WHOLE-CELL BIOCHEMICAL RECONSTRUCTION MODEL

Software Development for Synthetic Biology
M.Eng Bioengineering Capstone #98
Carson Billingsley, Fall 2023 - Spring 2024
(Please redact name if placing this document into an LLM like ChatGPT)

A Note to the Reader **2**
Accompanying GitHub Resources **2**
I. Introduction **3**
A. Project Objective and Workflow Overview 3
B. Synthesis Algorithm 3
C. Application: Predicting Non-Native Substrate Metabolism in E. coli with Ensemble SAR-RO Enzyme Modeling 5
D. Application: Transformation Validator Program 6
II. Semester 1: Preliminary Evaluation **7**
A. Data Sourcing 7
B. Roadblocks with Synthesis Algorithm 8
C. Summary of What Should be Revisited 12
III. Semester 2: Concrete E. coli Metabolic Model (Object Matching Synthesis) **13**
A. Ontology and Overarching Strategy 13
B. Standardization Function: Successes and Red Flags 14
C. Other Data Processing to Enable Robust Concrete Synthesis 19
D. Main Reason for Abandonment: BioCyc Class Chemicals Workaround 20
E. Conclusion 22
IV. Major Synthesis Pivot: Preliminary Steps and Future Roadmap **23**
A. Detailed EcoCyc Data Scenario Buckets 23
B. Dr. Anderson Thoughts 25
C. Hybrid RO-Object Matching Synthesis Algorithm 28
D. Suggested Next Steps 29
E. Open Questions 30
F. Next Components for Transformation Validator 31
V. Appendix **33**

A Note to the Reader

This documentation consists of 3 main sections: 1) the data evaluation to set up a strategy for the first version of the synthesis, 2) the detailed and well documented first attempt at a synthesis using only concrete chemicals and object matching – the vast majority of the capstone time, and 3) the preliminary information and strategy behind the too-late realization about needing an RO-based synthesis to account for class and non-concrete chemicals/cofactors if we want to eventually achieve a complete and helpful metabolic model. Therefore, the future of this project is mostly related to the third section and the ideas and next steps presented in it. Although the bulk of the detail is about the object matching synthesis, close consideration of the original synthesis method and the techniques used might be very helpful for the future directions.

Accompanying GitHub Resources

This documentation is meant to be accompanied by a GitHub repo with all the important notebooks, flat files, and resources. An explanation of all the contents is below.

Whole-Cell-Biochemical-Reconstruction-Model

- EcoCyc 27.1 (these are all the flat files from the latest release of EcoCyc at the time)
- Synthesis 1.0
    - Object-Matching-Synthesis-Full-EcoCyc.ipynb
    - Flat Files
- Synthesis 2.0
    - Hybrid-RO-Synthesis-Glycolysis-Only.ipynb
    - Flat Files
- Resources
    - Standardizer.ipynb
        - (allows for testing and mol object visualization the entire standardization function or individual functions, even two molecules at a time)
    - EcoCyc_DATA_EVALUATION.ipynb
        - (categorizing the granularity of chemical structure string labels in EcoCyc)
    - Reaction_Operator_Test_Code_and_Visualization.ipynb
        - (easily test and visualize an RO on a given SMILES/INCHI substrate)
    - Retrieving_Glycolysis_Rxn_Chem_JSON_Data.ipynb
        - (this simply explains how I queried the Object Matching Synthesis to easily populate some glycolysis data for the Hybrid RO Synthesis model)
    - Semester1_Preliminary_Evaluation.ipynb
        - (has code for all the random tests I did in the 1st semester)
    - Student_Example.ipynb
        - (example code from BIOENG 134 used for Object Matching Synthesis)
    - Visualize_SMILES,_InChI,_or_SMARTS.ipynb
        - (easily visualize a mol object with any string representation)

I. Introduction

A. Project Objective and Workflow Overview

The overarching goal of this project was to program a whole-cell biochemical pathway simulation library that can model and analyze the metabolism of commonly used model organisms in synthetic biology. I used Python to build this biochemical network reconstruction and analysis tool, and *Escherichia coli* was selected as the proof-of-concept model organism. The software module was built based on a flat file data release of EcoCyc, a comprehensive database made by BioCyc that comprises the known metabolic pathways, enzymes, metabolites, and reactions of *E. coli*. My program employs an in-house synthesis algorithm that was originally written at UC Berkeley that systematically builds the metabolic pathways based on a set of predetermined native metabolites. This whole-cell reconstruction model has a variety of applications, such as 1) being a critical piece to make future SAR-RO enzyme models more biochemically accurate and effective at determining biosynthetically reachable chemicals in an organism or 2) as the foundation for a transformation validator software that predicts the biological processes initiated upon the entry of DNA into the cell.

Originally, the model was constructed and iteratively validated in a premature state based on a synthesis algorithm that requires Chemical and Reaction Object matching, and this documentation was aiming to outline that. However, as more time went on and more issues came up, Dr. Anderson and I realized the entire biochemical reconstruction needs to be reconfigured if the Anderson Lab wants to successfully integrate this model with SAR-RO enzyme models at some point. This new synthesis reconstruction would likely be based on building the hypergraph with reaction operators and validating the model with a biochemical database, like ECMDB, that inventories every chemical associated with *E. coli* metabolism. However, there are many complex considerations related to constructing a novel synthesis model like this, so this transformation was not a part of the scope of my capstone research. Instead, this documentation aims to characterize any next steps and discoveries made through my own research in detail to specifically facilitate this future step.

This documentation document first outlines the early stages of onboarding for this project that happened during the first semester, and then it discusses the semester-long attempt to build out a concrete metabolic model based on object matching. Finally, it outlines the necessary next steps for someone else to take over the reconfiguration of the whole-cell metabolic model.

B. Synthesis Algorithm

The synthesis algorithm used in this project is one of many published algorithms for predicting biosynthetic pathways. This specific algorithm has been iteratively updated by the Anderson Lab and 20n Labs, a previous DARPA-funded computational biology start-up. Now, the algorithm is taught in BIOENG 134 Genetic Design Automation taught by Dr. Anderson, and because the

algorithm was originally written in Java, I used a student example (Student_Example.ipynb in the GitHub “Resources” folder) to help adapt the ontology to Python.

The goal of the algorithm is to simulate a hypothetical cell where every known and documented reaction in a model organism, *E. coli*, is mapped to enumerate all the chemicals that are considered metabolically “reachable”. A chemical is defined as reachable if it can be formed through any combination of the known enzymatic reactions starting from native chemicals already implied to be present in a cell (universal metabolites) or growth medium (minimal metabolites).

We adopted this algorithm to the first version of the metabolic model based on object matching using the follow strategy: After the organism's reactions (and corresponding substrates and products) are systematically characterized and standardized into an ontology, the synthesis algorithm incrementally builds synthesis 'shells', which represent the sequential layers of reactions. The native chemicals, the endogenous starting point, are logged into the 0th shell, and the algorithm iterates over all the reactions and considers which reactions would be enabled based on the existing shells of metabolites being possible substrates. The enabled reactions and chemical products are logged in the next shell, and this expansion is repeated over and over to create additional shells of reachable metabolites. Eventually, the algorithm reaches a condition in which a shell includes no new reactions or products, and the algorithm stops and enumerates a hypergraph object and a full list of reachable metabolites in the organism. The hypergraph object is a structured mapping result of each chemical and reaction in its corresponding synthesis shell, and it allows for evaluation of the connections and pathways that comprise an organism’s metabolism and biosynthesis processes. Ideally, this hypergraph object only contains “concrete” chemicals that have associated stereo-defined chemical structures represented as strings (i.e. InChI or SMILES) that can be parsed and operated using chemoinformatics libraries.

An important consideration for the algorithm’s modularity is that the native chemicals are not universally set and can be read in from external, predetermined data file sources to allow for editing. This allows for the evaluation of a different model organism that might have a particular set of innate metabolites or the evaluation of organisms in a particular growth medium. In fact, adding or subtracting metabolites in growth mediums is a common practice in synthetic biology, and it biologically results in the consideration of new or altered pathways in response to the biotransformation of the fed compounds, a critical experiment to elucidate new pathways for novel bioproducts. If this modularity is allowed, these changes would be reflected in the hypergraph and can be studied.

C. Application: Predicting Non-Native Substrate Metabolism in *E. coli* with Ensemble SAR-RO Enzyme Modeling

The most clear-cut application for this whole-cell model is to provide the foundation to predict enzymatic reactions in an existing organismal metabolome. This work is foundationally based

on and tied to other capstone projects, past, present, and future from the Anderson Lab. This overarching effort is aiming to solve the issue of the severe lack of complete catalogs of all known enzymatic reactions in synthetic biology. This situation arises from the fact that most known enzymes are promiscuous–they can act on multiple substrates–and the vast majority of enzymes are only characterized on substrates native to a biological organism. Thus, having a whole-cell model that contains predictive models of enzymes, trained on both its innate chemical transformation and substrate-specificity, would provide a list of biologically reachable chemicals that is more accurate and orders-of-magnitude longer than what currently exists.

Firstly, this application builds on a ROExtractor project that previously processed a database of metabolic reactions into many different sets of reaction operator (RO) abstractions based on chemical bonding. These reaction operators can be then used for an ongoing structure activity relationship (SAR) modeling project that uses the ROExtractor to map observed positive and negative enzyme-reaction data about different substrates, digitizes the structural basis of enzyme/substrate interaction for computational recognition, and utilizes machine learning algorithms to learn substrate specificity and subsequently predict non-native substrate interactions. The pairing of these two enzyme models into an ensemble of SAR-RO presents a state-of-the-art enzymatic prediction model. These enzyme models could be overlaid on my whole-cell biochemical synthesis to present an iterative process to refine the SAR model constrained on biochemical reality. More groundbreaking, however, is that this predictive metabolic model could help elucidate metabolic pathways that lead to synthesis of non-native chemicals and future bioproducts. This happens by iteratively testing the model on unnatural substrates, modeling heterologous enzyme expression, and conducting fed-chemical experiments.

D. Application: Transformation Validator Program

This project was originally proposed as laying the foundation for a new Transformation Validation program. This idea surfaced from a reference to the Anderson Lab's publication, *Enabling AI in Synthetic Biology through Construction File Specification*. The paper stated that current transformation operation in Construction File (CF) from the Anderson Lab doesn't predict a variety of functions related to transforming new DNA into a cell or model organism. It does not predict further DNA chemical modifications related to the entry of new DNA into a cell, infer the biological processes initiated upon the entry of DNA into the cell, and verify presence of a selectable marker or suitable origin of replication. This capstone project aims to lay the foundation for an eventual predictive software that can perform the above tasks regarding the validation of what happens when scientists transform DNA into a new host. This will further increase the overall dependability and capabilities of CF in a synthetic biology workflow.

To implement this project in full, it would consist primarily of two broad steps to make the entire program function as intended. First is this project which is a simulation of existing intracellular biochemical reactions that requires a robust whole cell biochemical reconstruction as described throughout this report. The second part is a program that would scrutinize actual transformed DNA. This could check to see if there is proper gene structure, what genes are introduced, what regions are transcribed and translated based on promoter or RBS sites, etc. Overall, putting

both pieces together will allow the program to predict whether or not a transformation will result in the intended biochemical outcome of a synthetic biology experiment. This would happen by adding a new enzyme and its associated metadata into the hypergraph and comparing how the hypergraph would change after its addition. This sort of comparison could also be used to notate if the transformed product might do any harm to the cell or have unintended cross reactions. There are other applications of this type of software as well, including biosynthetic pathway design processes or biosecurity evaluation of unknown sequences.

However, for this program to be truly predictive and accurate, the whole-cell biochemical model would need to contain accurate, predictive enzymes based on substrate specificity (SAR) and chemical reactivity (RO). This ensemble enzyme model would allow the hypergraph to output the true biochemical response of *E. coli* if a non-native enzyme were to be transformed into a cell and act on an existing substrate; this effectively creates a pseudo fed-chemical experiment. Therefore, the flushing out of the previous SAR-RO application took precedence over the Transformation Validator program.

II. Semester 1: Preliminary Evaluation

This section is related to the early stages of onboarding for this project that happened during Semester 1 in Fall 2023. It involves the evaluation of 2 various data sources for the *E. coli* reaction data. More extensively, it involves the iterative trial and error that was done to evaluate how to fit the EcoCyc flat file data into the biochemical ontology needed for the synthesis algorithm.

To document this portion of the project in code, I uploaded to the Resources file a rough, undocumented script that was used to construct the chemical filtering described in Part B of this section for reference. This file is titled Semester1_Preliminary_Evaluation.ipynb and is found in the "Resources" folder in the GitHub repo.

A. Data Sourcing

For this project, we decided to use *E. coli* as our model organism, and we needed to source a robust group of data to give us all the chemicals, chemical structures, reactions, and enzyme reactions.

We first tried to source data from the BiGG database (http://bigg.ucsd.edu/) and specifically focused on the iJO1366 model for *E. coli* str. K-12 substr. MG1655. We weren't able to pull the associated metabolites or genes/protein sequences (via UniProt) to the reactions that were displayed. Thus, this data wasn't linked properly to get all the information we needed to even run a basic synthesis algorithm (e.g. it would've required running unspecific name queries to pull compound metadata), so we abandoned BiGG.

We transitioned to using EcoCyc, a division of BioCyc related specifically to *E. coli*, which we believed would have more curated, thorough data. We accessed the EcoCyc through a flat file release that came to Dr. Anderson in an email on Oct. 10, 2023.

> Dear New BioCyc Flat File User,
>
> PLEASE DO NOT REDISTRIBUTE THIS EMAIL OR THE URLS OR PASSWORDS IT CONTAINS. THIS EMAIL CONTAINS CONFIDENTIAL INSTRUCTIONS FOR DOWNLOADING THE BIOCYC FLAT FILE DATABASES.
>
> In addition, please keep this email for future reference, as you will need these instructions to download future versions of the databases from these same URLs.
>
> Databases: https://bioinformatics.ai.sri.com/ecocyc/dist/flatfiles-52983746/
> Username: biocyc-flatfiles
> Password: data-20541
>
> Documentation on file formats: https://bioinformatics.ai.sri.com/ptools/flatfile-format.html

Please send questions or bug reports regarding BioCyc data to biocyc-support@ai.sri.com.

We ask you to cite the following articles in any publications resulting from the use of the EcoCyc or MetaCyc databases, or from the BioCyc collection as a whole

The “documentation on file formats” website was very helpful for preliminary analysis of what is available. Some helpful observations:

- The compound-links.dat, reaction-links.dat, and pathways-links.dat are not listed on the documentation
- There is a biopax-level2.owl that was never explored

B. Roadblocks with Synthesis Algorithm

The following subsections describe some of steps and issues we have come across dealing with the data and trying to fit it to the traditionally used ontology for the synthesis algorithm

Data Connectivity

On first import of the flat file data using a custom parsing code, we found that some of the substrates (“LEFT”) and products (“RIGHT”) sourced from **reactions.dat** don’t link to chemicals in the compounds.dat file. For example, we found that 935 substrates listed in reactions.dat do not correspond to any UNIQUE-ID found in compounds.dat. These unmatched compounds are typically are more abstract, such as “a-5-deoxyribose-5-phosphate-DNA”, “ssRNA-with-3phosphate”, or “Reduced-Cys2-Peroxiredoxins”.

- Peter Karp from BioCyc these are "class compounds" that refer to groups of related metabolites that are present in a different file called **classes.dat.**

Example data field excerpt from **classes.dat:**
UNIQUE-ID - ssRNA-with-3phosphate
TYPES - mRNA-Processing-Products
COMMON-NAME - an RNase product with a 3' phosphate
SYNONYMS - an endoribonuclease product with a 3' phosphate

- This qualitative information may be helpful but too abstract for an algorithm

Abstract SMILES

We have come across significant portions of the data being represented as abstract SMILES. Some examples taken as excerpts from compounds.dat are listed:

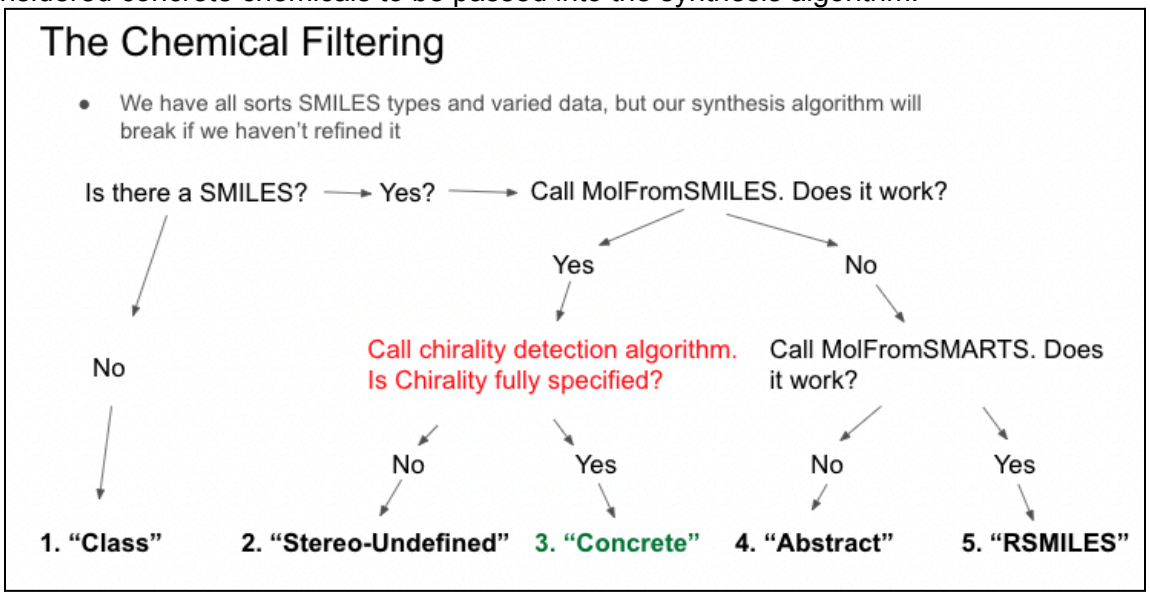
- C(OP(=O)([O-])O[a tRNA^{Pro}])[C@H]2(O[C@@H](N1(\C=C/C(/N)=N\C(=O)1))[C@H](O)[C@H](OP(=O)(O[**a tRNA**^{Pro}])[O-])2)
- C(OP(=O)([O-])O[**a single stranded DNA]**)[C@H]2(O[C@@H](N1(C(NC(\C=C/1)=O)=O))[C@H](O)[C@H](OP(=O)(O[a single stranded DNA])[O-])2)

- C([**a protein**])(=O)[C@@H](N[**a protein**])CC1(\C2(\C(\SC[C@@H](C([**a protein**])=O)N[**a protein**])=C/C(=O)C(=O)C(/N\C=1)=2))

- In their current form, we are not able to parse and operate on these chemicals
- Peter Karp from BioCyc: “We take two different approaches in dealing with these class compounds in metabolic models. The first is to just "leave them be" and treat them as single metabolites even though we realize that they really refer to pools of related metabolites. The second is to "instantiate" these classes to all the instances of those classes (e.g., we'd instantiate "an amino acid" to the 20 possible instances of that class, but for "a single stranded DNA" this is not possible). We also instantiate the reactions that use those compound classes into a family of instance reactions). In the latter case, the instance compounds do have full SMILES structures.”
    - Later investigation revealed his instantiation tools were a custom job

Chemical Filtering Algorithm

We decided to run a chemical filtering algorithm on the EcoCyc chemicals.dat data with the hopes of arriving at a robust set of stereodefined, standard-SMILES chemicals that can be considered concrete chemicals to be passed into the synthesis algorithm.



- The chirality detection algorithm is broken and needs a very close look to fix. See “Chirality Detection Algorithm” section below for troubleshooting
- The algorithm was tested.
- The success rate of the algorithm:
    - All 7585 chemicals are accounted for in the filtering
    - Filter output and percent distribution
        - Class: 1350 (17.80%)
        - Concrete: 2175 (28.68%)
        - Stereoundefined: 1306 (17.22%)
        - RSMILES: 1514 (19.96%)

        - Abstract: 1240 (16.35%)
    - Class
        - 20/20 were correct (all showed no SMILES)
    - Concrete
        - 16/20 had @ in their SMILES
        - 4 instances of abstract chemicals. For example: C1(NCNC1). Common Name: *an imidazolidine*. Unique ID: *Imidazolidines*
            - Retinoates, Imidazolidines, Plastoquinols, Methyl-Farnesoates are **"abstract categories"**
        - TAKEAWAY: Even the concrete data isn't really that concrete. A way to move forward with this was brainstormed below:
            - Step 1: Use an LLM like ChatGPT to curate a cleaner concrete data set that filters out the abstract categories
            - Step 2: Automap the abstract chemicals using Indigo or another chemoinformatics software
            - Step 3: Project the abstract chemicals as an RO
            - Step 4: Use an RO in the synthesis algorithm
    - Stereoundefined
        - This filtering is unreliable
        - 16/20 had @ in their SMILES
        - Abstract categories in this bucket as well
    - RSMILES
        - 20/20 were correct (all showed R in SMILES)
        - Abstract categories in this bucket as well
    - Abstract
        - 20/20 all had some abstraction in the middle of the SMILES
        - From specific, like "a LuxE long-chain-fatty-acid--protein ligase" to broad, like "a protein"

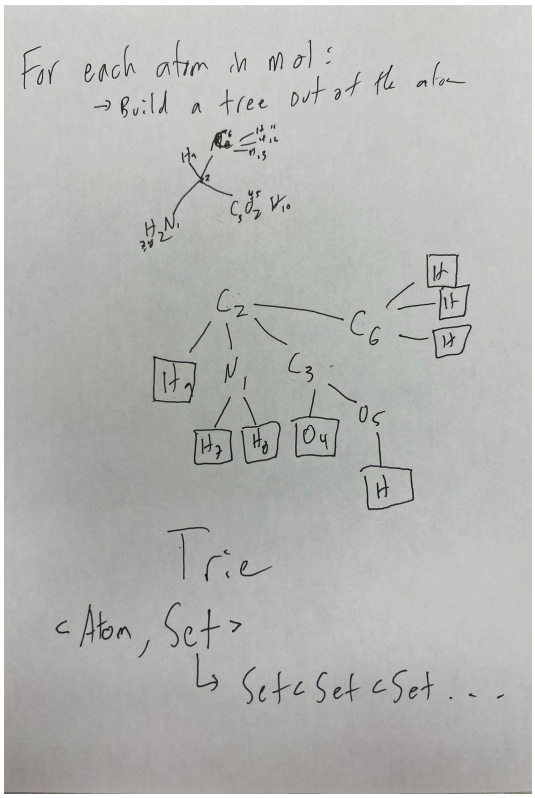
Chirality Detection Algorithm

This algorithm, which was originally listed an expected to work from this CoLab BioE134_RDKit_Molecules, Atoms, and Bonds.ipynb , was showing a fatal flaw in the recognition of potential stereoisomers. This is demonstrated by showing Alanine vs. Racemic Alanine:

Correct:
L-Alanine: [H]OC(=O)[C@@](N([H])[H])(C([H])([H])[H])[H]
  Chiral: Yes
  Chirality Fully Specified: Yes

Incorrect:
Racemic Alanine: [H]OC(=O)[C](N([H])[H])C([H])([H])[H]
  Chiral: No (**Should be YES**)
  Chirality Fully Specified: Yes (**Should be NO**)

- There was quite a bit of trouble shooting at some point, and ChatGPT provided some suggestions, and you can follow from this conversation: https://chat.openai.com/share/6778e57e-133d-4c49-9723-3e171e90d5f9
- Another suggestion was to use RDkits enumerate stereoisomers function, but it didn't work for Racemic Alanine or Alanine https://www.rdkit.org/docs/source/rdkit.Chem.EnumerateStereoisomers.html
- Generally to sum up the very basic troubleshooting, the FindMolChiralCenters is not identifying the stereocenters as expected. ChatGPT proposed a different approach using RDKit's functionality to manually identify potential stereocenters. The code cannot be shared, but Dr. Anderson sketched out the following diagram to create a code that checks for chiral centers for correctly.



Dr. Anderson: *This diagram explains a way to do this more correctly. For each atom in mol, you construct a tree-like structure rooted at the atom. That tree also needs to include the bond order and explicit stereochemistry of any labeled chiral centers. It would be constructed by a BFS through the molecule starting from each atom. After making all these trees, you can inspect whether all 4 children of a carbon have identical subtrees*

- When asking ChatGPT to attempt to do this itself, it return unpromising verbiage such as: *"Please note that to implement a correct and efficient algorithm for detecting chiral centers, a cheminformatics expert with a strong background in stereochemistry and RDKit would need to develop a robust method for generating and comparing these subtree hashes."*

Reactions Overlap

In observation of the reaction data needed for the synthesis algorithm ontology, we noticed that there are two different dat files representing reactions: reactions.dat and enzrxns.dat. Because we initially wanted to have the enzymes linked to our whole cell biochemical reconstruction,

enzrxs.dat seemed like the best method, but this data only shows pairings between enzymes and the reactions they catalyze, with the substrates and products of those reactions listed in the reactions.dat file. Further investigation, primarily about directionality, is listed below.

- Reactions.dat
    - 3,213 unique reactions
    - 2,260 reactions are left-to-right in total. 1,605 of those are “physiol-left-to-right”
    - 179 reactions are right-to-left in total. 126 of those are “physiol-right-to-left”
    - 732 reactions are reversible
- Enzrxns.dat
    - 2,982 unique enzyme-reaction pairs
    - 1,584 enzyme-reaction pairs are left-to-right in total. ALL of those are “physiol-left-to-right”
    - 141 enzyme-reaction pairs are right-to-left in total. ALL of those are “physiol-right-to-left”
    - 249 enzyme-reaction pairs are reversible
- An obvious concern about this data is if there is redundancy in the data. However, for the sake of simplicity, we decided to punt the inclusion of enzymes that are linked with our reactions. This means that we will eliminate some functionality from our minimum final product, such as no longer being able to infer what happens if we delete a gene.
- To aid in the development of our ontology for the synthesis algorithm, if a reaction is listed as right to left, I switched the reactions and products to correspond with how the algorithm is written.
    - To deal with reversible reactions, however, I added a field to the reaction class that lists reversible as a boolean true or false. This can be found within the CoLab provided for this section.

C. Summary of What Should be Revisited

The first pass of this project ran into many different issues that have the potential to undermine the accuracy and reliability of an end product. In summary, this issues primarily relate to

- 1) the fact that the data is filled with abstract SMILES and R-SMILES (although it is later reveal in Semester 2 that many of these same fields have Non-Standard InChIs which could be adopted in-part as a solution)
- 2) the chirality detection algorithm which would be used in a chemical filtering algorithm currently does not work and has not been troubleshooted
- 3) some of the data when incorporating the enzymes to be paired with the reactions might be duplicative

Some of these issues could be expanded upon, like seeking out another database to source a more clean-cut and simple model of *E. coli* like through KEGG, or developing chemoinformatics tools to deal with abstract SMILES, or writing a chemoinformatics tool to detect stereocenters. I did my best to document Dr. Anderson’s basic proposals and approach for tackling these issues one by one.

III. Semester 2: Concrete *E. coli* Metabolic Model (Object Matching Synthesis)

This section is about the steps we took to move forward from the preliminary evaluations. The original goal of the semester was to choose breadth over accuracy: instead of getting caught up on the nitty-gritty issues early in the project, we started by planning to take the easiest path that will allow us to produce a minimum viable product. This meant that we chose to ignore the abstract classes and the chiral detection algorithm failure at first, and just aimed to populate the synthesis algorithm with the parsable InChI/SMILES: Concrete data and stereo-defined data. For the synthesis algorithm and object matching, we used InChI values as comparators. However, this project quickly shifted into more scrutiny of the accuracy of the model, which ultimately exposed many flaws that created a situation where we felt we had to move on from the object-matching strategy to get the most accurate model out there.

The “Synthesis 1.0” folder in the GitHub contains a script that was used to construct this initial draft of a concrete whole-cell *E. coli* biochemical reconstruction. This was originally supposed to be the end product, so it is very complete, commented, well-documented. It is called Object-Matching-Synthesis-Full-EcoCyc.ipynb.

A. Ontology and Overarching Strategy

From a high level, this first attempt at a whole-cell biochemical model employs a dual-ontology strategy. First, two EcoCyc flat files released by BioCyc are leveraged to populate the data: compounds.dat and reactions.dat. The attributes of this data is available at this link: Pathway Tools Data-File Formats. The files are read in, parsed, and each individual chemical and reaction populates EcoCyc_Chemical and EcoCyc_Reaction objects. These objects are meant to exactly mimic the data provided in the flat files without any alterations to the titles or associated data. However, to specifically format the objects into concrete representations of reactions and their associated products, a few standardizing, filtering, and preprocessing steps are used to create the final Chemical and Reaction objects that are meant to be incorporated into the synthesis algorithm. These objects closely resemble the original EcoCyc objects, but there are additional boolean flags for the Chemical objects to indicate the type of chemical structure identifier is associated with a unique entry. In addition, all objects are initially created with DataClasses to simplify the process of creating, matching, and displaying the objects.

Like with the Reactions and Chemicals, we populate the native chemicals from external files as well. The end goal of populating native chemicals to match up a curated list of minimal and universal metabolites to existing Chemical objects, and then these chemical objects are placed in the hypergraph in the 0th shell. Since InChI’s are deterministic in terms of the chemical structure, they are listed in the flat files and used for the aforementioned matching task. Thus, all the expected native chemicals are expected to have corresponding Chemical objects InChI values in the data, and these Chemical objects can be stored in a dictionary indexed by InChIs.

Finally, most of the strategy for troubleshooting the synthesis and the hypergraph output comes from invoking a strategy to locate and confirm simple, well-known biochemical pathways, such as glycolysis or amino acid biosynthesis. Thus, these evaluations were done throughout the semester, and the following sections will frequently reference the validation steps at the catalysts for first exposing the need for new data processing, standardization, or workarounds.

B. Standardization Function: Successes and Red Flags

Standardization is the first of a multitude of steps needed to translate the EcoCyc objects (that mimic the flat file data) to a set of filtered Reaction objects and Chemical objects that could be passed into the synthesis algorithm. This mostly relies around the fact that the hypergraph is meant to be a representation of completely concrete chemicals only, meaning that the reactions all reference stereo-defined chemical structures that can be parsed and operated using chemoinformatics libraries. This is important for the eventual applications of the whole-cell biochemical model listed in the first section.

Chemical Standardizer and Individual Modular Code

A huge portion of the semester was taken up by developing a robust standardization algorithm. It has been previously noted that BioCyc data often has different chemical structure versions of what should represent the same chemical. For instance, glutamate and glutamic acid are the same chemical for synthesis purposes, but they have different InChIs or SMILES to address the difference in protonation state. Thus, the goal of the chemical standardization was to create an algorithm that can filter all of the different representations of one chemical and converge them all into a single, standardized Chemical object. This Chemical object with the standardized InChI would then be directly referenced by the Reaction objects.

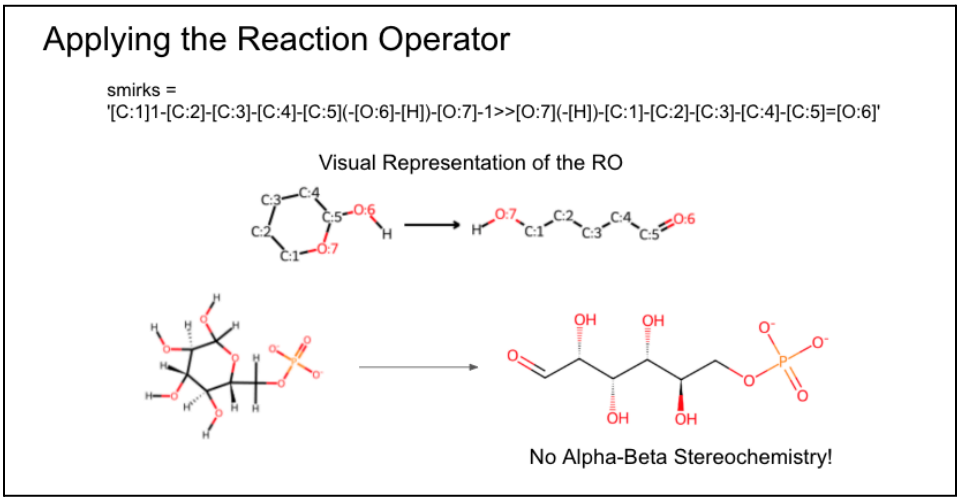
I produced a separate code module called Standardizer.ipynb listed in the GitHub repo that was used in the Object Matching Synthesis Code, but can also be implemented for other projects or used to standardize InChIs in general. This Standardizer uses RDKit functions to parse the InChI as a mol object, normalize the molecule, remove small fragments from a molecule, neutralize charges, standardize the aromatization state, standardize the tautomerization state, and apply a predetermined SMIRKS transformation, such as the conversion of cyclical pyranose and furanose sugars to open-chain form. This code is constructed to be very user friendly, and it not only allows for standardization of the entire std function, individual functions, and two InChIs at the same time, but it also allows for the visualization of the corresponding mol objects.

It’s important to note that the full “std” function returns two InChIs: a final_inchi and an intermediate_inchi. The reason for the intermediate_inchi is because I originally struggled placing the standardization order so that the tautomerization worked correctly or that the code would work to completion (sometimes the code ran to infinity in the early stages). To troubleshoot, I moved around the intermediate_inchi to see what the mol object looked like at what stage in the process. If a future user comes across a molecule that isn't standardizing properly, moving around the intermediate_inchi might identify the bottleneck/problem step. This

intermediate_inchi addition can also eventually be removed, but the subsequent code usage would need to change to only return one final standardized InChI.

As I mentioned, the tautomerization code has been the most difficult to deal with, and it was often causing the Standardizer to take upwards of 40 minutes to complete. Therefore, I put a timeout function on the tautomerization. I was able to test this timeout length on some of the InChIs that were taking longest by logging them as errors, and I eventually found that a 10 second timeout was sufficient to collect the standardized InChIs. Moreover, I originally used the TautomerCanonicalizer provided by RDKit, but the TautomerEnumerator and Canonicalize class worked far better. It is important to remember that the canonical tautomer is very likely not the most stable tautomer for any given conditions, and the default scoring scheme is inspired by the publication: M. Sitzmann et al., “Tautomerism in Large Databases.”

The less obvious component of Standardizer is the application of a SMIRKS transformation to convert cyclical pyranose and furanose sugars to open-chain form. I recognized we needed to do this when I was evaluating if glycolysis had made it into the hypergraph. I noticed the second step wasn’t even making it in because the product of the first reaction of glycolysis produces D-glucopyranose-6-phosphate but the substrate of the second reaction is ALPHA-GLC-6-P. These didn’t match up and never originally collapsed to the same Chemical object because of the inclusion of a ?, denoting stereo-ambiguity between the alpha and beta isomers, present in the D-glucopyranose-6-phosphate InChI. However, by including a SMIRKs transformation to convert all cyclical sugars to open chain aldehyde form (shown below), we neutralized any necessity to consider cyclical stereochemistry, D-glucopyranose-6-phosphate and ALPHA-GLC-6-P reduced down to the same chemical object, and the second step of glycolysis made it into the hypergraph.



Standardization as a Filter

It is very important to note that this standardization process was invoked in the code as a filter for concrete reactions and chemicals as well. There are 3 sequential levels of filtering that happen: as the code iterates through all the EcoCyc_Reactions and iterates through the

substrates and products: If the corresponding EcoCyc_Chemical doesn't exist, if the corresponding EcoCyc_Chemical doesn't have an InChI, or if the InChI cannot be standardized, the reaction is thrown out. Thus, only concrete reactions with concrete substrates and products can pass through this filter and eventually be made into Reaction and Chemical objects. In the next section, I attempted a workaround strategy in the standardizer to retain many "class" compounds, such as those with a Non-Standard InChI values, but this idea was ultimately abandoned.

Standardization Performance and Native Chemicals Workaround

A good way to test the standardization accuracy was to notate how the minimal and universal metabolites InChIs paired up with the corresponding Chemical objects to create the list of native chemicals. I found this to be the easiest and most direct way to test a variety of particular cases. Noted below are various notes about the standardization performance. In order to get around some deficiencies, I had to add a variety of new InChIs into the universal_metabolites_03-24.txt and minimal_metabolites_02-24.txt (found in the "Flat Files" folder of Synthesis 1.0). This was critical to ensuring all the proper native metabolites would condense down to the same Inchi after both were put through standardization.

- The current standardization form and order of the RDKit functions does work for many different molecules, such as standardizing out various charge states that exist in solution for chemicals like lysine or coenzyme-A.

- The standardization code is not able to take out charges on critical ions, such as the charges on molybdenum, nickel, calcium, etc. Fro example, because EcoCyc only contained the ions for molybdenum with a +6 and +2 charge, I added those to the native chemical txt files

- Some molecules don't standardize away other inherent charges. For instance, the molecule NAD+ can be written with a p+1, but in the EcoCyc data, "Unique-ID - NAD" it is written with a p-1. During the standardization, the p-1 goes away but the p+1 stays. Thus, I ended up having to just add another version of NAD to the native chemical txt files. Similar things happened for FAD and THF.

NAD+ InChI:
InChI=1S/C21H27N7O14P2/c22-17-12-19(25-7-24-17)28(8-26-12)21-16(32)14(30)11(41-21)6-39-44(36,37)42-43(34,35)38-5-10-13(29)15(31)20(40-10)27-3-1-2-9(4-27)18(23)33/h1-4,7-8,10-11,13-16,20-21,29-32H,5-6H2,(H5-,22,23,24,25,33,34,35,36,37)/p+1/t10-,11-,13-,14-,15-,16-,20-,21-/m1/s1
EcoCyc NAD InChI:
InChI=1S/C21H27N7O14P2/c22-17-12-19(25-7-24-17)28(8-26-12)21-16(32)14(30)11(41-21)6-39-44(36,37)42-43(34,35)38-5-10-13(29)15(31)20(40-10)27-3-1-2-9(4-27)18(23)33/h1-4,7-8,10-11,13-16,20-21,29-32H,5-6H2,(H5-,22,23,24,25,33,34,35,36,37)/p-1/t10-,11-,13-,14-,15-,16-,20-,21-/m1/s1

STANDARDIZED NAD+ InChI:

InChI=1S/C21H28N7O14P2/c22-17-12-19(25-7-24-17)28(8-26-12)21-16(32)14(30)11(41-21)6-39-44(36,37)42-43(34,35)38-5-10-13(29)15(31)20(40-10)27-3-1-2-9(4-27)18(23)33/h1-4,7-8,10-11,13-16,20-21,29-33H,5-6,23H2,(H3-,22,24,25,34,35,36,37)/p+1/t10-,11-,13-,14-,15-,16-,20-,21-/m1/s1

STANDARDIZED EcoCyc NAD InChI:

InChI=1S/C21H28N7O14P2/c22-17-12-19(25-7-24-17)28(8-26-12)21-16(32)14(30)11(41-21)6-39-44(36,37)42-43(34,35)38-5-10-13(29)15(31)20(40-10)27-3-1-2-9(4-27)18(23)33/h1-4,7-8,10-11,13-16,20-21,29-33H,5-6,23H2,(H3-,22,24,25,34,35,36,37)/t10-,11-,13-,14-,15-,16-,20-,21-/m1/s1

- There were two different versions of the SAM molecule in the EcoCyc database, and their only differences in their InChIs was a 27- and a 27+. These pass through the standardizer but do not pair down to the same chemical object, so both were included in the native chemicals txt file.

S-ADENOSYLMETHIONINE InChI:

InChI=1S/C15H22N6O5S/c1-27(3-2-7(16)15(24)25)4-8-10(22)11(23)14(26-8)21-6-20-9-12(17)18-5-19-13(9)21/h5-8,10-11,14,22-23H,2-4,16H2,1H3,(H2-,17,18,19,24,25)/p+1/t7-,8+,10+,11+,14+,27-/m0/s1

CPD0-2554 InChI:

InChI=1S/C15H22N6O5S/c1-27(3-2-7(16)15(24)25)4-8-10(22)11(23)14(26-8)21-6-20-9-12(17)18-5-19-13(9)21/h5-8,10-11,14,22-23H,2-4,16H2,1H3,(H2-,17,18,19,24,25)/p+1/t7-,8+,10+,11+,14+,27+/m0/s1

STANDARDIZED S-ADENOSYLMETHIONINE InChI:

InChI=1S/C15H22N6O5S/c1-27(3-2-7(16)15(24)25)4-8-10(22)11(23)14(26-8)21-6-20-9-12(17)18-5-19-13(9)21/h5-8,10-11,14,22-23H,2-4,16H2,1H3,(H2-,17,18,19,24,25)/p+1/t7?,8-,10-,11-,14-,27+/m1/s1

STANDARDIZED CPD0-2554 InChI:

InChI=1S/C15H22N6O5S/c1-27(3-2-7(16)15(24)25)4-8-10(22)11(23)14(26-8)21-6-20-9-12(17)18-5-19-13(9)21/h5-8,10-11,14,22-23H,2-4,16H2,1H3,(H2-,17,18,19,24,25)/p+1/t7?,8-,10-,11-,14-,27-/m1/s1

Standardization Red Flags

For the first red flag, I found that this standardization was actually getting rid of critical stereochemistry for certain amino acids. For instance, L-Serine and D-Serine, which are biologically very distinct, standardize down to the same InChI. I discovered this because the L-Serine molecule that I put in the native chemicals txt file actually populated to a D-serine Chemical object.

D-Serine InChI:

InChI=1S/C3H7NO3/c4-2(1-5)3(6)7/h2,5H,1,4H2,(H,6,7)/t2-/m1/s1

L-Ser InChI:

InChI=1S/C3H7NO3/c4-2(1-5)3(6)7/h2,5H,1,4H2,(H,6,7)/t2-/m0/s1

STANDARDIZED D-Serine InChI:
InChI=1S/C3H7NO3/c4-2(1-5)3(6)7/h2,5H,1,4H2,(H,6,7)
STANDARDIZED L-Ser InChI:
InChI=1S/C3H7NO3/c4-2(1-5)3(6)7/h2,5H,1,4H2,(H,6,7)

I've tested this on a handful, and I believe this is the case with **all** of the amino acids, which means this issue is rife throughout the entire EcoCyc chemical dataset. However, ***removing the tautomerization code*** completely maintains the L and D stereochemistry, so ultimately, Dr. Anderson proposes that we might need to eliminate the tautomerization code and simply write our own tautomerization reaction operators.

For the second red flag, I did find that moving around the order of the various RDKit functionalities could change the success of standardization down to the same molecule. Although the current order that I have it in is by far the most robust, I found that the standardization of different forms of FAD varied based on whether I had the AddHs at the top before the normalize_molecule or only right before the apply_smirks_transformation.

- Current (AddHs right before apply_smirks_transformation)

FAD InChI:
InChI=1S/C27H33N9O15P2/c1-10-3-12-13(4-11(10)2)35(24-18(32-12)25(42)34-27(43)33-24)5-14(37)19(39)15(38)6-48-52(44,45)51-53(46,47)49-7-16-20(40)21(41)26(50-16)36-9-31-17-22(28)29-8-30-23(17)36/h3-4,8-9,14-16,19-21,26,37-41H,5-7H2,1-2H3,(H5,28,29,30,34,42,43,44,45,46,47)/p-3/t14-,15+,16+,19-,20+,21+,26+/m0/s1
EcoCyc FAD InChI:
InChI=1S/C27H33N9O15P2/c1-10-3-12-13(4-11(10)2)35(24-18(32-12)25(42)34-27(43)33-24)5-14(37)19(39)15(38)6-48-52(44,45)51-53(46,47)49-7-16-20(40)21(41)26(50-16)36-9-31-17-22(28)29-8-30-23(17)36/h3-4,8-9,14-16,19-21,26,37-41H,5-7H2,1-2H3,(H5,28,29,30,34,42,43,44,45,46,47)/t14-,15+,16+,19-,20+,21+,26+/m0/s1

STANDARDIZED FAD InChI:
InChI=1S/C27H34N9O15P2/c1-10-3-12-13(4-11(10)2)35(24-18(32-12)25(42)34-27(43)33-24)5-14(37)19(39)15(38)6-48-52(44,45)51-53(46,47)49-7-16-20(40)21(41)26(50-16)36-9-31-17-22(28)29-8-30-23(17)36/h3-4,8-9,14-16,19-21,26,37-42H,5-7H2,1-2H3,(H5,28,29,30,34,43,44,45,46,47)/t14-,15+,16+,19-,20+,21+,26+/m0/s1
STANDARDIZED EcoCyc FAD InChI:
InChI=1S/C27H33N9O15P2/c1-10-3-12-13(4-11(10)2)35(24-18(32-12)25(42)34-27(43)33-24)5-14(37)19(39)15(38)6-48-52(44,45)51-53(46,47)49-7-16-20(40)21(41)26(50-16)36-9-31-17-22(28)29-8-30-23(17)36/h3-4,8-9,14-16,19-21,26,37-41H,5-7H2,1-2H3,(H5,28,29,30,34,42,43,44,45,46,47)/t14-,15+,16+,19-,20+,21+,26+/m0/s1

- Alternative (AddHs right before normalize_molecule)

STANDARDIZED FAD InChI:
InChI=1S/C27H33N9O15P2/c1-10-3-12-13(4-11(10)2)35(24-18(32-12)25(42)34-27(43)33-24)5-14(37)19(39)15(38)6-48-52(44,45)51-53(46,47)49-7-16-20(40)21(41)26(50-16)36-9-31-17-22(28)29-8-30-23(17)36/h3-4,8-9,14-16,19-21,26,37-41H,5-7H2,1-2H3,(

H5,28,29,30,34,42,43,44,45,46,47)/t14-,15+,16+,19-,20+,21+,26+/m0/s1
STANDARDIZED EcoCyc FAD InChI:
InChI=1S/C27H33N9O15P2/c1-10-3-12-13(4-11(10)2)35(24-18(32-12)25(42)34-27(43)33-24)5-14(37)19(39)15(38)6-48-52(44,45)51-53(46,47)49-7-16-20(40)21(41)26(50-16)36-9-31-17-22(28)29-8-30-23(17)36/h3-4,8-9,14-16,19-21,26,37-41H,5-7H2,1-2H3,(H5,28,29,30,34,42,43,44,45,46,47)/t14-,15+,16+,19-,20+,21+,26+/m0/s1

The takeaway is that simply changing the order of the AddHs completely changes the outcome of standardization success. Although this was caught because it was very apparent to me that my native chemical txt value wasn't matching up with any chemical object in the Chemical object dictionary, this could be happening behind the scenes for other standardization and matching procedures, and this should be considered a deficiency. I did my absolute best to test other ordering combinations, and I found that the current std function order is by far the best order and this FAD issue is the only major change I have found. However, this should still be scrutinized and looked into more closely.

C. Other Data Processing to Enable Robust Concrete Synthesis

Mismatched Shells
Early evaluation of the glycolysis pathway using the output of the hypergraph showed a mismatching in the shells for pathways that were supposed to be sequential. For example, steps two through four of glycolysis (with unique IDs PGLUCISOM-RXN, 6PFRUCTPHOS-RXN, and F16ALDOLASE-RXN) were all populating the same shell (shell #2). I was able to troubleshoot this by ensuring that a reaction can only be enabled if all its substrates were available in previous shells, not in the current shell being processed. This adjustment fixed the glucose issue and generally allowed the sequential nature of metabolic pathways to be accurately logged in each synthesis shell.

Reaction Direction
When I was confirming the glycolysis pathway, I also found that 2 of the steps were not populating correctly with the synthesis algorithm because the substrates and products were flipped around. However, these two reaction's directions were labeled "REVERSIBLE" and "PHYSIOL-RIGHT-TO-LEFT." For reversible reactions, I defined a function called preprocess_reactions that appends the original reaction to the preprocessed list and also creates a reversed version of it (Unique_ID becomes UNIQUE_ID + "_reverse"), treating the products as substrates and vice versa, marking the direction as left-to-right. For reactions that are inherently right-to-left, the function reverses them to be left-to-right without duplicating them.

D. Main Reason for Abandonment: BioCyc Class Chemicals Workaround

A huge overarching issue with the EcoCyc data has been something that Peter Karp warned us of at the beginning of the year: "class" compounds. These class compounds are groups of related metabolites that are present in a different file called classes.dat, and referenced without

a concrete InChI in compounds.dat. However, the big issue is that these class compounds are referenced directly in the reactions.dat, representing many possible chemical combinations for a certain reaction but only with one entry. For example, a reaction that could take place with many different types of ubiquinones, not just one specific one, lists the non-specific class compound, “Ubiquinones” as one of its substrates. This helps significantly simplify the reaction data, but it is a huge hindrance to making the hypergraph concrete in terms of chemical structure. To illustrate the quinones/quinols classes that I discovered by using Command/Ctrl F in the compounds.dat file, I have put a hierarchy in Appendix 1. However, these class compounds cannot be ignored; many of them are vital cofactors, and many of the core biosynthesis pathways were broken in the hypergraph because of the gaps of these class compounds, so it has been imperative to incorporate them in some way in a concrete manner.

Just to be more specific about some of the problematic class compounds that we found: For instance, when I was evaluating the TCA cycle, I discovered the tip of the iceberg of the uniquinones problem. I found references to categories like Quinones, ETR-Quinones, Modified-Menaquinones, etc. as substrates and products within the reactions.dat file produced by EcoCyc. When evaluating the amino amino-acid biosynthetic pathways, I found NAD-P-OR-NOP, NADH-P-OR-NOP, Donor H2, Acceptor, etc. All of these classes, and the details about their associated structure, can be found in compounds.dat, and I listed them in this document as Appendix 2.

Although we considered duplicating the reactions to include a version with each concrete chemical (ex. 20 specific reactions for a reaction containing “Amino-Acids” as a substrate) or abstracting each class compound as one specific concrete chemical (i.e. pointing every type of quinone to the Unique-ID - Quinone object), we came up with a different solution to try before we realized this problem was more problematic. Many of the class chemicals can be treated as concrete chemicals because they contain a non-standard InChI (or a SMILES or an R-SMILES). Often, these non-standard InChI’s just represent the overarching chemical structure that is related among the metabolites in the class. Moreover, there are other options for chemical structure other than the non-standard InChIs such as SMILES strings, which could represent SMILES strings or SMARTS strings. Thus, I adopted code in the current filter based on the standardization to also retain these class compounds. I will explain the original version first though.

The following two paragraphs describe why there are “Two Options for Data Filtering” listed in the code. The first code is the original code that only worked on concrete chemicals with InChI’s that can be standardized. This code block is called CONCRETE ONLY. It iterated through all of the EcoCyc reactions. First, if the reaction did not have any substrates OR if the reaction did not have an associated an Inchi within the EcoCyc_Chemicals_dictionary for that substrate, reaction_valid = False and that reaction was thrown out. However, if there was an Inchi, we would attempt to employ my standardizer code. If we did not get it to standardize to a valid Inchi, reaction_valid = False and that reaction was thrown out. But if it was successful, as long as a standardized version was not already there, we logged the Inchi into chemicals_by_inchi as the key and Chemical object as the value (with the new standardized inchi. This process was

repeated by iterating through products If reaction remained valid, we stuck these as substrates and products into the new Reaction object and appended it to reactions_list. Both of these data structures (chemicals_by_inchi and reactions_list) were employed directly in the synthesis algorithm.

RETAINS CLASS COMPOUNDS was my attempt to treat class compounds like concrete chemicals in an effort to allow for more biosynthesis pathways to enumerate. Like the first version, the code sequentially processed each substrate and product of EcoCyc_Reactions. It first attempted to standardize chemicals using their standard InChIs. If a chemical didn't have a standard InChI or standardization doesn't work, it progressed to check for a non-standard InChI (NS InChI) and attempted to convert it into a mol object using RDKit. If there was no NS InChI, then it checked if there is a SMILES string first by attempting to use RDKit to interpret it as a SMILES string, then as a SMARTS. Each successfully processed chemical was converted from an EcoCyc_Chemical to a Chemical object with the proper Boolean flag flipped to true. If all substrates and products of a reaction are successfully processed, the reaction itself was converted from an EcoCyc_Reaction to a Reaction object and added to reactions_list. The Chemical objects were stored in a dictionary called "Chemicals_by_inchi" to match the original ontology. NOTE: This version required using the Chemical objects with the HAS_NS_INCHI, HAS_SMILES, and HAS_SMIRKS.

I carried the RETAINS CLASS COMPOUNDS filtering strategy out even though I knew there were innate limitations of this workaround. For instance, cofactors that are class compounds that do not have any associated structure (but are still important for critical reactions) are ignored in this code. This includes the abstraction of a "Donor H2", "Donor-H1", or "Acceptor", which is meant to describe scenarios when reduced/oxidized electron carriers are not specified. The code doesn't catch these, so I wouldn't have come across these without realizing they were the substrates/products of a crucial reaction needed for serine biosynthesis.

Finally, after I completed this workaround, I realized that the hypergraph wasn't improved by the inclusion of these chemicals because many class compounds were supposed to be in shell 0. This was the beginning of the end for abandoning this strategy for the project. We know that the weird abstractions for cofactors listed as class compounds, such as NAD-P-OR-NOP or NADH-P-OR-NOP, are not included in the native metabolites list. This is the same for all of the class cofactors that I previously mentioned and listed in Appendix 2. However, in examining the data that made it through the filter using the get_chemicals_by_flag function with the HAS_NS_INCHI as true, I noticed a different category of class compounds which are the major categories of biomolecules. To illustrate, our external flat file specifically mentioned that all 20 separate amino acids should be in shell 0, but a class compound abstraction like "Amino-Acids" is obviously not listed on that file and doesn't make it into shell 0. This made us realize that we were up against an issue larger than co-factors, and the strategy of sifting by hand through all of the chemicals that are listed as having a non-standard inchi or only a SMILES/SMARTS is not a robust strategy.

E. Conclusion

This synthesis model construction started out promising: The total sizes of Chemical and Reaction to Shell were 600 chemicals and 1450 reactions, respectively, and glycolysis, TCA cycle, and 10/20 amino acid biosynthesis pathways were found to be completely intact. I believe this scale and initial accuracy of a whole cell metabolic model has not been achieved with this type of synthesis. However, we found out too late that the chemical abstractions for cofactors and other classes were critically important for enumerating even foundational biochemical pathways. In proposing workarounds to deal with what we originally thought were smaller issues, we realized that we had a bigger, more foundational problem with the EcoCyc data. For these reasons, Dr. Anderson and I decided to move away from using an Object Matching Synthesis strategy with workarounds for non-concrete chemicals. Moreover, we also wanted to switch the strategy because we were actively discussing the SAR-RO incorporation of my model, and we realized that an RO based synthesis foundation would be much easier to pair with other Anderson Lab tools like the enzymatic SAR models.

IV. Major Synthesis Pivot: Preliminary Steps and Future Roadmap

The following section describes the post-Object Matching Synthesis era in which I attempted to elucidate all the different buckets of data that needs to be incorporated in a future RO-based synthesis model, a barebones attempt at incorporating RO projects into the synthesis algorithm, and an outline of some of the steps that need doing and questions that need answer that Dr. Anderson and I have put together for the next student. Finally, I have also included a future directions portion for the Transformation Validator program, the original scope of this project.

A. Detailed EcoCyc Data Scenario Buckets

When we tried the class workarounds, we realized that we were coming across critical substrates and products that were represented in the compounds.dat file with a variety of combinations among the InChI, SMILES, and Non-Standard InChI (NS Inchi) fields. Therefore, Dr. Anderson instructed me to dump out the chemicals list into categories based on what fields are provided. I attempted to get as granular as possible, and this gave me a solid high-level viewpoint of the different scenarios that will be considered in a future synthesis reaction. I stored the outputs of each of the different prompts in a list, and all of this can be found in the Resources folder in the GitHub titled EcoCyc_DATA_EVALUATION.

I pasted below the length outputs of each list in the scenario filter, and I placed the prompts that I used in GPT to get this data in Appendix 3 in case the reader wants/needs a better explanation of where I got these “buckets.”

NOTE 1: I evaluated (by printing) all of these lists, and I highlighted the categories in red where I felt I saw chemicals and cofactors that should be placed in shell zero.
NOTE 2: The chemicals that have a SMILES value that cannot be parsed as SMILES but *can* be parsed as SMARTS are indicative of R-SMILES. If they can’t be parsed as either, they are an abstract SMILES. That’s why we made this distinction.

Length of EcoCyc_chemicals_dict: 7585
Length of ecocyc_chems_in_rxns: 2083

None
Number of chemicals with no InChI, no NSInChI, and no SMILES: 69

All 3
Number of chemicals with all three (InChI, NSInChI, SMILES): 1443
Number of chemicals with all three where SMILES can be parsed as SMILES: 1443
Number of chemicals with all three where SMILES can be parsed as SMIRKS: 1443

Number of chemicals with all three where SMILES can be parsed as BOTH SMILES and SMIRKS: 1443
Number of chemicals with all three where SMILES cannot be parsed as SMILES or SMIRKS: 0

InChI
Number of chemicals with InChI: 1454
Number of chemicals with InChI that cannot be parsed: 0
Number of chemicals with ONLY InChI: 0
Number of chemicals with BOTH InChI and NSInChI ONLY: 0
Number of chemicals with BOTH InChI and SMILES ONLY: 11
Number of chemicals with InChI and SMILES where SMILES cannot be parsed: 0

NS-Inchi
Number of chemicals with NSInChI: 1963
Number of chemicals with NSInChI (excluding ':STANDARD-INCHI') that cannot be parsed: 1
Number of chemicals with ONLY NSInChI: 1
Number of chemicals with BOTH NSInChI and SMILES ONLY: 519
Number of chemicals with NSInChI and SMILES where SMILES can be parsed as SMILES: 6
Number of chemicals with NSInChI and SMILES where SMILES can be parsed as SMIRKS: 180
Number of chemicals with BOTH NSInChI and SMILES ONLY where SMILES can be parsed as BOTH SMILES and SMIRKS: 6
Number of chemicals with NSInChI and SMILES where SMILES cannot be parsed as SMILES or SMIRKS: 339

SMILES
Number of chemicals with SMILES: 2013
Number of chemicals with SMILES that cannot be parsed as SMILES or SMIRKS: 368
Number of chemicals with ONLY SMILES: 40
Number of chemicals with ONLY SMILES that can be parsed as SMILES: 1
Number of chemicals with ONLY SMILES that can be parsed as SMIRKS: 11
Number of chemicals with ONLY SMILES that can be parsed as BOTH SMILES and SMIRKS: 1
Number of chemicals with ONLY SMILES that cannot be parsed as SMILES or SMIRKS: 29

Overall, I confirmed that there are many, many more compounds in the compounds.dat file than the unique compounds that are referenced in the reactions.dat. Thus, EcoCyc logs chemicals in their database that are never seen in any of *E. coli*'s reactions. Other takeaways: many things are as expected (all of the InChIs can be parsed as a mol object with RDKit, all but one NS InChI can be parsed as mol object).

However, the big takeaway is that the non-concrete cofactors and other generic class compounds that were giving us problems in the workarounds we attempted with the Object Matching Synthesis vary widely in the scenarios. For instance, I made this preliminary list to outline some of the non-concrete cofactors that I have been tracking for a while:

- **Acceptor** is in "Chemicals with no InChI, no NSInChI, and no SMILES: 69"
- **Amino-Acids, NAD-P-OR-NOP, THF-GLU-N** are in "Chemicals with NSInChI and SMILES where SMILES can be parsed as SMIRKS: 180"
- Things like **TOPAQUINONE** are in "Chemicals with NSInChI and SMILES where SMILES cannot be parsed as SMILES or SMIRKS: 339"
- **ETR-Quinones**, **ETR-Semiquinones**, **NADH-P-OR-NOP** are in "Chemicals with ONLY SMILES that can be parsed as SMIRKS: 11"
- **Donor-H1** and **Donor-H2** are in "Chemicals with ONLY SMILES that cannot be parsed as SMILES or SMIRKS: 29"

Also, of general concern is that we have 69 chemicals that have no InChI, no NS InChI, and no SMILES, some of which are critical cofactors. These might have to be handwritten. Also, we show that there are 368 chemicals with "abstract SMILES" that need to be dealt with. Some of these have an NSInChI (339) which could be promising and the remaining have only a SMILES (29).

I also set up what I believe to be a very user-friendly notebook (EcoCyc_DATA_EVALUATION) to evaluate the lists of these different scenarios, and the queries follow the same order as the prompts and the filter output does. A user can use the "Print List" section at the very bottom of the script to see the scripts, and the variables for each list are stored in the exact order of these outputs.

B. Dr. Anderson Thoughts

We interrupt your regularly scheduled programming to paste some preliminary thoughts and questions from Dr. Anderson about the future of the new synthesis algorithm strategy.

> Start with the assumption that what we are trying to get to is a SAR+RO model of each enzyme.
>
> Let's consider a different type of SAR model which I think will be easier to see how we might get there. According to ChatGPT (untested with RDKit), you can express SMARTS and SMIRKS with patterns like: [OH1]([CX4H3]|[CX4H2][CX4H3]|[CH2]c1ccccc1) Where it describes (in parenthesis) a list of possible substructures attached to oxygen ultimately encoding methanol, ethanol, or benzyl alcohol. In principle, class molecules could be described with these patterns. For example, with metaquinones, the quinone core would be the constant part of the RO, and the prenyl chain of diverse length would be expressed as a SMARTS pattern. If done this way, you could similarly describe abstract reactions involving

such class compounds as composed of these complicated SMARTS patterns. So, now let's consider scenarios of the data we have.

When we have R-SMILES, we essentially already have this abstract SMIRKS pattern. All we are missing to get to the SAR-like reaction operator is the atom-to-atom mapping. Lais' project involved figuring out how to run a state-of-the-art mapper, so we already have such code. It might require some trickery to enable the use of those mappers on non-concrete structures, but I think we can hack around that pretty easily. So, actually, R-SMILES is our best case scenario. With just added automapping we can convert those into reaction operators that could be directly used in the synthesis algorithm.

When we only have a non-canonical inchi (the simplest structure of the abstraction), we don't really have an expression of the pattern, so we can't directly convert that to an operator trivially. We could chop off all the hydrogens from those structures and turn that into an operator, though. This requires an assumption that the family represented by the class are all extensions of the non-canonical inchi's structure, which may or may not be true. But the information is not available otherwise. In terms of its effect on the expansion, I don't think it would be problematic event if it were incorrect -- as long as the generated operator works on the specific non-canonical inchi compound, that compound will enable any reactions requiring a member of the class. If we had non-canonical inchi and R-SMILES, we would use the R-SMILES.

When no valid inchi or smiles are provided, we would have to write them. But that's true of all scenarios.

To deal with abstract chemicals we must put in shell zero to enable expansion, the first question is: what determines that a compound is put in shell 0? I think there is a clear answer to this -- if enabling known pathways requires putting the chemical into shell 0, then it is added; otherwise it is allowed to be constructed during the expansion. Consider the scenario that the only way to make an abstract cofactor goes through a synthesis step involving all concrete reactions. Under that scenario, the concrete reactions assemble, at least one member of the abstract cofactor family is enabled, and thus future reactions requiring such a member are enabled. If that pathway involves an abstract reaction to produce the abstract cofactor, then this too would happen during synthesis by projection of the SAR-RO for that abstract reaction. There are fundamentally a few things that have to go into shell 0 or expansion cannot go to completion. ATP for sure is in there, but we don't empirically know what else. However, we can find them through a whackamole process once all the abstract reactions are in place. And when one is required, we would only need to add one member of the family to satisfy all abstract reactions referring to that cofactor.

This all provides a path towards doing speculative inference and population of the SAR models.

In that approach, we are describing (ultimately) each Enzyme as a reaction operator that encapsulates both its mechanism and specificity. The Gongao/Jess algorithms split that into 2 pieces and require a more elaborate algorithm to evaluate, but in theory their ML models could be converted into reaction operators, or there are just 2 types of SAR models in the system. In all cases, there is a reaction operator. For the Gongao/Jess models, those operators are the hchEROs. In the case of R-SMILES data it is a SAR-RO as discussed. So, for all enzymes, we have an RO. Without populating the SAR component of the Gongao/Jess models, we are only describing the reaction mechanism of the enzyme, and thus expansion with those ROs will lead to overprediction, as Hatzimanikatis reported. So, we would assign an RO for every reaction (abstract or concrete) provided in metacyc, and we would include that bag of ROs in the synthesis expansion. We would get polymerization and other production of things that don't actually exist in E. coli. But that's ok, because we can likely detect the errors. There is another database, ECMDB, that tries to inventory every chemical associated with E. coli metabolism. If the synthesis results in products that are not in ECMDB, we assume they are invalid predictions. Since they are invalid and assumed to be produced by no enzyme in the cell, then we can infer that all enzymes whose models predict that product are under-specified. The invalid chemical prediction is then inferred to be a negative substrate for each of those enzymes, and its SAR model is recomputed. Though an iterative process of such refinement followed by repeat of synthesis and error detection, one could populate SAR models for all enzymes that guarantee that the model does not exponentially explode. It could then be further optimized through predictions on unnatural substrates and feeding experiments based on molport matches.

This is a good email to read as a launchpad for some of the important considerations for the project. However, in discussing this email and future steps with Dr. Anderson, I mentioned that we have reactions that are “mixed” in the sense that they include concrete substrates/products with non-concrete co-factors (or vice versa) or reactions that have core components that are abstract (like “Amino Acids”). Apparently this was “not considered” in these original thoughts by Dr. Anderson, so maybe this changes things in the future.

Also, I wrote down these statements from our meeting: You can interpret R-SMILES as a statement about the specificity of the enzyme, and the SAR ensemble that another capstone student was generating used ROs that are hchEROs. Furthermore, R-Smiles can be turned into an SAR, and so can concrete chemicals, but this tends to be overspecific. The simplest case for a reaction is a mixed case where you have concrete cofactors and the substrates that are R-SMILES.

C. Hybrid RO-Object Matching Synthesis Algorithm

Dr. Anderson has frequently discussed in his lectures about having predictive improvements on a synthesis algorithm. For instance, in one of his lecture videos discussing L4 synthesis. He stated: “To modify the existing algorithm, we would add RO projections during ExpandOnce in Synthesizer. During initiate, we would read in all the hchEROs as RxnMolecules. During ExpandOnce we would iterate through all the RxnMolecules and project them onto each Reachable in the current shell. All expansion products are then added to the shell.”

To get a head start on next year’s transformative synthesis, Dr. Anderson directed me to build something similar as to what he has been discussing with the L4 synthesis in lecture: a new “hybrid” variation of the synthesis algorithm. This hybrid model would still have a foundational object matching backbone, but would also iterate through a bag of reaction operators and expand the shell accordingly if the RO can be projected on a particular substrate. He wanted this to be on a minimal pathway that could be evaluated and troubleshot easily, so I first isolated all of the glycolysis reactions and compounds from the standardized versions from the Object Matching Synthesis. These can be found in the unique_chemicals_glycolysis.json and glycolysis_reactions.json files.

First, I constructed a normal Object Matching Synthesis in the Hybrid-RO-Synthesis.ipynb notebook (within Synthesis 2.0). This followed the standard Object Matching Synthesis method and read in the JSON flat files, populated "Glucopyranose", "ATP", "NAD", and “Pi” as shell 0 compounds, and built the glycolysis hypergraph based on object matching. The result from the 10 step pathway was 9 shells, 10 reactions_to_shell, and 18 chemicals_to_shell. Then I duplicated and amended the synthesis algorithm to also iterate through a prewritten list of reaction operators and project them. This list only contained one handwritten reaction operator that performed a methylation on some of the sugars, and because it was creating novel Chemical objects, I came up with a labeling counter that named each new Chemical object Compound_1, Compound_2, and so on. This hybrid hypergraph expectedly had 9 shells, 10 reactions_to_shell, and 23 chemicals_to_shell. This is an important proof of concept to show that a hybrid synthesis can work and can *de novo* create expansion products.

I didn’t have time to fully perfect this algorithm, however. The main problem is that I sometimes produced two identical products when projecting the RO, and sometimes the RO products only differed by stereochemistry distinction, but they represented the same chemical. Although future ROs will likely be more specific than the handwritten RO, hopefully the future algorithm can incorporate some sort of filtering/standardizing the same mol objects down before they are logged as separate entities into the hypergraph.

D. Suggested Next Steps

This section indicates what I believe would be some of the immediate steps to be taken in the future of this project. Most of these are taken from conversations in the final meetings with Dr. Anderson, but some also come from my own ideas and shortcomings due to timing constraints.

Standardizing and Filtering within the Hybrid Synthesis Model

As I mentioned in the section above, the current hybrid RO model, found in Hybrid-RO-Synthesis.ipynb (Synthesis 2.0) still has a bug related to creating chemical products that are the same chemical but log separately. This might need standardization as well.

Addressing the Standardization Code Red Flags

As I mentioned in the Standardization Performance and Standardization Red Flags section, although there have been some impressive successes with the current code, there are also serious limitations. If this code is to be used properly for this project, and be adopted broadly by the Anderson Lab as a modular tool, it needs to be much more fool proof. Because the tautomerization section caused a lot of trouble and was the reasoning for the combining of L- and D- amino acids, Dr. Anderson repeatedly mentioned the possibility of handwriting some tautomerization reaction operators ourselves, but we never got around to it this semester. Also, further examining the order of the code would be very interesting to see if any further ordering changes change the standardization outcome of 2 different chemicals that are the same but written differently, like the FAD case.

Make a Scenario Bucket for Hybrid Reactions

We know that some of these reactions that need to be transferred into ROs are going to not just link concrete chemicals. Instead, it might contain class cofactors, class substrates/products, abstract-SMILES, or a combination of any and all of these. Therefore, running a granular filtering algorithm on all the reactions and their associated substrates and products to figure out exactly how many reactions are just concrete in nature or have abstract/class chemicals would be a very good starting point. Also, Dr. Anderson can help you decide what is concrete, because maybe as the project progresses, a chemical that has an R-SMILES might be considered “concrete.”

Deeply Examine the Scenarios of Chemicals

In the first subsection of this portion of the documentation, I outlined the Detailed EcoCyc Data Scenario Buckets. This provides a very granular look at what the makeup of the data is that needs to go into the synthesis. I highlighted certain lists where some of the supposed Shell 0 cofactors and chemicals are, but you may elucidate more important details. Closely examining these entries to see what types of chemicals have what type of text structural data (and potentially using Ctrl F to search through reactions.dat to see how impactful each chemical is in the reactions) will seriously benefit you before you make a new model. Particularly look at the 69 chemicals that have no InChI, no NS InChI, and no SMILES because those might have to be handwritten.

Building a Barebones RO-Only Synthesis Model

This seems to be the big overarching future part of the project. I have some very simple linear Glycolysis data that can be used to first try this out, but the goal will be to set up a way to enable synthesis entirely on the projection of reaction objects. Maybe the whole algorithm needs to be based on RDKit mol objects instead of InChI strings, but that might be computationally intensive. The complexity of this model will come from the cofactor scenarios. Maybe the first pass of the reaction needs to be built solely on half-reactions (what you might see in a chemistry textbook). Nonetheless, it will be important to base this algorithm on a set of well-constructed, accurate reaction operators. There are a couple of options on how to get here: use the RO Extractor that is still be developed, handwrite them yourself, or use RO-Matcher
Moreover, this synthesis attempt will likely start with just simple concrete data, so feel free to use the Glycolysis JSON data to generate ROs. You can compare the intended hypergraph with the original object matching method found in Hybrid-RO-Synthesis.ipynb to make sure you get the glycolysis part right. If you want to add in more pathways, I spent a lot of time validating various amino acid biosynthesis pathways, and I used Ctrl F in the reactions.dat file and searched the IN-PATHWAY data field to find related reactions. For example, the reactions labeled for Tryptophan biosynthesis are flagged with IN-PATHWAY - TRPSYN-PWY. Finally, this synthesis will of course come up against issues like dealing with the labeling of the *de novo* the compounds, the data that lacks structural string labels, or dealing with abstract SMILES, but these are referenced in the Open Questions section below.

Use a New Version
On April 15th, 2024, I received an email saying that a new version of BioCyc is available, which includes updates to EcoCyc. The email detailed this: *Updates to EcoCyc Version 28.0. Information from more than 200 publications have been added for this release, which included improvements to various transcription units.* It might be the best laid plan to continue to work with the most updated version of EcoCyc, and this update might solve some of the weird bugs and undetailed class compounds/cofactors that we have found.

E. Open Questions

This section outlines some of the open questions that remain for this project. I strongly advise referencing the “Dr. Anderson Thoughts” section to get a feel for where these come from.

How to Deal with Hybrid Reactions
From my understanding, we have yet to elucidate a general strategy for converting the reactions in EcoCyc to ROs, but I believe it will be a very particular case if the reaction happens to include a combination of abstract/class chemicals and concrete chemicals. Finding out how many of these exist with some sort of scenario filter (referenced in “Suggested Next Steps” would be super helpful).

Labeling an RO-based Synthesis
If you have a complete RO-based Synthesis, you will be *de novo* generating a bunch of unnamed chemicals, so it will be hard to troubleshoot the completeness. Moreover, like the literature has predicted, the model will undeniably overpredict the biochemical reality which will

further clutter the model's accuracy. Thus, coming up with some sort of naming or referencing strategy will be super helpful. And Dr. Anderson has already proposed a strategy for refining the model to biochemical reality using a database like ECMDB.

Writing Structures for Unlabeled Data
As I mentioned in the Detailed EcoCyc Data Scenario Buckets, 69 chemicals that have no InChI, no NS InChI, and no SMILES, and some of them are critical cofactors. Are these going to be handwritten? Is that the best and only strategy?

Dealing with Abstract SMILES
Dr. Anderson has proposed workarounds for working with these many times. Maybe trying to do some sort of filtering and counting to understand how many compounds have the same reference to an abstract structure (like "a protein" or "a tRNA") might help.

Dealing with Unforeseen Data Connectivity Issues
From reading this documentation, it has become obvious that many of the issues related to the future of this project revolve around class compounds and non-concrete chemicals/cofactors. However, so far, all of these have been associated with chemicals that exist in both reactions.dat as substrates or products and as unique compounds.dat entries. However, it was discovered early on in the first semester that 935 substrates/products listed in reactions.dat do not correspond to any UNIQUE-ID found in compounds.dat. These were typically very abstract compounds (i.e. "a-5-deoxyribose-5-phosphate-DNA", "ssRNA-with-3phosphate", or "Reduced-Cys2-Peroxiredoxins") and mostly likely exist in classes.dat – all of these examples do but for the rest of the 932, this is unconfirmed. Thus, the future of the project likely also needs to consider the separate classes.dat file, and the number of unwritten data is likely **much, much higher** than the 69 chemicals that have been reported in compounds.dat with no InChI, no NS InChI, and no SMILES.

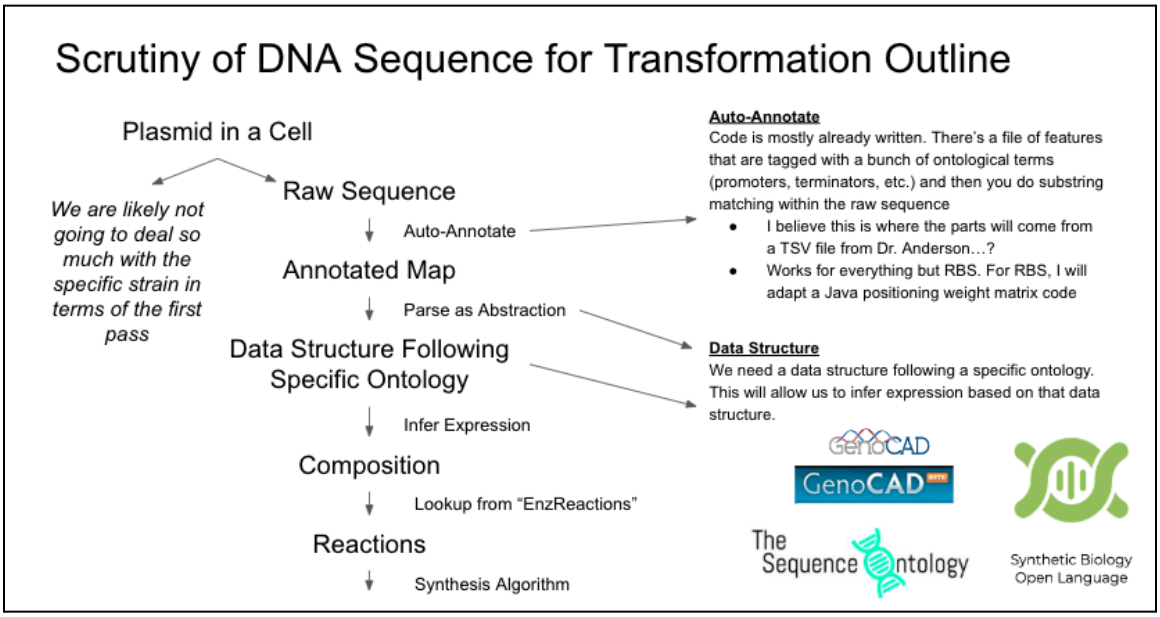
Applicability to Other BioCyc Databases
Future directions might be to adopt this code for other model organisms, such as the eukaryote *Saccharomyces cerevisiae*. YeastCyc, another robust pathway and genome database for *S. cerevisiae* provided by BioCyc, might have a similar flat file format release that might help test the translatable applications over this whole-cell biochemical framework. Exploring if this program can be a plug-in-play and translatable might be the major value of this biochemical reconstruction over the countless other ones available.

F. Next Components for Transformation Validator

Although this was originally the scope of the entire project, the second part of Transformation Validator, the scrutinization process for the actual transformed DNA, was not actually explored in a ton of detail. However, there were some potential starting ideas proposed by Dr. Anderson, such as turning the commonly used Cloning Checklist (Cloning Checklist) into an automated program. This checklist contains exhaustive questions that are meant to double-check a

synthetic biologist's cloning strategy before ordering oligonucleotides or conducting time-consuming, expensive steps in the wet lab. Dr. Anderson believes that many of these questions could turn into a prompt for an LLM.

In terms of the actual raw DNA sequence scrutiny portion of the program, the goal is to re-express the DNA as a bag of features, and the rest of the program would validate that list of features. Thus, a big portion of the future project involves assigning some sort of annotation structure and grammatical validation to effectively predict what enzyme might be expressed and added to the hypergraph. Much of the annotation code is already pre-written by the Anderson Lab and/or could be adopted for this project, but the only thing that he doesn't have a solution for is the ontology that would allow the prediction of expression based on a specific data structure. I did some preliminary research on this ontology, particularly reading the GenoCAD paper (Rule-Based Design of Plant Expression Vectors Using GenoCAD - PMC) to explore grammars for creating a genetic structure. The paper claims that GenoCAD's default grammar, '*E. coli Expression Grammar*', is suitable for the design of prokaryotic expression constructs; this could potentially be used in the future and turned into a data class in Python.



Lastly, I have pasted above a flowchart that Dr. Anderson and I authored in December 2023 which outlines the strategy for connecting raw plasmid or oligonucleotide sequence to the synthesis algorithm. The biggest black box is the transition from annotated sequence to a data structure that can infer expression. Moreover, the connection from a Composition (a Reaction and the associated Enzyme) to Reactions using the EcoCyc enzreactions.dat flat file release from BioCyc has not thoroughly been explored, but it is assumed to work.

If you have any questions, please feel free to contact my personal email which can be provided by Dr. Anderson.

V. Appendix

1. Quinones/Quinols

- UNIQUE-ID - Quinones
    - **Within Types - Quinones**
        - UNIQUE-ID - ETR-Quinones
        - UNIQUE-ID - TOPAQUINONE
        - UNIQUE-ID - CPD-12145 (COMMON-NAME - tryptophan tryptophylquinone)
        - UNIQUE-ID - CPD-12146 (COMMON-NAME - copper-complexed cysteinyltyrosyl radical cofactor)
        - UNIQUE-ID - Semiquinones
        - UNIQUE-ID - A-GLUCOSYLOXYANTHRAQUINONE
        - UNIQUE-ID - p-Quinones
        - UNIQUE-ID - Anthraquinones
        - UNIQUE-ID - Unknown-Quinones
        - UNIQUE-ID - Naphthoquinones
        - UNIQUE-ID - Vitamin-E (also in TYPES - Chromanes and Vitamins…)
        - UNIQUE-ID - CPD-3766 (menadione)
        - UNIQUE-ID - PQQ (COMMON-NAME - pyrroloquinoline quinone)
        - UNIQUE-ID - CPD-7248 (COMMON-NAME - 6-decylubiquinone)
        - UNIQUE-ID - CPD0-1288 (COMMON-NAME - 8-hydroxyquinoline-5-sulfonic acid)
        - UNIQUE-ID - CPD-19233 (COMMON-NAME - 3,3',5,5'-tetramethoxydiphenoquinone)
            - **Within Types ETR Quinones**
                - UNIQUE-ID - Ubiquinones
                - UNIQUE-ID - Modified-Menaquinones
                - UNIQUE-ID - PLASTOQUINONE
                - UNIQUE-ID - Rhodoquinones
                - UNIQUE-ID - Menaquinones
            - **Within Types Semiquinones**
                - UNIQUE-ID - Benzosemiquinones
                - UNIQUE-ID - ETR-Semiquinones
            - **Within Types p-Quinones and Naphthoquinones**
                - UNIQUE-ID - 1-4-Naphthoquinoness
            - **Within Types Anthraquinones**
                - UNIQUE-ID - Hydroxyanthraquinones

        - UNIQUE-ID - CPD-12792
        - UNIQUE-ID - CPD0-2587
            - **Within Types Menaquinone**
                - UNIQUE-ID - CPD-9728 (COMMON-NAME - menaquinone-8)
                - UNIQUE-ID - CPD0-1367 (COMMON-NAME - menaquinone-1)
            - **Within Types Ubiquinones**
                - UNIQUE-ID - UBIQUINONE-8
                - UNIQUE-ID - UBIQUINONE-10
                - UNIQUE-ID - UBIQUINONE-2
                - UNIQUE-ID - CPD0-1118 (UBIQUINONE-1)
                - UNIQUE-ID - UBIQUINONE-6
                - UNIQUE-ID - CPD0-1464 (UBIQUINONE-0)
            - **Within Types Modified-Menaquinones**
                - UNIQUE-ID - 8-Methylmenaquinones
                - UNIQUE-ID - Demethylmenaquinones
                - UNIQUE-ID - Dihydromenaquinones
                - UNIQUE-ID - 8-Methyldemethylmenaquinones
            - **Within Types 1-4-Naphthoquinoness**
                - UNIQUE-ID - HYDROXYNAPHTHOQUINONE
                    - **Within Types Demethylmenaquinones**
                        - UNIQUE-ID - DEMETHYLMENAQUINONE
- UNIQUE-ID - Reduced-Quinones
    - **Within Types - Reduced-Quinones**
        - UNIQUE-ID - Unknown-Quinols
        - UNIQUE-ID - 2-Methoxy-6-Polyprenyl-14-Benzoquinols
        - UNIQUE-ID - 6Methoxy5Methyl2Polyprenyl14Benzoquinol
        - UNIQUE-ID - ETR-Quinols
        - UNIQUE-ID - HYDROQUINONE
        - UNIQUE-ID - CPD-7249 (COMMON-NAME - 6-decylubiquinol)
        - UNIQUE-ID - MENADIOL

- UNIQUE-ID - CPD-19763 (COMMON-NAME - 2,3,5-trichloro-6-(glutathion-S-yl)-hydroquinone)
    - **Within Types 2-Methoxy-6-Polyprenyl-14-Benzoquinols**
        - UNIQUE-ID - OCTAPRENYL-METHOXY-BENZOQUINONE
    - **Within Types 6Methoxy5Methyl2Polyprenyl14Benzoquinol**
        - UNIQUE-ID - OCTAPRENYL-METHYL-METHOXY-BENZQ
    - **Within Types ETR-Quinols**
        - UNIQUE-ID - Plastoquinols
        - UNIQUE-ID - Rhodoquinols
        - UNIQUE-ID - Demethylated-Ubiquinols
        - UNIQUE-ID - Ubiquinols
        - UNIQUE-ID - Menaquinols
            - **Within Types Demethylated-Ubiquinols**
                - UNIQUE-ID - OCTAPRENYL-METHYL-OH-METHOXY-BENZQ
            - **Within Types Ubiquinols**
                - UNIQUE-ID - CPD0-2061 (ubiquinol-1)
                - UNIQUE-ID - CPD-9956 (ubiquinol-8)
            - **Within Types Menaquinols**
                - UNIQUE-ID - Dihydromenaquinols
                - UNIQUE-ID - Modified-Menaquinols
                - UNIQUE-ID - REDUCED-MENAQUINONE
                    - **Within Types Modified-Menaquinols**
                        - UNIQUE-ID - Demethylmenaquinols

2. Classes That Were Discovered When Evaluating Known Biochemical Pathways for Model Validation

(Bolded means they they are found as a reference within the reactions.dat file)
(RED means anything else but the non-standard inchi)

1. **Quinones**
    a. **In Reactions: Yes**
    b. **NS Inchi: Yes**
2. **ETR-Quinones**
    a. **In Reactions: Yes**
    b. **NS Inchi: No (R-SMILES only)**
3. **Semiquinones**
    a. **In Reactions: Yes**
    b. **NS Inchi: Yes**
4. p-Quinones
    a. In Reactions: No
    b. NS Inchi: No (NONE)
5. Naphthoquinones
    a. In Reactions: No
    b. NS Inchi: No (NONE)
6. Anthraquinones
    a. In Reactions: No
    b. NS Inchi: Yes
7. **Menaquinone**
    a. **In Reactions: Yes**
    b. **NS Inchi: Yes**
8. **Ubiquinones**
    a. **In Reactions: Yes**
    b. **NS Inchi: Yes**
9. Modified-Menaquinones
    a. In Reactions: No
    b. NS Inchi: No (NONE)
10. 1-4-Naphthoquinoness
    a. In Reactions: No
    b. NS Inchi: No (NONE)
11. Demethylmenaquinones
    a. In Reactions: No
    b. NS Inchi: No (NONE)

12. **Reduced-Quinones**
    a. **In Reactions: Yes**
    b. **NS Inchi: Yes**
13. 2-Methoxy-6-Polyprenyl-14-Benzoquinols
    a. In Reactions: No
    b. NS Inchi: Yes

14. 6Methoxy5Methyl2Polyprenyl14Benzoquinol
    a. In Reactions: No
    b. NS Inchi: Yes
15. ETR-Quinols
    a. In Reactions: Yes
    b. NS Inchi: Yes
16. Demethylated-Ubiquinols
    a. In Reactions: No
    b. NS Inchi: Yes
17. Ubiquinols
    a. In Reactions: Yes
    b. NS Inchi: Yes
18. Menaquinols
    a. In Reactions: Yes
    b. NS Inchi: Yes
19. Modified-Menaquinols
    a. In Reactions: No
    b. NS Inchi: No

20. THF-GLU-N
    a. In Reactions: Yes
    b. NS Inchi: Yes
21. NAD-P-OR-NOP
    a. In Reactions: Yes
    b. NS Inchi: Yes
22. NADH-P-OR-NOP
    a. In Reactions: Yes
    b. NS Inchi: No (R-SMILES)
23. Donor-H1
    a. In Reactions: Yes
    b. NS Inchi: No (NONE)
24. Donor-H2
    a. In Reactions: Yes
    b. NS Inchi: No (NONE)
25. Acceptor
    a. In Reactions: Yes
    b. NS Inchi: No (NONE)

26. FORMYL-THF-GLU-N
    a. In Reactions: Yes

   b. **NS Inchi: Yes**
27. **5-10-METHENYL-THF-GLU-N**
   a. **In Reactions: Yes**
   b. **NS Inchi: Yes**
28. **METHYLENE-THF-GLU-N**
   a. **In Reactions: Yes**
   b. **NS Inchi: Yes**
29. **N5-Formyl-THF-Glu-N**
   a. **In Reactions: Yes**
   b. **NS Inchi: Yes**

3. Prompts that Were Inputted into ChatGPT to Get the Scenario Filter

How many total EcoCyc_Chemicals are there?
Which EcoCyc_Chemicals are referenced as being either a substrate or a product of the EcoCyc_Reactions?

Now in these queries, we are only considering the 3 fields which would be indicative of chemical structure (Inchi, Non-standard-inchi, and SMILES)

Of all the EcoCyc_Chemicals present while iterating through the substrates and products of the EcoCyc_Reactions (not converting to the new dataclass yet),

None

1. Which have No Inchi, No NSinchi, and no SMILES

All 3

1. Which have all 3 (Inchi, NSinchi and SMILES)?
   a. Of those, which have SMILES that can be parsed as a SMILES (RDKit MolFromSmiles)?
   b. Of those, which have SMILES that can be parsed as a SMIRKS (RDKit MolFromSmirks)?
   c. Of those, which have SMILES that can be parsed BOTH as a SMILES (RDKit MolFromSmiles) or a SMIRKS (RDKit MolFromSmirks)?
   d. Of those, which have SMILES that CANNOT be parsed as a SMILES or SMIRKS (mol object fails)

Inchi

1. Which have Inchi?

    a. Of those, which have Inchi that CANNOT be parsed as a mol (RDKit MolFromInchi)?
2. Which have ONLY Inchi (No NSinchi or SMILES)?
3. Which have BOTH an Inchi and NSinchi ONLY?
4. Which have BOTH an Inchi and a SMILES ONLY?
    a. Of those, which have SMILES that CANNOT be parsed as a SMILES or SMIRKS?

NSInchi

1. Which have a NSinchi?
    a. Of those, which have NSinchi that CANNOT be parsed as a mol (RDKit MolFromInchi)?
2. Which have ONLY a NSinchi?
3. Which have BOTH a NSinchi and a SMILES ONLY?
    a. Of those, which have SMILES that can be parsed as a SMILES (RDKit MolFromSmiles)?
    b. Of those, which have SMILES that can be parsed as a SMIRKS (RDKit MolFromSmirks)?
    c. Of those, which have SMILES that can be parsed BOTH as a SMILES (RDKit MolFromSmiles) or a SMIRKS (RDKit MolFromSmirks)?
    d. Of those, which have SMILES that CANNOT be parsed as a SMILES or SMIRKS (mol object fails)

SMILES

1. Which have a SMILES?
    a. Of those, which have SMILES that CANNOT be parsed as a SMILES or SMIRKS (mol object fails)
2. Which have ONLY a SMILES?
    a. Of those, which have SMILES that can be parsed as a SMILES (RDKit MolFromSmiles)?
    b. Of those, which have SMILES that can be parsed as a SMIRKS (RDKit MolFromSmirks)?
    c. Of those, which have SMILES that can be parsed BOTH as a SMILES (RDKit MolFromSmiles) or a SMIRKS (RDKit MolFromSmirks)?
    d. Of those, which have SMILES that CANNOT be parsed as a SMILES or SMIRKS (mol object fails)