projects.

Chapter 7: Conclusions and future work

1 Introduction

Implementing data standards in synthetic biology has been slow and has yielded limited success thus far. Traditional methods and tools to promote the usage of standards often require users to interface closely with the underlying structure, learn new languages decoupled from established domain languages or undertake additional manual steps to ensure that data conforms to standards. Moreover, introducing standards alone does not inherently resolve all issues associated with unstructured data; challenges such as reliability and redundancy persist. These issues were identified during a review of specification tools, datasets and their qualities, databases, how they store and serve data, and the techniques used to present the information. This research aimed to make the integration of standards into workflows more accessible, primarily by leveraging the power of networks, which allow for the modelling, manipulation, and analysis of complex systems. The topics presented in this thesis, encompassing ShortBOL, the weighted knowledge graph (WKG), and the network visualisation of genetic designs, collectively mark a significant stride forward in addressing these challenges. These tools are designed around robust standards, enhancing traceability and promoting interoperability across different systems and users in the field. This standard-backed approach ensures that every piece of data and every genetic design can be integrated and utilised across various platforms and projects, facilitating a more cohesive and collaborative research environment. Moreover, by leveraging the power of networks, these tools efficiently capture and unify complex biological data, offering a holistic view of genetic designs that is both comprehensive and comprehensible. This network-centric methodology significantly reduces the cognitive load on users, simplifying complex concepts and processes into more manageable and understandable components. It enables practitioners to easily navigate, manipulate, and derive insights from data, reducing the time and effort traditionally required in genetic design. Integrating network visualisation into this suite of tools further enhances this approach, providing dynamic and scalable visual representations of genetic data that enhance understanding and facilitate more informed decision-making. Together, these tools effectively tackle the challenges of standardisation and complexity in synthetic biology. They lay the groundwork for future developments in the field, offering methods that could lead to more efficient, accessible, and collaborative ways of working. The final chapter will revisit each research chapter's main discoveries and contributions relative to the motivation for this work. Additionally, the implications of the findings for current research practices will be reiterated, and suggestions for future work will be addressed.

2 Specifying design data by abstracting language

Chapter 3 explored how domain-specific languages (DSLs) directly mapped to the standard language (SBOL) can reduce the gap between standard and natural language. The outcome was

ShortBOL, an extensible DSL where libraries of custom entities can be defined. ShortBOL has two main advantages. Firstly, it enables the specification of genetic designs where users can create reusable abstract templates accessible to other users. Language within synthetic biology is essentially unconstrained, and many-many mappings may occur between terms, i.e. a word may have different definitions between different groups. Therefore, enforcing a universal standard language would be infeasible. Next, the template system allows for the language to be extended by any user. It has the potential for any number of terms to be mapped together, enabling the language to be set by the practitioners and increasing the likelihood of comprehension. The extensibility is achievable because ShortBOL is essentially a generated template library that can define arbitrary terms (providing it can be defined in RDF) and is not coupled directly to SBOL (instead, the SBOL is a specific template library). Also, ShortBOL bridges the gap between abstract methods, such as SBOL visual and verbose ones, such as programming implementations. The outcome is a language that can be comprehended easily but can still handle complexity (an issue that arises with the visual approaches) and compiles into SBOL a standard format. A potential criticism of ShortBOL is that the library developed simultaneously with the language, which is not truly a user-facing set of terms and, in some places, directly maps to an SBOL term. Therefore, one potential improvement of ShortBOL would be to invest further time into creating a user-facing set of terms (and capturing their meaning and interrelationships) used within the field instead of locally defined terminology. This future work can be divided into two projects: a social and technical challenge. The social challenge is to decide on a set of terms encompassing the largest language used within the community. Even if a set of terms that are more user-facing than the current ones are decided upon, the terms will invariably have a bias towards the creators and their backgrounds. For example, from a synthetic biology perspective, a gene may encompass a composite of promoter, RBS, CDS and terminator. However, others from a natural biology perspective may define a gene as the coding region and binding sites of transcription factors. Other definitions may include additional elements, such as enhancers, introns, and other regulatory sequences, which play roles in gene expression, regulation, and function. Although it is used to describe nucleotide-based sequences encoding some related information, the definition has changed over time as more discoveries are made—furthermore, the definition changes based on the specific domain of biology[216]. The gene is a single example of the differences in the nomenclature within biology, and while some things have specific agreed-upon names, others are abstract and open to interpretation. Therefore, the challenge is to identify an inclusive set of terms which will require the input of large groups of practitioners within the community, and the outcome may be a much larger set of terms that overlap, for example, words with multiple meanings or an inherent abstraction hierarchy of terms and grouped within other terms. Because the language is much broader and more complex, the set of terms decided upon will contain many nuances that may overlap or are contained within one another and will be more challenging to define than the relatively simple set of terms decided upon during language development. As a result, future technical efforts should focus on structuring these complex and nuanced terms into a well-defined and inclusive ontology. This ontology must encompass a comprehensive array of terms and their intricate relationships and hierarchies. This task is critical for accurately representing the language's richness and depth.

3 Enhancing design data using weighted knowledge graphs

The fourth chapter initially examined the ongoing challenges with existing data capture systems and their underlying data. Many issues were identified, namely a narrow range of datatypes and no guarantee of correctness. Therefore, weighted knowledge graphs were introduced to reduce uncertainty within synthetic biology design data by creating a centralised and canonical knowledge system. The weighted knowledge graphs, meta characteristics, metadata and integration strategies provide several advantages. Firstly, encoding functional information reduces perceived complexity, alignment with expected outcomes through defined functional modules, flexibility for iterative refinement, enabling a rational and systematic approach to designing biological systems, and allowing adjustments in detail levels to suit specific requirements. Also, the ability to quantify accuracy reduces the burden on the user to identify the correctness of a piece of information, which enhances trust in the data's reliability and facilitates more informed decision-making processes. A standardised network approach ensuring features such as canonicity enables the data to remain consistent, providing computational tractability so more focus can be applied to analysis instead of integration strategies. In conclusion, the weighted knowledge graph provides an environment that aims to reduce the burdens often put on practitioners or developers of tooling to quantify the information from databases before using it. A significant issue not addressed during this research is bridging the gap between ubiquitous handwritten informal storage methods and formal standards such as SBOL. Even if, in future, standards are employed in mass, legacy information previously informally captured must be either manually or automatically translated. However, automatically formalising natural language is challenging because it is ambiguous. For example, text can have multiple interpretations depending on the context, the meaning of a word or phrase can change based on context, and natural language can refer to entities that do not have specific dictionary definitions, such as genetic part names. Resolving this ambiguity requires a deep understanding of the vast amount of domain-specific language and the ability to infer meaning, and translating this contextual information into a formal system is challenging. During this research, formalising free text was outside of the scope. In cases where written information was used, it was formalised using established and simple methods such as fuzzy string matching. Some efforts, such as SBKS[136], are underway to tackle this challenge and are well-needed additions to standardisation within synthetic biology. From the establishment of the WKG, chapter 5 explored two use cases. The first was an advanced query system using the weighted knowledge graph to handle more complex queries and return more accurate results. This query system was achieved by exploiting the unique features of the weighted knowledge graph, namely, robust datatypes encoded using semantic labels, capture provenance, capture confidence, usage and functional information. This approach has several advantages over its contemporaries. Incorporating confidence metrics enables the promotion of information that is more likely to be correct, boosting trust in query results. Also, leveraging provenance data to identify analogous or identical entities allows entities to be clustered, aiding users in finding more relevant information. The ability to accommodate abstract query requirements allows users to search for information using broader, less specific criteria, expanding the scope of retrievable data. Integration of user feedback mechanisms enables ongoing dataset updates and result refinement, ensuring that the information retrieved remains

accurate and up-to-date and can utilise the greater knowledge of a community. The main limitation of this query system is the inability to handle written inputs properly. This limitation is an extension of the issue with the WKG and has used simple matching when a more complex natural language processing solution is necessary. Specifically, in this case, the ability to understand the context of a larger text block. For example, if the query input is aimed at searching for a repression system, it is crucial to understand which part of the query is the repressor and which is repressed. Chapter Five also displays how the knowledge graph can be used to automate the enhancement of existing genetic designs, thus reducing users' requirements to interface with the underlying standard. The enhancement takes advantage of the canonical feature of the weighted knowledge graph and the techniques used to create conceptually identical but syntactically different entities. Furthermore, once a match between the existing design and the weighted knowledge graph is found, the extra information encoded within the weighted knowledge graph is transferred to the design. Leveraging canonicity to normalise entities within the design network enables seamless data transfer and integration of information into existing designs without large amounts of manual work. Next, integrating functional data beyond raw sequence information streamlines the construction of designs, providing a higher level of abstraction crucial for advancing design methodologies. By correcting the absence of crucial data and introducing a system that streamlines the addition of explicit and comprehensive descriptions, the approach removes significant entry barriers in implementing standards, reducing the manual definition of intricate information and simplifying the perceived complexity of underlying data structures. The main limitation of this enhancement system, however, is its scope. For example, The modules introduced were relatively small by abstracting individual regulatory systems. Therefore, future work would be to expand the scope of this enhancement system to include the integration of larger and more complex modules, allowing for more comprehensive and sophisticated genetic designs. This work would involve increasing the variety and complexity of modules available in the knowledge graph. Doing so could further automate and streamline the design process, accommodating more advanced and intricate genetic constructs. Additionally, enhancing the system to better handle the complexities of multidimensional data and interdependencies within larger modules could significantly reduce the time and expertise required to develop functional genetic systems. In summary, synthetic biology's most valuable assets are its practitioners and their extensive knowledge. However, the current landscape of knowledge transfer underutilises these resources, leading to significant duplication of effort and wasted time. The introduction of the WKG offers a solution by enabling the sharing of specific information, including newly designed genetic parts and researchers' experiences. Encoding this knowledge enhances comprehension for users unfamiliar with the domain and facilitates problem-solving by leveraging solutions already identified by others, ultimately saving valuable time. Both use cases underscore a user-centric approach and emphasise the inefficiencies in current working methods, highlighting the potential for significant time and effort savings.

4 Tailored data representation by scaling complexity

The final chapter established a pipeline to visualise genetic designs by presenting several network representations in line with structures commonly seen within biological data. The key to the work was projections, which are individual processes to create subgraphs representing different aspects

of the data by focusing on representing a subset of the complete design, likely by creating a new user-facing network structure. Once the projections are established, many can be scaled in complexity by merging or expanding edges to further fit the level of detail in the representation to fit the user's needs. These choices and several aspects can be "mixed and matched" based on the requirements. Users can select the labels for nodes and connections relevant to the projection. Also, they can determine which algorithm should be used to calculate the positions of nodes on the canvas for layout purposes. There are also options to choose which categorical data is represented by the colours of the nodes and edges or what the sizes of nodes and edges represent, using them to encode continuous data. When combined, the network visualisation approach provides choices, which is often a limitation of other established visualisation techniques. Currently, the number of projections is limited, focusing solely on functional and structural visualisation. A future project may explore visualising a broader array of data types prevalent in genetic design. For instance, incorporating visualisations that portray gene expression levels under various conditions or illustrating epigenetic modifications could substantially deepen the comprehension of gene function and regulation. This future work could include the development of interactive visualisation that demonstrates the intricate interactions of genes within complex biological pathways or networks. Such models could also clarify the impact of environmental factors on gene expression, offering a more dynamic and comprehensive view of genetic processes. These enhancements would not only foster a better understanding of genetic mechanisms but also aid in developing more effective genetic engineering strategies. During this chapter, other data types, such as metabolic networks and experimental protocols, were visualised. However, these visualisations represented only preliminary efforts and have significant potential for expansion and refinement. For instance, future work could focus on creating more detailed and interactive visualisations that provide deeper insights into the interplay between the designed systems and the metabolic networks. Additionally, enhancing the visual representation of experimental protocols could involve adding layers of information, such as step-by-step procedural details, equipment used, and time frames for each step. Expanding this visualisation to illustrate the genetic design and the corresponding protocol for constructing the genetic construct is an ambitious yet promising direction. This dual visualisation would necessitate a comprehensive mapping system to link the genetic design elements with the specific steps and conditions outlined in the experimental protocols. Such an integrated visual tool would aid in better understanding the practical aspects of genetic engineering and facilitate more efficient planning and execution of experimental designs. This future work could be expanded to use a network approach as a unifier for all information within the development process. For example, the design, simulations, build protocol, experimental data, and performance measurements can all be captured as a network that can be visualised using representations that display how each information group maps together.

5 Conclusion

The tools and methodologies developed in this research were designed to overcome existing challenges in establishing standards within synthetic biology by developing solutions backed by a data standard. A pivotal outcome was the creation of ShortBOL, a novel language for specifying genetic designs, which emphasises natural language terms over more complex standard terminolo-

gies. This approach focused on an extensible framework that allowed ShortBOL to incorporate customisable templates and a level of abstraction that facilitated the creation of modular designs that could be easily adapted to various needs. Further enhancing the utility of these developments, the weighted knowledge graph emerged as a key asset. It functioned as a central repository, merging data from various sources into a unified, comprehensive dataset. This outcome not only offered a holistic view of different genetic entities beyond singular databases but also maintained the uniqueness of information through its emphasis on canonicity. Additionally, by encoding weights into the graph, the system could address and quantify the uncertainties often inherent in open-source data. This feature was particularly valuable for automating tasks based on the data model, such as refining genetic designs or providing reliable access to complex information. Regarding data representation, the research focused on network visualisation as an effective tool for depicting complex systems. This method differs from existing approaches that present only singular facets of genetic designs. Network visualisation enabled the projection of various sub-networks, including interaction networks or parts hierarchies, thus offering a more complete and integrated view of the genetic designs. However, this research also highlighted significant challenges that still impede the widespread adoption of standards in synthetic biology. One of the most notable issues is the difficulty converting vast amounts of written information into structured, standardised data. This gap remains a critical hurdle for standardisation within synthetic biology. In conclusion, the work presented here substantially contributes to mitigating these challenges and enhancing the appeal of standard adoption in synthetic biology. It supports the broader objectives of the field by fostering a community that can effectively share knowledge and reduce the burdens on practitioners. This research provides practical tools and methodologies, raises crucial questions, and identifies areas for future exploration, setting the stage for further advancements in the standardisation and effectiveness of synthetic biology practices.

Acronyms

API	Application Programming Interface
AraC	Arabinose Repressor
BBa	BioBrick Part
CAD	Computer-aided Design
CDS	Coding Sequence
CRISPR	Clustered Regularly Interspaced Short Palindromic Repeats
CSV	Comma-Separated Values
DBTL	Design, Build, Test, and Learn
DSL	Domain-Specific Language
ENA	European Nucleotide Archive
EPU	Entrez Programming Utilities
FTP	File Transfer Protocol
GEC	Genetic Engineering of Cells
Genbank	Genetic Sequence Database
GRN	Gene Regulatory Network
ICE	Inventory of Composable Elements
IDE	Integrated Development Environment
IGEM	International Genetically Engineered Machine
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
KEGG	Kyoto Encyclopedia of Genes and Genomes
LacI	Lactose Repressor
LCP	Living Computer Project
libSBOLj	libSBOL Java Library
NCBI	National Center for Biotechnology Information
PDF	Portable Document Format
PFIN	Probabilistic Functional Integrated Networks
PPI	Protein-Protein Interaction Network
PROV-O	Provenance Ontology
pTet	Tetracycline Promoter
pySBOL	Python SBOL Library
RCS	Reactome Content Service
RDF	Resource Description Framework
RNA	Ribonucleic Acid
SB	Synthetic Biology
SBML	Systems Biology Markup Language

SBO	Systems Biology Ontology
SBOL	Synthetic Biology Open Language
SEVA	Standard European Vector Architecture
ShortBOL	ShortBOL
SPARQL	SPARQL Protocol and RDF Query Language
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
VPR	Virtual Parts Repository
WCC	Weakly Connected Components
WKG	Weighted Knowledge Graph
XML	eXtensible Markup Language

Bibliography

1. Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "A Network Approach to Genetic Circuit Designs". *ACS Synthetic Biology* 11:9, 2022. PMID: 36044984, pp. 3058–3066. DOI: 10.1021/acssynbio.2c00255. eprint: https://doi.org/10.1021/acssynbio.2c00255. URL: https://doi.org/10.1021/acssynbio.2c00255.
2. Matthew Crowther et al. "ShortBOL: A Language for Scripting Designs for Engineered Biological Systems Using Synthetic Biology Open Language (SBOL)". *ACS Synthetic Biology* 9:4, 2020. PMID: 32129980, pp. 962–966. DOI: 10.1021/acssynbio.9b00470. eprint: https://doi.org/10.1021/acssynbio.9b00470. URL: https://doi.org/10.1021/acssynbio.9b00470.
3. Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno. "GENETTA: a Network-Based Tool for the Analysis of Complex Genetic Designs". *ACS Synthetic Biology*, 2023.
4. Hasan Baig et al. "Synthetic biology open language (SBOL) version 3.0.0". *Journal of Integrative Bioinformatics* 17:2-3, 2020, p. 20200017. DOI: doi:10.1515/jib-2020-0017. URL: https://doi.org/10.1515/jib-2020-0017.
5. Esteban Martínez-García et al. "SEVA 4.0: an update of the Standard European Vector Architecture database for advanced analysis and programming of bacterial phenotypes". *Nucleic Acids Research* 51:D1, 2022, pp. D1558–D1567. ISSN: 0305-1048. DOI: 10.1093/nar/gkac1059. eprint: https://academic.oup.com/nar/article-pdf/51/D1/D1558/48441442/gkac1059.pdf. URL: https://doi.org/10.1093/nar/gkac1059.
6. Shankar Mukherji and Alexander Oudenaarden. "Synthetic biology: Understanding biological design from synthetic circuits". *Nature reviews. Genetics* 10, 2009, pp. 859–71. DOI: 10.1038/nrg2697.
7. Christina M Agapakis. "Designing synthetic biology". *ACS synthetic biology* 3:3, 2014, pp. 121–128.
8. Christopher Langton. *Artificial life: proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems*. Routledge, 2019.
9. Ibrahim Aldulijan et al. "Functional synthetic biology". *Synthetic Biology* 8:1, 2023, ysad006.
10. Curtis Madsen et al. "Synthetic biology open language (SBOL) version 2.3". *Journal of integrative bioinformatics* 16:2, 2019.
11. George Church et al. "Realizing the potential of synthetic biology". *Nature reviews. Molecular cell biology* 15, 2014. DOI: 10.1038/nrm3767.
12. Priscilla Purnick and Ron Weiss. "The second wave of synthetic biology: From modules to systems". *Nature reviews. Molecular cell biology* 10, 2009, pp. 410–22. DOI: 10.1038/nrm2698.

13. Ernesto Andrianantoandro et al. "Synthetic biology: new engineering rules for an emerging discipline". *Molecular Systems Biology* 2, 2006, pp. 2006.0028–2006.0028.
14. Yaakov Benenson. "Biomolecular computing systems: principles, progress and potential". *Nature Reviews Genetics* 13:7, 2012, pp. 455–468.
15. Jennifer AN Brophy and Christopher A Voigt. "Principles of genetic circuit design". *Nature methods* 11:5, 2014, pp. 508–520.
16. Martyn Amos and Angel Goni-Moreno. "Cellular Computing and Synthetic Biology". In: *Computational Matter*. Springer, 2018, pp. 93–110.
17. Simon Ausländer, David Ausländer, and Martin Fussenegger. "Synthetic biology—the synthesis of biology". *Angewandte Chemie International Edition* 56:23, 2017, pp. 6396–6419.
18. Ernesto Andrianantoandro et al. "Synthetic biology: new engineering rules for an emerging discipline". *Molecular systems biology* 2:1, 2006, pp. 2006–0028.
19. Alec AK Nielsen et al. "Genetic circuit design automation". *Science* 352:6281, 2016, aac7341.
20. Chunbo Lou et al. "Synthesizing a novel genetic sequential logic circuit: a push-on push-off switch". *Molecular systems biology* 6:1, 2010, p. 350.
21. Ari E Friedland et al. "Synthetic gene networks that count". *science* 324:5931, 2009, pp. 1199–1202.
22. Seth L Shipman et al. "Molecular recordings by directed CRISPR spacer acquisition". *Science* 353:6298, 2016, aaf1175.
23. Shunsuke Kawasaki et al. "RNA and protein-based nanodevices for mammalian post-transcriptional circuits". *Current Opinion in Biotechnology* 63, 2020, pp. 99–110.
24. Adison Wong et al. "Layering genetic circuits to build a single cell, bacterial half adder". *BMC biology* 13:1, 2015, pp. 1–16.
25. Ye Chen et al. "Genetic circuit design automation for yeast". *Nature Microbiology* 5:11, 2020, pp. 1349–1360.
26. Gabriele Lillacci, Yaakov Benenson, and Mustafa Khammash. "Synthetic control systems for high performance gene expression in mammalian cells". *Nucleic acids research* 46:18, 2018, pp. 9855–9863.
27. Fankang Meng and Tom Ellis. "The second decade of synthetic biology: 2010–2020". *Nature Communications* 11:1, 2020, pp. 1–4.
28. Victor De Lorenzo et al. "The power of synthetic biology for bioproduction, remediation and pollution control: the UN's Sustainable Development Goals will inevitably require the application of molecular biology and biotechnology on a global scale". *EMBO reports* 19:4, 2018, e45658.
29. Shimyn Slomovic, Keith Pardee, and James J Collins. "Synthetic biology devices for in vitro and in vivo diagnostics". *Proceedings of the National Academy of Sciences* 112:47, 2015, pp. 14429–14435.

30. Ángel Goñi-Moreno et al. "Deconvolution of gene expression noise into spatial dynamics of transcription factor–promoter interplay". *ACS synthetic biology* 6:7, 2017, pp. 1359–1369.
31. Avigdor Eldar and Michael B Elowitz. "Functional roles for noise in genetic circuits". *Nature* 467:7312, 2010, pp. 167–173.
32. Felix Moser et al. "Dynamic control of endogenous metabolism with combinatorial logic circuits". *Molecular systems biology* 14:11, 2018, e8605.
33. Angel Goñi-Moreno and Pablo I Nikel. "High-performance biocomputing in synthetic biology–integrated transcriptional and metabolic circuits". *Frontiers in bioengineering and biotechnology* 7, 2019, p. 40.
34. Huseyin Tas et al. "Contextual dependencies expand the re-usability of genetic inverters". *Nature communications* 12:1, 2021, pp. 1–9.
35. Alice Boo, Tom Ellis, and Guy-Bart Stan. "Host-aware synthetic biology". *Current Opinion in Systems Biology* 14, 2019, pp. 66–72.
36. Ronghui Zhu et al. "Synthetic multistability in mammalian cells". *Science* 375:6578, 2021, eabg9765.
37. Lewis Grozinger et al. "Pathways to cellular supremacy in biocomputing". *Nature communications* 10:1, 2019, p. 5250.
38. Paul S. Freemont. "Synthetic biology industry: data-driven design is creating new opportunities in biotechnology". *Emerging Topics in Life Sciences* 3:5, 2019, pp. 651–657. ISSN: 2397-8554. DOI: 10.1042/ETLS20190040. eprint: https://portlandpress.com/emergtoplifesci/article-pdf/3/5/651/869674/etls-2019-0040c.pdf. URL: https://doi.org/10.1042/ETLS20190040.
39. Jacob Beal et al. "Communicating structure and function in synthetic biology diagrams". *ACS synthetic biology* 8:8, 2019, pp. 1818–1825.
40. Yiannis N Kaznessis. "Models for synthetic biology". *BMC systems biology* 1, 2007, pp. 1–4.
41. Linda J Kahl and Drew Endy. "A survey of enabling technologies in synthetic biology". *Journal of biological engineering* 7, 2013, pp. 1–19.
42. Richard Kitney and Paul Freemont. "Synthetic biology–the state of play". *FEBS letters* 586:15, 2012, pp. 2029–2036.
43. Jianzhi Zhang et al. "Accelerating strain engineering in biofuel research via build and test automation of synthetic biology". *Current Opinion in Biotechnology* 67, 2021, pp. 88–98. ISSN: 0958-1669. DOI: https://doi.org/10.1016/j.copbio.2021.01.010. URL: https://www.sciencedirect.com/science/article/pii/S095816692100015X.
44. Benjamin Pouvreau, Thomas Vanhercke, and Surinder Singh. "From plant metabolic engineering to plant synthetic biology: The evolution of the design/build/test/learn cycle". *Plant Science* 273, 2018. Synthetic Biology Meets Plant Metabolism, pp. 3–12. ISSN: 0168-9452. DOI: https://doi.org/10.1016/j.plantsci.2018.03.035. URL: https://www.sciencedirect.com/science/article/pii/S0168945217311809.

45. Albert-László Barabási and Márton Pósfai. *Network science*. Cambridge University Press, Cambridge, 2016. URL: http://barabasi.com/networksciencebook/.

46. Lothar Krempel. "Network visualization". *The SAGE handbook of social network analysis*, 2011, pp. 558–577.

47. Jingwen Yan et al. "Network approaches to systems biology analysis of complex disease: integrative methods for multi-omics data". *Briefings in Bioinformatics* 19:6, 2017, pp. 1370–1381. ISSN: 1477-4054. DOI: 10.1093/bib/bbx066. eprint: https://academic.oup.com/bib/article-pdf/19/6/1370/27119423/bbx066.pdf. URL: https://doi.org/10.1093/bib/bbx066.

48. Mathieu Bouchard, Anne-Laure Jousselme, and Pierre-Emmanuel Doré. "A proof for the positive definiteness of the Jaccard index matrix". *International Journal of Approximate Reasoning* 54:5, 2013, pp. 615–626.

49. MK Vijaymeena and K Kavitha. "A survey on similarity measures in text mining". *Machine Learning and Applications: An International Journal* 3:2, 2016, pp. 19–28.

50. Bruce Golden. "Shortest-path algorithms: A comparison". *Operations Research* 24:6, 1976, pp. 1164–1168.

51. Stephen P. Borgatti. "Centrality and network flow". *Social Networks* 27:1, 2005, pp. 55–71. ISSN: 0378-8733. DOI: https://doi.org/10.1016/j.socnet.2004.11.008. URL: https://www.sciencedirect.com/science/article/pii/S0378873304000693.

52. William JR Longabaugh, Eric H Davidson, and Hamid Bolouri. "Computational representation of developmental genetic regulatory networks". *Developmental biology* 283:1, 2005, pp. 1–16.

53. Michael Hecker et al. "Gene regulatory network inference: data integration in dynamic models—a review". *Biosystems* 96:1, 2009, pp. 86–103.

54. Daniel R Rhodes et al. "Probabilistic model of the human protein-protein interaction network". *Nature biotechnology* 23:8, 2005, pp. 951–959.

55. Andrea Franceschini et al. "STRING v9. 1: protein-protein interaction networks, with increased coverage and integration". *Nucleic acids research* 41:D1, 2012, pp. D808–D815.

56. Alexiou Athanasios et al. "Protein-protein interaction (PPI) network: recent advances in drug discovery". *Current drug metabolism* 18:1, 2017, pp. 5–10.

57. Yoichi Murakami et al. "Network analysis and in silico prediction of protein–protein interactions with applications in drug discovery". *Current opinion in structural biology* 44, 2017, pp. 134–142.

58. Suresh Kumar. "COVID-19: A drug repurposing and biomarker identification by using comprehensive gene-disease associations through protein-protein interaction network analysis", 2020.

59. Alicia L Richards, Manon Eckhardt, and Nevan J Krogan. "Mass spectrometry-based protein–protein interaction networks for the study of human diseases". *Molecular systems biology* 17:1, 2021, e8792.

60. Sepideh Sadegh et al. "Network medicine for disease module identification and drug repurposing with the NeDRex platform". *Nature Communications* 12:1, 2021, p. 6848.

61. Insuk Lee et al. "A probabilistic functional network of yeast genes". *science* 306:5701, 2004, pp. 1555–1558.

62. Katherine James, Anil Wipat, and Jennifer Hallinan. "Integration of full-coverage probabilistic functional networks with relevance to specific biological processes". In: *Data Integration in the Life Sciences: 6th International Workshop, DILS 2009, Manchester, UK, July 20-22, 2009. Proceedings 6*. Springer. 2009, pp. 31–46.

63. Vincent Lacroix et al. "An introduction to metabolic networks and their structural analysis". *IEEE/ACM transactions on computational biology and bioinformatics* 5:4, 2008, pp. 594–617.

64. Oliver Fiehn and Wolfram Weckwerth. "Deciphering metabolic networks". *European Journal of Biochemistry* 270:4, 2003, pp. 579–588.

65. Maciek R Antoniewicz. "Dynamic metabolic flux analysis—tools for probing transient states of metabolic networks". *Current opinion in biotechnology* 24:6, 2013, pp. 973–978.

66. Lee J Sweetlove and Alisdair R Fernie. "Regulation of metabolic networks: understanding metabolic complexity in the systems biology era". *New Phytologist* 168:1, 2005, pp. 9–24.

67. Douglas E Biancur et al. "Compensatory metabolic networks in pancreatic cancers upon perturbation of glutamine metabolism". *Nature communications* 8:1, 2017, p. 15965.

68. URL: https://autoprotocol.org/.

69. Felipe Pezoa et al. "Foundations of JSON schema". In: *Proceedings of the 25th international conference on World Wide Web*. 2016, pp. 263–273.

70. Gabriele Gramelsberger. "The simulation approach in synthetic biology". *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences* 44:2, 2013, pp. 150–157.

71. Michael Hucka et al. "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models". *Bioinformatics* 19:4, 2003, pp. 524–531.

72. Fei He, Ettore Murabito, and Hans V Westerhoff. "Synthetic biology and regulatory networks: where metabolic systems biology meets control engineering". *Journal of The Royal Society Interface* 13:117, 2016, p. 20151046.

73. Roberta Kwok. "Five hard truths for synthetic biology: can engineering approaches tame the complexity of living systems? Roberta Kwok explores five challenges for the field and how they might be resolved". *Nature* 463:7279, 2010, pp. 288–291.

74. Chris J Myers. *Engineering genetic circuits*. CRC Press, 2016.

75. Adrian L Slusarczyk, Allen Lin, and Ron Weiss. "Foundations for the design and implementation of synthetic genetic circuits". *Nature Reviews Genetics* 13:6, 2012, pp. 406–420.

76. Thomas E Gorochowski. "Agent-based modelling in synthetic biology". *Essays in biochemistry* 60:4, 2016, pp. 325–336.

77. Yuting Zheng and Ganesh Sriram. "Mathematical modeling: bridging the gap between concept and realization in synthetic biology". *Journal of Biomedicine and Biotechnology* 2010, 2010.

78. Stéphanie Rialle et al. "BioNetCAD: design, simulation and experimental validation of synthetic biochemical networks". *Bioinformatics* 26:18, 2010, pp. 2298–2304.

79. Zach Zundel et al. "A validator and converter for the synthetic biology open language". *ACS Synthetic Biology* 6:7, 2017, pp. 1161–1168.

80. Hiroaki Kitano. "Computational systems biology". *Nature* 420:6912, 2002, pp. 206–210.

81. Goksel Misirli. "Data integration strategies for informing computational design in synthetic biology". PhD thesis. Newcastle University, 2013.

82. Anže Verbič, Arne Praznik, and Roman Jerala. "A guide to the design of synthetic gene networks in mammalian cells". *The FEBS Journal* 288:18, 2021, pp. 5265–5288.

83. Evan Appleton et al. "Design automation in synthetic biology". *Cold Spring Harbor perspectives in biology* 9:4, 2017, a023978.

84. Lachlan J Munro and Douglas B Kell. "Intelligent host engineering for metabolic flux optimisation in biotechnology". *Biochemical Journal* 478:20, 2021, pp. 3685–3721.

85. Jürgen Pleiss. "The promise of synthetic biology". *Applied microbiology and biotechnology* 73, 2006, pp. 735–739.

86. Michal Galdzicki et al. "The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology". *Nature biotechnology* 32:6, 2014, pp. 545–550.

87. Elizabeth HC Bromley et al. "Peptide and protein building blocks for synthetic biology: from programming biomolecules to self-organized biomolecular systems". *ACS chemical biology* 3:1, 2008, pp. 38–50.

88. Deepak Chandran, Frank T Bergmann, and Herbert M Sauro. "TinkerCell: modular CAD tool for synthetic biology". *Journal of biological engineering* 3:1, 2009, pp. 1–17.

89. Sascha Rollié, Michael Mangold, and Kai Sundmacher. "Designing biological systems: systems engineering meets synthetic biology". *Chemical Engineering Science* 69:1, 2012, pp. 1–29.

90. Gregory Stephanopoulos. "Synthetic biology and metabolic engineering". *ACS synthetic biology* 1:11, 2012, pp. 514–525.

91. Michael M Kämpf and Wilfried Weber. "Synthetic biology in the analysis and engineering of signaling processes". *Integrative Biology* 2:1, 2010, pp. 12–24.

92. Shankar Mukherji and Alexander Van Oudenaarden. "Synthetic biology: understanding biological design from synthetic circuits". *Nature Reviews Genetics* 10:12, 2009, pp. 859–871.

93. Aidan Hogan et al. "Knowledge graphs". *ACM Computing Surveys (CSUR)* 54:4, 2021, pp. 1–37.
94. Sameh K Mohamed, Aayah Nounu, and Vít Nováček. "Biological applications of knowledge graph embedding models". *Briefings in bioinformatics* 22:2, 2021, pp. 1679–1693.
95. Eric Miller. "An introduction to the resource description framework." *D-lib Magazine*, 1998.
96. Mike Uschold and Michael Gruninger. "Ontologies: Principles, methods and applications". *The knowledge engineering review* 11:2, 1996, pp. 93–136.
97. Balakrishnan Chandrasekaran, John R Josephson, and V Richard Benjamins. "What are ontologies, and why do we need them?" *IEEE Intelligent Systems and their applications* 14:1, 1999, pp. 20–26.
98. Steffen Staab and Rudi Studer. *Handbook on ontologies*. Springer Science & Business Media, 2010.
99. Pasquale Pagano, Leonardo Candela, and Donatella Castelli. "Data interoperability". *Data Science Journal* 12, 2013, GRDI19–GRDI25.
100. Jieying Chen et al. "Ontology extraction for large ontologies via modularity and forgetting". In: *Proceedings of the 10th International Conference on Knowledge Capture*. 2019, pp. 45–52.
101. Ran Chao et al. "Engineering biological systems using automated biofoundries". *Metabolic Engineering* 42, 2017, pp. 98–108.
102. Zhen Zhang et al. "libSBOLj 2.0: A Java Library to Support SBOL 2.0". *IEEE Life Sciences Letters* 1:4, 2015, pp. 34–37. DOI: 10.1109/LLS.2016.2546546.
103. Bryan A. Bartley et al. "pySBOL: A Python Package for Genetic Design Automation and Standardization". *ACS Synthetic Biology* 8:7, 2019. PMID: 30424601, pp. 1515–1518. DOI: 10.1021/acssynbio.8b00336. eprint: https://doi.org/10.1021/acssynbio.8b00336. URL: https://doi.org/10.1021/acssynbio.8b00336.
104. Maureen A. O?Malley. "Making Knowledge in Synthetic Biology: Design Meets Kludge". *Biological Theory* 4:4, 2009, pp. 378–389. DOI: 10.1162/biot_{a}_00006.
105. Peter McMahan and James Evans. "Ambiguity and Engagement". *American Journal of Sociology* 124:3, 2018, pp. 860–912. DOI: 10.1086/701298. eprint: https://doi.org/10.1086/701298. URL: https://doi.org/10.1086/701298.
106. Federica Ciocchetta and Jane Hillston. "Bio-PEPA: A framework for the modelling and analysis of biological systems". *Theoretical Computer Science* 410:33, 2009. Concurrent Systems Biology: To Nadia Busi (1968–2007), pp. 3065–3084. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2009.02.037. URL: https://www.sciencedirect.com/science/article/pii/S0304397509001662.
107. Rosara Lakin et al. "Visual DSD: A design and analysis tool for DNA strand displacement systems". *Bioinformatics (Oxford, England)* 27, 2011, pp. 3211–3. DOI: 10.1093/bioinformatics/btr543.

108. Michael Zhang et al. "SBOLDesigner 2: An Intuitive Tool for Structural Genetic Design". *ACS Synthetic Biology* 6:7, 2017. PMID: 28441476, pp. 1150–1160. DOI: 10.1021/acssynbio.6b00275. eprint: https://doi.org/10.1021/acssynbio.6b00275. URL: https://doi.org/10.1021/acssynbio.6b00275.

109. James Alastair McLaughlin et al. "VisBOL: Web-Based Tools for Synthetic Biology Design Visualization". *ACS Synthetic Biology* 5:8, 2016. PMID: 26808703, pp. 874–876. DOI: 10.1021/acssynbio.5b00244. eprint: https://doi.org/10.1021/acssynbio.5b00244. URL: https://doi.org/10.1021/acssynbio.5b00244.

110. Logan Terry et al. "SBOLCanvas: A Visual Editor for Genetic Designs". *ACS Synthetic Biology*, 2021. DOI: 10.1021/acssynbio.1c00096.

111. *Benchling*. https://www.benchling.com/. Accessed: 25/07/2023.

112. Dennis M. Ritchie. "The Development of the C Programming Language". In: *History of Programming Languages—II*. Association for Computing Machinery, New York, NY, USA, 1996, pp. 671–698. ISBN: 0201895021. URL: https://doi.org/10.1145/234286.1057834.

113. Erin H Wilson et al. "Genotype specification language". *ACS Synthetic Biology* 5:6, 2016, pp. 471–478.

114. Lesia Bilitchenko, Adam Liu, and Douglas Densmore. "The Eugene language for synthetic biology". In: *Methods in enzymology*. Vol. 498. Elsevier, 2011, pp. 153–172.

115. Michael Pedersen and Andrew Phillips. "Towards programming languages for genetic engineering of living cells". *Journal of the Royal Society, Interface / the Royal Society* 6 Suppl 4, 2009, S437–50. DOI: 10.1098/rsif.2008.0516.focus.

116. James Alastair McLaughlin et al. "sboljs: Bringing the Synthetic Biology Open Language to the Web Browser". *ACS Synthetic Biology* 8:1, 2019, pp. 191–193. DOI: 10.1021/acssynbio.8b00338. eprint: https://doi.org/10.1021/acssynbio.8b00338. URL: https://doi.org/10.1021/acssynbio.8b00338.

117. Zhen Zhang et al. "LibSBOLj 2.0: A Java library to support SBOL 2.0". *IEEE Life Sciences Letters* 1, 2016, pp. 1–1. DOI: 10.1109/LLS.2016.2546546.

118. Iyad Zayour and Hassan Hajjdiab. "How much integrated development environments (ides) improve productivity?" *J. Softw.* 8:10, 2013, pp. 2425–2431.

119. Domitilla Del Vecchio. "Modularity, context-dependence, and insulation in engineered biological circuits". *Trends in biotechnology* 33:2, 2015, pp. 111–119.

120. Thomas Decoene et al. "Standardization in synthetic biology: an engineering discipline coming of age". *Critical reviews in biotechnology* 38:5, 2018, pp. 647–656.

121. Uriel Urquiza-Garcıéa, Tomasz Zieliński, and Andrew J Millar. "Better research by efficient sharing: evaluation of free management platforms for synthetic biology designs". *Synthetic Biology* 4:1, 2019, ysz016.

122. James Alastair McLaughlin et al. "SynBioHub: A Standards-Enabled Design Repository for Synthetic Biology". *ACS Synthetic Biology* 7:2, 2018. PMID: 29316788, pp. 682–688. DOI: 10.1021/acssynbio.7b00403. eprint: https://doi.org/10.1021/acssynbio.7b00403. URL: https://doi.org/10.1021/acssynbio.7b00403.

123. James Brown. "The iGEM competition: building with biology". *IET Synthetic Biology* 1:1, 2007, pp. 3–6.

124. Göksel Mısırlı et al. "A Computational Workflow for the Automated Generation of Models of Genetic Designs". *ACS Synthetic Biology* 8, 2018. DOI: 10.1021/acssynbio.7b00459.

125. Timothy S Ham et al. "Design, implementation and practice of JBEI-ICE: an open source biological part registry platform and tools". *Nucleic acids research* 40:18, 2012, e141–e141.

126. Joanne Kamens. "The Addgene repository: an international nonprofit plasmid and data resource". *Nucleic Acids Research* 43:D1, 2015, pp. D1152–D1157.

127. Minoru Kanehisa. "The KEGG database". In: *'In silico'simulation of biological processes: Novartis Foundation Symposium 247*. Vol. 247. Wiley Online Library. 2002, pp. 91–103.

128. Göksel Mısırlı et al. "Virtual Parts Repository 2: Model-driven design of genetic regulatory circuits". *ACS Synthetic Biology* 10:12, 2021, pp. 3304–3315.

129. Bettina Berendt et al. "A roadmap for web mining: From web to semantic web". In: *Web Mining: From Web to Semantic Web: First European Web Mining Forum, EWMF 2003, Cavtat-Dubrovnik, Croatia, September 22, 2003, Invited and Selected Revised Papers*. Springer. 2004, pp. 1–22.

130. Timothy Lebo et al. "Prov-o: The prov ontology". *W3C recommendation* 30, 2013.

131. Dennis A Benson et al. "GenBank". *Nucleic acids research* 41:D1, 2012, pp. D36–D42.

132. Hasan Baig et al. "Synthetic biology open language visual (SBOL visual) version 2.2". *Journal of Integrative Bioinformatics* 1:ahead-of-print, 2020.

133. James Alastair McLaughlin et al. "SynBioHub: a standards-enabled design repository for synthetic biology". *ACS synthetic biology* 7:2, 2018, pp. 682–688.

134. Chris J Myers et al. "A standard-enabled workflow for synthetic biology". *Biochemical Society Transactions* 45:3, 2017, pp. 793–803.

135. Christina D Smolke. "Building outside of the box: iGEM and the BioBricks Foundation". *Nature biotechnology* 27:12, 2009, pp. 1099–1102.

136. Jeanet Mante et al. "Synthetic Biology Knowledge System". *ACS synthetic biology* 10:9, 2021, pp. 2276–2285.

137. Mathew M Jessop-Fabre and Nikolaus Sonnenschein. "Improving reproducibility in synthetic biology". *Frontiers in bioengineering and biotechnology* 7, 2019, p. 18.

138. Marc P Raphael, Paul E Sheehan, and Gary J Vora. "A controlled trial for reproducibility". *Nature* 579:7798, 2020, pp. 190–192.

139. Ryan McDaniel and Ron Weiss. "Advances in synthetic biology: on the path from prototypes to applications". *Current opinion in biotechnology* 16:4, 2005, pp. 476–483.

140. Gil Alterovitz, Taro Muso, and Marco F Ramoni. "The challenges of informatics in synthetic biology: from biomolecular networks to artificial organisms". *Briefings in bioinformatics* 11:1, 2010, pp. 80–95.

141. Ernesto Andrianantoandro et al. "Synthetic biology: new engineering rules for an emerging discipline". *Molecular Systems Biology* 2:1, 2006, p. 2006.0028. DOI: https://doi.org/10.1038/msb4100073. eprint: https://www.embopress.org/doi/pdf/10.1038/msb4100073. URL: https://www.embopress.org/doi/abs/10.1038/msb4100073.

142. Steven A Benner and A Michael Sismour. "Synthetic biology". *Nature reviews genetics* 6:7, 2005, pp. 533–543.

143. Mario A Marchisio and Jörg Stelling. "Computational design tools for synthetic biology". *Current opinion in biotechnology* 20:4, 2009, pp. 479–485.

144. Ibrahim Aldulijan et al. "Functional Synthetic Biology". *Synthetic Biology* 8:1, 2023, ysad006. ISSN: 2397-7000. DOI: 10.1093/synbio/ysad006. eprint: https://academic.oup.com/synbio/article-pdf/8/1/ysad006/49937590/ysad006.pdf. URL: https://doi.org/10.1093/synbio/ysad006.

145. Priscilla EM Purnick and Ron Weiss. "The second wave of synthetic biology: from modules to systems". *Nature reviews Molecular cell biology* 10:6, 2009, pp. 410–422.

146. Matthias Heinemann and Sven Panke. "Synthetic biology—putting engineering into biology". *Bioinformatics* 22:22, 2006, pp. 2790–2799.

147. Niel Chah. "OK Google, What Is Your Ontology? Or: Exploring Freebase Classification to Understand Google's Knowledge Graph". *ArXiv* abs/1805.03885, 2018.

148. Alain Barrat et al. "The architecture of complex weighted networks". *Proceedings of the national academy of sciences* 101:11, 2004, pp. 3747–3752.

149. Albert-Laszlo Barabasi and Zoltan N Oltvai. "Network biology: understanding the cell's functional organization". *Nature reviews genetics* 5:2, 2004, pp. 101–113.

150. Debajyoti Mukhopadhyay, Debasis Giri, and Sanasam Ranbir Singh. "An approach to confidence based page ranking for user oriented web search". *ACM SIGMOD Record* 32:2, 2003, pp. 28–33.

151. Daniel W Margo and Margo I Seltzer. "The Case for Browser Provenance." In: *Workshop on the Theory and Practice of Provenance*. 2009.

152. Jacob Beal et al. "An end-to-end workflow for engineering of biological networks from high-level specifications". *ACS Synthetic Biology* 1:8, 2012, pp. 317–331.

153. Albert-László Barabási and Eric Bonabeau. "Scale-free networks". *Scientific american* 288:5, 2003, pp. 60–69.

154. Timothy S Jones et al. "Genetic circuit design automation with Cello 2.0". *Nature protocols* 17:4, 2022, pp. 1097–1113.

155. Trey Ideker and Nevan J Krogan. "Differential network biology". *Molecular systems biology* 8:1, 2012, p. 565.

156. Jin-Hee Cho et al. "Uncertainty-based false information propagation in social networks". *ACM Transactions on Social Computing* 2:2, 2019, pp. 1–34.

157. Tomas Fencl, Pavel Burget, and Jan Bilek. "Network topology design". *Control Engineering Practice* 19:11, 2011, pp. 1287–1296.

158. Arvind Kumar Yadav, Rohit Shukla, and Tiratha Raj Singh. "Chapter 22 - Topological parameters, patterns, and motifs in biological networks". In: *Bioinformatics*. Ed. by Dev Bukhsh Singh and Rajesh Kumar Pathak. Academic Press, 2022, pp. 367–380. ISBN: 978-0-323-89775-4. DOI: https://doi.org/10.1016/B978-0-323-89775-4.00012-2. URL: https://www.sciencedirect.com/science/article/pii/B9780323897754000122.

159. Ron Milo et al. "Network motifs: simple building blocks of complex networks". *Science* 298:5594, 2002, pp. 824–827.

160. Alvin Tamsir, Jeffrey J Tabor, and Christopher A Voigt. "Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'". *Nature* 469:7329, 2011, pp. 212–215.

161. Luis Serrano. *Synthetic biology: promises and challenges*. 2007.

162. Yunming Zhang et al. "Optimizing ordered graph algorithms with graphit". In: *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*. 2020, pp. 158–170.

163. Santo Fortunato. "Community detection in graphs". *Physics reports* 486:3-5, 2010, pp. 75–174.

164. Richard J Anderson and Heather Woll. "Wait-free parallel algorithms for the union-find problem". In: *Proceedings of the twenty-third annual ACM symposium on Theory of computing*. 1991, pp. 370–380.

165. Michael Sutton, Tal Ben-Nun, and Amnon Barak. "Optimizing Parallel Graph Connectivity Computation via Subgraph Sampling". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2018, pp. 12–21. DOI: 10.1109/IPDPS.2018.00012.

166. JV Mante. "Promotion of Data Reuse in Synthetic Biology". PhD thesis. University of Colorado at Boulder, 2022.

167. Chengwei Lei and Jianhua Ruan. "A novel link prediction algorithm for reconstructing protein–protein interaction networks by topological similarity". *Bioinformatics* 29:3, 2013, pp. 355–364.

168. Marko Gosak et al. "Network science of biological systems at different scales: A review". *Physics of life reviews* 24, 2018, pp. 118–135.

169. Erzsébet Ravasz et al. "Hierarchical organization of modularity in metabolic networks". *science* 297:5586, 2002, pp. 1551–1555.

170. D Ewen Cameron, Caleb J Bashor, and James J Collins. "A brief history of synthetic biology". *Nature Reviews Microbiology* 12:5, 2014, pp. 381–390.

171. Sergi Valverde et al. "The software crisis of synthetic biology". *BioRxiv*, 2016, p. 041640.

172. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. "Semantics and complexity of SPARQL". *ACM Transactions on Database Systems (TODS)* 34:3, 2009, pp. 1–45.

173. Uriel Urquiza-García, Tomasz Zieliński, and Andrew J Millar. "Better research by efficient sharing: evaluation of free management platforms for synthetic biology designs". *Synthetic Biology* 4:1, 2019. ysz016. ISSN: 2397-7000. DOI: 10.1093/synbio/ysz016. eprint: https://academic.oup.com/synbio/article-pdf/4/1/ysz016/33640438/ysz016.pdf. URL: https://doi.org/10.1093/synbio/ysz016.

174. Antonio Fabregat et al. "The reactome pathway knowledgebase". *Nucleic acids research* 46:D1, 2018, pp. D649–D655.

175. Jim Webber. "A programmatic introduction to neo4j". In: *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. 2012, pp. 217–218.

176. Nadime Francis et al. "Cypher: An evolving query language for property graphs". In: *Proceedings of the 2018 international conference on management of data*. 2018, pp. 1433–1445.

177. Eric W Sayers et al. "GenBank". *Nucleic acids research* 47:D1, 2019, pp. D94–D99.

178. Michal Galdzicki et al. "Data Model Standardization for Synthetic Biomolecular Circuits and Systems". *Design and Analysis of Biomolecular Circuits: Engineering Approaches to Systems and Synthetic Biology*, 2011, pp. 281–293.

179. Iñaki Sainz de Murieta, Matthieu Bultelle, and Richard I Kitney. "Toward the first data acquisition standard in synthetic biology". *ACS synthetic biology* 5:8, 2016, pp. 817–826.

180. Anna Bernasconi. "Data quality-aware genomic data integration". *Computer Methods and Programs in Biomedicine Update* 1, 2021, p. 100009.

181. Savas Konur et al. "Toward full-stack in silico synthetic biology: integrating model specification, simulation, verification, and biological compilation". *ACS Synthetic Biology* 10:8, 2021, pp. 1931–1945.

182. Hasan Baig et al. "Synthetic biology open language visual (SBOL Visual) version 2.3". *Journal of Integrative Bioinformatics* 18:3, 2021.

183. Belén Calles, Ángel Goñi-Moreno, and Viéctor de Lorenzo. "Digitalizing heterologous gene expression in Gram-negative bacteria with a portable ON/OFF module". *Molecular Systems Biology* 15, 2019.

184. Jacob Beal et al. "Reproducibility of fluorescent expression from engineered biological constructs in E. coli". *PloS one* 11:3, 2016, e0150182.

185. Bradley Brown et al. "Capturing Multicellular System Designs Using Synthetic Biology Open Language (SBOL)". *ACS Synthetic Biology* 9:9, 2020, pp. 2410–2417.

186. Amy N Langville and Carl D Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton university press, 2006.

187. Göksel Mısırlı et al. "Data integration and mining for synthetic biology design". *ACS synthetic biology* 5:10, 2016, pp. 1086–1097.

188. Vivienne Sze et al. "Efficient processing of deep neural networks: A tutorial and survey". *Proceedings of the IEEE* 105:12, 2017, pp. 2295–2329.

189. Leo Breiman. "Random forests". *Machine learning* 45, 2001, pp. 5–32.

190. Gunther Eysenbach et al. "The role of ChatGPT, generative language models, and artificial intelligence in medical education: a conversation with ChatGPT and a call for papers". *JMIR Medical Education* 9:1, 2023, e46885.

191. Annunziata Lopiccolo et al. "A last-in first-out stack data structure implemented in DNA". *Nature communications* 12:1, 2021, pp. 1–10.

192. Richard Kitney et al. "Enabling the advanced bioeconomy through public policy supporting biofoundries and engineering biology". *Trends in biotechnology* 37:9, 2019, pp. 917–920.

193. Joanne Kamens. "The Addgene repository: an international nonprofit plasmid and data resource". *Nucleic Acids Research* 43:D1, 2014, pp. D1152–D1157. ISSN: 0305-1048. DOI: 10.1093/nar/gku893. eprint: https://academic.oup.com/nar/article-pdf/43/D1/D1152/7330438/gku893.pdf. URL: https://doi.org/10.1093/nar/gku893.

194. Kim D Pruitt, Tatiana Tatusova, and Donna R Maglott. "NCBI Reference Sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins". *Nucleic acids research* 33:suppl_1, 2005, pp. D501–D504.

195. Esteban Martıénez-Garcıéa et al. "SEVA 4.0: an update of the Standard European Vector Architecture database for advanced analysis and programming of bacterial phenotypes". *Nucleic Acids Research* 51:D1, 2023, pp. D1558–D1567.

196. Artem Lysenko et al. "Representing and querying disease networks using graph databases". *BioData mining* 9:1, 2016, pp. 1–19.

197. Michael Baitaluk et al. "BiologicalNetworks: visualization and analysis tool for systems biology". *Nucleic acids research* 34:suppl_2, 2006, W466–W471.

198. Avi Ma'ayan. "Introduction to network analysis in systems biology". *Science signaling* 4:190, 2011, tr5–tr5.

199. Hiroaki Kitano. "Systems biology: a brief overview". *science* 295:5560, 2002, pp. 1662–1664.

200. Guillermo G Zampar et al. "Temporal system-level organization of the switch from glycolytic to gluconeogenic operation in yeast". *Molecular systems biology* 9:1, 2013, p. 651.

201. Marc Legeay et al. "Visualize omics data on networks with Omics Visualizer, a Cytoscape App". *F1000Research* 9, 2020.

202. Christiane VR Hütter et al. "Network cartographs for interpretable visualizations". *Nature Computational Science* 2:2, 2022, pp. 84–89.

203. Brent Berlin and Paul Kay. *Basic color terms: Their universality and evolution*. Univ of California Press, 1991.

204. Matthias Heinemann and Sven Panke. "Synthetic biology—putting engineering into biology". *Bioinformatics* 22:22, 2006, pp. 2790–2799. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl469. eprint: https://academic.oup.com/bioinformatics/article-pdf/22/22/2790/48838388/bioinformatics_22_22_2790.pdf. URL: https://doi.org/10.1093/bioinformatics/btl469.

205. Percy Liang and Mayur Naik. "Scaling abstraction refinement via pruning". In: *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2011, pp. 590–601.

206. Alvin Tamsir, Jeffrey J. Tabor, and Christopher A. Voigt. "Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'". *Nature* 469:7329, 2011, pp. 212–215. ISSN: 1476-4687. DOI: 10.1038/nature09565. URL: https://doi.org/10.1038/nature09565.

207. Aidan Tinafar, Katariina Jaenes, and Keith Pardee. "Synthetic biology goes cell-free". *BMC biology* 17:1, 2019, pp. 1–14.

208. Max Chavarría et al. "A Metabolic Widget Adjusts the Phosphoenolpyruvate-Dependent Fructose Influx in Pseudomonas putida". *mSystems* 1, 2016, e00154–16. DOI: 10.1128/mSystems.00154-16.

209. Fusun Yaman, Aaron Adler, and Jacob Beal. "AI challenges in synthetic biology engineering". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

210. Justin J Miller. "Graph database applications and concepts with Neo4j". In: *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*. Vol. 2324. 36. 2013, pp. 141–147.

211. Carson Sievert. *Interactive web-based data visualization with R, plotly, and shiny*. CRC Press, 2020.

212. Michael E Smoot et al. "Cytoscape 2.8: new features for data integration and network visualization". *Bioinformatics* 27:3, 2011, pp. 431–432.

213. Albert-László Barabási. "Network science". *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371:1987, 2013, p. 20120375.

214. Jonathan Tellechea-Luzardo et al. "Fast biofoundries: coping with the challenges of biomanufacturing". *Trends in Biotechnology*, 2022.

215. Jacob Beal et al. "The long journey towards standards for engineering biosystems: Are the Molecular Biology and the Biotech communities ready to standardise?" *EMBO reports* 21:5, 2020, e50521.

216. Petter Portin and Adam Wilkins. "The evolving definition of the term "gene"". *Genetics* 205:4, 2017, pp. 1353–1364.