HW1: Exploring Design Spaces in Knox

- **Submission date: 03/20/2026 (11:59 PM)**
- Complete in teams of 2
- Contact Professor Densmore or James for questions

Goals:

1. Learn how to download and run applications from GitHub.
2. Explore Knox ecosystem (User Interface + Neo4j Database).
3. Explore GOLDBAR, Operators, and Enumeration.
4. Rule Generation with GOLDBAR Generator.
5. Using GOLDBAR Rule Evaluation Algorithm for rule-based machine learning.

Deliverables:

1. Bug Report → Google Form
2. Tutorial Walk Through (PDF) → Blackboard
3. Rule-based Machine Learning (Jupyter Notebook + PDF version) → Blackboard

---

Extra Credit Options (Optional):

1. Rule Capturing Between Models
    a. Topic Area: Machine Learning
2. Human Language to GOLDBAR
    a. Topic Areas: Natural Language Processing, Large Language Models, AI agents

---

Outline:

1. Introduction & Reading
2. Knox and Dependencies Installation
3. Running and Stopping Knox & Neo4j
4. Tutorial Walk Through
5. Rule-based Machine Learning
6. Bug Report
7. Extra Credit Options
8. Appendix

Introduction:

Knox is an open-source computational framework designed to facilitate combinatorial biological design through the integration of the GOLDBAR formalism with graph database technology. Developed as a tool for synthetic biology and genetic engineering applications, Knox provides researchers with a powerful platform to specify, visualize, enumerate, and evaluate complex biological design spaces.

Core Components

1. **User Interface**: A web-based graphical interface that enables users to interact with design spaces, apply operators, visualize rules, and execute various design operations without requiring extensive programming knowledge.
2. **Neo4j Graph Database**: A backend database system that stores and manages design spaces as graph structures, allowing for efficient querying, manipulation, and relationship mapping between biological components.

Some Key Capabilities

1. **Define Design Constraints**: Create rule-based specifications using GOLDBAR syntax to constrain biological designs according to specific requirements (e.g., "part A must come before part B" or "part C cannot be repeated").
2. **Visualize Design Spaces**: Generate intuitive graph representations of design spaces that show all possible designs satisfying given constraints.
3. **Enumerate Designs**: Systematically generate all valid designs within a constrained space, enabling comprehensive exploration of design possibilities.
4. **Apply Operators**: Combine, modify, and refine design spaces using logical operators (AND, OR) and compositional operators (JOIN, MERGE, REPEAT, REVERSE).
5. **Generate Rules Systematically**: Use the GOLDBAR Generator to automatically create comprehensive rule sets for testing and machine learning applications.
6. **Evaluate Rule Performance**: Assess how well different rules predict design success through rule-based machine learning workflows.

The GOLDBAR Framework

At the heart of Knox lies GOLDBAR, a formal grammar for specifying combinatorial biological designs. GOLDBAR provides operators and syntax that allow designers to express complex compositional rules in a human-readable yet computationally tractable format. This framework bridges the gap between high-level design intent and low-level implementation details, making it easier to communicate, reproduce, and optimize biological designs.

Through its combination of formal specification, graph-based representation, and machine learning integration, Knox enables both systematic design exploration and data-driven design optimization.

Reading:

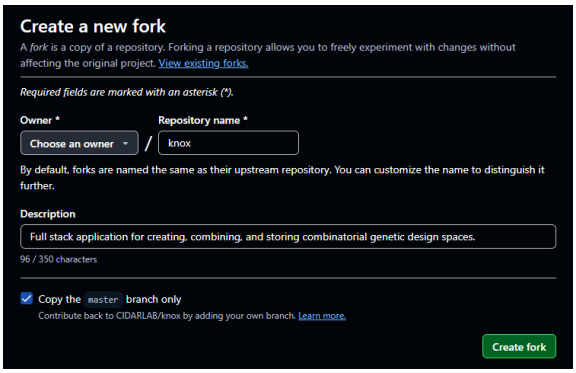
The first thing you should do is read “GOLDBAR: A Framework for Combinatorial Biological Design”. This paper will introduce the formalism of GOLDBAR and the rationale behind having the framework. Additionally, you should review the slides on Knox.

Link: https://pubs.acs.org/doi/10.1021/acssynbio.4c00296

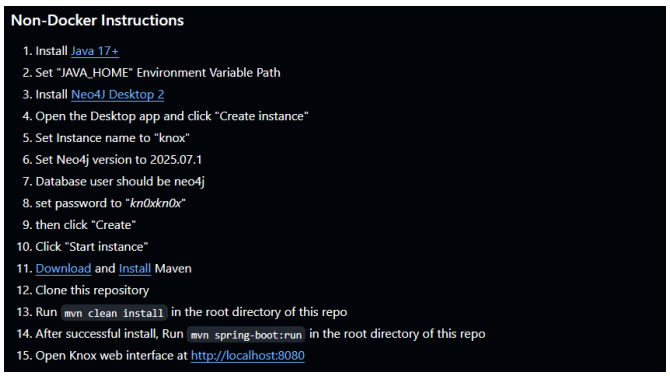
Knox and Dependencies Installation:

Here we will go over how to install Knox and Neo4j on your computer. I suggest all group members do this as it is good practice to learn how to install open-source software from GitHub, especially if you plan to do software development.

1. Go to the Knox webpage (link)
2. Create a Fork of Knox



3. Follow Non-Docker Instructions
    a. Clone your Fork!

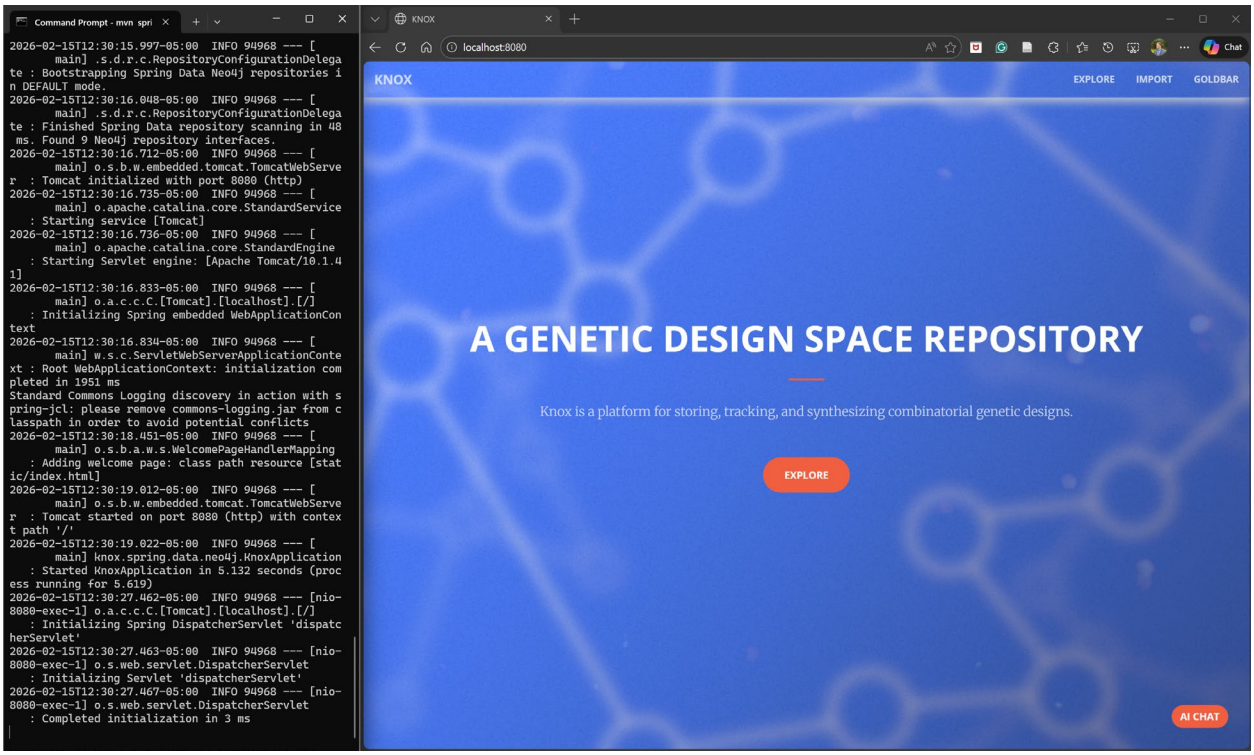


4. No need to Follow AI Chat / Agent Integration

Running Knox & Neo4j

1. Open a terminal in the root of Knox's Directory.
    a. Ex: your path\knox
2. To start Knox run: "mvn spring-boot:run'.
3. Open Neo4j desktop and press "start instance".
4. Open a browser and search http://localhost:8080

Note: Do not close the terminal while Knox is running.



This is how I suggest you have Knox open when running so you can see information via the terminal.

Stopping Knox & Neo4j

1. Click the terminal running Knox and press **CTRL+C**.
2. Then type "y" and press **ENTER.**
3. Open Neo4j Desktop Tab and press "stop instance".
4. Now you can close all windows.

Tutorial Walk Through

In this walkthrough we will explore some of Knox's capabilities. As you go through this tutorial, you will take screenshots, along with writing a figure caption per screenshot. The screenshots with captions along with answers to questions should be submitted as a PDF through blackboard.

Please Include all team members name in the PDF. Please have deliverables organized.

Example:

Submit a screenshot of the graph representation and describe what it specifies.

How would you specify this graph in GOLDBAR?

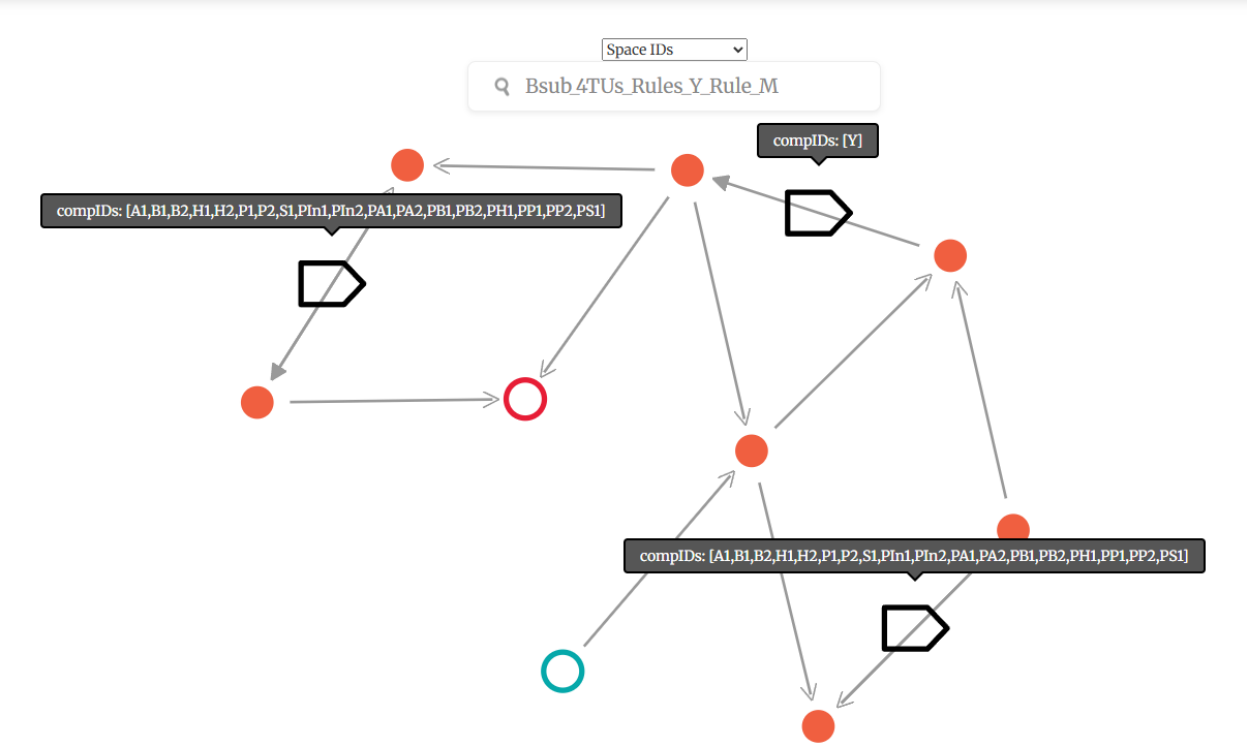


Figure 1. Bsub_4Tus_Rules_Y_Rule_M Design Space Graph Representation. This rule specifies that all designs must include part "Y".

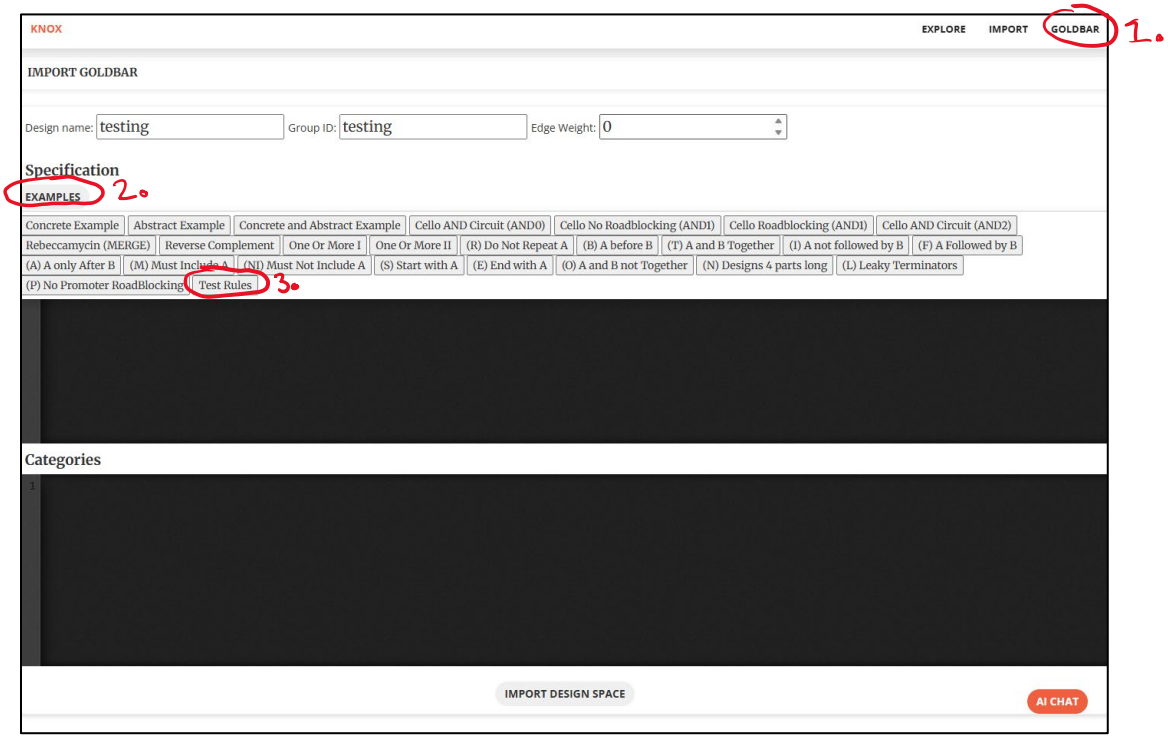
The GOLDBAR would be "one-or-more(zero-or-more(any_except_Y) then Y) then zero-or-more(any_except_Y)"

1) Rule Tests

Sanity check. Run a test on your Knox to make sure GOLDBAR rules are being synthesized properly.

Steps:

1. Navigate to GOLDBAR tab.
2. Click “EXAMPLES” button.
3. Click “Test Rules” button.
4. Test Outcomes will populate in the specification box.



Deliverables:

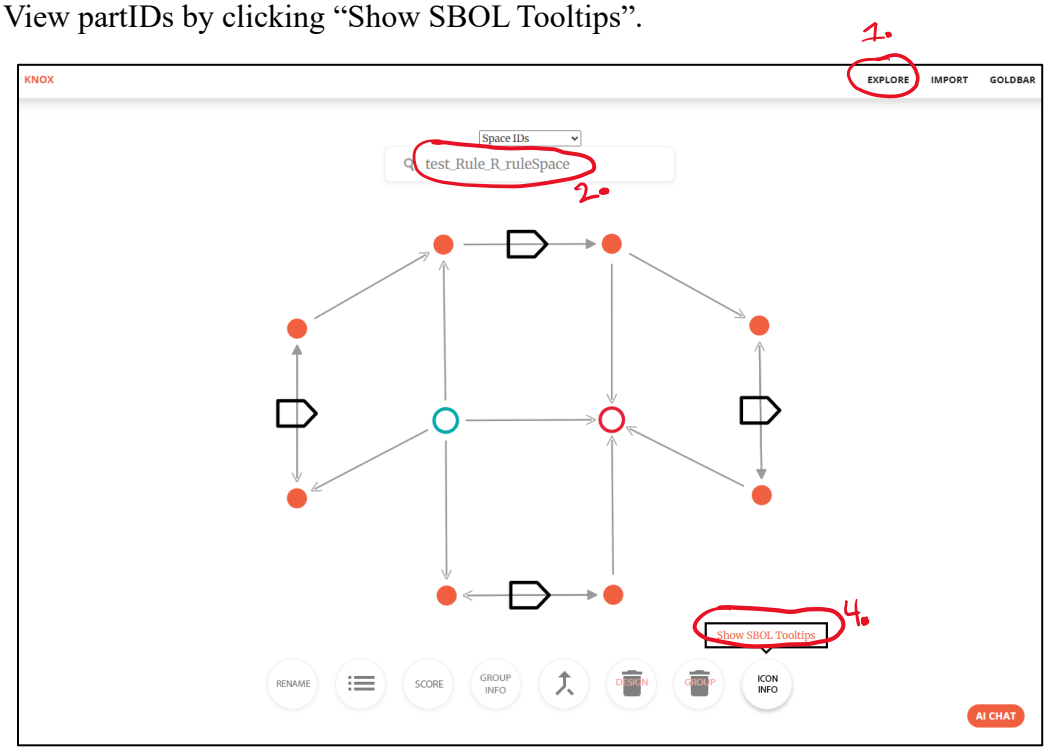
1. Was the “outcome” for each rule “pass”?
    a. If not, reach out to James immediately.
2. Take a screenshot of rule “R” outcome.

2) Visualizing a Design Space

Knox lets you visualize design spaces via the user interface.

Steps:

1. Navigate to Explore Page.
2. Search for “test_Rule_R_ruleSpace”
3. Zoom in on the design space graph.
4. View partIDs by clicking “Show SBOL Tooltips”.



Deliverables:

1. Take a screenshot of the design space.
2. What does this rule do?

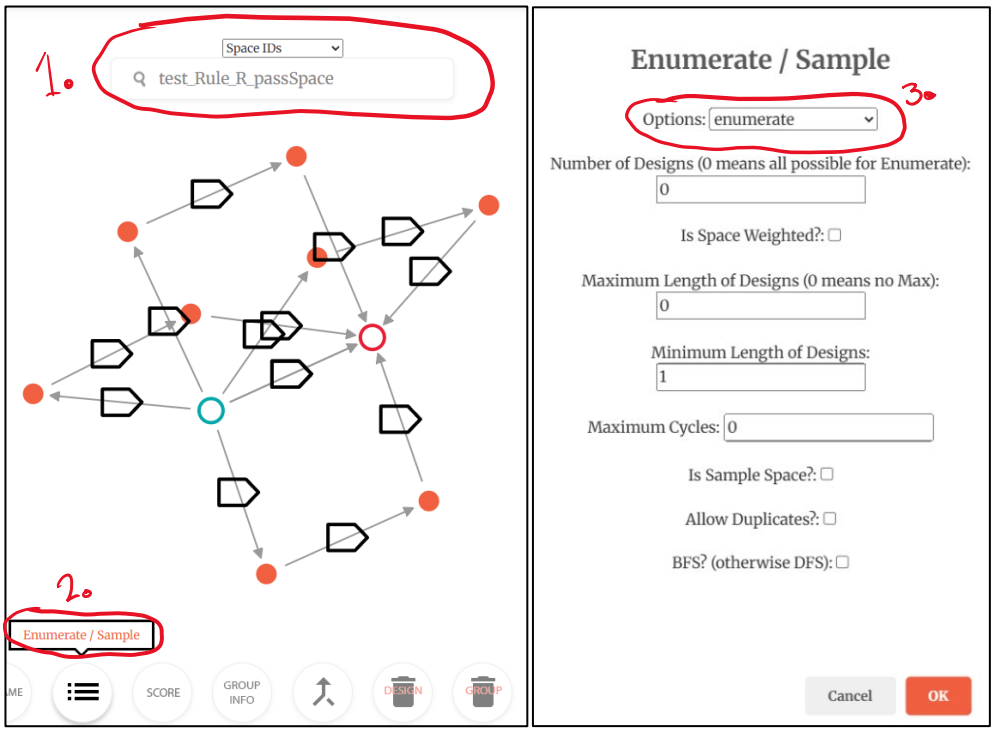
Hint: go to “Import” tab and select “GOLDBAR Generator” sub tab to see rule descriptions.

3) Enumerating a Design Space

Design Spaces can be enumerated and sampled from through the user interface.

Steps:

1. Visualize “test_Rule_R_passSpace” via search bar.
2. Click “Enumerate/Sample” button.
3. Select “enumerate” in Options.



Deliverables:

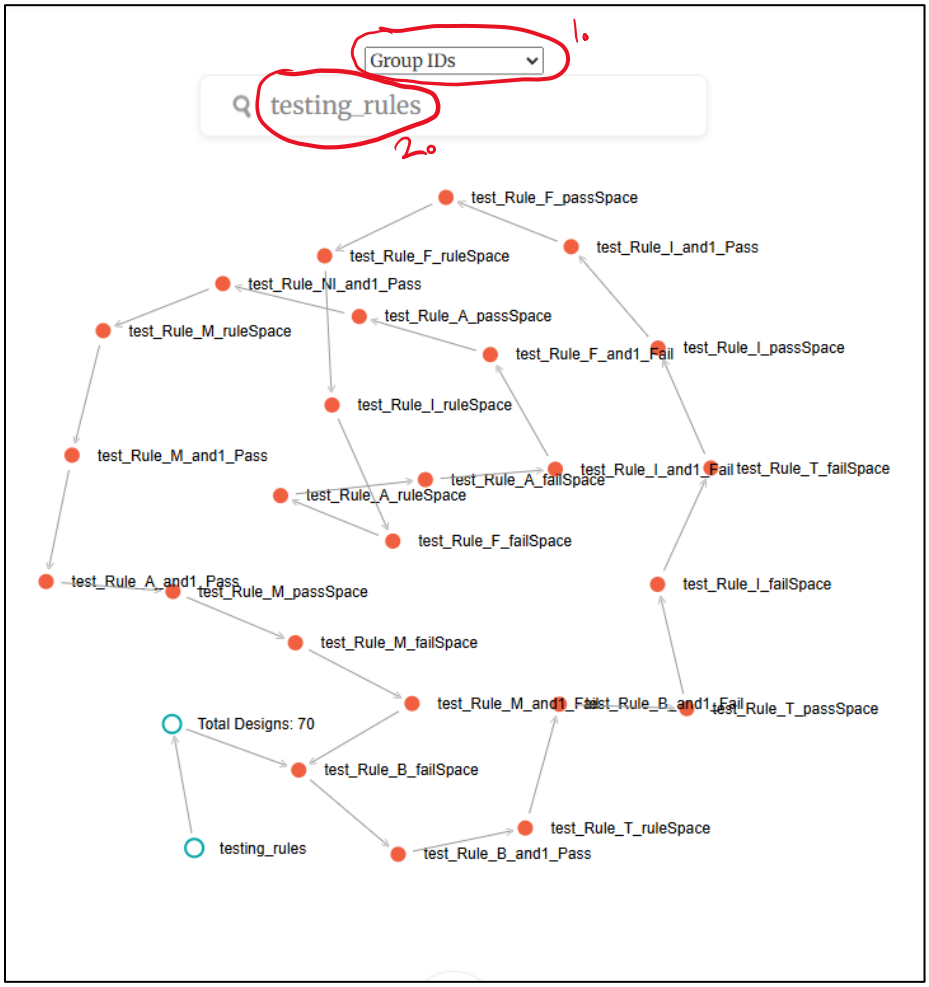
1. Take a screenshot of the output of the enumeration.
2. Why do these designs “pass” or follow the rule “test_Rule_R_ruleSpace”?
3. Repeat deliverable (1) for design space “test_Rule_R_failSpace”.
4. Why do these designs “fail” or not follow the rule “test_Rule_R_ruleSpace”?

4) Visualizing a Group of Design Spaces

Knox groups design spaces through their corresponding Group ID. Knox then lets you visualize the group, up to 25 design spaces will be shown through the user interface.

Steps:

1. Toggle search bar to “Group IDs”.
2. Select “testing_rules”.



Deliverables:

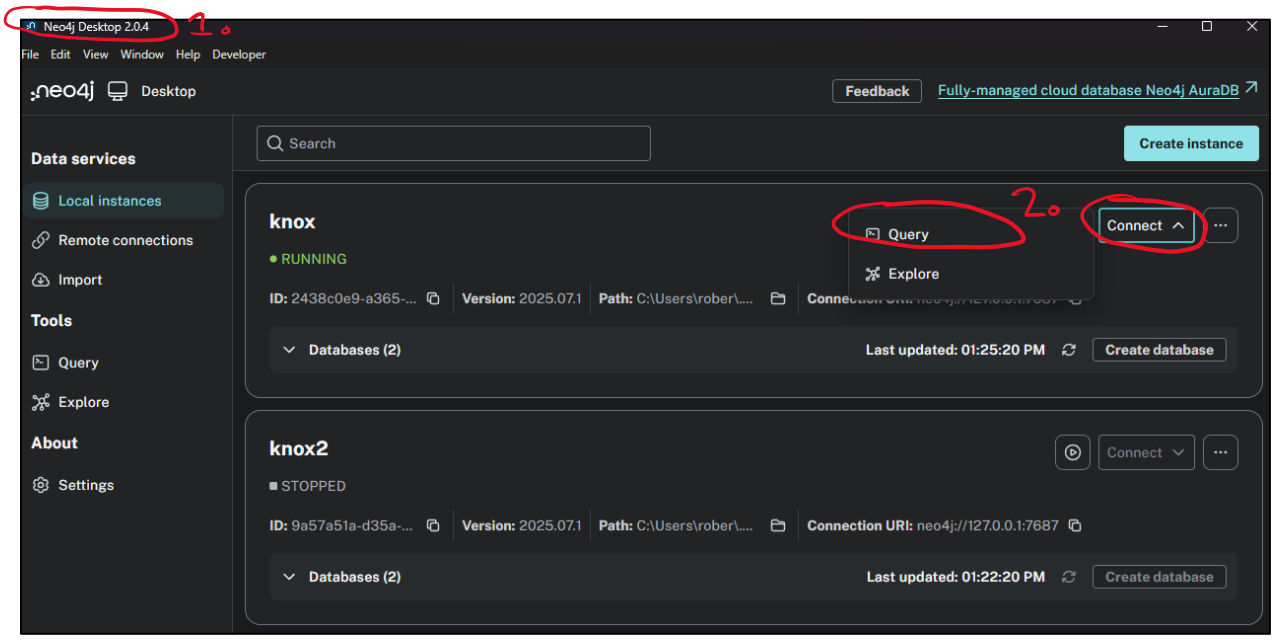
1. Take a screenshot of the group space.

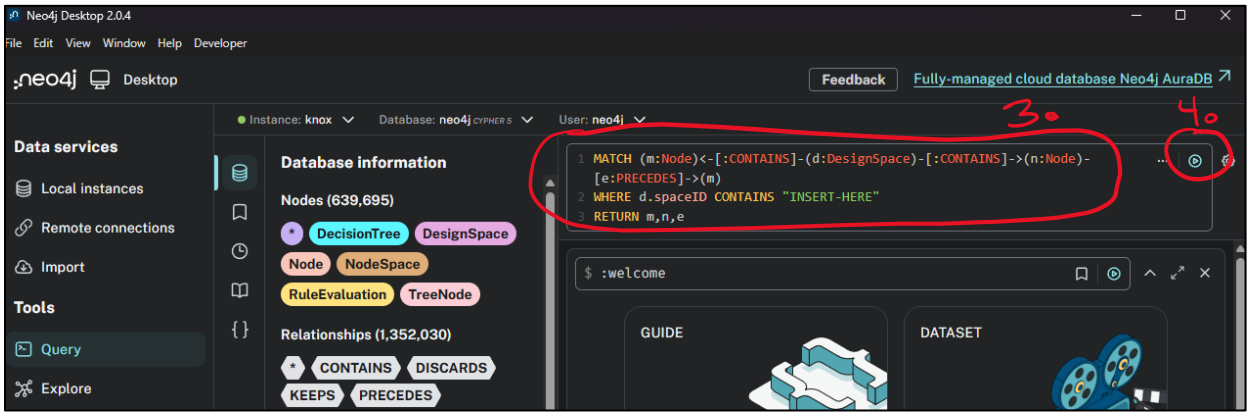
5) Visualizing a Design Space in Neo4j

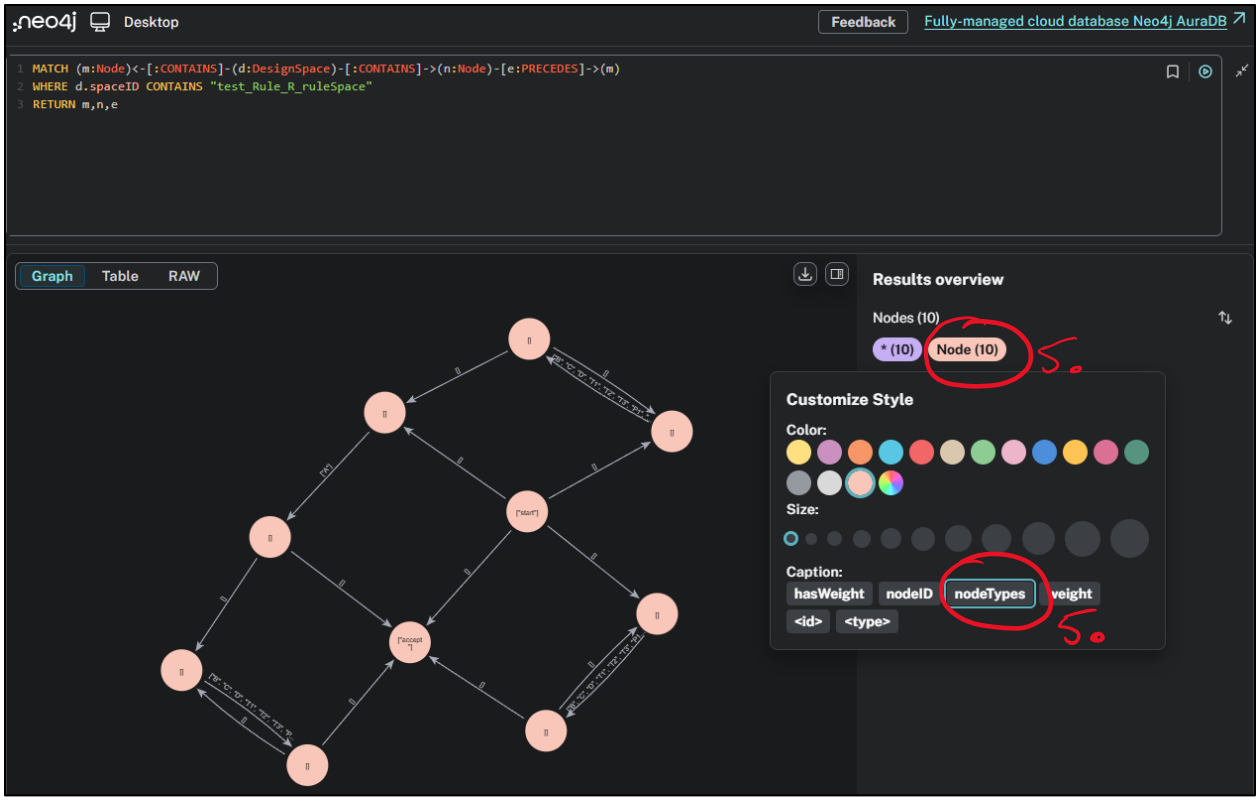
We can also perform visualizations through Neo4j, the database that is storing our design space graphs.

Steps:

1. Open Neo4j Window.
2. Click “Connect” and then “Query”.
3. Write the following Query. Replace “INSERT-HERE” with design space ID.
4. Run Query.
5. Click “Node” and select “nodeTypes”.
6. Click “PRECEDES” and select “componentIDs”







Deliverables:

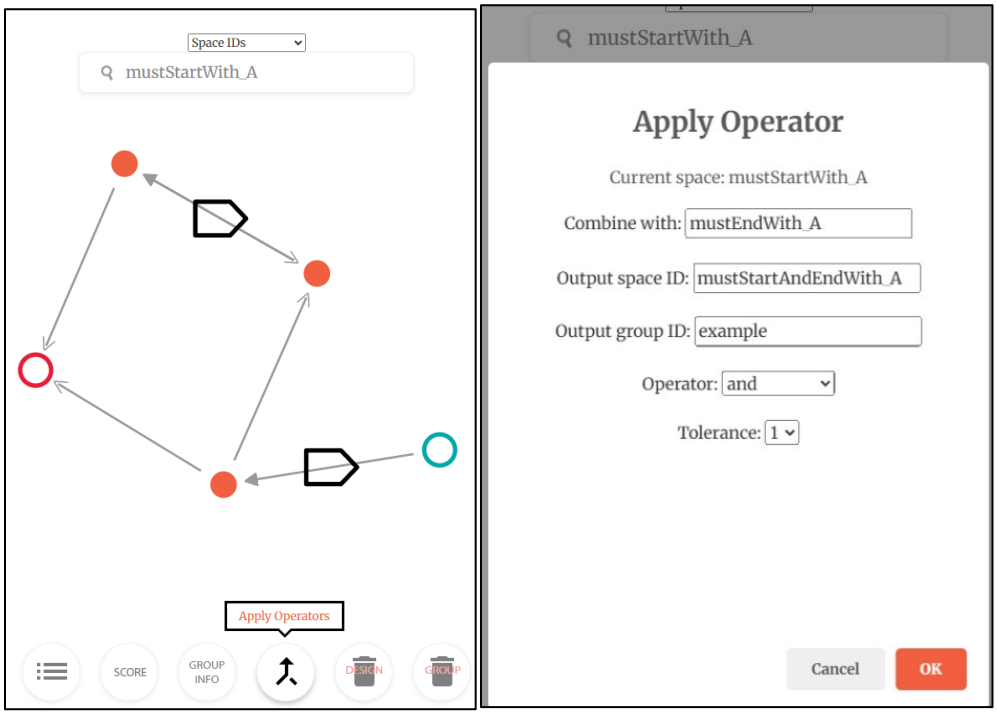
1. Take a screenshot of Neo4j design space graph for any rule other than “R”.
2. Describe what that rule enforces.

6) Apply GOLDBAR AND Operator on Rule Spaces

Knox enables users to apply additional operators to design spaces after they have been created.

Steps:

1. Go to “GOLDBAR” tab.
2. Click “Examples”.
3. Click “(S) Start with A”.
4. Click “IMPORT DESIGN SPACE”.
5. Repeat Steps 2-4 for “(E) End with A”.
6. Visualize “mustStartWith_A” on “Explore” Tab
7. Click “Apply Operators”
8. Set up Apply Operator as seen below and click “OK”.



Deliverables:

1. Take a screenshot of the visualization of “mustStartAndEndWith_A”.
2. What does this rule space describe?
3. Take a screenshot of the enumeration.
4. What does each design have in common?
5. How would you write “mustStartAndEndWith_A” in GOLDBAR?
6. If you repeated steps (6-7) and applied the “or” operator, what would that rule describe?

7) Combining Design Spaces

Let's explore the many different GOLDBAR operations that can be used.

Steps:

1. Go to "GOLDBAR" tab.
2. Select Example "(R) Do Not Repeat A".
3. Change "Group ID" to "testing".
4. Click "IMPORT DESIGN SPACE".
5. Now replace GOLDBAR specification box with design "A then B then C".
6. Change "Design name" to "ABC"
7. Click "IMPORT DESIGN SPACE".
8. Now replace GOLDBAR specification box with design "D then B then A".
9. Change "Design name" to "DBA"
10. Click "IMPORT DESIGN SPACE".
11. Apply the following operators between spaces "ABC" and "DBA"
    a. "join"
        i. Note: this operation is the same as "then" in GOLDBAR
    b. "or"
    c. "and" – tolerance set "1"
        i. Note: this operation is the same as "and1" in GOLDBAR
    d. "merge" – both tolerances set to "0"
    e. "repeat" – cardinality set to "one-or-more"
        i. Note: this operation is the same as "one-or-more()" in GOLDBAR
12. Apply the following operator on the output of the "join".
    a. "reverse" – Reverse Part Orientations set to "True" – Note: same as "reverse-comp"
13. Apply the "and" operator between the output of the "merge" and "do_not_repeat_A".

Deliverables:

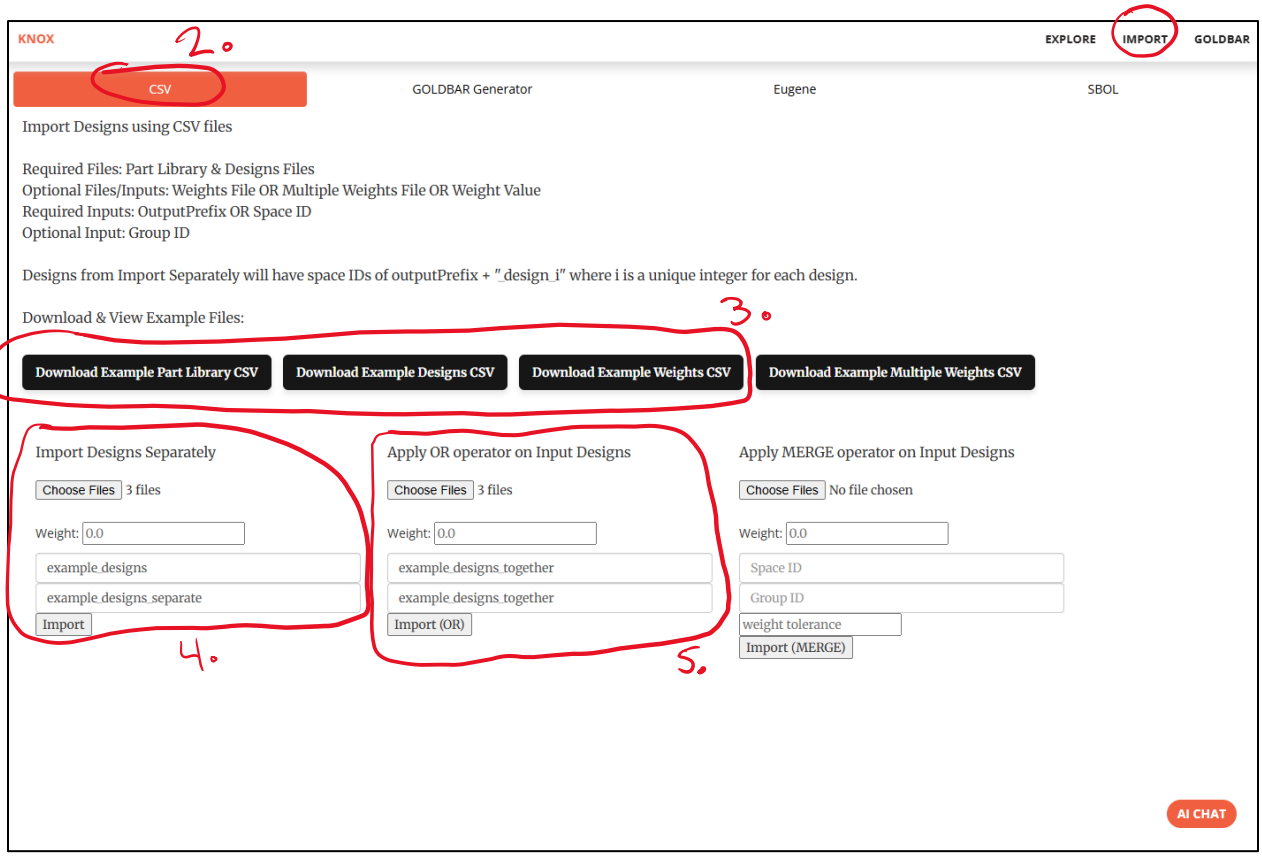
1. Take a screenshot of the Group Visualization (All spaces including output of operators should have the Group ID "testing"). There should be 10 design spaces in the group.
2. Describe briefly what each operation does.
3. For operators "join", "or", "and", "merge", and "repeat" does the order in which you apply the operation matter?
    a. Ex: is "ABC join DBA" the same as "DBA join ABC"?
4. How would you write the GOLDBAR for step 13?

8) Importing Designs

Knox enables users to import designs beyond GOLDBAR. Let's look at how Knox imports designs in the CSV file format.

Steps:

1. Go to "IMPORT" tab.
2. Go to "CSV" sub tab.
3. Download circled example files.
4. Import design separately.
5. Apply OR operator on Input Designs.



Deliverables:

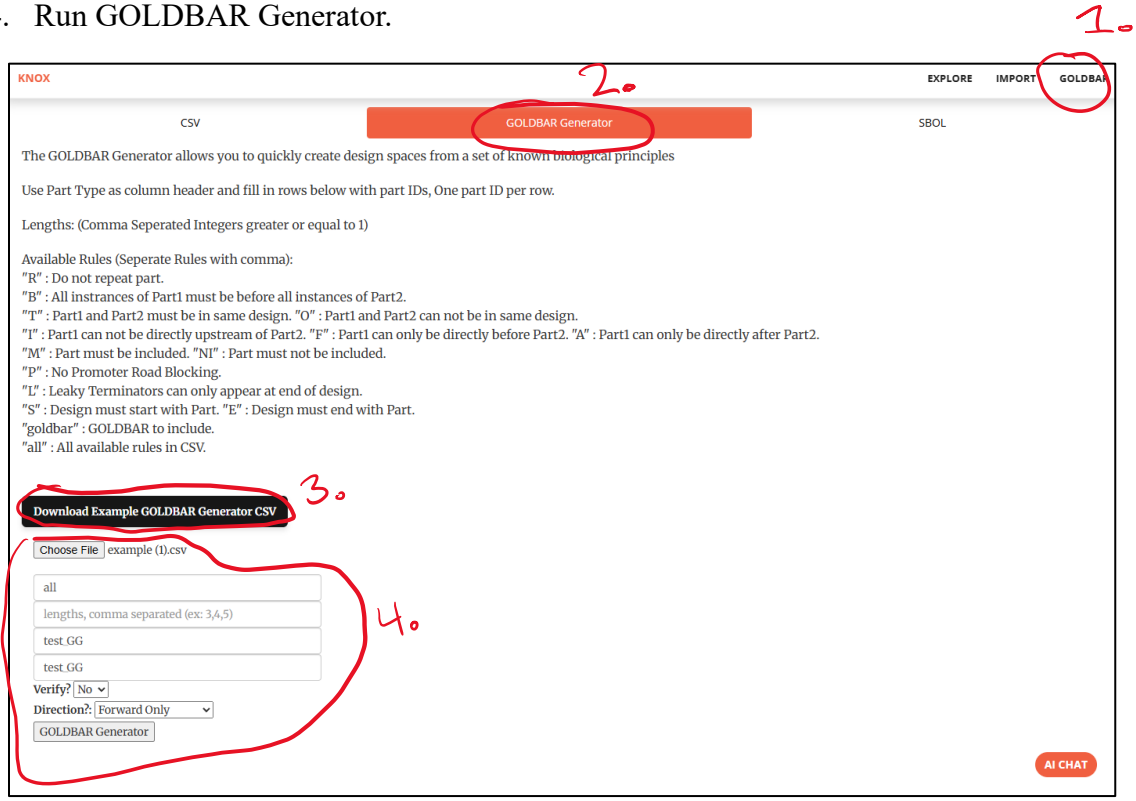
1. Take a screenshot of graph visualization of "example_designs_together".
2. Take a screenshot of the enumeration of "example_designs_together".
3. What is the difference between the two import options?

9) Rule Generation with GOLDBAR Generator

Often, we need a lot of rules to represent the design space. We can produce rules quickly with the GOLDBAR Generator. The GOLDBAR Generator also allows you to verify that your rules are not conflicting (ex: “do not include A” and “must include A”). This verification is done by applying the “and” operator between all the synthesized rules.

Steps:

1. Go to “IMPORT” tab.
2. Go to “GOLDBAR Generator” sub tab.
3. Download Example File.
4. Run GOLDBAR Generator.



Deliverables:

1. Take a screenshot of the group visualization.
2. Rerun GOLDBAR Generator but this time:
    a. Set “rules” to “M,N”
    b. Set “lengths” to “2”
    c. Set “outputPrefix” to “test_GG_verify”
    d. Set “Group ID” to “test_GG_verify”
    e. Set “Verify?” to yes.
3. Take a screenshot of the graph visualization of “test_GG_verify_Verification”.
4. Why is the graph like this? What does this tell us about this set of rules?

10) Writing GOLDBAR

Now that you have spent some time going through GOLDBAR examples and using GOLDBAR operators now it is time to write some original GOLDBAR. Specifically, we are going to be writing complement GOLDBAR. A complement rule is a rule that encodes the opposite of its complement. For example, “must include” (M) and “not include” (NI) are complements. Counter example, “starts with” (S) and “ends with” (E) are not complements. The complement of (S) would be “does not start with”. The complement of (E) would be “does not end with”.

Steps:

1. Use the categories that are used for the rule examples.
2. Write the complement for rule S (starts with A).
3. You can test your rule by applying the AND operator between your GOLDBAR with “test_Rule_S_failSpace” and “test_Rule_S_passSpace”.
    a. Only the designs in the failSpace should be kept as it represents possible complements.
4. Repeat steps (1-2) for the complement for rule E (ends with A).
5. Repeat steps (1-2) for the complement for any rule (other than S, E, M, NI).

Deliverables:

1. Take a screenshot of the graph representation of the complement S rule.
2. What is the GOLDBAR for the complement S rule?
3. Did it correctly keep designs in the failSpace and eliminate designs in the passSpace?

4. Take a screenshot of the graph representation of the complement E rule.
5. What is the GOLDBAR for the complement E rule?
6. Did it correctly keep designs in the failSpace and eliminate designs in the passSpace?

7. What rule did you choose to find the complement for?
8. Take a screenshot of the graph representation of the complement rule you chose.
9. What is the GOLDBAR for the complement rule you chose?
10. Did it correctly keep designs in the failSpace and eliminate designs in the passSpace?

Rule-Based Machine Learning

At this point, we have a fast way to generate rules using the GOLDBAR generator and a way to verify if a design follows a rule using the GOLDBAR AND operator. This section will explore how we could utilize these two features to explore ways to separate designs based on whether a design follows a rule (features) and the design's score (label).

Basically, we want to see what set of rules cut's the entire design space such that we have an interpretable way to distinguish "good" and "poor" designs.

To do this, we will utilize two algorithms:

1. Knox's Rule Evaluation Algorithm
    a. knox/src/main/java/knox/spring/data/neo4j/domain/RuleEvaluation.java at master · CIDARLAB/knox
2. Scikit-learn's Decision Trees
    a. https://scikit-learn.org/stable/api/sklearn.tree.html

Knox's Rule Evaluation Algorithm

This algorithm works by applying the GOLDBAR AND operator between each rule and design. The algorithm generates two tables:

1. Purity Metrics Table
    a. Consists of metrics for how well each rule cut the space.
2. Design Kept/Eliminated Table
    a. Consists of results for if a design follows a rule
        i. where (0 = follows rule / kept, 1 = does not / eliminated)
    b. Scores column
        i. Which are the scores that were inputted into the algorithm.
    c. Labels column
        i. Designs are labeled inside the algorithm based on their score.

Please see examples for both tables in the appendix. You can also see examples for the data we are working with in the appendix.

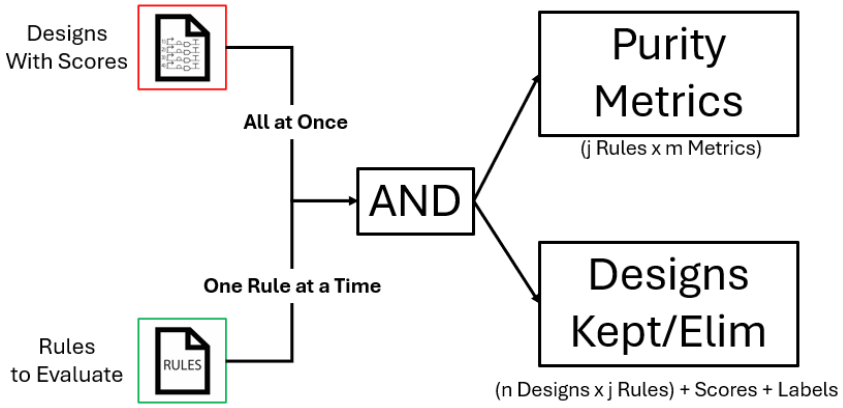


Figure 1. Rule Evaluation Algorithm. Key inputs are the designs with their scores, the rules to evaluate, and the labeling method (we will use the median). Each rule is ANDed with all the input designs.

Scikit-Learn Decision Trees

Decision Trees are a graph-based object with nodes and edges. The root node and intermediate nodes ask questions ("Do you follow Rule X"). The terminal or leaf nodes assign labels to the group of designs that make it to that leaf. For classification a label could be "good/poor" or "awful, poor, good, great". The label for a classification node is assigned by the majority class present in the node. For regression the label is a numeric value. The label for a regression node is the average of the data points in that node.

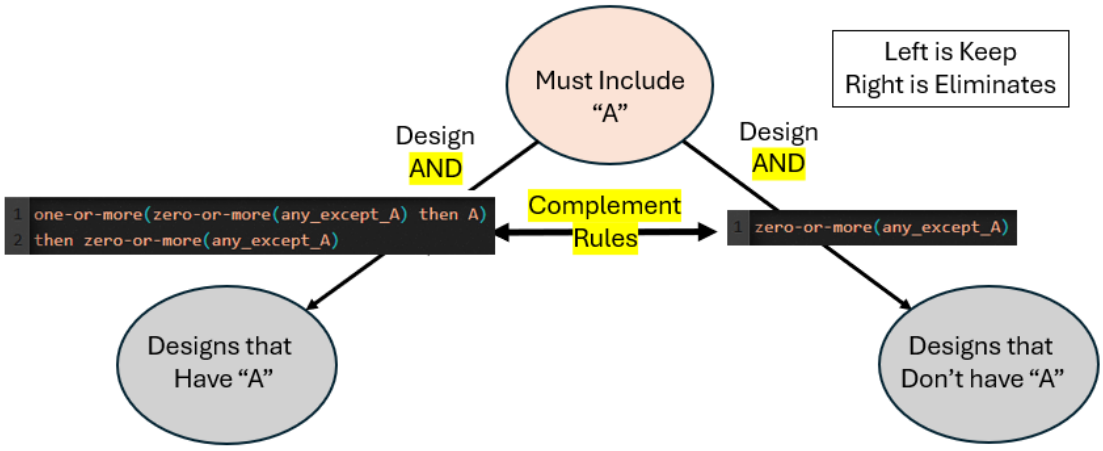


Figure 2. Example sub tree where the node's question is "Do you follow the Must Include A Rule". The designs that do follow this rule would go to the left child node. The designs that do not follow this rule would go to the right child node. We can think of this as the left child node follows the rule and the right child node follows the complement of the rule. In this case, the left child node follows "Must Include A", and the right child node follows "Not Include A".

Steps:

1. Import Designs. (Import Designs Separately)
    a. Using files
        i. “designs.csv”
        ii. “part_library.csv”
        iii. “gnn_predicted_scores.csv”
    b. Make sure to set the “Group ID” and “OutputPrefix” to something we haven’t used.
2. Generate Rules. (GOLDBAR Generator)
    a. Create a GOLDBAR Generator CSV file.
        i. Make sure there are at least 200 rules. (Use a variety)
    b. Generate Rules with GOLDBAR Generator
        i. Make sure to set “Group ID” to something we haven’t used (ex: “rules”)
3. Run Rule Evaluation.
    a. Using the Jupyter Notebook “Knox_RBML.ipynb”
    b. Run “ruleEvaluateByGroup” function.
        i. Set the evaluation name (evalName)
        ii. Set the designs group ID (groupID)
        iii. Set the rules group ID (ruleGroupID)
        iv. Set “labelingMethod” to “sign” or “median”
            1. “sign” will give each design a 0 (poor) if it is ≤ 0, otherwise 1 (good).
            2. “median” will give each design a 0 if it is $\leq$ median, otherwise 1.
4. View the “purity_metrics_df” and “designToRule_df”.
5. Save each DataFrame as a CSV file.
6. Explore Sklearn.tree to see how to train Trees.
7. Build at least three Trees
    a. Binary Classification Tree
    b. Regression Tree
    c. Multi-Classification (Binned) Tree
        i. Turn design scores into quantiles
        ii. Ex: $(awful < Q1 \leq poor < Q2 \leq good < Q3 \leq great)$
8. For Each Tree
    a. Train on 70% of the data.
    b. Test on 30% of the data.
    c. Plot Final Tree using “tree.plot_tree”
        i. Make sure to set “feature_names”
        ii. Also set “class_names” for Classification Trees
        iii. Try to make plotted trees easily legible.
    d. Save each tree plot as a “.png”

Deliverables:

Note: 1-4 can be written in markdown in Jupyter Notebook.

1. How many rules did you use? Why did you choose those rules?
2. What are we learning from each type of tree?
3. What are some interesting insights you noticed about some of the rules?
4. If you were to experimentally validate five of these rules, which would you choose? Why?
5. Submit your final Jupyter Notebook and PDF of the Jupyter Notebook on Blackboard.
6. Submit each plotted tree ".png" on Blackboard.
7. Submit GOLDBAR Generator CSV file on BlackBoard.
8. Submit DataFrame CSV files on BlackBoard.

Bug Report

Please complete this section once you have completed the walkthrough and Rule-Based Machine Learning sections. Any bugs that need immediate attention, please contact James. Fill out the following google form (only 1 per team).

https://forms.gle/J3pn1ycqZDNWQU7MA

Extra Credit Options

1) Rule Capturing Between Models

Complete this section if you are interested in diving deeper into exploring rule-based machine learning.

Steps:

1. Import true scores into python and turn into a list.
    a. Use “true_scores.csv”
2. Run the rule evaluation by designs function on the designs with a new evaluation name.
3. Import transformer scores into python and turn into a list
    a. Use “transformer_predicted_scores.csv”
4. Run the rule evaluation by designs function on the designs with a new evaluation name.
5. NOTE: you should now have three different rule evaluations
    a. One for GNN predicted scores
    b. One for Transformer predicted scores
    c. One for True scores
6. Use these rule evaluations to compare their purity metrics and build decision trees.

Deliverables:

Note: You can include the following in the same Jupyter Notebook as rule-based machine learning.

1. Which model more closely emulates the true grammar? How did you come to this conclusion?
2. Include any plots or figures that helped you come to this conclusion in the Jupyter Notebook.

2) Natural Language Processing

If you are interested in diving deeper into rule generation using large language models (LLMs) please contact us.

Appendix

Table 1. Example Designs Kept/Eliminated Table. For the "Label" column, 0 is a poor design and 1 is a good design. For the "Rule" columns, 0 means the design is kept / follows rule and 1 means the design is eliminated / does not follow the rule.

Design ID	Score	Label	Rule_1	Rule_2
ID_1	-0.5	0	0	1
ID_2	-0.7	0	0	1
ID_3	1.0	1	1	1
ID_4	3.0	1	1	0

Table 2. Example Purity Metrics Table part 1. We will use table 1 to fill in this part of the table (4 total designs).

Rule ID	Good Designs Kept	Good Designs Eliminated	Poor Designs Kept	Poor Designs Eliminated
Rule_1	0	2	2	0
Rule_2	1	1	0	2

Table 3. Example Purity Metrics Table part 2. Continuing the table. For this section, you can think of a 0 as "False" and a 1 as "True".

Rule ID	PoorPerfection	GoodPerfection	TotalPerfection
Rule_1	1	0	0
Rule_2	0	0	0

Table 4. Example Purity Metrics Table part 3. Continuing the table.

Rule ID	NumIncorrect	NumCorrect
Rule_1	4	0
Rule_2	1	3

Table 5. Example Purity Metrics Table part 4.

Rule ID	Impact	PoorEliminationPercent	GoodnessPercent	PoornessPercent
Rule_1	-100	0%	0%	100%
Rule_2	50	67%	50%	0%

Formulas for Purity Metrics

$T_d = Total\ Designs$

$P_d = Total\ Poor\ Designs$

$P_k = Total\ Poor\ Designs\ Kept$

$P_e = Total\ Poor\ Designs\ Eliminate$

$G_d = Total\ Good\ Designs$

$G_k = Total\ Good\ Designs\ Kept$

$G_e = Total\ Good\ Designs\ Eliminated$

Poor Perfection is True when a rule only eliminates good designs.

Good Perfection is True when a rule only eliminates poor designs.

Total Perfection is True when a rule does not eliminate any designs.

NumIncorrect – Number of Incorrect Eliminations / Keeps.

$NumIncorrect = P_k + G_e$

NumCorrect – Number of Correct Eliminations / Keeps.

$NumCorrect = P_e + G_k$

Impact – This metric measures the ratio of number of correct to total possible correct while penalizing number of incorrect.

$$Impact = 100 * \frac{NumCorrect - NumIncorrect}{T_d} \quad \in \quad [-100, 100]$$

PoorEliminationPercent – This metric measures the ratio of number of designs eliminated that are poor to the total number of designs eliminated. Note, this metric can be not a number (nan) if no designs are eliminated (Total Perfection).

$$PoorEliminationPercent = \ 100 * \frac{P_e}{P_e + G_e} \quad \in \quad [0, 100\%]$$

GoodnessPercent – This metric measures the ratio of good designs remaining to total good designs.

$$GoodnessPercent = 100 * \frac{G_k}{G_d} \quad \in \quad [0, 100\%]$$

PoornessPercent – This metric measures the ratio of poor designs remaining to total poor designs.

$$PoornessPercent = 100 * \frac{P_k}{P_d} \quad \in \quad [0, 100\%]$$

Dataset – Genetic Circuits

We will be working with genetic circuits that are 8 parts in length consisting of 4 transcription units.

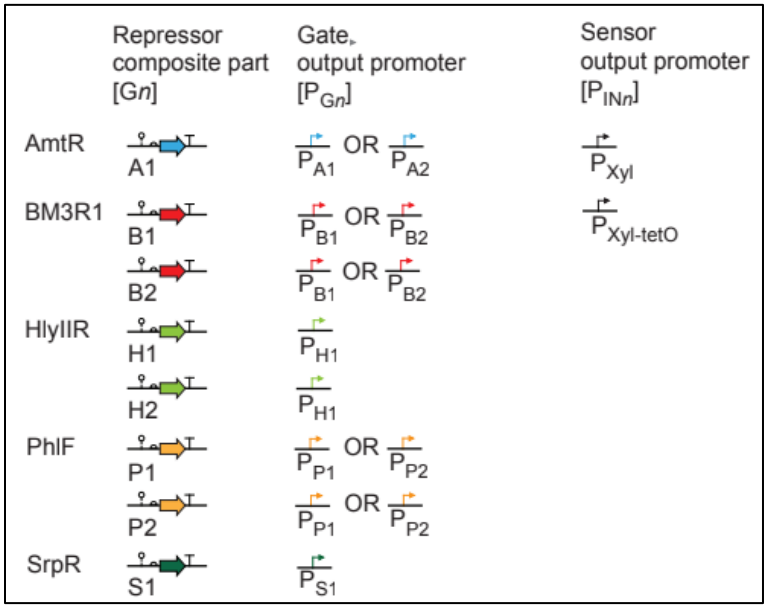


Figure 3. Part Library for our dataset.

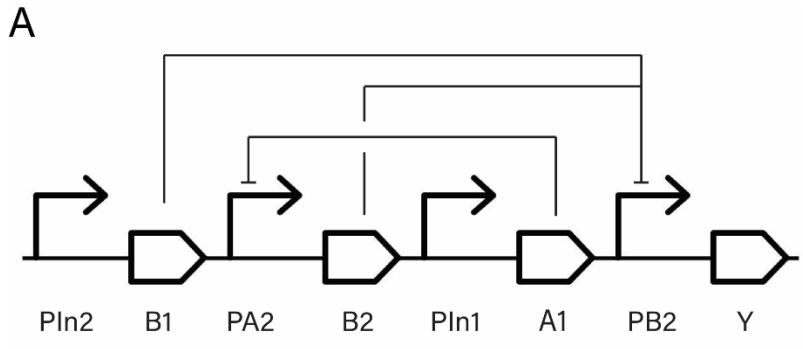


Figure 4. Example Design in our dataset.

The score for each design was generated using Cello. Here is the formula:

$$score = \ln\left(0.5 * \frac{Lowest\ On\ Expression\ Level}{Highest\ Off\ Expression\ Level}\right)$$

Where a higher score means there is a greater expression level between an "on" state than a "off" state.