

**SBOLEXPLOER: DATA INFRASTRUCTURE AND DATA  
MINING FOR GENETIC DESIGN REPOSITORIES**

by  
Michael Zhang

A thesis submitted to the faculty of  
The University of Utah  
in partial fulfillment of the requirements for the degree of

Master of Science  
in  
Computing

School of Computing  
The University of Utah  
May 2019

Copyright © Michael Zhang 2019

All Rights Reserved

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of Michael Zhang  
has been approved by the following supervisory committee members:

Chris Myers , Chair(s) 11 Feb 2019  
Date Approved

Ryan Stutsman , Member 14 Feb 2019  
Date Approved

Jeffrey Phillips , Member 12 Feb 2019  
Date Approved

by Ross Whitaker , Chair/Dean of  
the Department/College/School of Computing  
and by David B. Kieda , Dean of The Graduate School.

## ABSTRACT

Biology is a very noisy field. Experiments are difficult to reproduce, the mechanisms behind life are not well understood, and data that we do obtain is difficult to make sense of. Much like traditional engineering fields where engineers draw from a library of reusable parts for their designs, experimental and synthetic biologists have designed biological circuits by drawing from a library of genetic constructs. However, these so-called genetic parts are poorly understood and are therefore limited in their usefulness. Additionally, there are hundreds of thousands of parts and sequences that have been either created or discovered. For my thesis, I filter through this biological noise to provide genetic circuit designers a powerful way to search for and access the genetic parts that are useful to them.

This thesis is focused on creating SBOLEplorer, a system that is used to provide intuitive search within the SynBioHub genetic design repository. SynBioHub integrates genetic construct data from various sources and transforms and stores this data in a standardized data model. By tackling the intricate data mining and data infrastructure problems associated with large-scale semi-structured and noisy data, the search, transformation, and storage of data in genetic design repositories can be enhanced. In particular, this thesis focuses on improving the usability of genetic part repositories' search capabilities. By clustering SynBioHub's genetic parts into many derived collections, duplicate parts are merged. From there, a graph analysis algorithm is used to rank collections of parts by popularity and usefulness. Finally, data infrastructure challenges relating to indexing, storing, serving, and distributed search are solved. The end goal of SBOLEplorer is to integrate these findings into SynBioHub and other genetic design repositories' data representation, search functionality, and data infrastructure.

# CONTENTS

<b>ABSTRACT</b> .....	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b> .....	<b>vi</b>
<b>CHAPTERS</b>	
<b>1. INTRODUCTION</b> .....	<b>1</b>
1.1 Genetic Part Selection .....	1
1.2 Genetic Design Repositories .....	2
1.3 Contributions .....	3
1.4 Thesis Overview .....	5
<b>2. BACKGROUND</b> .....	<b>6</b>
2.1 Synthetic Biology .....	6
2.2 Genetic Circuits .....	7
2.3 The Synthetic Biology Open Language .....	10
2.4 Software Tools .....	14
<b>3. GRAPH ANALYSIS</b> .....	<b>17</b>
3.1 Data Exploration .....	17
3.2 Ranking .....	19
3.3 Implementation .....	20
3.4 Analysis .....	24
3.5 Summary .....	27
<b>4. CLUSTERING</b> .....	<b>28</b>
4.1 Methods .....	28
4.2 Implementations .....	32
4.2.1 Hierarchical Single Link Clustering .....	32
4.2.2 UCLUST .....	37
4.3 Summary .....	38
<b>5. DATA INFRASTRUCTURE</b> .....	<b>41</b>
5.1 Inverted Index .....	41
5.2 Search Implementation .....	42
5.2.1 Metrics .....	46
5.3 Workflow and System Design .....	47
5.3.1 Distributed Search .....	52
5.4 Summary .....	54
<b>6. CONCLUSION</b> .....	<b>56</b>

6.1 Summary .....	56
6.2 Future Work .....	56
6.2.1 Infrastructure .....	57
6.2.2 Data Augmentation .....	57
6.2.3 Data Visualization .....	57
6.2.4 Better Metrics .....	58
<b>REFERENCES .....</b>	<b>59</b>

## ACKNOWLEDGEMENTS

I want to thank my advisor, Professor Chris Myers. Chris accepted me into his lab when I was a freshman undergraduate. I have learned a lot from Chris since then, and he has been integral in challenging me and making me a better researcher. He is never afraid of getting his hands dirty in the nitty-gritty details, and he is one of the most passionate and motivated people I know. His idioms will continue to intrigue me.

I also want to thank Professor Ryan Stutsman and Professor Jeff Phillips for their efforts on the supervisory committee. Ryan has been inspirational. I can always count on him to feed my curiosity for distributed systems and infrastructure. His courses have been a highlight of my time at the University of Utah, and I will continue to think back to our conversations for the duration of my career. Jeff, whose mathematical intuition is sometimes beyond my comprehension, has done a fantastic job at introducing me to all things data and new Markovian perspectives on life.

I want to thank my lab mates, Leandro Watanabe, Tramy Nguyen, Zhen Zhang, Meher Samineni, Zach Zundel, Jet Mante, Pedro Fontanarrosa, Oliver Flatt, Igor Durovic, and Samuel Bridge. They have all been great friends. I will look back fondly at my time with you all in the steadily deteriorating lab space on the 4th floor of MEB.

Finally, I want to thank my parents and little brother. They provide the foundation upon which I can focus on learning and developing. I would not be here if it weren't for their boundless encouragement.

This work is funded by the National Science Foundation under grants CCF-1218095 and DBI-1356041. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# CHAPTER 1

## INTRODUCTION

In synthetic biology, experimental biologists build new and novel genetic circuits, or networks of biological logic gates, that perform a variety of applications [16]. These applications range from making cats glow green to identifying and disrupting cancer cells [20, 36]. To build these genetic circuits, experimental biologists search for well-characterized and well-studied genetic parts. These genetic parts form the building blocks for more complicated genetic circuits. Each part is unique in its function, ideal operating environment, and ability to interface with other parts. Unlike traditional electrical engineering parts such as the resistor or transistor, biological parts are noisy and particular. Specifically, this means biological parts are difficult to use since each part is unique, and standardized parts vary depending on the target organism and target function.

### 1.1 Genetic Part Selection

To address the issues with selecting the right parts to use in genetic circuit design, experimental biologists have compiled massive amounts of data on a wide variety of parts. This data is stored in genetic design repositories and is encoded in a variety of standardized and non-standardized data models. As a result of how unstructured this encoding and storing process is, the data is also messy and unstructured even when each part is encoded in a standardized format. How do you find the right part? How do you know if it is even there? While a massive amount of data exists, it is difficult to search through and find trends in the data in a meaningful way.

Depending on how the search query or data is represented, there are many different ways to search for and select genetic parts. One way of describing genetic parts is through its genetic sequence. Genetic sequences are character strings of varying length, and in the case of deoxyribonucleic acid (DNA), are composed of base pairs cytosine (C), guanine (G), adenine (A), and thymine (T). Each base pair forms a character in the genetic



sequence string, and therefore pairs of sequences can be compared through a variety of techniques. Commonly, fuzzy local alignments are computed between a query sequence and a database of sequences using the Basic Local Alignment Search Tool (BLAST) [3]. However, other forms of sequence search are also useful. For example, optimizing for global alignment or edit distance, searching for exact matches that fit a certain template akin to regular expression matching, and searching through an annotated corpus of parts with semantic relationships are all useful ways to search for genetic parts. Offering solutions for all these kinds of searches is important for any repository hosting genetic circuit designs. Unfortunately, the best search solutions often depend on the available data shapes, and some repositories that have parts with interesting semantic relationship data and linkage metadata don't utilize or mine the data to enhance search [41].

## 1.2 Genetic Design Repositories

Genetic design repositories are centralized data storage services for storing, sharing, and serving genetic designs. SynBioHub is an example of a genetic design repository [37, 41]. Specifically, SynBioHub is a repository of genetic designs encoded in the *Synthetic Biology Open Language* (SBOL), a standardized data exchange format for genetic circuit designs [6, 19, 52]. SynBioHub leverages the power of linked data to connect designs and parts with their usages in other designs and parts. By linking all part data using well-defined ontologies such as the *Sequence Ontology*, a graph of characteristics, definitions, and their relationships is formed [15]. Each edge in this graph is of the form of a triple. Each triple encodes relationship data using a subject, a predicate, and an object. This means queries on the data can be constructed to filter based on the relationship data, follow triples to observe direct and transitive relationships, and show the general topology between all stored genetic designs. This also means the data has interesting linkage metadata and semantic relationship information. Unfortunately, having all the genetic designs stored in the same standardized data exchange format does not mean the data is immediately useful [1]. Making sense of semi-structured data is, therefore, one of the responsibilities of genetic design repositories like SynBioHub.

General semi-structured information retrieval systems tackle the problem of search using a variety of techniques. A good case study is the various attempts to make sense of the

unstructured data on the web. Similar to biological parts, web page data is the definition of messy and unkempt. Web pages link to one another similarly to how biological parts are composed hierarchically and through different versions. Also, web pages host content that is of varying quality, usefulness, and uniqueness. A common problem for non-curated collections of genetic circuit designs is the amount of seemingly duplicate designs for the same circuit or part. As a result, all sorts of techniques from graph analysis, clustering, and statistical modeling are used [8, 18, 49]. Unfortunately, certain search engines for biological parts are not as advanced, with some just performing basic substring matching arbitrarily [37, 41].

One of the largest datasets in SynBioHub currently is the *International Genetically Engineered Machine* (iGEM) dataset. The iGEM competition is an annual competition where university students compete against each other to design novel genetic circuits [57]. As such, the collection of these designs, the iGEM dataset, is a great proving ground for the algorithms and techniques discussed in this thesis. The iGEM dataset consists of many parts which are all stored in SQL tables and hosted at the Registry of Standard Biological Parts [30]. This registry provides a search interface, but it does not utilize the latent structure hidden in the parts and their relationships. SynBioHub has converted the dataset into SBOL, and all evaluations of SBOLExplorer are done using this dataset as a benchmark.

By extending SynBioHub and other genetic design repositories with SBOLExplorer and its improved algorithms and infrastructure, synthetic biologists will be able to more intelligently search, extract, and utilize the genetic parts they require for their circuit designs. These enhancements give users of genetic design repositories a better way to interpret and extract meaning from the massive amounts of semi-structured data present. Experimental biologists will be able to make better decisions when designing their circuits, and researchers will be able to understand better the genetic design topology found within these repositories.

### 1.3 Contributions

Data is most powerful when it is easily interpretable. Genetic design repositories exist, but they provide difficult interfaces for their users. The data is semi-structured, noisy, not

normalized, and inconsistent. This makes it difficult for experimental biologists to find parts, determine their characteristics, and view trends within the various designs. Without a clear way to explore the data, genetic design repositories are of reduced value. The main contributions of the proposed research attempt to address these issues.

A popular popularity ranking algorithm is *PageRank*, which operates on a graph where the links to a part or page are indicative of that part or page's popularity [49]. PageRank was also developed with ranking the web in mind. This is useful given the similar challenges between the web and a genetic design repository hosting SBOL parts. Applying PageRank to the SBOL graph introduces a novel way to reason about datasets like the iGEM dataset.

The ability to naively query SBOL is powerful, but the scale of the data limits a human's ability to reason and understand its insights. By clustering genetic parts, duplicate and similar entries can be grouped, merged, or removed. Because the data is user-generated and user-submitted, there are many inconsistencies in its metadata. By clustering SBOL parts with the goal of enhancing search, SBOLExplorer introduces and explores interesting trade-offs between different clustering techniques.

SBOLExplorer uses data infrastructure in the form of *Elasticsearch*. Storing a preprocessed set of SBOL designs in an inverted index allows for fast retrieval and calculation of search results. By calculating a unique ranking based off of keyword match, PageRank, and clustering results, SBOLExplorer determines relevancy to queries for genetic parts in an intelligent way that leverages the uniqueness of SBOL.

The result is a service that genetic design repositories storing SBOL can integrate with to provide better search functionality. SynBioHub is currently integrated with SBOLExplorer and benefits from SBOLExplorer's data analysis and data infrastructure. Users do not interact with SynBioHub any differently, but they do notice that search results are suddenly much more relevant compared to before.

This thesis focuses on SBOLExplorer and the technology it utilizes, the architecture it's built on, the data it helps users find and understand, how it contributes to the SBOL workflow, and the future work and benefits it enables. The main contributions of this thesis culminate in a service that enhances the search capabilities of genetic design repositories by introducing new ways to better understand, reason, and interact with large collections

of SBOL parts.

## 1.4 Thesis Overview

Chapter 2 goes over the background necessary to set the context of this work. Specifically, the field of synthetic biology is introduced, genetic circuits are defined and examples are shown, SBOL and its motivations are discussed, and software tools and workflows in the domain of synthetic biology are explored.

Graph popularity ranking algorithms can also be designed to exploit the linked structure between parts. This is especially important for genetic design repositories since popularity can be an indicator of how to order results to user queries. The details of SBOLExplorer's graph analysis are discussed in Chapter 3.

Clustering based off of sequence and SBOL structure is a good way to simplify and extract meaning from SBOL datasets. However, this introduces challenges in how to represent the data, and specifically how different SBOL parts within a large collection should be compared against each other. Genetic sequence clustering formulations and trade-offs are discussed in Chapter 4.

The end goal is to use the aforementioned data analysis to provide better search for genetic design repositories. As such, SBOLExplorer needs to be a service that genetic design repositories can plug into and query. To serve these queries efficiently, data infrastructure is required. Chapter 5 goes over how Elasticsearch is used to provide a fast and incremental search index, and how clustering and PageRank together contribute to the ranking of the search results. SBOLExplorer has been integrated into the SynBioHub part repository. This chapter also describes the resulting workflow integration, architecture, and improvements to SynBioHub's search. Tools that rely on SynBioHub now benefit from improved search. SBOLDesigner is used as a case study here. Also, SBOLExplorer exposes an API that other genetic design repositories can integrate with. This includes features like advanced search, permissioned search, and distributed search. A metric that compares search result relevancy shows that using SBOLExplorer more than doubles the perceived relevance of search results.

Finally, Chapter 6 concludes this thesis by giving a summary of the work and by presenting future research directions.

## CHAPTER 2

### BACKGROUND

In this chapter, Section 2.1 introduces the field of synthetic biology. Section 2.2 dives a little bit deeper and describes the concept of a genetic circuit. To exchange designs of genetic circuits between software tools and genetic part repositories, SBOL is used, as described in Section 2.3. Finally, Section 2.4 goes over some of the software tools that support SBOL. This will give context regarding how SBOLExplorer fits into the field and within the SBOL workflow.

#### 2.1 Synthetic Biology

Synthetic biology is a relatively new field born out of systems biology, electrical engineering, and bioinformatics [16]. Specifically, synthetic biology enhances these parent fields by taking fundamental engineering principles such as standards, abstraction, and decoupling, and applying them to genetic circuit design. Systems biology, defined as the study of complex biological systems and their pathways of operation and interaction, is used to accelerate the creation of new and novel genetic circuits. Electrical engineering concepts such as circuits and logic gates function as great models for the design process. Finally, synthetic biology borrows from bioinformatics and makes use of massive amounts of data and computational resources.

The direct precursor to synthetic biology was genetic engineering, which focused on modifying organisms' genomes in order to manipulate the characteristics of those organisms. While genetic engineering has existed since the 1960s, it has never really adopted true engineering fundamentals. Therefore, even though synthetic biology is a re-branding of genetic engineering, the focus is to adopt standards, abstraction, and decoupling. This effort can be seen in the adoption of the SBOL standard, the use of software tools to rapidly iterate on the design of genetic circuits and their models, simulations, and compositions, and the separation of different tools for different tasks.

SBOL also addresses the problem of reproducibility in synthetic biology. Experiments are inherently complex, and information necessary for reproducing the results found in groundbreaking papers is often incomplete. As a result, much of the research in this field is of reduced value to the broader scientific community. Data models such as SBOL and tools that utilize the data model are therefore integral in providing a means to address the issue of reproducibility. Without these blueprints, genetic circuits lack DNA sequence information, proper characterization data, and circuit layout details. To more thoroughly solve the reproducibility problem, tools that utilize SBOL must also support the full range of ways experimental biologists express their designs.

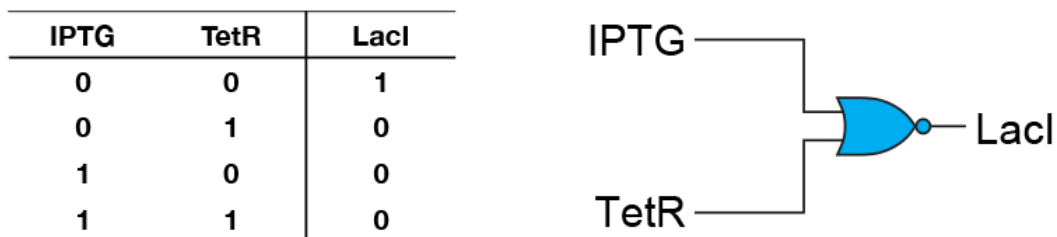
## 2.2 Genetic Circuits

In synthetic biology, genetic circuits are classified as sensor and actuator networks operating in the biological domain. In the traditional electrical circuit, inputs such as buttons or switches control outputs such as LEDs and motors. This is accomplished through the physical and electrical properties of a variety of components such as resistors, capacitors, inductors, and LEDs. In genetic circuits, much is the same. Standardized sequences of DNA control expression of various proteins that results in external factors like cells glowing green [20]. The most significant difference is that the mode of expression is due to the transcription and translation of genes into protein, and not of electrons flowing through wires. This is commonly referred to as the central dogma of molecular biology [12], and is shown in Figure 2.1. Because of these similarities, genetic circuits with similar function to traditional electrical circuits can be built.

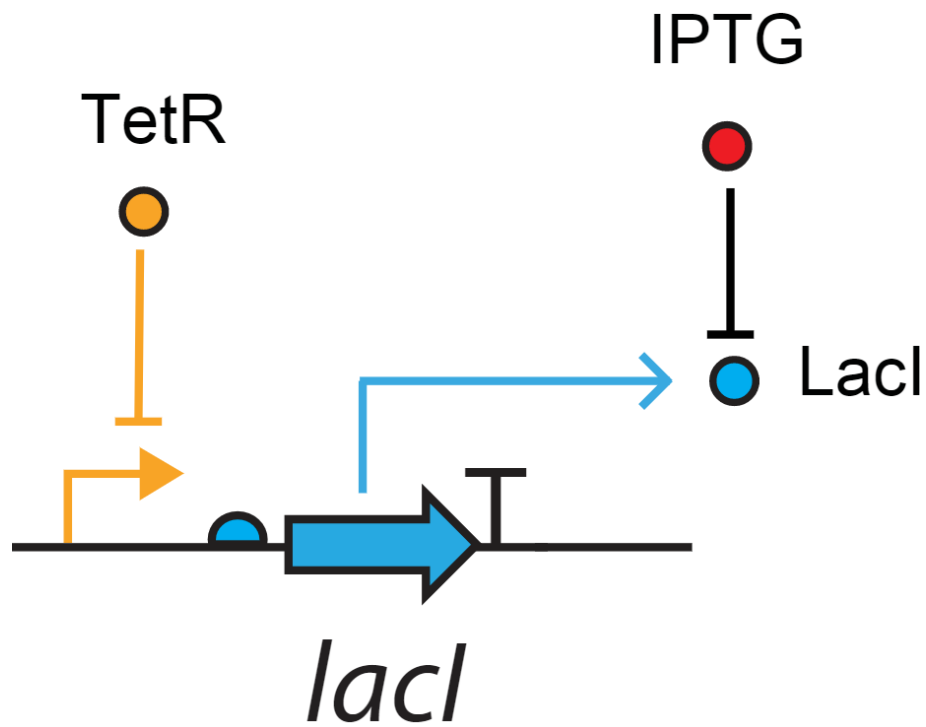
For example, one such circuit is the NOR gate. A truth table and electrical circuit schematic are shown in Figure 2.2. IPTG and TetR are the inputs, and LacI is the output. The truth table shows that LacI is only high when both inputs are low. In every other case, LacI is low. The Boolean NOR function can also be obtained from a genetic circuit, as shown in Figure 2.3. The circuit schematic describes a backbone of DNA with four glyphs of DNA sequences, defined as parts, drawn on top of it. The yellow bent arrow is called a *promoter*, and facilitates transcription of DNA to RNA. The half circle *ribosome binding site* and arrow *coding sequence* are parts that get transcribed into RNA. The T shaped *terminator* is used to stop transcription. After the RNA is created, a ribosome binds to the



**Figure 2.1.** The central dogma of molecular biology. DNA is transcribed into RNA, and RNA is translated into protein. In genetic engineering and synthetic biology, experimental biologists change the DNA, and the resulting protein is altered. For example, by modifying a genome's DNA to encode for green fluorescent protein, researchers can determine if their genetic circuit is operating correctly by shining ultraviolet light on the cells [20].



**Figure 2.2.** The truth table and traditional schematic drawing for a NOR gate. The inputs are IPTG and TetR, and the output is LacI. LacI is only produced when TetR is low, and LacI is only functional when IPTG is not present. When IPTG is present, it binds to LacI prohibits it from acting as a repressor. Therefore, the only time when LacI is likely functional in the cell is when both TetR and IPTG are low.



**Figure 2.3.** The biological circuit schematic of a traditional NOR gate. TetR is repressing the promoter, and IPTG binds with LacI to form a complex that removes free LacI from the system. When neither TetR nor IPTG is present, LacI can be produced normally and is free to control downstream promoters in the system.



*ribosome binding site* and translates the *coding sequence* into a protein. In this case, the *coding sequence* contains the blueprint for the LacI protein. These standard parts are analogous to the electrical components of an electrical circuit.

With the previous understanding of these genetic parts' behaviors, the Boolean NOR function is realized. TetR is a protein that inhibits the function of the *promoter*, and IPTG is a small molecule that binds to LacI and effectively removes its ability to act as a transcriptional repressor. When TetR is present, LacI doesn't get produced in the first place; when IPTG is present, LacI is neutralized before it can act. The states where LacI is low coincide with the rows in the truth table in Figure 2.2 that show a zero for LacI. When neither TetR nor IPTG is present, LacI can be expressed, resulting in a high state. This high state is the only possible state where LacI is high, and coincides with the row in the truth table in Figure 2.2 that shows a one for LacI.

The NOR gate is a universal gate, meaning all other gates and Boolean systems can be built just from NOR. For example, a multiple input AND function could be built [36]. The inputs could be proteins associated with carcinogenic cells, and the output could be green fluorescent protein. When all the inputs are present, all the cancer cells will glow green. Alternatively, the function could be NAND, and cause some vital protein for the cancer cell to stop being produced. Now that we have realized a genetic NOR circuit, all other Boolean systems can theoretically be built using genetic circuits.

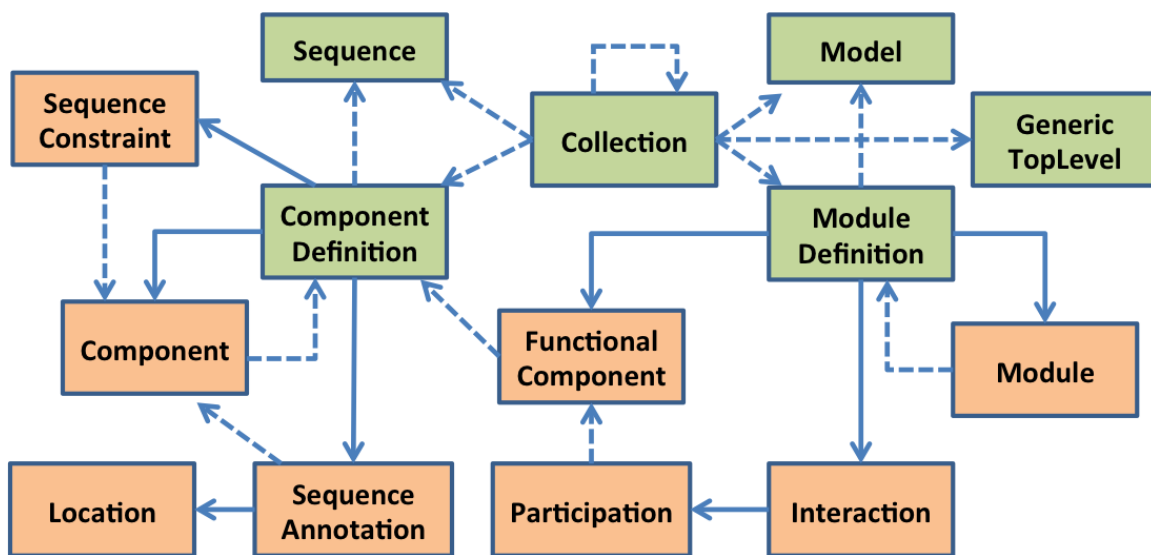
### 2.3 The Synthetic Biology Open Language

SBOL allows genetic parts to be composed in a machine interpretable and extensible way. Specifically, this means each part has required fields that provide all sorts of useful characterization data. Parts can also be composed hierarchically, allowing for designs to be easily created in a modular and referential manner. In particular, SBOL is powerful because it has a very rich and thorough data model for representing genetic parts and their references. It makes software tool workflows possible through its language and tool agnostic interoperability and provides an easy way for new tools, such as SBOExplorer, to tap into the existing SBOL data and ecosystem [53]. SBOExplorer relies on the genetic circuit data being represented in a normalized, consistent, and easily reasonable format, so the SBOL data model is especially important for this thesis' applications.

SBOL is a tightly specified standard for describing these genetic circuits [19]. SBOL consists of two parts: the SBOL data model and the SBOL Visual standard [6,52]. The data model, as shown in Figure 2.4, is a specification of objects and their relationships. Circuits such as the genetic NOR in Figure 2.3 can be encoded in SBOL and passed around as an electronic file. Each distinct part is represented by a *ComponentDefinition*, with each instantiation or usage of the part being represented by a *Component*. *SequenceConstraints* and *SequenceAnnotations* define a relative and absolute order respectively of the parts on the backbone. The actual base pairs are stored in a *Sequence*. Many parts and designs are organized into *Collections* which also include functional information represented as *ModuleDefinitions* and *Interactions*. This centralized file format allows software tools to communicate with each other.

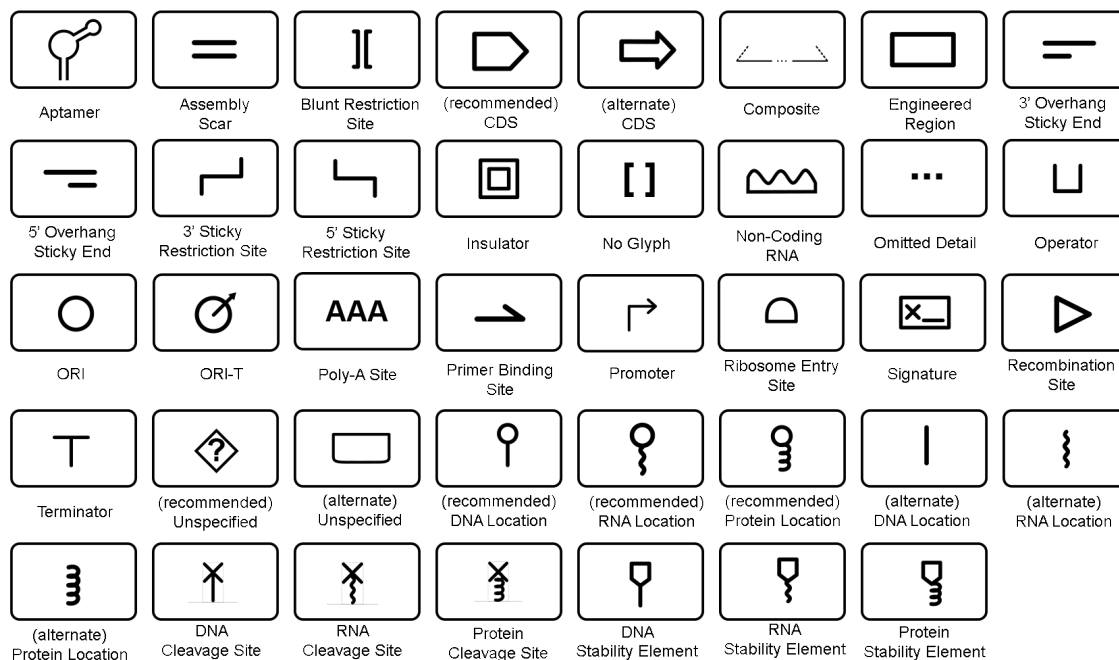
The SBOL Visual standard provides a standardized set of schematic glyphs to describe visualizations of genetic circuits [11, 52]. For example, Figure 2.3 is a depiction of a genetic circuit using SBOL Visual glyphs. Specifically, the genetic parts are drawn in accordance with how their glyphs are defined. These definitions are shown in Figure 2.5. For example, whenever a *promoter* must be pictorially represented, a bent arrow going to the right can be drawn, and its meaning can be assumed.

Both the SBOL data and visual standards are necessary for reproducibility and interoperability in synthetic biology and further shows how engineering principles have influenced the field. When SBOL is not used to describe research results and genetic circuit layouts, the information published in papers is usually incomplete. Incomplete knowledge of the genetic system results in unreproducible results and a loss in trust for the findings. For this reason, in 2016, *ACS Synthetic Biology* set a precedent by adopting the SBOL Visual and data standards as the official method for depicting and digitally storing genetic constructs [26]. The use of SBOL in publications and the deposition of this data into public repositories tremendously aids reproducibility in this field. However, for biologists to generate these designs in SBOL, they need a workflow with tools that have features that enable a straightforward way for creating, storing, and searching through these constructs [53].



**Figure 2.4.** The SBOL data model. Top level classes are drawn in green, and supporting classes are drawn in yellow. Together, this data specification allows the description of a broad range of genetic circuits. Software tools import and export this format to allow for interoperability and decoupling from tool to tool. Figure from Beal et al. [6].

## Nucleic Acid Glyphs



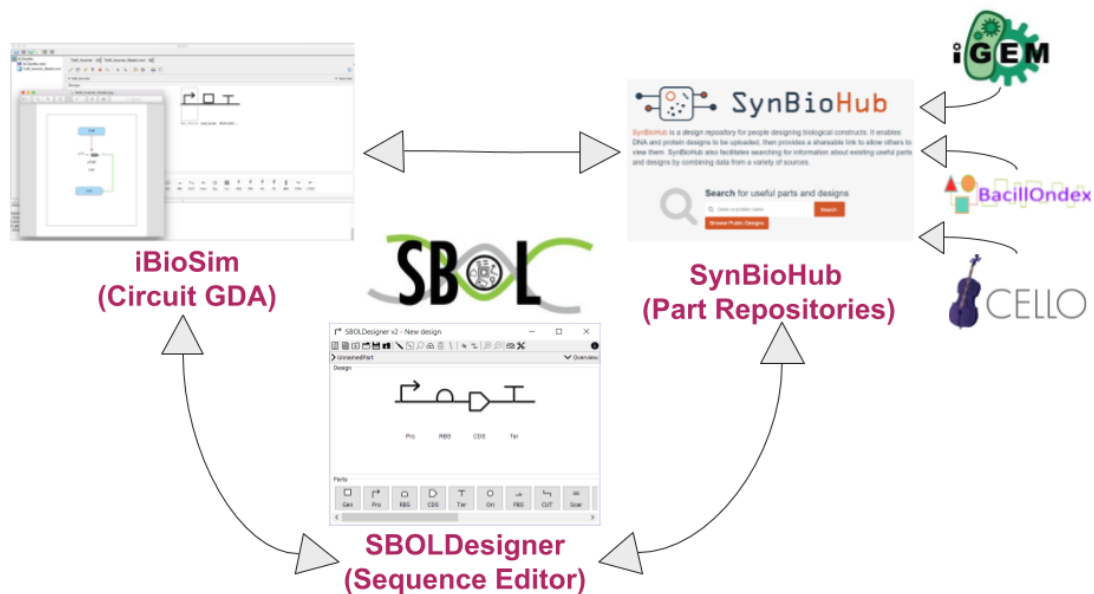
**Figure 2.5.** The SBOL Visual set of defined nucleic acid glyphs. Each glyph represents a type of genetic part that genetic circuits are built from. The visual standard specifies rules and best practices for how these parts should be drawn in software tools, in figures, and on whiteboards. Glyphs that represent molecular species and interactions also exist. Figure from Cox et al. [11].

## 2.4 Software Tools

Biologists use tools to create their designs. Part repositories, simulation tools, modeling tools, and sequence level computer-aided design tools interact through the SBOL standard. Software tools help experimental biologists abstract their designs and more efficiently prototype their circuits [53]. These tools can also be decoupled by allowing SBOL to be the common language. Because of SBOL, even a tool developed in isolation can contribute to the workflow.

An example workflow that creates and captures genetic designs is depicted in Figure 2.6. In this workflow, SBOLDesigner is a sequence editor that integrates support of the SBOL data standard and SBOL Visual symbols. In particular, SBOLDesigner can obtain DNA sequences and other important metadata from the SynBioHub parts repository [37, 41]. These components can then be composed and edited within SBOLDesigner to create a complete structural design of a genetic circuit. These new composite designs can then be uploaded to a genetic design repository like SynBioHub to enable sharing, storing, and reuse. To add functional information about a genetic design, SBOLDesigner has been integrated into the modeling and simulation *genetic design automation* (GDA) tool, iBioSim [38]. The iBioSim software can be used to construct and analyze functional models using the *Systems Biology Markup Language* (SBML) [29]. These functional models once annotated using genetic designs produced by SBOLDesigner [54] can be converted into an SBOL document including functional information about the product of these genetic circuits and their interactions [44]. Once again, the complete genetic circuit with its functional information can be archived in a part repository for sharing. Throughout this process, researchers can collaborate and pass around files from institution to institution, located anywhere in the world. The SBOL standard provides the means to enable a lossless communication of data between these software tools and repositories.

This modularization of software tools reduces the need for each platform to reinvent the wheel and allows a single tool to add valuable features by connecting to a rich library of tools and repositories. For example, SBOLDesigner can be embedded in other tools and platforms that support SBOL. This gives modeling and simulation tools like iBioSim the ability to elegantly edit the structural portion of its genetic designs using SBOL Visual [32, 38]. Also, tools like SBOLExplorer can be integrated with genetic design repositories like



**Figure 2.6.** A workflow consisting of SBOLDesigner, SynBioHub, and iBioSim [38]. Genetic parts from various databases are hosted in the SynBioHub repository [23, 37, 41, 43, 45]. SBOLDesigner and iBioSim can download these parts and use them to construct complete genetic designs. Specifically, iBioSim takes care of modeling and simulation, and SBOLDesigner takes care of sequence level design. In this workflow, SBOLEplorer is not used. This makes searching through parts stored in SynBioHub difficult for every tool in this workflow. Figure from Zhang et al. [62].

SynBioHub as long as the two communicate through the SBOL data standard. Additionally, the experimentalist can select from many options and is not limited to the in-house toolchain of a single platform.

As a result of part reuse and design sharing between different software tools, it is essential to have a central repository of shared genetic designs. This repository acts as a hub for storing designs, sharing designs, kick-starting designs, and searching for designs. By improving the search capabilities of genetic circuit design repositories like SynBioHub, the entire workflow is improved, and each tool benefits from higher utility. For biologists, the value brought by a complete genetic design automation workflow is quickly growing, and will only continue to become more established in the future. As such, it is important to make the workflow frictionless by enhancing biologists' abilities to utilize their software tools and genetic circuit design repositories through better search and exploration capabilities.

## CHAPTER 3

### GRAPH ANALYSIS

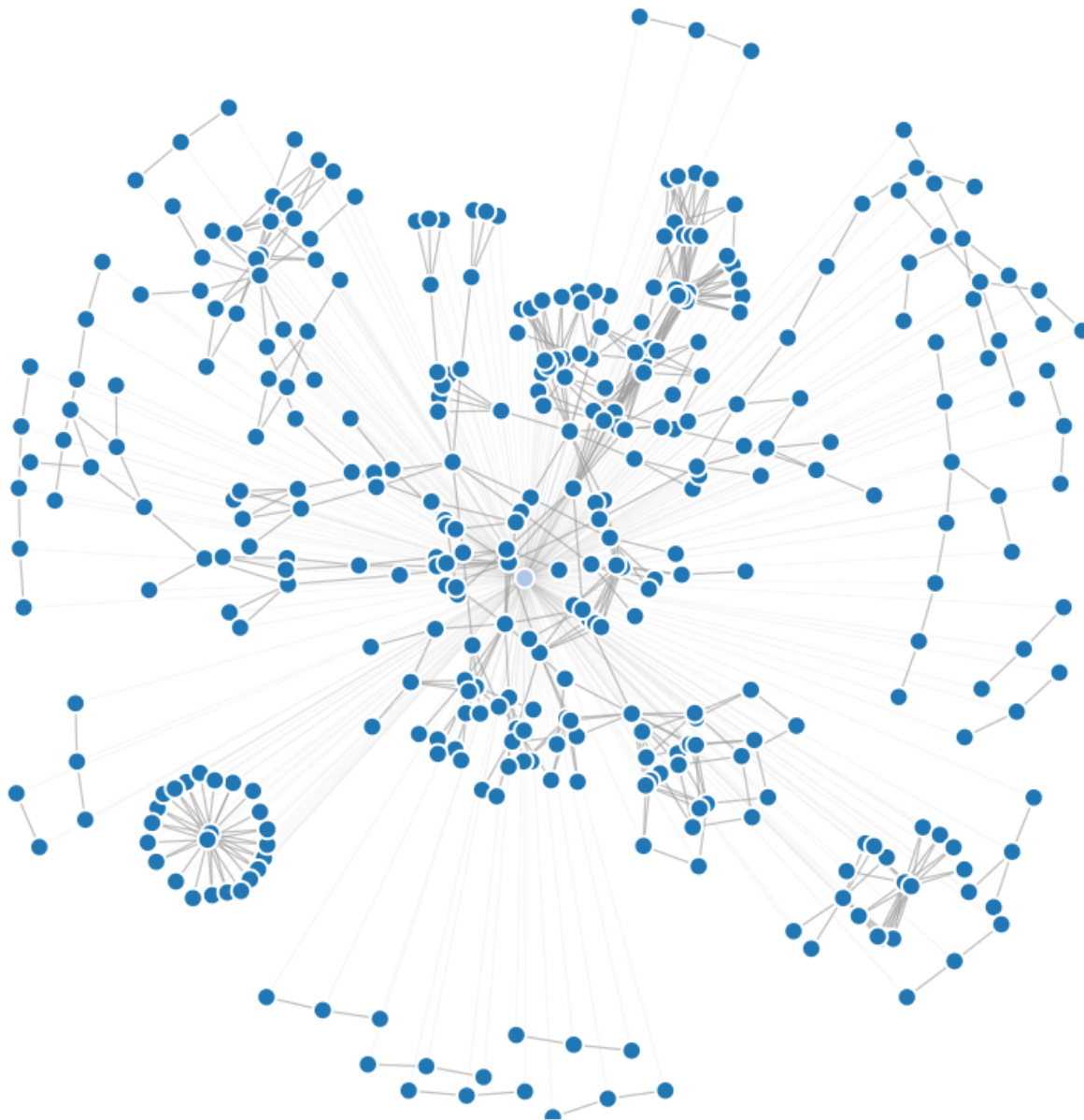
Analyzing the link structure of the graph formed by SBOL parts is a good way of determining a part's popularity. In this chapter, Section 3.1 describes the data within SynBioHub using a graph visualization. Section 3.2 goes over why having popularity ranking for parts is important for deciding on search results and how the PageRank algorithm lends itself well to ranking parts encoded in SBOL. Section 3.3 describes the PageRank implementation used in SBOLEplorer and how the probability transition matrix can be extracted from a graph database storing SBOL as RDF triples. Specifically, the dataset is small enough to run PageRank periodically on a single server as long as sparse matrices are used. Section 3.4 shows the results of running PageRank on the iGEM dataset and analyzes the distribution of rankings and how it relates to what is expected from the iGEM dataset. Finally, Section 3.5 summarizes this chapter and its context within SBOLEplorer.

#### 3.1 Data Exploration

Figure 3.1 shows a high-level visualization of a subset of the genetic design topology stored in SynBioHub. In this visualization, nodes represent genetic parts, and edges represent usages within another part. Compared to a traditional line by line textual representation of these genetic parts, the structure found within genetic parts can be seen, and information such as degree, connectedness, and density can be reasoned about.

The visualization side of SBOLEplorer is an application that runs in the user's web browser. The visualization itself is created in JavaScript using D3's force directed graph package. The data comes from running a *SPARQL Protocol and RDF Query Language* (SPARQL) query on the SynBioHub repository. SynBioHub subsequently fetches the part data by sending the SPARQL query to its backing store, Virtuoso. Virtuoso is a specialized *resource description framework* (RDF) triplestore [17]. This data is interpreted and transformed into a graph representation. Because SynBioHub stores all its part information in the form of





**Figure 3.1.** A visualization of a subset of the parts within SynBioHub. Blue circles represent parts, and a link between a pair of blue circles represents a usage within another part. Most of the nodes are within a single connected component, and some of the nodes are in isolated connected components. Some connected components, such as the ring in the bottom left corner, are unique in that they form spokes around a central set of parts. This is an example of a node with a high degree being surrounded by nodes with low degree. Also, many tree-like patterns can be seen. This shows that SynBioHub stores a sparse, as opposed to a dense, graph.

SBOL compliant triples, any result from any query can be visualized using SBOLExplorer. If it were not for the power of linked data and SBOL as a standard, arbitrary queries would not have been possible. Each query would return unstructured data, and special purpose code would have to be written to transform the query results into a format understandable by the graph visualization.

SynBioHub is a web frontend for its Virtuoso database. SynBioHub provides an API that allows other services to query its contents. Virtuoso is a specialized RDF triplestore that stores all the SBOL encoded triple data in a native graph format. SBOLExplorer's frontend asks the user for information they want to use to search. This form data is then parsed into a SPARQL query and sent to SynBioHub's API. The triples that are returned represent the data that we want to visualize. Each row is transformed into an adjacency list representation that the graph visualization understands. This is what the user will observe as the result of their query.

## 3.2 Ranking

It would be useful to have the ability to rank parts by popularity. When determining search result orderings, parts will be evaluated by both their importance and similarity to other parts.

The PageRank algorithm has been used to rank websites on the internet [49]. However, it also has many applications in social networks, information networks, road networks, biology, chemistry, neuroscience, and physics [21]. Fundamentally, PageRank uses Markov chain analysis to compute the relative importance of each node in the graph. Conceptually, each node's importance is determined by the importance of the nodes that have a directed edge pointing towards it. The more nodes that link to a node, and the more important those nodes are, the more important the linked node is in the network. By creating a probability transition matrix of the network, the relative importance of each node can be calculated using a variety of algorithms. If the graph is not ergodic (i.e., the graph is not connected, is periodic, or has absorbing and transient states), then the importance vector is not guaranteed to converge to a measure of the relative importance of each node. However, many interesting networks are not ergodic, so the PageRank algorithm simulates ergodicity by incorporating random jumps, also known as teleportation or taxation, into the

probability transition matrix. Therefore, the PageRank importance vector is guaranteed to converge.

PageRank can be applied in bioinformatics to analyze DNA sequences [21, 31]. Graphs of SBOL parts add another level of structure to plain genetic sequence data, so the original PageRank technique can also be applied to parts in genetic design repositories like SynBioHub. Each part is encoded in SBOL, and consists of metadata linking it to the parts it uses or references, and the characteristics it has (ex. type, role, collection, author, etc.). PageRank can then operate on a graph where each node is a part, and each edge is a part referencing another part through some predicate relationship. By itself, the SynBioHub graph is not ergodic because regardless of whether it is periodic, it is not connected and has absorbing and transient states. However, PageRank's random jump property allows the application of Markov analysis due to it simulating ergodicity through teleportation. By determining the relative importance of each part, SynBioHub can intelligently decide the relative usefulness of each part and present this order in the search results accordingly.

### 3.3 Implementation

Traditionally, the MapReduce programming framework has been used in a cluster environment to compute the PageRank of large graphs [13]. MapReduce is useful for a variety of real work tasks on large datasets, including genome analysis [40]. More recently, Apache Spark has displaced MapReduce as the big data framework of choice due to its ability to share data in *resilient distributed datasets* (RDDs) between different parts of the pipeline without needing to materialize the data on disk [60, 61]. PageRank is an iterative algorithm and therefore benefits from the use of RDDs. A common approach of computing PageRank at scale is to repeatedly matrix multiply the PageRank vector with the probability transition matrix until the PageRank vector converges. Between iterations, while the PageRank vector has not converged, the vector state can be represented as an RDD as opposed to being flushed onto disk. The efficiency benefits of this reduction in disk I/O have been confirmed [50, 55].

However, this approach is not needed for the scale of the iGEM dataset since the entire dataset can fit into the memory of one computer [42, 46]. This is possible since the probability transition matrix can be represented as a sparse matrix. Each node in the graph

is a part, and each edge is a connection to another part. The definition of connection is a RDF triple within the Virtuoso database where the subject and object values are SBOL *TopLevels* (ex. *ComponentDefinitions*, *Sequences*, *Collections*, etc.) and the predicate value is a relationship between the subject and object. This forms a subject, predicate, object triple that can encode a variety of SBOL relationships (ex. *Component*, *wasDerivedFrom*, etc). The number of connections for each part in the iGEM dataset is on the order of tens to hundreds, which is much smaller than the total number of parts. By storing the probability transition matrix as a decomposition into multiple sparse matrices, only the actual edges that exist, the nonzero entries of the adjacency matrix, have to be stored in memory.

The actual process of extracting these RDF triples from the Virtuoso graph databases uses SPARQL. SPARQL provides the ability to write a query that extracts certain fields depending on how they match template triples. The query that extracts relationships between parts is shown in Figure 3.2. Entities that should be ranked all have a triple where the subject and object fields correspond to a unique *uniform resource identifier* (URI) and the predicate indicates that subject and object are *TopLevels*. That is how the *uri\_query* fetches all relevant entities within the graph database. The *link\_query* then asks for all the links or edges between entities, forming a sort of adjacency list of connections. As shown in the *link\_query*, the parent and child values represent different entities that are connected via a *oneLink* or transitively via a *twoLinkOne*  $\rightarrow$  *tmp*  $\rightarrow$  *twoLinkTwo* path. Both direct paths and one hop transitive paths are required due to how SBOL structures its part  $\rightarrow$  sub-part usage relationships as *Component*  $\rightarrow$  *ComponentDefinition* paths in the graph.

The PageRank sparse matrix implementation is shown in Algorithm 1. To compute the PageRank vector, the algorithm initializes a vector  $q$  of length  $n$ , the number of nodes in the graph. Then, this vector is multiplied with the probability transition matrix  $P$  until it converges. This way, the only mutable state that has to be maintained in memory is the vector  $q$ . If the PageRank vector were calculated instead by taking the power of  $P$ , then  $P$  would quickly become dense and too large to fit in memory. Therefore, storing only  $q$  is necessary since storing a dense  $n * n$  matrix would exceed the memory on a single machine.  $P$  is stored efficiently and implicitly in the graph object by decomposing it into its relevant parts. Specifically,  $P$  is a  $n * n$  matrix where each column represents a node's outgoing links to other nodes. If node  $j$  links to node  $i$ , then  $P_{ij}$  will be  $\frac{1}{\text{numberOfOutgoingLinksForNode}j}$ .

```

1  uri_query:
2    PREFIX sbh: <http://wiki.synbiohub.org/wiki/Terms/synbiohub#>
3
4    SELECT DISTINCT ?subject
5    WHERE
6    {
7      ?subject sbh:topLevel ?subject
8    }
9
10
11 link_query:
12   PREFIX sbh: <http://wiki.synbiohub.org/wiki/Terms/synbiohub#>
13
14   SELECT DISTINCT ?parent ?child
15   WHERE
16   {
17     ?parent sbh:topLevel ?parent .
18     ?child sbh:topLevel ?child .
19
20     {
21       ?parent ?oneLink ?child
22     }
23     UNION
24     {
25       ?parent ?twoLinkOne ?tmp . ?tmp ?twoLinkTwo ?child
26     }
27   }

```

**Figure 3.2.** The SPARQL queries for extracting entity and entity relationship information from the Virtuoso graph database. The *uri\_query* query fetches all the unique URIs, and the *link\_query* query fetches all the entity relationship information. These two queries can be thought of as fetching all the vertices and all the edges of the graph. Together, the data returned from these queries is used to construct an adjacency list which is then turned into a probability transition matrix for use in the PageRank algorithm.

---

**Algorithm 1** PageRank using a sparse matrix representation of a graph
 

---

```

1: procedure PAGERANK(GRAPH, TOLERANCE)
2:   pageranks  $\leftarrow \left\{ \frac{1}{\text{graph.numberOfNodes}} \right\}_{\text{graph.numberOfNodes}}$ 
3:   delta  $\leftarrow \infty$ 
4:   while delta > tolerance do
5:     newPageranks  $\leftarrow \{0\}_{\text{graph.numberOfNodes}}$ 
6:     danglingContrib  $\leftarrow \frac{\text{sum}(\text{pageranks}[\text{graph.danglingPages}])}{\text{graph.numberOfNodes}}$ 
7:     teleportationContrib  $\leftarrow \frac{1}{\text{graph.numberOfNodes}}$ 
8:     for node in graph do
9:       linkContrib  $\leftarrow \text{sum} \left( \frac{\text{pageranks}[\text{inLink}]}{\text{graph.numberOfOutLinks}[\text{inLink}]} \text{ for } \text{inLink} \text{ in } \text{graph.inLinks}[\text{node}] \right)$ 
10:      newPagerank  $\leftarrow .85 * (\text{linkContrib} + \text{danglingContrib}) + .15 * \text{teleportationContrib}$ 
11:      newPageranks[node]  $\leftarrow \text{newPagerank}$ 
12:     newPageranks  $\leftarrow \frac{\text{newPageranks}}{\text{sum}(\text{newPageranks})}$ 
13:     delta  $\leftarrow \text{L1Norm}(\text{pageranks} - \text{newPageranks})$ 
14:     pageranks  $\leftarrow \text{newPageranks}$ 
15:   return pageranks

```

---

All other indices in column  $j$  are 0. Each time  $q_{t+1} = P * q_t$  is computed,  $P$  is also scaled by  $P_{scaled} = .85 * P + .15 * \frac{1}{n}$ . The full recurrence relation is shown in the equation below.

$$q_{t+1} = ((1 - \beta)P + \beta Q) * q_t$$

The scaling factor is denoted as  $\beta$  and is usually .15. The  $Q$  matrix represents a  $n * n$  matrix consisting of only entries with the value  $\frac{1}{n}$ . This scaling acts as the teleportation probability by augmenting the probability transition matrix to leak a small probability from a node to all other nodes in the graph. This also effectively makes the graph fully connected, which is integral to the Markov ergodicity requirement.

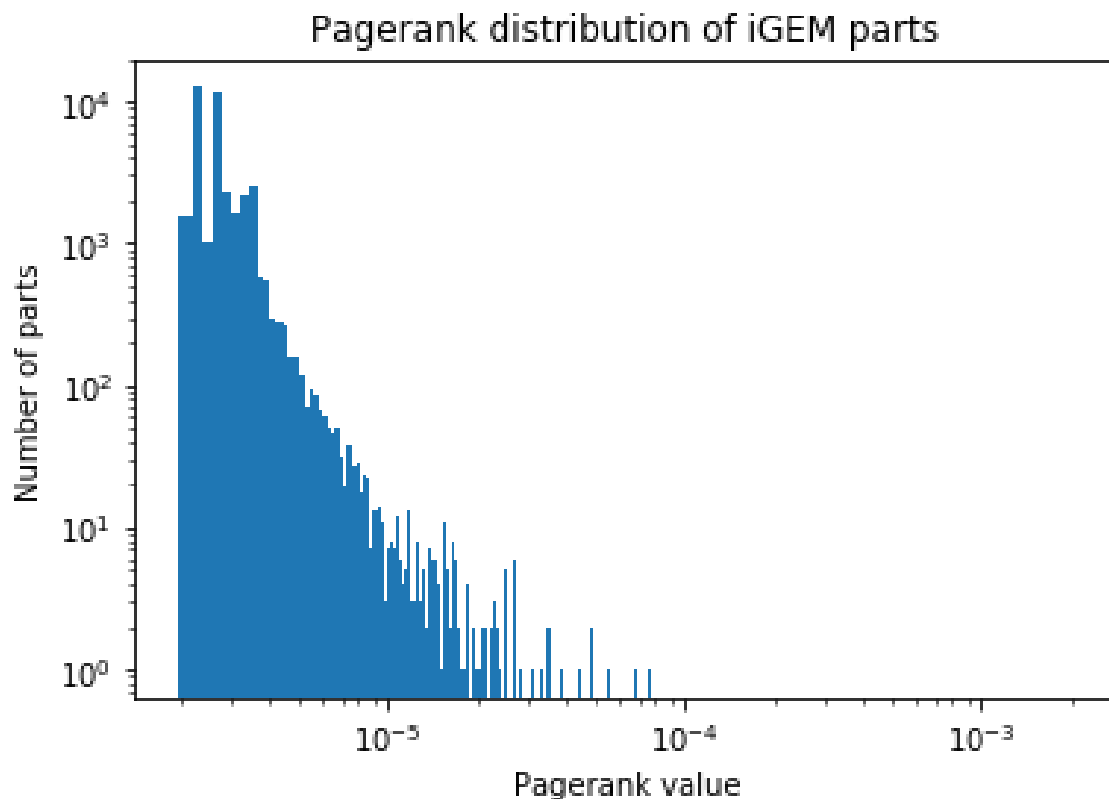
The key part of the algorithm is realizing that  $P$  can be decomposed into a set of dangling pages and linked pages [46]. By adding these decompositions up and weighting them according to the PageRank recurrence relation, the new PageRank value for a node in the graph can be computed. The dangling pages represent the nodes in the graph that have no outgoing links. When this happens, there should be a uniform probability of any page being visited next. The linked pages represent the set of possible pages the current page links to, and are needed to determine which pages should be non-zero in the probability transition matrix. Finally, the teleportation probabilities are represented by the  $Q$  matrix, which can be calculated since it only consists of  $\frac{1}{n}$  values. These representations are efficient since the dangling pages take  $\mathcal{O}(n)$  space, the linked pages take  $\mathcal{O}(E)$  space,

and the teleportation probabilities take  $\mathcal{O}(1)$  space. Since the graph is sparse,  $\mathcal{O}(E)$  tends to be close to  $\mathcal{O}(n)$ . As a result, this is much more efficient to store than a dense  $n * n$  matrix and enables PageRank on the iGEM dataset to be run on a single machine.

### 3.4 Analysis

Figure 3.3 shows the distribution of PageRank scores for all the parts in the iGEM dataset. The x-axis shows the PageRank value, and the y-axis shows the number of parts that have that value. Both the x-axis and y-axis are logarithmically scaled. The plot shows that there is an exponentially large number of low popularity parts, and an exponentially small number of high popularity parts. Many of the designs within the iGEM dataset are novel genetic circuits created by different teams competing in the iGEM competition. Each of these circuits uses more fundamental building block parts and isn't used by many other complete designs. As a result, the reuse of parts is focused on only a couple thousand more popular and well-understood parts. Consequently, these well-characterized parts can be found in many of the more complicated complete designs. This disparity in the different types of part reuse is characterized well by the PageRank distribution shown and is consistent with our intuition of the iGEM dataset.

Table 3.1 shows the top 10 most popular parts in the iGEM dataset by PageRank. Each part is identified by its unique URI. These URIs are computed by concatenating <https://synbiohub.org/public/igem/> with the part's *displayId*. Each part also has a role which defines the type of the part and a brief highlight of some more detailed information on what the part is. The parts are ordered by their PageRank values. Most of these parts are simpler building block parts that are used in many of the more complicated or complete composite genetic circuit designs. These parts are also popular due to their prolific use in the larger synthetic biology community. One possible cause of this could be the part's reliability, depth of understanding and characterization, and pragmatism when designing genetic circuits in the lab. Since these parts are reused in many other popular genetic designs, they benefit from PageRank's definition of popularity and easily rise to the top in many more broad search queries. Given the prevalence of these parts in the literature and other genetic designs, this is expected and intended behavior for a search engine and is consistent with what is expected from the iGEM dataset.



**Figure 3.3.** The distribution of PageRank values is shown. The x-axis shows the PageRank value, and the y-axis shows the number of parts which have that value. Note that both the x-axis and y-axis are logarithmically scaled. This means there is an exponentially large number of low popularity parts and an exponentially small number of high popularity parts. This is consistent with our intuition about the iGEM dataset. There are many parts, most of which are composite designs which are not used. However, there are a few basic building block parts which are well understood and used in many different designs. Note that the sum of all the PageRank values must add up to 1.



**Table 3.1.** A table of the top ten highest PageRank parts. Most of these parts are basic building blocks that can be found in many more complete composite genetic circuit designs. As such, they have high amounts of reuse and high PageRank scores. Note that the actual PageRank scores sum up to one since they represent a probability distribution over all the parts. The PageRank scores shown in the table are scaled for ease of comparison.

Table of top 10 most popular parts				
	URI	Role	Part information	PageRank
1	BBa_B0034	RBS	RBS based on Elowitz repressilator (Elowitz 1999).	2.0444828
2	BBa_B0012	Terminator	Transcription terminator for the E.coli RNA polymerase.	1.4841141
3	BBa_B0010	Terminator	Transcriptional terminator consisting of a 64 bp stem-loop.	1.4321551
4	BBa_B0015	Terminator	Double terminator consisting of BBa_B0010 and BBa_B0012.	0.7491525
5	BBa_R0040	Promoter	Sequence for pTet inverting regulator driven by the TetR protein.	0.4765939
6	BBa_E0040	CDS	Green fluorescent protein derived from jellyfish <i>Aequorea victoria</i> wild-type GFP (SwissProt: P42212).	0.4580074
7	BBa_B0030	RBS	Strong RBS based on Ron Weiss thesis.	0.3802979
8	BBa_B0032	RBS	Weak1 RBS based on Ron Weiss thesis.	0.3728527
9	BBa_R0010	Promoter	Inverting regulatory region controlled by LacI (lacI regulated).	0.3709392
10	BBa_R0011	Promoter	Inverting regulatory region controlled by LacI (lacI regulated, lambda pL hybrid).	0.2486734

### 3.5 Summary

Relative popularity between different parts is important knowledge to have when determining how to rank search results. PageRank is a canonical algorithm for finding the popularity of nodes in a graph and applies itself very well given the graph structure of SBOL encoded in RDF. The more parts that link to me, the more popular I am. The more popular the parts are that link to me are, then the even more popular I am.

After extracting an adjacency list representation of the graph from the Virtuoso graph database using SPARQL, a probability transition matrix is created. This matrix is kept sparse by decomposing it into simpler structures that are more memory efficient to represent. Then, the PageRank algorithm is run on the modified sparse matrix representation to generate a vector containing each part's global popularity. It turns out that the iGEM dataset contains an exponentially small number of high popularity parts and an exponentially large number of low popularity parts. In other words, the roughly 10 percent of parts which are widely used are actually popular, while the rest are hardly ever used. This makes sense given that the iGEM dataset contains both building block parts that are used by many designs as well as more complete genetic designs that are built up of many smaller parts. SBOLExplorer uses this popularity ranking along with clustering results as factors for deciding how to order parts in search results to user queries.

# CHAPTER 4

## CLUSTERING

Clustering is a useful way to extrapolate information from a dataset in an unsupervised manner. Genetic circuits can be thought of as a graph of interlinking parts, but the resulting graph appears more like a hairball and is difficult to reason about. A better aspect of data that could provide insights is each part's genetic sequence. This sequence data is further explored, and clustering techniques on sequence data are discussed in Section 4.1. Different implementations and their results on the iGEM dataset are discussed in Section 4.2. Experiments using different kinds of clustering techniques and distance metrics are performed. The iGEM dataset and our goal of searching through it are shown to be a unique use case compared to many other genetic sequence exploration goals. The chapter is summarized in Section 4.3.

### 4.1 Methods

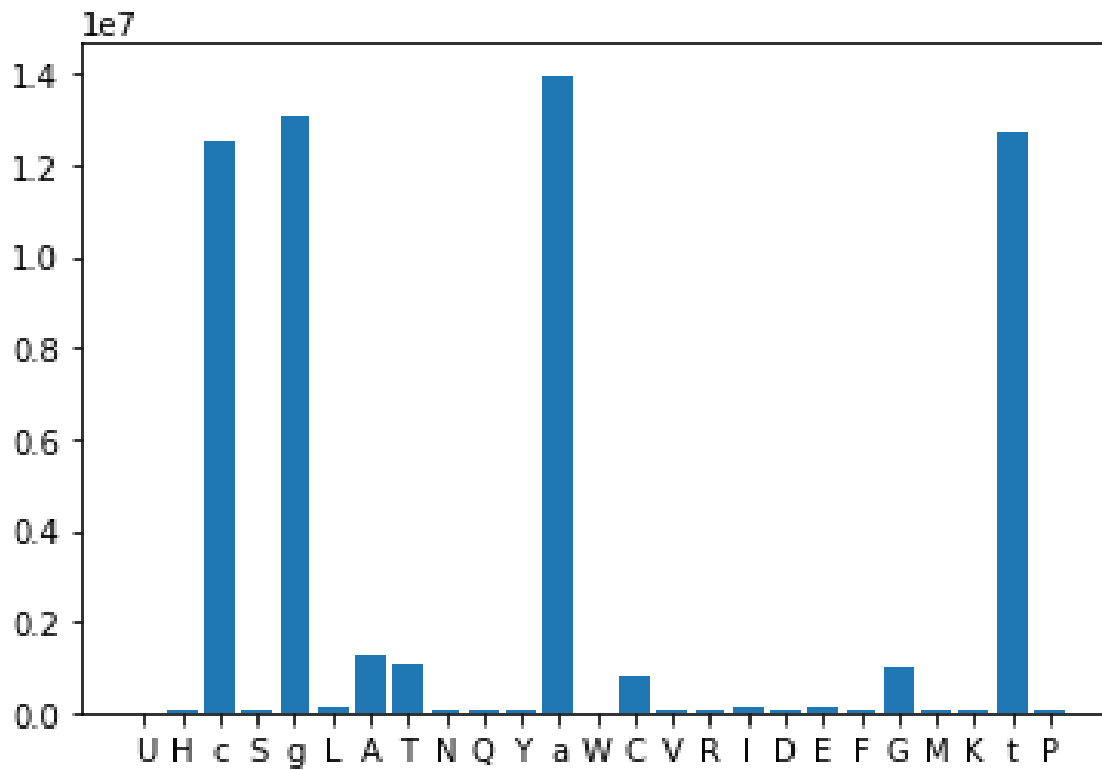
Similarity and clustering techniques can be applied to SynBioHub genetic part data to make it cleaner and easier to reason about. Of the over 50k parts, many are likely to be duplicates and similarity between parts is largely unknown. By clustering the parts, the dataset can be tidied up (merging duplicates, adding references/usages), and similarity between parts can be better understood [35]. Additionally, this genetic part data is unique because of its encoding in the SBOL data model and reliance on DNA sequence information.

Traditionally, sequence homology has been used to determine the shared ancestry in the evolutionary history of life [7]. Two sequences which are similar are more likely to share a common ancestor. These sequences are defined as long strings of base pairs where each base is represented by a letter in the alphabet. Guanine (G), adenine (A), thymine (T), and cytosine (C) are the physically present base pairs, with the other letters encoding for variations of one or more combinations of G, A, T, and C. The various combinations are

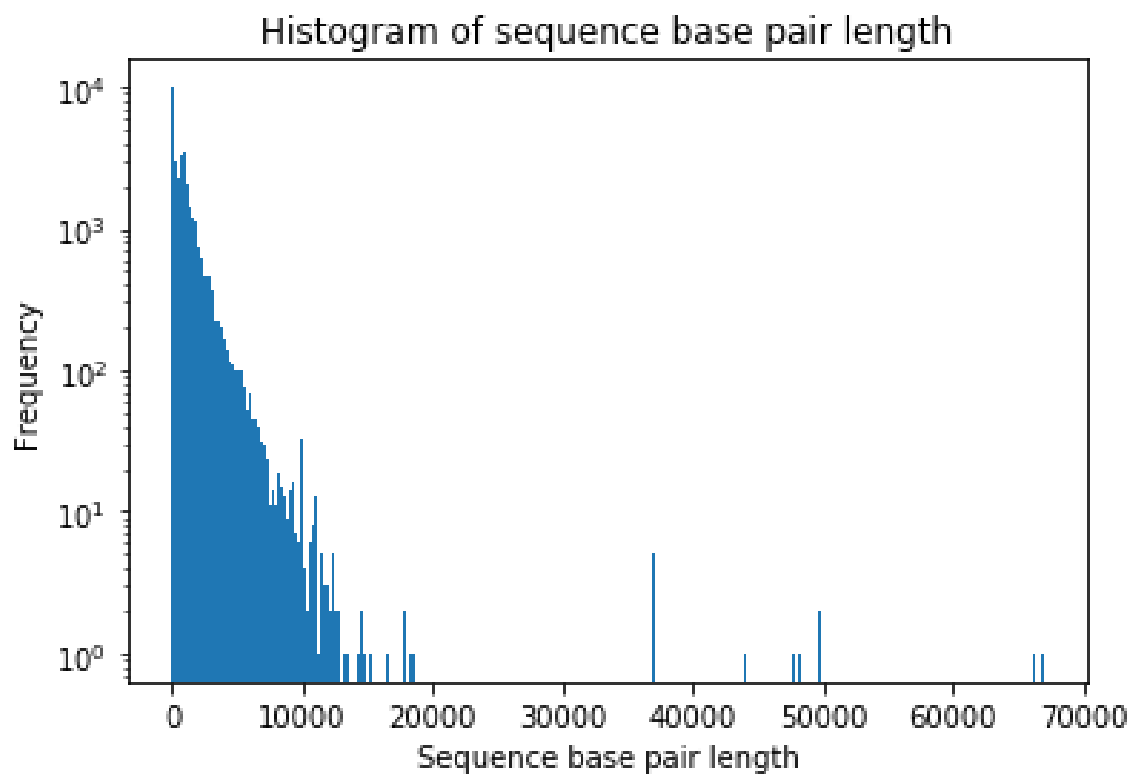
formalized by the *International Union of Pure and Applied Chemistry* (IUPAC) in the IUPAC nucleic acid notation. Figure 4.1 shows the base pair frequencies and Figure 4.2 shows the base pair length frequencies of the iGEM genetic part corpus in SynBioHub. Each part that has a specified sequence has an associated SBOL sequence object. The sequence object contains an elements string that is an encoded genetic sequence. These sequences are used to calculate part similarity and clusterings. By clustering similar genetic parts based off of sequence similarity, we are utilizing research in sequence homology to merge duplicated user submitted parts in SynBioHub.

Alignment-based methods and alignment-free methods are the two main ways for calculating sequence similarity [2, p. 398] [28]. Traditional alignment-based methods using Levenshtein or edit distance, which calculates distance as the number of insertions, deletions, and substitutions needed to transform one string into another string, give a good measure of global alignment [34]. Unfortunately, the optimal dynamic programming algorithm is  $\mathcal{O}(m * n)$  in time complexity where  $m$  and  $n$  are the lengths of the two strings. Due to the typically long nature of DNA sequences, this is not feasible. Other algorithms that focus more on local alignment, such as Smith-Waterman's local alignment score, also exist [58]. For certain tasks, local alignment can be a better measure of how similar two sequences are since it focuses on regions of similarity within long sequences that might overall be quite different, but it still has performance bottlenecks similar to global alignment. Alignment-free approaches apply techniques from data mining to speed up the similarity metric computation [4]. Rather than focusing on string comparison, alignment-free methods break each sequence into a feature vector. These compressed encodings of each sequence are then compared. Often, the feature vectors lose information relating to the exact position of bases. However, the computational speedup can be considerable.

The *basic local alignment search tool* (BLAST) approximates alignments that optimize a measure of local similarity, the maximal segment pair score [3]. By giving up precision and adopting smart heuristics, BLAST can be many orders of magnitude faster than traditional alignment-based similarity measures. BLAST is also very popular in the industry and has many variations that optimize for different use cases. Its searching and clustering performance has also been studied extensively and optimized [14]. BLAST+, a rewritten and repackaged version of BLAST with a new command line programmatic interface, is



**Figure 4.1.** The x-axis represents the base pairs found in the genetic part sequence data, and the y-axis represents how many instances of each base pair exists. There is no documented difference between the uppercase and lowercase base pairs. All base pairs other than G/g, A/a, T/t, C/c represent some combination of those base pairs. The IUPAC standard defines the combination.



**Figure 4.2.** The x axis represents the base pair lengths found in the genetic part sequence data, and the y axis represents how many instances of each base pair length exists.

especially useful for software applications that want to utilize BLAST [9].

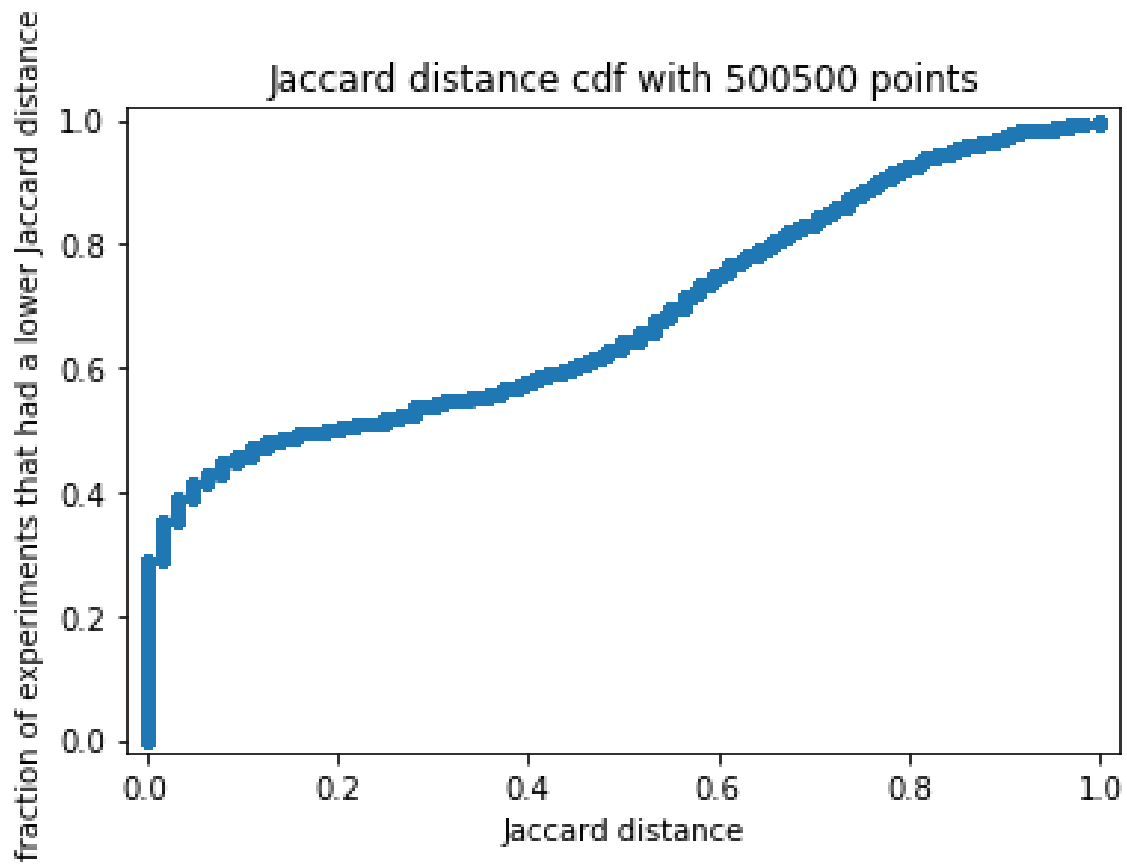
Initial analysis of the iGEM corpus using Jaccard distance shows that many pairs of sequences are very similar. Analysis of the distance distribution of a random subset of the data can inform how many clusters should exist. The cumulative density plot of the Jaccard distances is shown in Figure 4.3. Due to a large number of pairs with very low distance scores, there seem to be many similar pairs of sequences which should get clustered together. By analyzing this distribution, a distance cut off can be determined which will terminate the clustering. This serves as a proxy for determining how many clusters should exist, and makes sense given the core of the problem is to merge duplicate parts.

## 4.2 Implementations

Different clustering techniques and distance metrics are explored. Section 4.2.1 highlights a variant of hierarchical single link clustering that is tuned to the problem at hand. Both Jaccard distance and BLAST are experimented with as metrics. It turns out that both of these local alignment metrics have flaws given the iGEM dataset and our goals. Section 4.2.2 fixes these issues by using a greedy algorithm based off of global alignment. It turns out that UCLUST is much better suited to our goal of deduplicating sequences in the iGEM dataset.

### 4.2.1 Hierarchical Single Link Clustering

Hierarchical single link clustering is a natural approach to clustering genetic parts because of its simplicity, efficiency, and parallelizability [48, 56]. In every iteration, single link clustering always puts the two most similar parts in the same cluster. However, this requires computing the full cross product of all distances between pairs of parts, which becomes the bottleneck of this algorithm. Additionally, precomputing the full cross product can be impossible given a large enough number of parts because of main memory capacity limitations. Fortunately, hierarchical clustering gives hierarchical structure and deterministic clustering order, but the results are the same in any other clustering order as long as the end condition is identical. If our method determines the end condition using a distance threshold calculated using the distance cumulative density plot, then the end



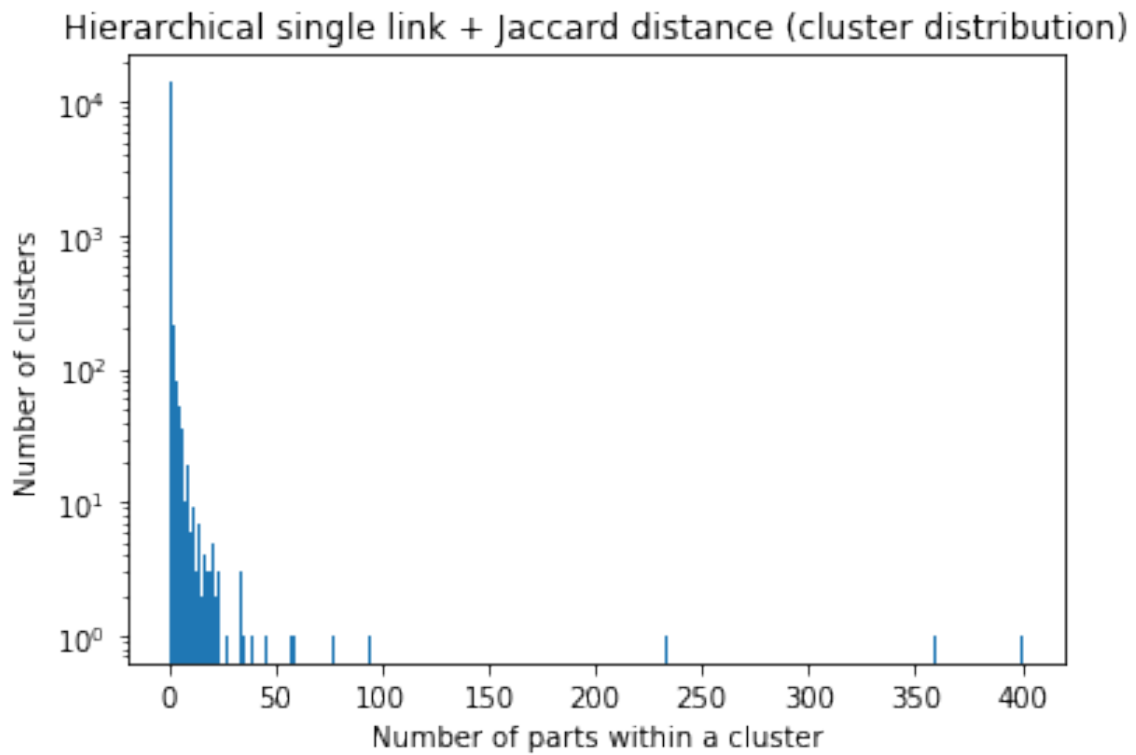
**Figure 4.3.** A cumulative density plot between all pairs of a random 1000 part subset is shown. There are many similar pairs of sequences which should get clustered together. This distribution can help inform a cut off for hierarchical clustering.



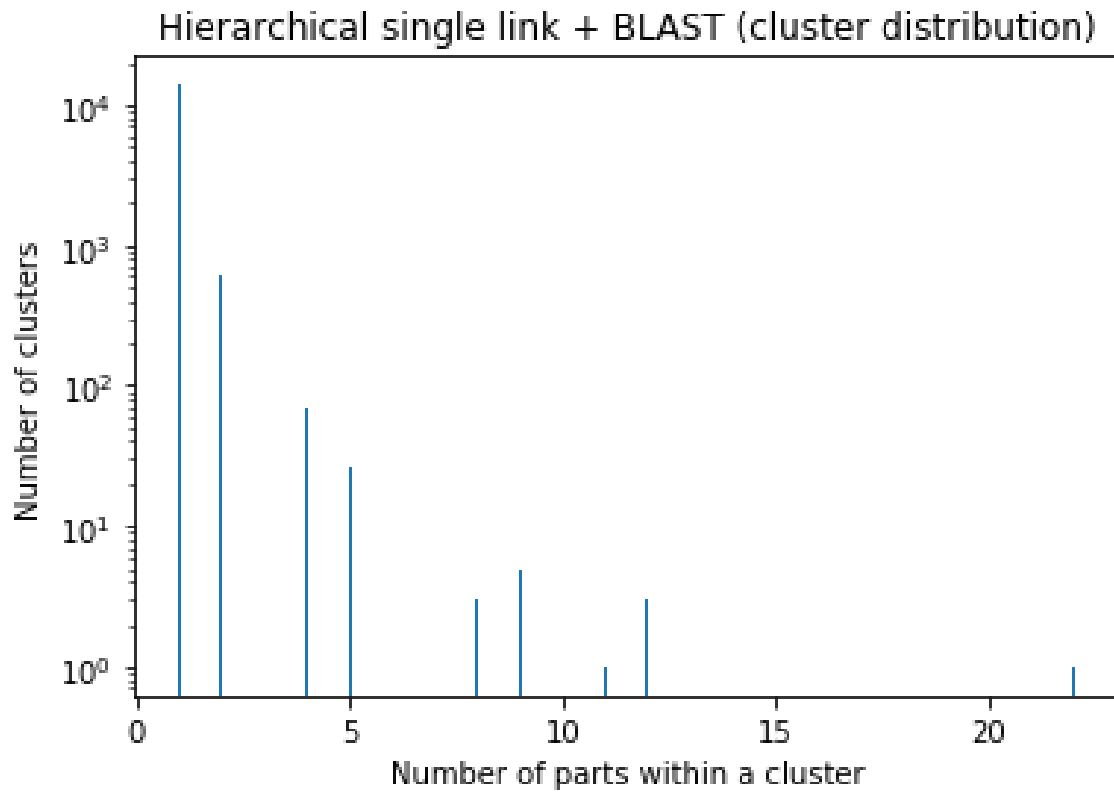
condition will be equal. By allowing an arbitrary clustering order, the hierarchy information used for determining phylogeny is lost. Thankfully, this is irrelevant for the purpose of merging duplicate parts. As a result, we can store the state of the clusters in a disjoint set data structure, and perform the clustering by streaming over the distances for many partitions of the original cross product in many parallelized threads. Each thread needs to perform the distance computation, and either union or reject depending on whether it exceeds the distance threshold. Naive hierarchical clustering has a time complexity of  $\mathcal{O}(n^3)$  and a space complexity of  $\mathcal{O}(n^2)$ . The time complexity of this algorithm is  $\mathcal{O}(n^2 \log^* n)$  where  $\log^* n$  is the iterated logarithm, the space complexity is  $\mathcal{O}(n)$ , and it has a high degree of parallelism.

For performance reasons, this algorithm was implemented in C++. Variants using Jaccard distance and BLAST were evaluated. The execution speed using the Jaccard distance was decent, with most runs finishing within 10 to 15 minutes. However, the produced clusters are not desirable. Many thresholds for both Jaccard and BLAST were compared, and all had drawbacks. Figure 4.4 shows the distribution of clusters that a run of the hierarchical single link clustering with a Jaccard distance threshold of 0.0125 produces. While most of the clusters are less than a couple hundred parts, and many clusters consist of only a single part, there was also one outlier cluster that contained around 30,000 parts. The smaller clusters are all quite meaningful, with many of them containing parts that share very similar sequences. Different Jaccard thresholds all share similar clustering distributions. This suggests that the Jaccard distance view of the dataset is not the best for grouping together a small amount of very similar sequences, which is what this dataset is assumed to be. The Jaccard distance is too sensitive to small similarities between distances that otherwise are not very similar. As a result, most of the dataset clusters into the one outlier cluster, which isn't very meaningful.

Figure 4.5 shows the distribution of clusters that a run of the hierarchical single link clustering using BLAST produces. BLAST was originally developed as a tool for searching databases of sequences for a match with a single query sequence. As such, BLAST is also very sensitive to whether a local alignment between the query sequence and a database sequence is possible. Most of the sequences are also either in clusters of size one or in a giant cluster that contained half the dataset. This is true even when varying the



**Figure 4.4.** A run of hierarchical single link clustering using a Jaccard distance threshold of 0.0125 is shown. The x-axis represents the number of clusters, and the y-axis represents the size of each of the clusters. One outlier cluster with around 30,000 parts is omitted. While the smaller clusters shown are meaningful, the large outlier cluster is not since it contains a large amount of the parts that share some small amount of similarity.



**Figure 4.5.** A run of hierarchical single link clustering using a BLAST distance threshold of 0.00001 is shown. The x-axis represents the number of clusters, and the y-axis represents the size of each of the clusters. Most of the parts are in many small clusters of a single element. One outlier cluster with around 30,000 parts is omitted. This outlier cluster is similar to the one found when using Jaccard distance.

threshold. The sequences in the large cluster have highly similar substrings with only a couple of base pair changes between a substring of the query sequence and a substring of the matched database sequence. As a result, any sequence that contains even a small substring match with the large cluster is clustered with the large cluster. Given BLAST's purpose of identifying local alignments with high accuracy, this is not the best way to view the dataset. Another drawback of BLAST was its relative computational inefficiency. Clustering the entire iGEM dataset took around an hour.

Jaccard distance represents an alignment-free technique, and BLAST represents a local alignment technique. Both of these views of the iGEM dataset are not suitable given their sensitivities to local or small similarities between sequences.

### 4.2.2 UCLUST

Given the experimental hierarchical single link clustering results shown in the previous subsection, a different clustering approach is needed. The UCLUST clustering algorithm is a greedy algorithm that defines each cluster using a centroid sequence and assigns new sequences to existing centroids using a greedy heuristic [14]. Specifically, UCLUST defines a global alignment threshold that it applies to compare a new sequence to all the existing centroids, and places the new sequence in the first centroid that falls within the threshold. If no existing centroids fall within the threshold, then the new sequence becomes its own centroid. As such, the input order with which the sequences get processed in is important. For this task, the sequences are sorted by decreasing length so longer sequences are processed first, and are therefore more likely to become centroids.

UCLUST can get much better results than hierarchical single link clustering for the iGEM dataset. Computational performance is on the order of hierarchical single link clustering using Jaccard distance and is much faster than the BLAST variant. Due to the use of global alignment, the generated clusters are also a lot more meaningful. Each sequence in a reported cluster has a high degree of global cohesiveness with all other sequences in the cluster. A vital part of this is how alignment is defined. The alignment identity is computed as the number of matching letters divided by the length of the shorter sequence. While this is more imprecise than something like edit distance, it is also a lot more efficient to compute while retaining some similar important characteristics. Specifically, the sequences in a

generated cluster all have to be globally similar, both in pairwise base pairs and overall sequence length.

Figure 4.6 shows the output of UCLUST clustering on the entire iGEM collection, and Figure 4.7 shows the distribution of clusters produced by UCLUST. Like the other clustering approaches, most of the sequences are in clusters by themselves. However, all sequences that are clustered together have a high degree of global alignment with the average hit identity being 99.5025, and there is not a single outlier cluster that contains most of the sequences.

### 4.3 Summary

Our goal for clustering was to have a sort of fuzzy deduplication of sequences within the iGEM dataset. There are many parts within the iGEM dataset, but many of these parts are slight variations of other more popular parts. Therefore, to provide a diversity of search results for queries, it's important we have a good understanding of which parts are unique and which parts should be ranked highly in the search results. Having only slight variations of the same part appear on the first page of search results does not provide a good user experience.

To this end, different clustering techniques and distance metrics are explored. Through this process, the best formulation and view of the dataset for SBOLEplorer is understood. Specifically, the goal of deduplicating the iGEM dataset lends itself to global alignment algorithms. This is in contrast with the traditional clustering formulation of determining phylogeny and therefore requires different tools compared to more traditional clustering approaches in this field. Since SBOLEplorer should deal with an incrementally changing dataset, performance is also considered. The result is the use of the UCLUST algorithm.

```

-bash-4.3$ ./usearch10.0.240_i86linux32 -cluster_fast ../ingest/synbiohub.fsa -id 0.75
                                         -sort length -uc uclust_results.uc
usearch v10.0.240_i86linux32, 4.0Gb RAM (16.4Gb total), 4 cores
(C) Copyright 2013-17 Robert C. Edgar, all rights reserved.
http://drive5.com/usearch

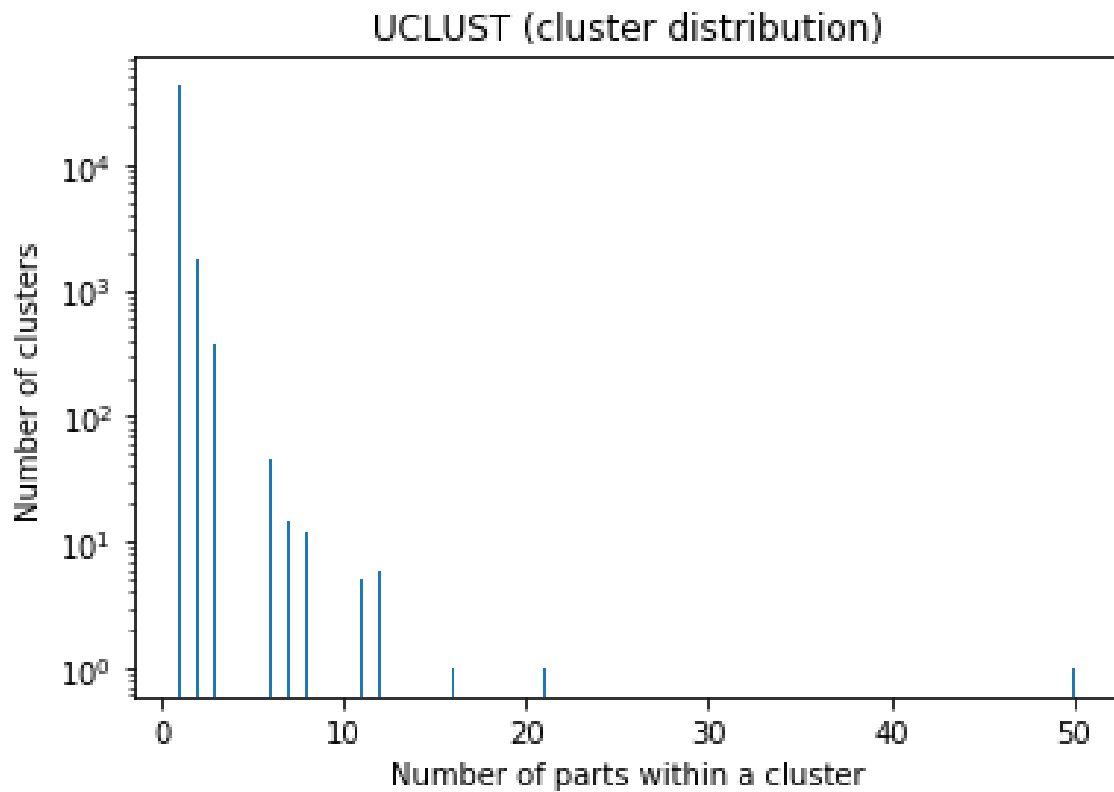
License: michael13162@gmail.com

00:00 101Mb   100.0% Reading ../ingest/synbiohub.fsa
00:01 99Mb    100.0% DF
00:01 100Mb  48548 seqs, 45116 uniques, 42935 singletons (95.2%)
00:01 100Mb  Min size 1, median 1, max 50, avg 1.08
00:01 102Mb  100.0% DB
00:01 103Mb  Sort length... done.
09:58 243Mb  100.0% 44308 clusters, max size 50, avg 1.1

      Seqs 45116 (45.1k)
      Clusters 44308 (44.3k)
      Max size 50
      Avg size 1.1
      Min size 1
Singletons 41724 (41.7k), 92.5% of seqs, 94.2% of clusters
      Max mem 243Mb
      Time 09:57
Throughput 75.6 seqs/sec.

```

**Figure 4.6.** A run of UCLUST clustering using an alignment threshold of 0.75 is shown. Most of the parts are in many small clusters of a single element, and there are no outlier clusters with most of the elements. All sequences within a cluster have a high degree of global similarity. Out of 48548 sequences, there are 45116 unique sequences, and 44308 clusters. The average hit identity is 99.5025.



**Figure 4.7.** The distribution of cluster sizes produced by the UCLUST algorithm is shown. The x-axis represents the number of clusters, and the y-axis represents the size of each of the clusters. Most of the parts are in many small clusters of a single element, and there are no outlier clusters with most of the elements.

## CHAPTER 5

### DATA INFRASTRUCTURE

In this chapter, SBOLEplorer's data infrastructure is explained. SBOLEplorer relies on a conceptually simple data structure for retrieving search results. Section 5.1 details this data structure, the inverted index. Section 5.2 discusses how the inverted index is concretely used, the pipeline that feeds it data, and how the results of querying the index get transformed into the final search results. This process combines the insights from both clustering, PageRank, and keyword score to make an informed decision on what ordering of results the user should be presented back with. The final search results are analyzed using a simple metric. Section 5.3 describes how SBOLEplorer fits into the overall workflow and how it interacts with different parts of the system. SBOLEplorer's architecture and distributed search functionality are also described. Finally, Section 5.4 summarizes this chapter.

#### 5.1 Inverted Index

To allow for efficient real-time queries, SBOLEplorer must leverage data infrastructure built for the task. Specifically, an inverted index data structure can be used to speed up the generation of search results [59]. The inverted index maps part metadata and part keywords to a set of parts that contain the metadata or keyword. This set of parts is called the postings list. The resulting parts are then ordered with input from the PageRank ranking, the clustering results, and the query relevance. Upon a query from SynBioHub or another genetic design repository, SBOLEplorer packages the search results up and returns them to be displayed to the user or software tool.

The traditional index data structure has been used in database applications to enable the efficient retrieval of documents given the document identifier. Some examples of popular indices include B-trees, hash tables, and skip lists [10, 33, 51]. All of these data structures map document identifiers to the contents held in the document or the documents



themselves. An inverted index does the opposite; it maps document contents to document identifiers. The inverted index enables a user to submit a query based on the content they are looking for and efficiently find all the documents that hold this content. Inverted indices have found use in many search and general document retrieval applications [59]. This includes querying RDF data, which is what SBOL is serialized in [24].

It is useful to be able to find genetic parts by searching for their characteristics. Instead of doing a full-text string comparison search using the names and descriptions of all parts in the database, an inverted index refers to its postings list for each keyword in the given query. Note that this is a different kind of search compared to what is often found in bioinformatics. Instead of the query being a genetic sequence that the user wants matched, the user is searching by keyword to identify a set of relevant genetic parts in a graph of genetic designs. Because searching a prebuilt index by keyword is a boost in efficiency compared to searching by performing many string comparisons, additional genetic part metadata beyond name and description can be considered in searches. Due to the limited textually descriptive metadata found within user submitted parts in SynBioHub, it is useful to also rely on data analysis such as PageRank and clustering in constructing the search results for a given query.

## 5.2 Search Implementation

Apache Lucene is a library that provides a powerful and fast information retrieval engine built on top of the inverted index data structure that developers can easily incorporate into their projects [39]. Elasticsearch uses Apache Lucene under the hood to provide a distributed search engine complete with sharding (horizontal partitioning of data over many machines to spread the load) and an easy to use *representational state transfer* (REST) *application programming interface* (API) [22]. Due to the benefits of having a clean API that takes away the need to manage the index, Elasticsearch is used by SBOLEplorer to answer search queries quickly. Also, if needed, Elasticsearch can be used to facilitate the painless addition of more machines to the inverted index cluster. A separate SBOLEplorer server provides a simple API to SynBioHub or any other genetic part repository. Whenever a user submits a search query to SynBioHub, SynBioHub needs to query the SBOLEplorer server's search endpoint, and SBOLEplorer consults Elasticsearch and the PageRank and

clustering results to return SynBioHub a list of search results sorted by relevance.

To populate the inverted index, SBOLEplorer needs to ingest data into Elasticsearch. SBOLEplorer gets its data from the Virtuoso graph database, clusters the sequences to get an idea on similarities between parts, runs the PageRank algorithm to get an idea on different part's relative popularities, and then ingests the amalgamated results into Elasticsearch.

The data that is indexed also includes searchable metadata about the part including *subject* URI, *displayId*, *version*, *name*, *description*, *type*, and *graph*. Figure 5.1 shows the SPARQL query used to get all this metadata. Each entity has a *subject* which is its URI. The URI has a *graph* predicate with the graph in Virtuoso that the entity belongs in. This is used as a form of *access control list* (ACL) and is used to implement permissioned search. There is a public graph that every user can access, and each user has his or her own private graph which contains parts that only they have access to. SynBioHub handles user authentication and tells SBOLEplorer what graphs to search in. The *from* line is used to specify specific graph URIs, and the *criteria* line is used to filter parts by other criteria further. Being able to filter by other criteria such as strict field matching or collection inclusion is used to implement advanced search. The *displayId*, *version*, *name*, *description*, and *type* metadata are all fields that users' queries search through. An entity's *displayId* and *name* differ in that the *displayId* is a part of its unique identifier, and the *name* is a regular human given name for the part. *Version* specifies a part's version, and *type* specifies the precise type of part, which can be DNA, RNA, protein, small molecule, etc. Note that the actual genetic sequence is not present in the index. While this could be supported, it would increase the memory footprint of Elasticsearch by a significant amount, and other tools like BLAST exist that provide search capabilities for genetic sequences.

Elasticsearch provides a *domain specific language* (DSL) for querying the inverted index. Figure 5.2 shows how SBOLEplorer constructs a query using this DSL. The query is serialized as a *JavaScript Object Notation* (JSON) payload to Elasticsearch's RESTFUL API. The main *query* block wraps a *function\_score* block that orders search results using a custom script. The script is defined in the *script\_score* block and specifies a function to be run on all results. This function calculates a score that depends on how relevant the result is to the search query and what the PageRank of the result is. The score is used by Elasticsearch to

```

1 part_query:
2   PREFIX dcterms: <http://purl.org/dc/terms/>
3   PREFIX sbh: <http://wiki.synbiohub.org/wiki/Terms/synbiohub#>
4   PREFIX sbol2: <http://sbols.org/v2#>
5
6   SELECT DISTINCT
7     ?subject
8     ?displayId
9     ?version
10    ?name
11    ?description
12    ?type
13    ?graph
14  <from>
15  WHERE {
16    <criteria>
17    ?subject a ?type .
18    ?subject sbh:topLevel ?subject .
19    GRAPH ?graph { ?subject ?a ?t } .
20    OPTIONAL { ?subject sbol2:displayId ?displayId . }
21    OPTIONAL { ?subject sbol2:version ?version . }
22    OPTIONAL { ?subject dcterms:title ?name . }
23    OPTIONAL { ?subject dcterms:description ?description . }
24  }

```

**Figure 5.1.** The SPARQL query used to fetch part metadata. This data is downloaded from the Virtuoso graph database and ingested into the inverted index along with PageRank results. Clustering results stay on SBOLEplorer. PageRank results are included in Elasticsearch due to its custom script scoring feature. The *subject* URI, *displayId*, *version*, *name*, *description*, and *type* metadata are all searchable fields. Depending on the logged in user, different sets of graphs are substituted in to the from line, which acts as a form of ACL.

```

1  elasticsearch_query = {
2    'query': {
3      'function_score': {
4        'query': {
5          'multi_match': {
6            'query': es_query,
7            'fields': [
8              'subject',
9              'displayId^3',
10             'version',
11             'name',
12             'description',
13             'type',
14             'keywords'
15           ],
16           'operator': 'or',
17           'fuzziness': 'AUTO',
18         }
19       },
20       'script_score': {
21         'script': {
22           # Math.log is a natural log
23           'source': "_score * Math.log(doc['pagerank'].value + 1)"
24         }
25       }
26     }
27   },
28   'from': 0,
29   'size': 10000
30 }

```

**Figure 5.2.** The query encoded in Elasticsearch’s DSL that is sent by SBOLEplorer to Elasticsearch’s RESTFUL API when a genetic design repository issues SBOLEplorer a search request. The query string is broken down into keywords that have to fuzzily match at least some of the fields in the part. All the parts that match at least one token are *ored* together. A custom function is defined in the *script\_score* block that orders results by their keyword score and PageRank value.

sort the results. The formula for calculating scores is shown below.

$$score_{part} = keywordScore_{part} * \ln(pagerank_{part} + 1)$$

The overall score is effectively the keyword score scaled by the natural logarithm of the PageRank. A value of 1 has to be added to the PageRank value since its normal values are between 0 and 1 which would result in a negative value after taking the logarithm. The keyword score is dependent on how similar Elasticsearch thinks the part's metadata fields are to the query string. Any field that matches any token in the query string gets included since the query is using the *or* operator. A *fuzziness* of *AUTO* means that even tokens similar to one of the keywords generate a match.

Having a diversity of results in the top results for a search query is useful in order to not let the first page of results get dominated by many variants of the same part. To accomplish this, the results from clustering are used. Before returning the ordered results from Elasticsearch, SBOLEplorer modifies each of the scores by keeping a hash table of cluster duplicates. Every time a part is emitted to the final results list, all the other parts in the cluster that the emitted part belongs to get added to the cluster duplicates set. Because many clusters only contain a single part, this hash table should not get too large. Now, whenever another part is added, SBOLEplorer halves that part's final score if it can be found in the cluster duplicates. Note that this must be done outside of Elasticsearch since Elasticsearch does not support arbitrary functions in its *script\_score* block. As a result of this postprocessing of the search results list, the highest ranked part determined by the keyword score and PageRank value shows up as it normally does in the list order, and any other parts in that cluster that also get matched by the query show up much lower down. This preserves the top spots of the list for parts which are all in different clusters. However, since the cluster duplicate score is only halved, it is still possible for it to beat out other parts from clusters that are not represented yet if it has a much higher keyword score or PageRank value.

### 5.2.1 Metrics

The resulting search quality is measured using a simple metric. Each part has a ranking in the search results. This ranking should ideally be ordered by how likely the user wants the part given their query. Using SBOLEplorer, the average ranking of parts for a given

query should be higher than the average ranking when doing arbitrary string matching in Virtuoso. Unfortunately, it is difficult to find an unbiased and algorithmic way for determining which part a user is looking for. Therefore, a close approximation using composite parts is used.

Composite parts are genetic circuits that represent a complete design and are built out of smaller sub-parts. These parts are usually represented using two levels of hierarchy. The root level part represents the design as a whole and includes a description of the entire composite part. The next level down is a sequence of smaller sub-parts which represent the actual building blocks of the design. A user search is approximated by using the root level part's description as the search query. Since the description describes the entire genetic design, it makes sense for it to describe somewhat the building block sub-parts the design is comprised of.

The metric is calculated by searching on the root level parts' description and looking at how far down each of the sub-parts are in the search results. A better overall ranking should have the sub-parts showing up higher in the search results. The average and median of the sub-part rankings are computed. Higher quality search results should result in a smaller average and median, and lower quality search results should result in a larger average and median. This process is done for 5,000 composite parts that include 22,972 sub-parts. Without SBOLEplorer, only 3,913 of these sub-parts even show up in the search results. Furthermore, the average position in the search results for those found is 1,639, while the median position is 974. With SBOLEplorer, 10,564 sub-parts are included in the search results. Furthermore, the average position in the search results improves to 138 with a median position of 8. These results are very encouraging since the description quality in the iGEM dataset is very uneven. In some cases, they are very lengthy and can be of limited relevance to the actual keywords of the sub-parts, while in others they can be very sparse in details. If a real human user would type a more succinct and direct query, we would expect even better results.

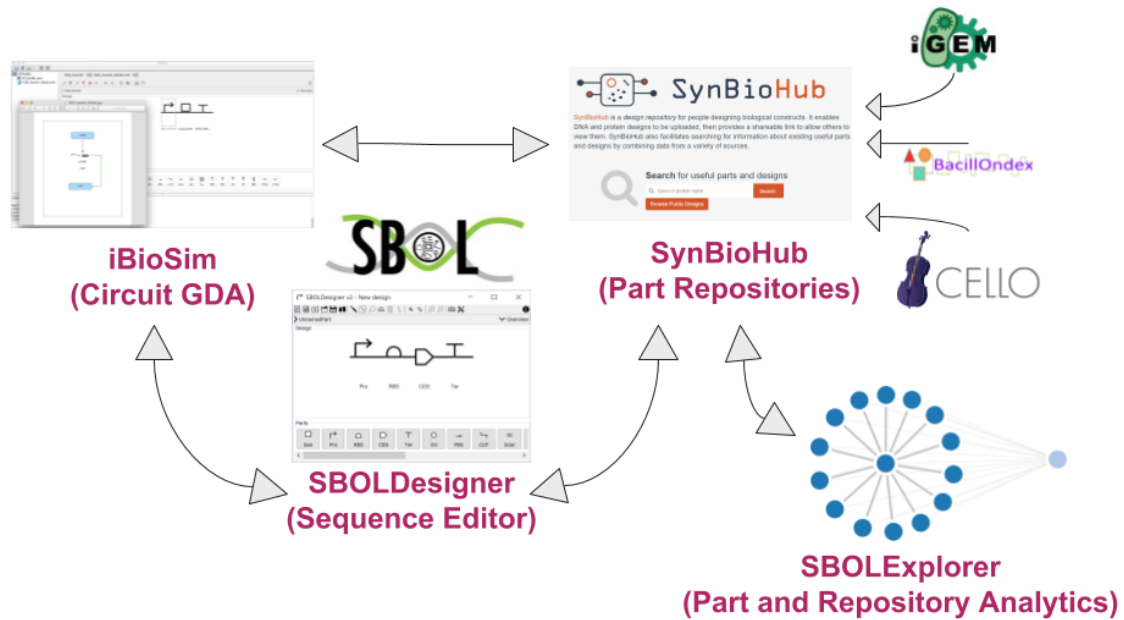
### **5.3 Workflow and System Design**

It is much easier to understand genetic design repository data after SBOLEplorer has processed it. Therefore, SBOLEplorer embeds itself nicely within the overall SBOL

workflow. As shown in Figure 5.3, experimental biologists use SBOLEplorer implicitly while browsing for parts in SynBioHub. Not only does this allow for a more efficient and thorough browsing experience through all relevant parts, experimental biologists are also able to better understand the context of their parts and its usages by other people. Users of SBOLDesigner, a sequence editing tool, also benefit by being able to refer to SBOLEplorer for their part searching and part selecting [47, 62]. In both cases, the search results are presented in an order that better represents what the user is looking for.

Figure 5.4 shows the result of searching for a ribosome binding site in SynBioHub. Clicking on a search result takes the user to a page with more information on the part. Figure 5.5 shows the result of performing the same search in SBOLDesigner. Clicking on a search result here lets the user download the specified part and place it into their genetic circuit design. In both cases, a query is sent from SynBioHub's backend server to SBOLEplorer. SBOLEplorer consults Elasticsearch and computes a final ranking weighted by the keyword score, PageRank, and clustering results.

Figure 5.6 shows how SBOLEplorer architecturally integrates into this workflow. SynBioHub provides a web application for users and a programmatic API for software tools such as SBOLDesigner, iBioSim, and Benchling. The genetic part data that SynBioHub serves is stored in a graph database called Virtuoso. Whenever a user or software tool issues a search, SynBioHub passes along the search query to SBOLEplorer for evaluation. SBOLEplorer returns a list of search results after consulting Elasticsearch and its internal data structures. Depending on the type of query, SBOLEplorer may also need to request data from Virtuoso. In this case, the responses to these queries are memoized in a *least recently used* (LRU) cache on SBOLEplorer's side. The search results consist of a list of URIs that uniquely identify a particular genetic part stored within Virtuoso. When the user selects a part to download or view in more detail, SynBioHub fetches the part from Virtuoso. Periodically, SBOLEplorer fetches newly submitted parts and performs clustering, graph analysis, and finally data infrastructure ingestion. This batch update and its periodicity are controlled by a cron job (a time-based job scheduler). SBOLEplorer itself consists of two processes. One is a *hypertext transfer protocol* (HTTP) server that responds to requests from SynBioHub or other genetic design repositories, and the other is Elasticsearch. Both of these processes are deployed inside of Docker containers along with



**Figure 5.3.** A workflow consisting of SBOLExplorer, SBOLDesigner, SynBioHub, and iBioSim [38]. Genetic parts from various databases are hosted in the SynBioHub repository [23,37,41,43,45]. SBOLDesigner and iBioSim can download these parts and use them to construct complete genetic designs. Specifically, iBioSim takes care of modeling and simulation, and SBOLDesigner takes care of sequence-level design. Throughout the process, parts and their relationships are understood through SBOLExplorer’s search service. Note that the user does not interact directly with SBOLExplorer. It is a backend service that provides search functionality for SynBioHub. Figure modified from Zhang et al. [62].



Search results for 'rbs' in SynBioHub. The search bar contains 'rbs' and a 'Search' button. Below the search bar, there are links for 'Advanced Search | Create Collection | SPARQL' and 'Showing 1 - 50 of 4907 result(s)'. A pagination bar shows '1 2 3 4 5 Next'. Three search results are displayed, each with a 'PUBLIC' badge in the top right corner:

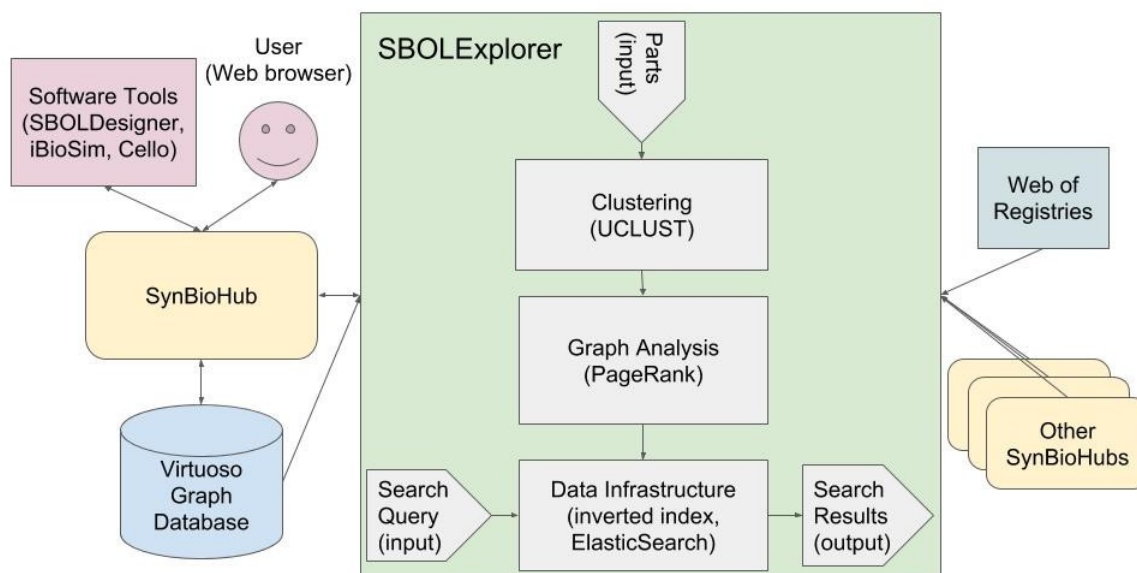
- BBa\_B0034**  
version 1  
RBS (Elowitz 1999) -- defines RBS efficiency
- BBa\_B0032**  
version 1  
RBS.3 (medium) -- derivative of BBa\_0030
- BBa\_B0030**  
version 1  
RBS.1 (strong) -- modified from R. Weiss

**Figure 5.4.** The results of searching for a ribosome binding site in SynBioHub. *BBa\_B0034* is immediately recognizable as the first search result, and is well known as the de facto Elowitz ribosome binding site. Users can click on a search result and view more details on the specified part.

SBOLDesigner v3 - New design interface. The design area shows a schematic with components: Pro, RBS, CDS, and Ter. A 'Part: RBS' dialog box is open, showing fields for Part type (DNA), Part role (RBS (Ribosome Binding Site)), Role refinement (None), Display ID (RBS), Name, and Description. A 'Select a part from registry' dialog box is also open, showing a search for 'rbs' in the 'local (http://localhost:7777/)' registry. The search results table is as follows:

Type	Display Id	...	Version	Description
Collection	categories	ig...	1	
Part	BBa_B0034	B...	1	RBS (Elowitz 1999) -- defines RBS efficiency
Part	BBa_B0032	B...	1	RBS.3 (medium) -- derivative of BBa_0030
Part	BBa_B0030	B...	1	RBS.1 (strong) -- modified from R. Weiss
Part	BBa_B0031	B...	1	RBS.2 (weak) -- derivative of BBa_0030
Part	BBa_B0033	B...	1	RBS.4 (weaker) -- derivative of BBa_0030
Part	BBa_K143021	R...	1	SpoVG ribosome binding site (RBS) for B. sub...
Part	BBa_K1362090	T...	1	strong T7 RBS
Part	BBa_J44001	B...	1	Reverse RBS (RBS<sub>rev</sub>) -- corre...
Part	BBa_K165002	B...	1	Kozak sequence (yeast RBS)
Part	BBa_K143020	R...	1	ColR ribosome binding site (RBS) for R. eublic...

**Figure 5.5.** The results of searching for a ribosome binding site in SBOLDesigner. Again, *BBa\_B0034* is immediately recognizable as the first search result. Users can click on their desired part to download it and place it into their design.



**Figure 5.6.** The architecture for SBOLEplorer and its relationship with SynBioHub is shown. SynBioHub has a web application for users using web browsers and a programmatic API for software tools. These interfaces allow users to interact with the data stored in Virtuoso, a graph database. SBOLEplorer's responsibility is to handle the search functionality that SynBioHub provides. By consulting its inverted index, search results that span the entire Virtuoso graph database can be returned in real time. New parts are also periodically ingested into the data infrastructure after clustering and graph analysis. Therefore, not only are searches quick to execute, they are also sorted by relevance and show a diversity of parts. SynBioHub then fetches the specifically requested part directly from Virtuoso if a user or software tool asks for it.


SynBioHub and Virtuoso.

SynBioHub and other genetic design repositories interact with SBOLExplorer through its API. The default */search* endpoint takes in a query string and returns the output results. However, to make integration with SynBioHub easier, SBOLExplorer can also be passed the SPARQL query that SynBioHub normally sends to Virtuoso. SBOLExplorer extracts the relevant query properties and figures out how best to utilize Elasticsearch, its internal data structures, and Virtuoso to best answer the query. For example, SBOLExplorer supports searching for similar parts, uses of a particular part, twin parts, and other advanced searches in addition to providing the normal keyword search functionality. Endpoints also exist for getting information on the state of SBOLExplorer, updating its index, incrementally updating its index, and updating its configuration. Figure 5.7 shows SynBioHub's administration page for SBOLExplorer. This page has information on whether SBOLExplorer is healthy and gives access to configurable fields for various parameters and endpoints. For example, SBOLExplorer's PageRank tolerance and UCLUST clustering identity can be tuned.

### 5.3.1 Distributed Search

Each institution usually sets up its own instance of SynBioHub. The genetic design data is stored in a managed genetic part repository in a standardized format, but the data is spread out over many locations. This makes it difficult to query and search through all SBOL parts.

A unique feature of SBOLExplorer is distributed search. Distributed search allows SBOLExplorer to perform its batch clustering, graph analysis, and data infrastructure ingestion by fetching parts from all SynBioHubs that it knows of. In normal search, only the local SynBioHub and Virtuoso instances are consulted. However, with distributed search turned on, SBOLExplorer first queries Web-of-Registries for a list of all registered SynBioHub instances, and then performs its update by fetching all the parts from all the instances that Web-of-Registries knows of. Web-of-Registries manages SynBioHub federation by indexing all known instances of SynBioHub and informing them about each other. This federation service permits powerful federated querying and data storage of SBOL parts. Since each SBOL part is uniquely identified by its URI, it is guaranteed that

 Note: Looks like SBOLEplorer is up and running :) Endpoint fields should end with '/', SynBioHub Public Graph should end with '/public', and SPARQL/Virtuoso Endpoint should end with '/sparql?'.

**SBOLEplorer Endpoint**   Searching Using SBOLEplorer

Update Index Right Now (check the box and click save, will take a couple minutes)

Use Distributed Search

**Pagerank Tolerance**

**UClust Clustering Identity**

**SynBioHub Public Graph**

**Elasticsearch Endpoint**  **Elasticsearch Index Name**

**SPARQL/Virtuoso Endpoint**

**Figure 5.7.** The administration page for SBOLEplorer in SynBioHub. This page lets administrators of SynBioHub configure SBOLEplorer, check its health, initiate an index update, and toggle whether to use SBOLEplorer and its distributed search functionality. SBOLEplorer's PageRank tolerance, UCLUST clustering identity threshold, SynBioHub public graph, Elasticsearch endpoint, Elasticsearch index name, and Virtuoso endpoint are configurable. The configuration state is communicated through SBOLEplorer's */config* endpoint.

each part is a unique SBOL part as long as SBOLEplorer filters the merged dataset for duplicate URIs.

SBOLEplorer can uniquely provide efficient search over all SynBioHubs due to its preprocessing. Clustering and PageRank are computed over all the SBOL parts stored in all SynBioHubs, and the results are loaded into the local Elasticsearch. Now, if a user wants to query over all SynBioHubs, a single request to the local SBOLEplorer instance is all that is required. Before SBOLEplorer's distributed search capabilities, the query would have to be sent to each SynBioHub individually, and the results would have to be merged. Now, all the necessary data for search is already efficiently stored in the local SBOLEplorer along with PageRank and clustering results over the entire graph. The complete SBOL data can be downloaded from the actual SynBioHub instance it belongs in by following the part's URI. While the update process of distributed search takes a lot longer since all parts from every SynBioHub must be fetched, and clustering and PageRank computations are performed over a much larger combined dataset, the efficiency benefits when searching over all parts is substantial. However, distributed search's index is not incrementally updated since it is polling for the data in batches. Therefore, newly submitted parts to non-local SynBioHubs are not reflected in the search results until the next update is run. In this way, SBOLEplorer can be thought of as a cache of parts that gets updated in batches.

Another benefit of distributed search is its ability to bootstrap a newly initialized instance of SynBioHub. New SynBioHub instances may have limited amounts of part data and are therefore of limited utility. By activating distributed search, even an empty instance of SynBioHub can have access to all the public part data of all other SynBioHubs. For example, with distributed search, the University of Utah SynBioHub instance (<https://synbiohub.utah.edu>) can search through parts from the NSF Living Computing Project SynBioHub (<https://synbiohub.programmingbiology.org>), Boston University CIDAR Lab SynBioHub (<https://synbiohub.cidarlab.org>), and reference instance of SynBioHub hosted by Newcastle University (<https://synbiohub.org>).

## 5.4 Summary

The use of an inverted index through Elasticsearch allows SBOLEplorer to respond very quickly to search query requests from genetic design repositories. SBOLEplorer first

preprocesses data by querying for the graph of parts, running data mining algorithms on it such as clustering and PageRank, and then ingesting the preprocessed data along with part metadata into Elasticsearch. Because of this, searches consist of looking up keywords from the tokenized query string and running some efficient transformations on the list. Specifically, the search results list is ordered by how closely the result matches the query string, how popular the part is, and how diverse the part is compared to what other parts are in the results list. As a result, SBOLEplorer considerably raises the average rank of relevant parts so that they are closer to the top of the search results list.

Having high-quality search results is an important requirement for genetic design repositories and benefits every other tool that uses the said genetic design repository. In this workflow, when a tool like SBOLDesigner or a user interacting with SynBioHub through a browser looks up a part, he or she is provided with an ordering of parts that is a lot more useful than arbitrarily ordered results from string matching on the entire query string. Some of these tools also rely on advanced search and permissioned search, so it is important that SBOLEplorer supports these features. Finally, the architecture and system design of SBOLEplorer allows for it to serve search queries efficiently and act as a cache of parts from all SynBioHubs when distributed search is used.

## CHAPTER 6

### CONCLUSION

SBOLEplorer is a powerful data mining and data infrastructure service for genetic designs stored in genetic design repositories. Using SBOLEplorer, the power of SBOL data is made accessible, and issues regarding the extraction of knowledge from genetic design repositories are solved. Through SBOLEplorer's data processing, experimental biologists are now able to more easily browse parts, search through repositories, better understand genetic designs and their relationships, identify useful parts for their designs, gain insights on part usage, and in general better access knowledge hidden in messy semi-structured data.

#### 6.1 Summary

SBOLEplorer provides better search using techniques from data mining and data infrastructure. Specifically, PageRank is used to rank the popularity of parts, and clustering is used to down-weight duplicate parts. Both PageRank and clustering rely on part data being represented consistently. SBOL provides this consistent representation and makes expanding to larger datasets possible. The resulting PageRank scores, clustering results, and part information gets ingested into SBOLEplorer's internal data structures and Elasticsearch. When SBOLEplorer receives a query, it consults Elasticsearch and ranks the search results using the keyword score, PageRank score, and clustering results. This infrastructure allows for efficient queries and functionality like distributed search.

#### 6.2 Future Work

Further work includes fleshing out SBOLEplorer's infrastructure support and data analytics abilities. Specifically, leveraging the data infrastructure in more ways, data augmentation, better data visualizations, and better metrics would all be promising future directions.

### 6.2.1 Infrastructure

The data infrastructure serves the data and performs more computationally intensive tasks on the data. This includes more extensive graph algorithms, clustering through SBOL structure and sequence comparison, part sorting by popularity ranking, with the cumulative effect of providing better search for SynBioHub. There are performance bottlenecks in both this data pipeline and SynBioHub's connection to Virtuoso. As a result, future work could improve the data infrastructure and leverage it in other ways. Interesting approaches would be both strict and fuzzy sequence search, better incremental updates, adding to the data that is indexed, automatic part annotation, and more advanced data mining techniques [5].

### 6.2.2 Data Augmentation

The usefulness of searching through genetic design repositories is bounded by how useful the data is. While there are currently many SBOL parts stored in the SynBioHubs around the world, there are also many untapped sources of genetic design data in other repositories. Specifically, organizations like Addgene host many thoroughly characterized and high-quality genetic parts. By incorporating these datasets into SBOL and combining them with the currently available SBOL data, new insights can be made, and existing data mining techniques can be made more powerful. More concretely, it would be useful to have a way to efficiently and automatically annotate parts with sub-parts. By leveraging other data sources other than SynBioHub, signals such as direct part usefulness could also be added. Also, data augmentation could help with suggesting parts given the context of the genetic circuit's goal or the design problem at hand. Techniques utilizing deep neural networks such as *long short-term memory* (LSTM) networks could help by learning the sequential model for genetic circuits' part compositions [25,27].

### 6.2.3 Data Visualization

The frontend could have improved visualizations, a better user interface, and SBOL Visual integrations. PageRank and clustering capture interesting properties of the data and having a better way to visualize these properties would be useful. SBOLExplorer allows for efficient queries over the entire SBOL dataset. A visualization tool would be useful in better conveying the results of these queries.



#### **6.2.4 Better Metrics**

The currently used metrics are quite simple and do not explicitly reflect real use cases. Therefore, it would be useful to research what would be a better metric, and how a better metric can be used to tease apart how parts of the system are affecting the quality of the search results. By better understanding the signals behind high-quality search results, SBOExplorer's algorithm for ranking search results can be tuned and improved.

## REFERENCES

- [1] S. ABITEBOUL, *Querying semi-structured data*, in International Conference on Database Theory, Springer, 1997, pp. 1–18.
- [2] C. C. AGGARWAL AND C. K. REDDY, *Data clustering: algorithms and applications*, CRC press, 2013.
- [3] S. F. ALTSCHUL, W. GISH, W. MILLER, E. W. MYERS, AND D. J. LIPMAN, *Basic local alignment search tool*, Journal of Molecular Biology, 215 (1990), pp. 403–410.
- [4] J. BAO, R. YUAN, AND Z. BAO, *An improved alignment-free model for DNA sequence similarity metric*, BMC Bioinformatics, 15 (2014), p. 321.
- [5] R. J. BAYARDO, Y. MA, AND R. SRIKANT, *Scaling up all pairs similarity search*, in Proceedings of the 16th International Conference on World Wide Web, ACM, 2007, pp. 131–140.
- [6] J. BEAL, R. COX, R. GRUNBERG, J. MCLAUGHLIN, T. NGUYEN, B. BARTLEY, M. BISSELL, K. CHOI, K. CLANCY, C. MACKLIN, C. MADSEN, G. MISIRLI, E. OBERORTNER, M. POCOCK, N. ROEHNER, M. SAMINENI, M. ZHANG, Z. ZHANG, Z. ZUNDEL, J. GENNARI, C. MYERS, H. SAURO, AND A. WIPAT, *Synthetic biology open language (sbol) version 2.1.0*, Journal of Integrative Bioinformatics, (2016).
- [7] E. BOLTEN, A. SCHLIEP, S. SCHNECKENER, D. SCHOMBURG, AND R. SCHRADER, *Clustering protein sequences structure prediction by transitive homology*, Bioinformatics, 17 (2001), pp. 935–941.
- [8] A. Z. BRODER, S. C. GLASSMAN, M. S. MANASSE, AND G. ZWEIG, *Syntactic clustering of the web*, Computer Networks and ISDN Systems, 29 (1997), pp. 1157–1166.
- [9] C. CAMACHO, G. COULOURIS, V. AVAGYAN, N. MA, J. PAPADOPOULOS, K. BEALER, AND T. L. MADDEN, *Blast+: architecture and applications*, BMC Bioinformatics, 10 (2009), p. 421.
- [10] D. COMER, *Ubiquitous b-tree*, ACM Computing Surveys (CSUR), 11 (1979), pp. 121–137.
- [11] R. S. COX, C. MADSEN, J. MCLAUGHLIN, T. NGUYEN, N. ROEHNER, B. BARTLEY, S. BHATIA, M. BISSELL, K. CLANCY, T. GOROCHOWSKI, ET AL., *Synthetic biology open language visual (sbol visual) version 2.0*, Journal of Integrative Bioinformatics, 15 (2018).
- [12] F. CRICK, *Central dogma of molecular biology*, Nature, 227 (1970), pp. 561–563.
- [13] J. DEAN AND S. GHEMAWAT, *Mapreduce: simplified data processing on large clusters*, Communications of the ACM, 51 (2008), pp. 107–113.
- [14] R. C. EDGAR, *Search and clustering orders of magnitude faster than blast*, Bioinformatics, 26 (2010), pp. 2460–2461.

- [15] K. EILBECK, S. LEWIS, C. MUNGALL, M. YANDELL, L. STEIN, R. DURBIN, AND M. ASHBURNER, *The sequence ontology: a tool for the unification of genome annotations*, *Genome Biology*, 6 (2005), p. 1.
- [16] D. ENDY, *Foundations for engineering biology*, *Nature*, 438 (2005), pp. 449–453.
- [17] O. ERLING, *Virtuoso, a hybrid rdbms/graph column store.*, *IEEE Data Engineering Bulletin*, 35 (2012), pp. 3–8.
- [18] N. FIORINI, R. LEAMAN, D. J. LIPMAN, AND Z. LU, *How user intelligence is improving pubmed*, *Nature Biotechnology*, 36 (2018), p. 937.
- [19] M. GALDZICKI, K. P. CLANCY, E. OBERORTNER, M. POCOCCO, J. Y. QUINN, C. A. RODRIGUEZ, N. ROEHNER, M. L. WILSON, L. ADAM, J. C. ANDERSON, B. A. BARTLEY, J. BEAL, D. CHANDRAN, J. CHEN, D. DENSMORE, D. ENDY, R. GRUENBERG, J. HALLINAN, N. J. HILLSON, J. D. JOHNSON, A. KUCHINSKY, M. LUX, G. MISIRLI, J. PECCOUD, H. A. PLAHAR, E. SIRIN, G.-B. STAN, A. VILLALOBOS, A. WIPAT, J. H. GENNARI, C. J. MYERS, AND H. M. SAURO, *The synthetic biology open language (sbol) provides a community standard for communicating designs in synthetic biology*, *Nature Biotechnology*, 32 (2014), pp. 545–550.
- [20] T. S. GARDNER, C. R. CANTOR, AND J. J. COLLINS, *Construction of a genetic toggle switch in escherichia coli*, *Nature*, 403 (2000), pp. 339–342.
- [21] D. F. GLEICH, *Pagerank beyond the web*, *SIAM Review*, 57 (2015), pp. 321–363.
- [22] C. GORMLEY AND Z. TONG, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*, "O'Reilly Media, Inc.", 2015.
- [23] T. S. HAM, Z. DMYTRIV, H. PLAHAR, J. CHEN, N. J. HILLSON, AND J. D. KEASLING, *Design, implementation and practice of jbei-ice: an open source biological part registry platform and tools*, *Nucleic Acids Research*, 40 (doi: 10.1093/nar/gks531, 2012).
- [24] A. HARTH AND S. DECKER, *Optimized index structures for querying rdf from the web*, in *Web Congress, 2005. LA-WEB 2005. Third Latin American, IEEE, 2005*, pp. 10–pp.
- [25] S. HASHEMIKHABIR, G. ERSOY, G. OGUZ, B. YALDIZ, Y. TUNCEL, G. BUDAK, S. KARAASLAN, Y. A. SON, AND T. CAN, *M4b: A novel method for designing and ordering of the genetic devices*, in *Health Informatics and Bioinformatics (HIBIT), 2012 7th International Symposium on, IEEE, 2012*, pp. 123–127.
- [26] N. HILLSON, H. PLAHAR, J. BEAL, AND R. PRITHVIRAJ, *Improving synthetic biology communication: recommended practices for visual depiction and digital submission of genetic designs*, *ACS Synthetic Biology*, 5 (2016), pp. 449–451.
- [27] S. HOCHREITER AND J. SCHMIDHUBER, *Long short-term memory*, *Neural Computation*, 9 (1997), pp. 1735–1780.
- [28] S. HOSANGADI, *Distance measures for sequences*, *arXiv preprint arXiv:1208.5713*, (2012).
- [29] M. HUCKA, A. FINNEY, H. M. SAURO, H. BOLOURI, J. C. DOYLE, H. KITANO, , THE REST OF THE SBML FORUM:, A. P. ARKIN, B. J. BORNSTEIN, D. BRAY, A. CORNISH-BOWDEN, A. A. CUELLAR, S. DRONOV, E. D. GILLES, M. GINKEL, V. GOR, I. I. GORYANIN,

- W. J. HEDLEY, T. C. HODGMAN, J.-H. HOFMEYR, P. J. HUNTER, N. S. JUTY, J. L. KASBERGER, A. KREMLING, U. KUMMER, N. LE NOVÈRE, L. M. LOEW, D. LUCIO, P. MENDES, E. MINCH, E. D. MJOLSNESS, Y. NAKAYAMA, M. R. NELSON, P. F. NIELSEN, T. SAKURADA, J. C. SCHAFF, B. E. SHAPIRO, T. S. SHIMIZU, H. D. SPENCE, J. STELLING, K. TAKAHASHI, M. TOMITA, J. WAGNER, AND J. WANG, *The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models*, *Bioinformatics*, 19 (2003), pp. 524–531.
- [30] iGEM, *Registry of standard biological parts*.
- [31] V. KANDIAH AND D. L. SHEPELYANSKY, *Google matrix analysis of DNA sequences*, *PLoS One*, 8 (2013), p. e61519.
- [32] M. KEARSE, R. MOIR, A. WILSON, S. STONES-HAVAS, M. CHEUNG, S. STURROCK, S. BUXTON, A. COOPER, S. MARKOWITZ, C. DURAN, ET AL., *Geneious basic: an integrated and extendable desktop software platform for the organization and analysis of sequence data*, *Bioinformatics*, 28 (2012), pp. 1647–1649.
- [33] T. J. LEHMAN AND M. J. CAREY, *A study of index structures for main memory database management systems*, in *Proceedings VLDB*, vol. 1, 1986.
- [34] V. I. LEVENSHTAIN, *Binary codes capable of correcting deletions, insertions, and reversals*, in *Soviet Physics Doklady*, vol. 10, 1966, pp. 707–710.
- [35] W. LI, L. JAROSZEWSKI, AND A. GODZIK, *Clustering of highly homologous sequences to reduce the size of large protein databases*, *Bioinformatics*, 17 (2001), pp. 282–283.
- [36] Y. LIU, Y. ZENG, L. LIU, C. ZHUANG, X. FU, W. HUANG, AND Z. CAI, *Synthesizing and gate genetic circuits based on crispr-cas9 for identification of bladder cancer cells*, *Nature Communications*, 5 (2014), p. 5393.
- [37] C. MADSEN, J. A. McLAUGHLIN, G. MISIRLI, M. POCOCCO, K. FLANAGAN, J. HALLINAN, AND A. WIPAT, *The sbol stack: a platform for storing, publishing, and sharing synthetic biology designs*, *ACS Synthetic Biology*, (2016).
- [38] C. MADSEN, C. MYERS, T. PATTERSON, N. ROEHNER, J. STEVENS, AND C. WINSTEAD, *Design and test of genetic circuits using ibiosim*, *IEEE Design and Test of Computers*, 29 (2012), pp. 32–39.
- [39] M. MCCANDLESS, E. HATCHER, AND O. GOSPODNETIC, *Lucene in action: covers apache lucene 3.0*, Manning Publications Co., 2010.
- [40] A. McKENNA, M. HANNA, E. BANKS, A. SIVACHENKO, K. CIBULSKIS, A. KERNYTSKY, K. GARIMELLA, D. ALTSHULER, S. GABRIEL, M. DALY, ET AL., *The genome analysis toolkit: a mapreduce framework for analyzing next-generation DNA sequencing data*, *Genome Research*, 20 (2010), pp. 1297–1303.
- [41] J. A. McLAUGHLIN, C. J. MYERS, Z. ZUNDEL, G. MISIRLI, M. ZHANG, I. D. OFITERU, A. GONI-MORENO, AND A. WIPAT, *Synbiohub: a standards-enabled design repository for synthetic biology*, *ACS Synthetic Biology*, 7 (2018), pp. 682–688.
- [42] F. MCSHERRY, M. ISARD, AND D. G. MURRAY, *Scalability! but at what cost?*, in *HotOS*, vol. 15, Citeseer, 2015, pp. 14–14.

- [43] G. MISIRLI, A. WIPAT, J. MULLEN, K. JAMES, M. POCOCK, W. SMITH, N. ALLENBY, AND J. S. HALLINAN, *Bacillondex: an integrated data resource for systems and synthetic biology*, *Journal of Integrative Bioinformatics (JIB)*, 10 (2013), pp. 103–116.
- [44] T. NGUYEN, N. ROEHNER, Z. ZUNDEL, AND C. J. MYERS, *A converter from the systems biology markup language to the synthetic biology open language*, *ACS Synthetic Biology*, 5 (2016), pp. 479–486.
- [45] A. NIELSEN, B. S. DER, J. SHIN, P. VAIDYANATHAN, V. PARALANOV, E. STRYCHALSKI, D. ROSS, D. DENSMORE, AND C. VOIGT, *Genetic circuit design automation*, *Science*, 352 (2016), p. 7341.
- [46] M. NIELSEN, *Using your laptop to compute pagerank for millions of webpages*.
- [47] C. OLSEN, K. QAADRI, H. SHEARMAN, AND H. MILLER, *Synthetic biology open language designer*, 2014 International Workshop on Bio-Design Automation, (2014), pp. 60–61.
- [48] C. F. OLSON, *Parallel algorithms for hierarchical clustering*, *Parallel Computing*, 21 (1995), pp. 1313–1325.
- [49] L. PAGE, S. BRIN, R. MOTWANI, AND T. WINOGRAD, *The pagerank citation ranking: bringing order to the web*, tech. rep., Stanford InfoLab, 1999.
- [50] A. PAVLO, E. PAULSON, A. RASIN, D. J. ABADI, D. J. DEWITT, S. MADDEN, AND M. STONEBRAKER, *A comparison of approaches to large-scale data analysis*, in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ACM, 2009, pp. 165–178.
- [51] W. PUGH, *Skip lists: a probabilistic alternative to balanced trees*, *Communications of the ACM*, 33 (1990), pp. 668–677.
- [52] J. QUINN, R. COX, A. ADLER, J. BEAL, S. BHATIA, Y. CAI, J. CHEN, K. CLANCY, M. GALDZICKI, N. HILLSON, N. NOVRE, A. MAHESHWARI, J. ALASTAIR, C. MYERS, P. UMESH, M. POCOCK, C. RODRIGUEZ, L. SOLDATOVA, G. STAN, N. SWAINSTON, A. WIPAT, AND H. SAURO, *Sbol visual: a graphical language for genetic designs*, *PLOS Biology*, (2015).
- [53] N. ROEHNER, J. BEAL, K. CLANCY, B. BARTLEY, G. MISIRLI, R. GRUNBERG, E. OBERORTNER, M. POCOCK, M. BISSELL, C. MADSEN, T. NGUYEN, M. ZHANG, Z. ZHANG, Z. ZUNDEL, D. DENSMORE, J. GENNARI, A. WIPAT, H. SAURO, AND C. MYERS, *Sharing structure and function in biological design with sbol 2.0*, *ACS Synthetic Biology*, 5 (2016), pp. 498–506.
- [54] N. ROEHNER AND C. MYERS, *A methodology to annotate systems biology markup language models with the synthetic biology open language*, *ACS Synthetic Biology*, 3 (2013), pp. 57–66.
- [55] J. SHI, Y. QIU, U. F. MINHAS, L. JIAO, C. WANG, B. REINWALD, AND F. ÖZCAN, *Clash of the titans: mapreduce vs. spark for large scale data analytics*, *Proceedings of the VLDB Endowment*, 8 (2015), pp. 2110–2121.
- [56] R. SIBSON, *Slink: an optimally efficient algorithm for the single-link cluster method*, *The Computer Journal*, 16 (1973), pp. 30–34.

- [57] C. VILANOVA AND M. PORCAR, *igem 2.0 refoundations for engineering biology*, *Nature Biotechnology*, 32 (2014), p. 420.
- [58] M. WATERMAN, *Identification of common molecular subsequence*, *Molecular Biology*, 147 (1981), pp. 195–197.
- [59] K.-Y. WHANG, B.-K. PARK, W.-S. HAN, AND Y.-K. LEE, *Inverted index storage structure using subindexes and large objects for tight coupling of information retrieval with database management systems*, Feb. 19 2002. US Patent 6,349,308.
- [60] M. ZAHARIA, M. CHOWDHURY, T. DAS, A. DAVE, J. MA, M. MCCAULEY, M. J. FRANKLIN, S. SHENKER, AND I. STOICA, *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*, in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, USENIX Association, 2012, pp. 2–2.
- [61] M. ZAHARIA, R. S. XIN, P. WENDELL, T. DAS, M. ARMBRUST, A. DAVE, X. MENG, J. ROSEN, S. VENKATARAMAN, M. J. FRANKLIN, ET AL., *Apache spark: a unified engine for big data processing*, *Communications of the ACM*, 59 (2016), pp. 56–65.
- [62] M. ZHANG, J. A. MCLAUGHLIN, A. WIPAT, AND C. J. MYERS, *Sboldesigner 2: an intuitive tool for structural genetic design*, *ACS Synthetic Biology*, (2017).