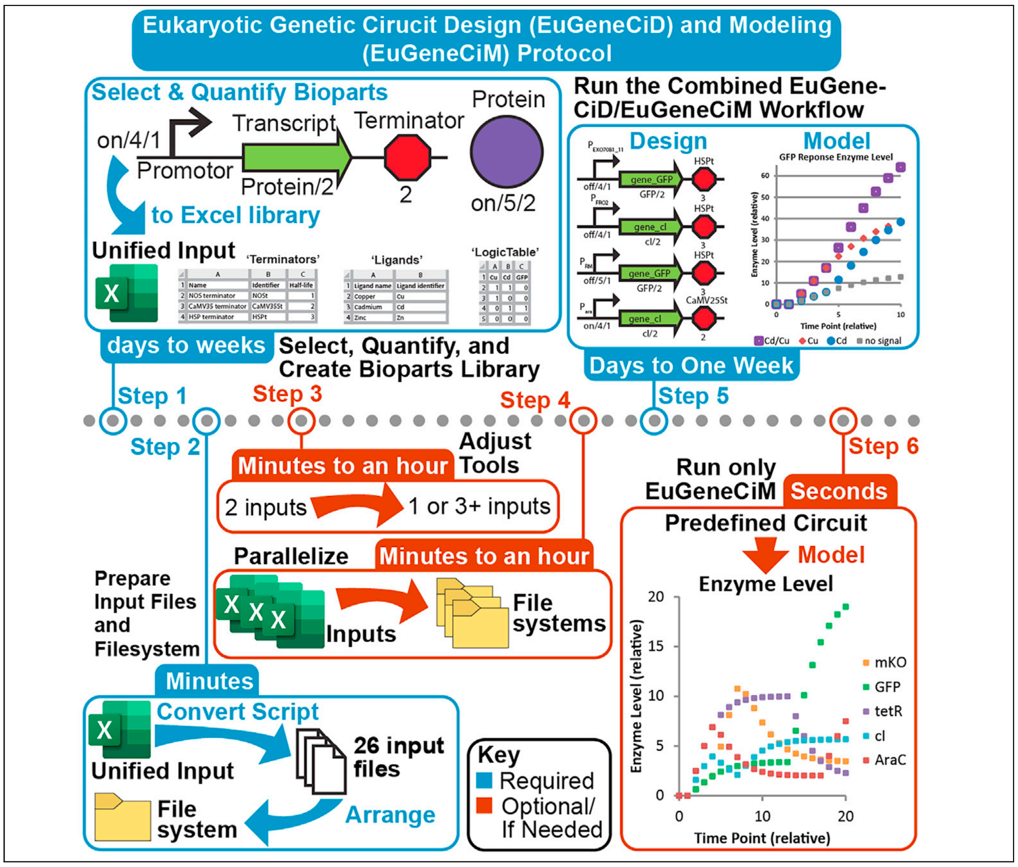
CellPress OPEN ACCESS

Protocol

Using EuGeneCiD and EuGeneCiM computational tools for synthetic biology



Synthetic biology often relies on the design of genetic circuits, utilizing "bioparts" (modular DNA pieces) to accomplish desired responses to external stimuli. While such designs are usually intuited, detailed here is a computational approach to synthetic biology design and modeling using optimization-based tools named Eukaryotic Genetic Circuit Design and Modeling. These allow for designing and subsequent screening of genetic circuits to increase the chances of *in vivo* success and contribute to the development of an application development pipeline.

Wheaton L. Schroeder, Anna S. Baber, Rajib Saha

wheaton@huskers.unl.edu (W.L.S.)
rsaha2@unl.edu (R.S.)

Highlights

Protocol for Eukaryotic Circuit Design (EuGeneCiD) and Modeling (EuGeneCiM) tools

Details modification of these tools for new applications and parallelization

Details reading, formatting, and curation of results for easier understanding

Instructions for combined and separate workflows for EuGeneCiD and EuGeneCiM

Schroeder et al., STAR Protocols 2, 100820
December 17, 2021 © 2021 The Author(s).
https://doi.org/10.1016/j.xpro.2021.100820

Protocol

Using EuGeneCiD and EuGeneCiM computational tools for synthetic biology

Wheaton L. Schroeder,[1,2,4,*] Anna S. Baber,[2,3] and Rajib Saha[1,2,5,*]

[1]Department of Chemical and Biomolecular Engineering, University of Nebraska – Lincoln, Lincoln, NE 68588, USA

[2]Center for Root and Rhizobiome Innovation, University of Nebraska – Lincoln, Lincoln, NE 68588, USA

[3]Department of Biomedical Engineering, University of Rochester, Rochester, NY 14627, USA

[4]Technical contact

[5]Lead contact

*Correspondence: wheaton@huskers.unl.edu (W.L.S.), rsaha2@unl.edu (R.S.)

https://doi.org/10.1016/j.xpro.2021.100820

SUMMARY

Synthetic biology often relies on the design of genetic circuits, utilizing "bio-parts" (modular DNA pieces) to accomplish desired responses to external stimuli. While such designs are usually intuited, detailed here is a computational approach to synthetic biology design and modeling using optimization-based tools named Eukaryotic Genetic Circuit Design and Modeling. These allow for designing and subsequent screening of genetic circuits to increase the chances of *in vivo* success and contribute to the development of an application development pipeline.

For complete details on the use and execution of this protocol, please refer to Schroeder, Baber, and Saha (2021).

BEFORE YOU BEGIN

Timing: Minutes to Hours

This section includes several steps which are necessary prerequisites to those included in the step-by-step method details as contained in this protocol. This initial section does not focus on sequential steps, rather on system requirements necessary to perform subsequent steps and is therefore numbered separately from the other sections. This includes needed programming languages, packages which expand upon the capabilities of said languages, suggested text editors, and FTPs (File Transfer Protocols, necessary if utilizing a computing cluster). Also detailed here is the location of a repository which contains code and files used throughout this work, some of which will need to be downloaded for successful completion of this protocol. Note that parts of this first "before you begin" section parallel that of Schroeder and Saha (2020a, 2020b) (also published in STAR Protocols) because the setup is somewhat similar.

1. If needed, install the latest version of programming languages used throughout this protocol, namely Perl and the General Algebraic Modeling System (GAMS).
   a. The Perl programming language can be downloaded from www.perl.org. Perl is generally pre-installed on the Mac Operating System (OS) and Linux/UNIX OSs; however, it is encouraged that users to ensure that the latest version of Perl is installed. This will ensure compatibility with modules downloaded from the Comprehensive Perl Archive Network (CPAN) which are used throughout this protocol. For Windows operating system users, the Strawberry Perl download, which can be found at http://strawberryperl.com/, is recommended as this is what was and is

used by the authors. Note that Windows OS users may need to uninstall Strawberry Perl and reinstall the updated version in order to update it. All Perl language downloads are free of charge.

   i. Necessary modules from CPAN can be easily downloaded using command line or terminal arguments once the Perl programming language is installed. To make installations of CPAN modules easier, first download the CPAN minus app using the command "`cpan App::cpanminus`". This results in easier module installation compared to the "cpan" command, as well as simpler and more easily understood command-line output during installation.
   ii. Once Perl and the CPAN minus app are installed, modules can then be installed using the command "`cpanm Model::Name`". For this protocol, three modules from CPAN need be downloaded: Spreadsheet::Read, Algorithm::Combinatorics, and Excel::Writer::XLSX. When each is installed, a few lines of output should be written to the command line. Successful installation will see the second to last line of output read "Successfully installed Name-Module-X.XX", where "Name-Module" indicates the module name and the X's denote the version of the installed module. If the module was not successfully installed, check and resolve any error messages.

b. The General Algebraic Modeling System (GAMS) programming language and platform used by the authors for optimization applications, namely for the EuGeneCiD and EuGeneCiM tools. GAMS is not free of charge, as the base module and CPLEX solver (used here) must both be purchased from www.gams.com.

Note: Other programming languages, including free languages like the Constraint-Based Reconstruction and Analysis (COBRA) package for python (COBRApy), are available for Mixed-Integer Linear Programming (MILP) applications. The authors have chosen to use the GAMS programming for MILP applications compared to these other possible languages for two overarching reason: solution speed and tractability. GAMS is a language more optimized for Mixed Integer Linear Programs (MILPs), and solves MILP problems faster than COBRApy. As discussed in Schroeder and Saha (2020a, 2020b), COBRApy lacks the computational speed or power to perform one of our previously developed tools, the OptFill tool (Schroeder and Saha, 2020a, 2020b), in a reasonable period of time for anything other than a small test problem. As the EuGeneCiD and EuGeneCiM algorithms take considerably more time to solve than OptFill when both are implemented in GAMS, it was decided that implementing these tools in COBRApy or MatLab was not an economic use of time. Should COBRApy become significantly quicker in solving MILP problems, it may be worth re-investigation the implementation of EuGeneCiD and EuGeneCiM in COBRApy in the future.

2. An advanced text editor such as Notepad++, which has capabilities beyond those of standard text editing software (e.g., notepad for Windows or TextEdit for Mac OS) is highly recommended. The authors, who use Windows as their Operating System (OS), use Notepad++ which can be downloaded from notepad-plus-plus.org. Those who use Mac OS in the authors' research group use BBEdit, available from https://www.barebones.com/products/bbedit/ or Sublime Text, available from https://www.sublimetext.com/, for their programming and text editing.
3. In this protocol, the authors make use of a High Performance Computing (HPC) resource (the Holland Computing Center, HCC) to perform parts of this protocol. The HPC resources are used only for running all code in the GAMS programming language (steps 17 and 23) to greatly reduce run time.
   a. Users of this protocol are encouraged to have HPC resources or gain access to such resources as this will increase speed for the slowest steps of this protocol (steps 17 and 23). Such resources may include a supercomputing cluster at their own institutions or supercomputing clusters available for research use. Follow that institution's or supercomputing cluster's recommendations on creating an account, accessing the cluster, cluster file systems, and any other pertinent recommendations.

   b. For managing files on the supercomputing cluster, the authors have used two approaches (using Windows OS). The first is the notepad plus plus File Transfer Protocol (nppFTP) plugin as the authors use the Windows OS. This is most useful for the transfer of one file at a time. For larger FTP needs, the authors have made use of the Globus research data management system (found at https://www.globus.org/), which can be used by anyone with a web browser and is compatible with all OSs. In addition, Mac OS users in the authors' laboratory use Cyberduck (available at https://cyberduck.io/) for their FTP needs.
   c. For connecting to the supercomputing cluster and command line protocols, the authors use PuTTY (compatible with Windows and Unix) which can be found at https://www.chiark.greenend.org.uk/~sgtatham/putty/.
4. Codes related to and needed for this protocol, as well as to Schroeder, Baber, and Saha (2021) (Schroeder et al., 2021), can be found in the GitHub repository associated with these works (found at https://github.com/ssbio/EuGeneCiDM or https://doi.org/10.5281/zenodo.4762590). Individual steps that require users to download a file will specify the file name and location.
5. For some of these protocol steps, an internet connection is required (generally this allows for communication with the supercomputing clusters or other HPC resources as well as for downloading resources from the associated GitHub).

Conceptualizing synthetic biology applications and formatting concepts as workflow inputs

Timing: minutes to days

Prior to using the Eukaryotic Genetic Circuit Design (EuGeneCiD) and Modeling (EuGeneCiM) tools, users must first conceptualize the desired synthetic biology application. There are a few restrictions as to what types of conceptualizations may be designed and modeled by EuGeneCiD and EuGeneCiM which are detailed in this section. Examples of applicable conceptualizations, shown in Schroeder, Baber, and Saha (2021), include building logic gate-based applications. After conceptualization, this section also details how to format these conceptualizations as inputs for EuGeneCiD and EuGeneCiM. An overview of all steps, major sections, necessity, numbering, and timing is shown in Figures 1 and 2, including the "before you begin" steps.

6. **Conceptualize a synthetic biology application.** Before using the integrated EuGeneCiD/EuGeneCiM workflow, the user must conceptualize a synthetic biology application. Specifically, this application should be for a system to express a particular phenotype (characterized by the expression or non-expression of one or more enzymes) in response to presence or absence of two signals. For cases with other than two signals, please see the section "Adjusting EuGeneCiD and EuGeneCiM for other than two signals". Below are some considerations for conceptualization.
   a. EuGeneCiD is not a dynamic tool; therefore, any conceptualization relying on dynamic behavior (such as a repressilator) cannot be designed by this tool.
   b. As these tools are for genetic circuits, it should be possible to characterize inputs and outputs as binary. This could be achieved through defining inputs and outputs by their presence, expression, or by achieving a threshold distinguishing two states.
   c. As this is a tool for eukaryotes, the chassis for implementing this concept should be a eukaryote.
7. **Define the concept in a logic table.** Once a concept has been created for a synthetic biology application, it must be defined using a logic table. A logic table states, in binary terms, the input (column(s) on the Left-Hand Side, LHS) and output (column(s) on the Right-Hand Side, RHS) signals of the desired circuits. A '0' indicates absence whereas a '1' indicates presence for signal, whereas a '0' indicates inactivity and a '1' indicates activity for output protein signals. The logic table should include all possible ligand combinations for the conceptualized circuit. An example logic table is shown in Figure 3A. These examples show input signals of the presence of Cadmium (Cd) and Copper (Cu), a logic of Cd NIMPLY Cu, and an output signal of GFP expression.

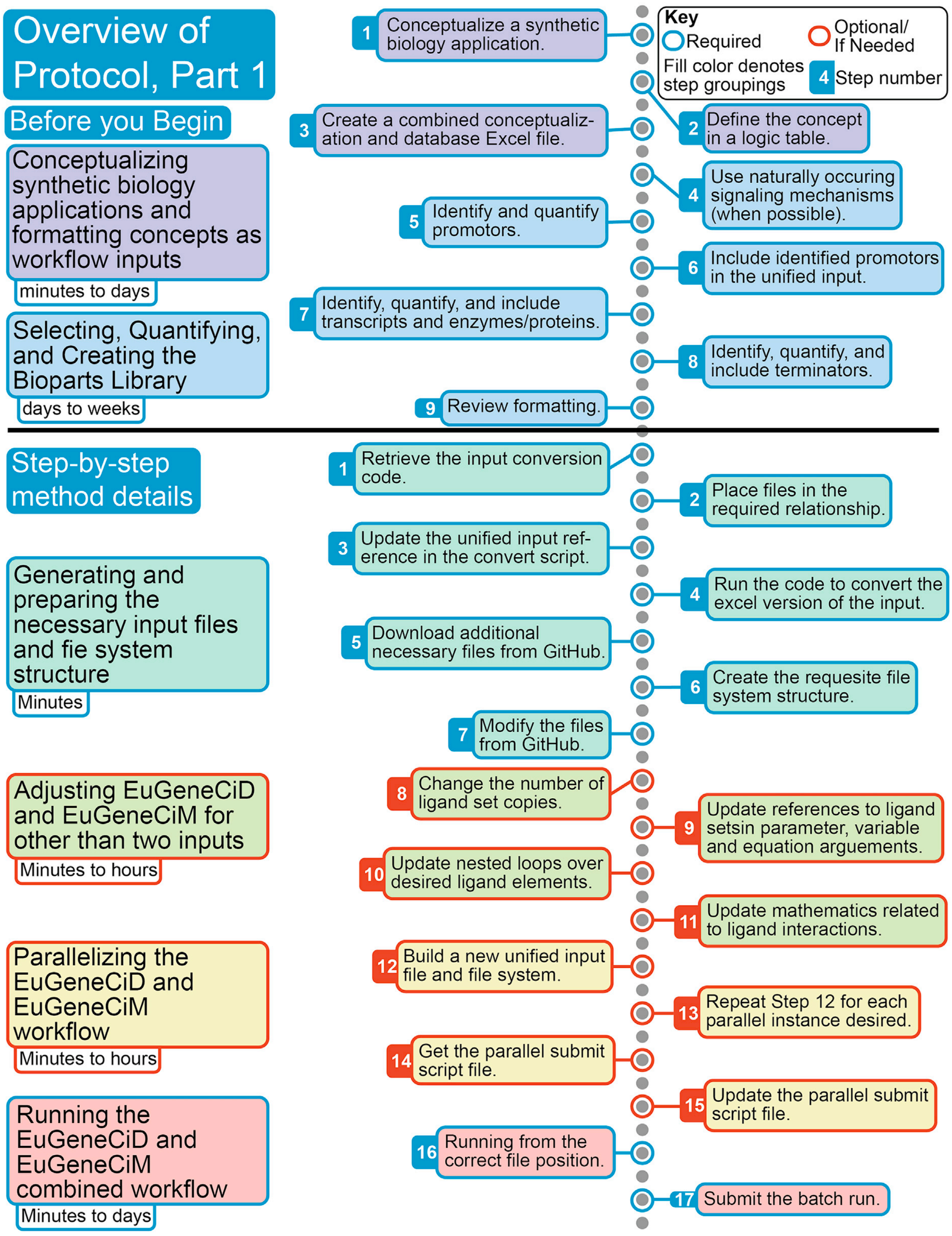

Overview of Protocol, Part 1
Before you Begin
Conceptualizing synthetic biology applications and formatting concepts as workflow inputs
minutes to days
Selecting, Quantifying, and Creating the Bioparts Library
days to weeks
Key
Required
Optional/ If Needed
Fill color denotes step groupings
4 Step number
1 Conceptualize a synthetic biology application.
2 Define the concept in a logic table.
3 Create a combined conceptualization and database Excel file.
4 Use naturally occuring signaling mechanisms (when possible).
5 Identify and quantify promotors.
6 Include identified promotors in the unified input.
7 Identify, quantify, and include transcripts and enzymes/proteins.
8 Identify, quantify, and include terminators.
9 Review formatting.
Step-by-step method details
Generating and preparing the necessary input files and fie system structure
Minutes
1 Retrieve the input conversion code.
2 Place files in the required relationship.
3 Update the unified input reference in the convert script.
4 Run the code to convert the excel version of the input.
5 Download additional necessary files from GitHub.
6 Create the requesite file system structure.
7 Modify the files from GitHub.
Adjusting EuGeneCiD and EuGeneCiM for other than two inputs
Minutes to hours
8 Change the number of ligand set copies.
9 Update references to ligand setsin parameter, variable and equation arguements.
10 Update nested loops over desired ligand elements.
11 Update mathematics related to ligand interactions.
Parallelizing the EuGeneCiD and EuGeneCiM workflow
Minutes to hours
12 Build a new unified input file and file system.
13 Repeat Step 12 for each parallel instance desired.
14 Get the parallel submit script file.
15 Update the parallel submit script file.
Running the EuGeneCiD and EuGeneCiM combined workflow
Minutes to days
16 Running from the correct file position.
17 Submit the batch run.

Figure 1. Overview of protocol, part 1
This figure provides a short descriptor of each steps of the protocol prior to step 17 of the step-by-step method details. This figure shows each section of steps as a different fill color and indicates if the step is required by the outline color of each step. Further, timing of each protocol section is detailed.

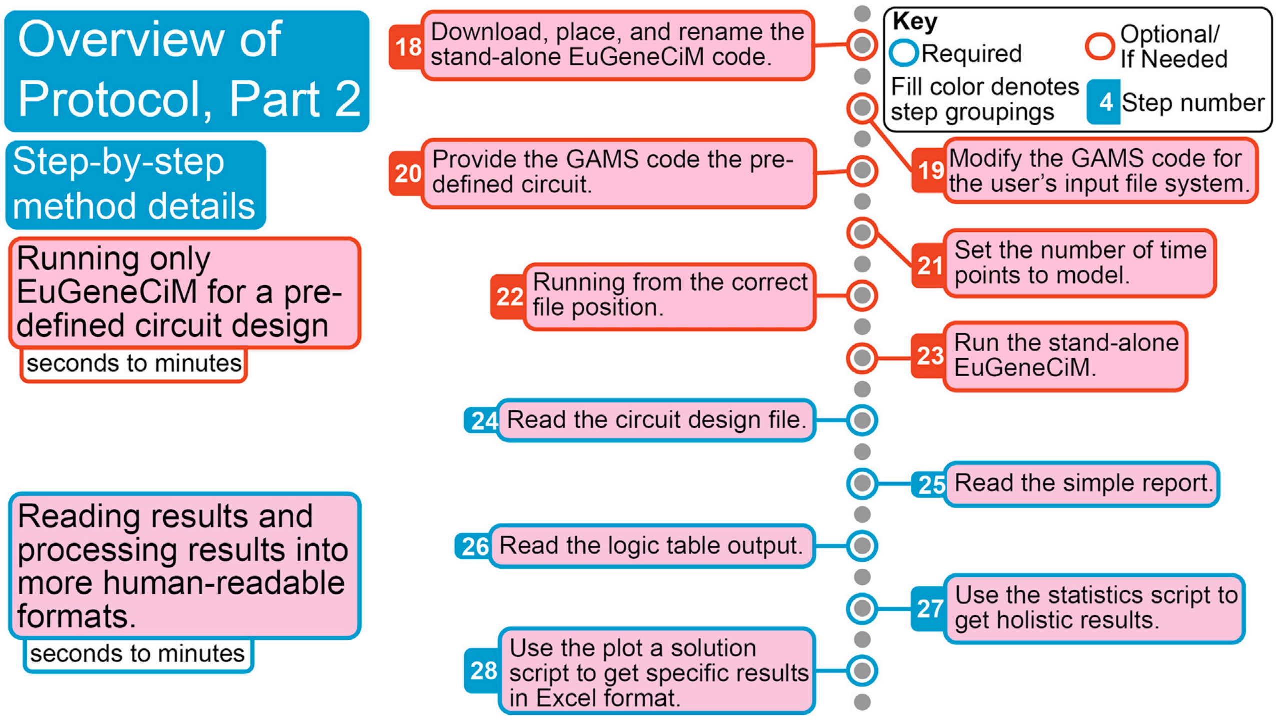

Overview of Protocol, Part 2
Step-by-step method details
Running only EuGeneCiM for a pre-defined circuit design
seconds to minutes
Reading results and processing results into more human-readable formats.
seconds to minutes
18 Download, place, and rename the stand-alone EuGeneCiM code.
19 Modify the GAMS code for the user's input file system.
20 Provide the GAMS code the pre-defined circuit.
21 Set the number of time points to model.
22 Running from the correct file position.
23 Run the stand-alone EuGeneCiM.
24 Read the circuit design file.
25 Read the simple report.
26 Read the logic table output.
27 Use the statistics script to get holistic results.
28 Use the plot a solution script to get specific results in Excel format.
Key
Required
Optional/ If Needed
Fill color denotes step groupings
4 Step number

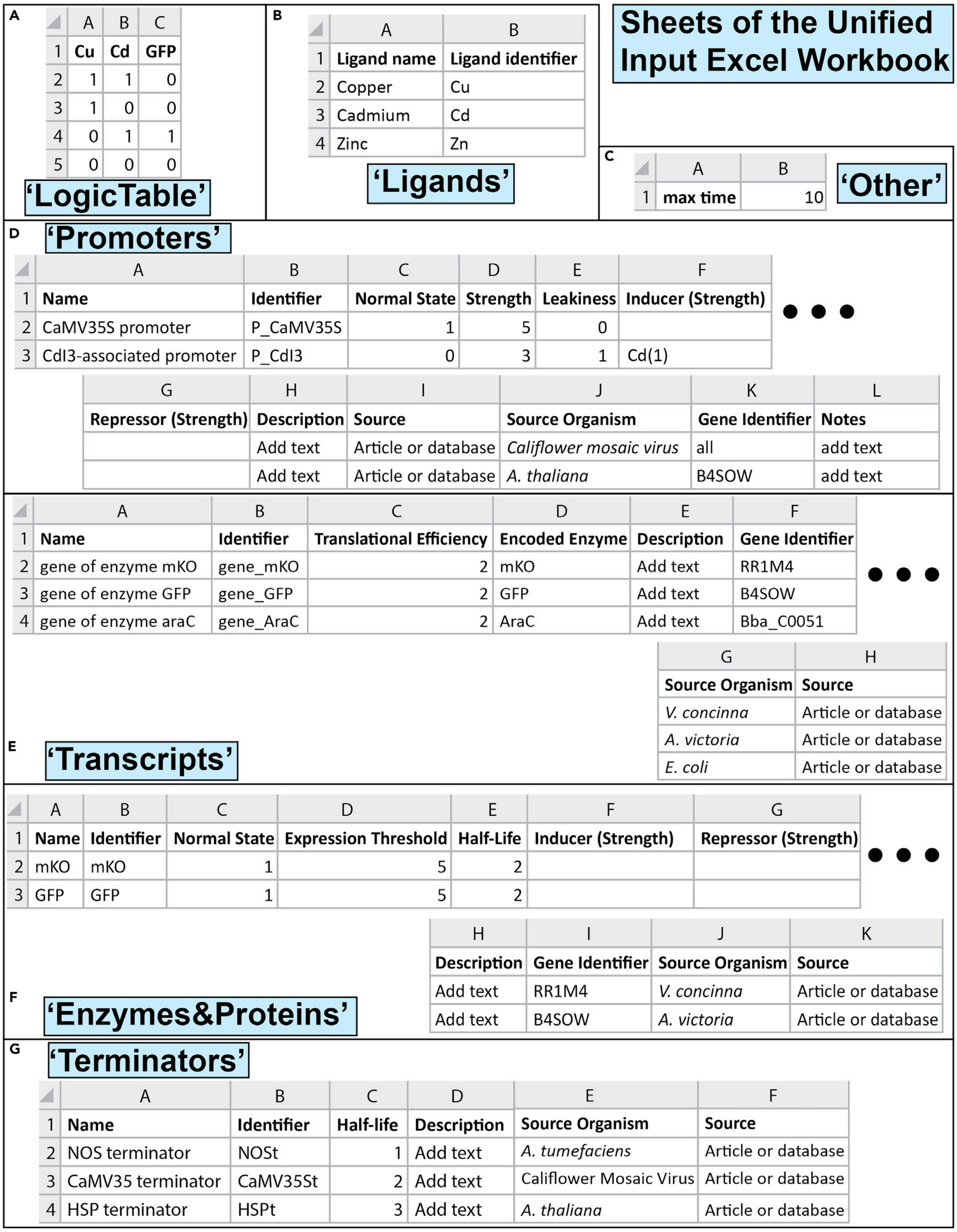
Figure 2. Overview of protocol, part 2

This figure provides a short descriptor of each steps of the protocol after step 17 of the step-by-step method details. This figure shows each section of steps as a different fill color and indicates if the step is required by the outline color of each step. Further, timing of each protocol section is detailed.

8. **Create a combined conceptualization and database Excel file.** Given that the combined EuGeneCiD/EuGeneCiM workflow code requires 26 unique input files, it was decided to define the conceptualization and the database bioparts library in a unified Microsoft Excel file (hereafter, referred to as the "unified input"), example sheets of which are shown in Figures 3A –3G.
   a. **Naming the input file.** This should be named "unifed_input_descriptor.xlsx", where "descriptor" stands in for a short descriptor which applies to the user's synthetic biology application conceptualization and/or database. For example, for the unified input for the Cadmium (Cd) NIMPLY Copper (Cu) circuit conceptualization in Schroeder, Baber, and Saha (2021), "descriptor" is replaced by "Cd_nimply_Cu", indicating that this was for a unified input containing bioparts, the logic gate conceptualization, and the input signals. This unified input could then be automatically converted to the required input files.
   b. **Creating some of the requisite sheets for the input file.** In this step, three of the sheets of this unified input related to defining the conceptualization will be created (examples of which are shown in Figures 3A–3C). Example unified input files can be found in the databases folder of theGitHub associated with this work (at github.com/ssbio/EuGeneCiDM/inputs/) named "unified_input_descriptor.xlsx", where "descriptor" stands for a descriptor applied to a synthetic biology conceptualization from Schroeder, Baber, and Saha (2021) (such as "Cd_nimply_Cu").

⚠ CRITICAL: Some type of descriptor, however brief, should be applied to the naming of the unified input file, otherwise downstream steps may throw an error.

Note: Throughout this protocol, descriptors in file names designated by the users of this protocol will be represented by the string "descriptor", while those in files downloaded from the GitHub associated with Schroeder, Baber, and Saha (2021) will generally be from the Cd



Sheets of the Unified Input Excel Workbook

A 'LogicTable'

	A	B	C
1	Cu	Cd	GFP
2	1	1	0
3	1	0	0
4	0	1	1
5	0	0	0

B 'Ligands'

	A	B
1	Ligand name	Ligand identifier
2	Copper	Cu
3	Cadmium	Cd
4	Zinc	Zn

C 'Other'

	A	B
1	max time	10

D 'Promoters'

	A	B	C	D	E	F
1	Name	Identifier	Normal State	Strength	Leakiness	Inducer (Strength)
2	CaMV35S promoter	P_CaMV35S	1	5	0	
3	CdI3-associated promoter	P_CdI3	0	3	1	Cd(1)

● ● ●

G	H	I	J	K	L
Repressor (Strength)	Description	Source	Source Organism	Gene Identifier	Notes
	Add text	Article or database	*Califlower mosaic virus*	all	add text
	Add text	Article or database	*A. thaliana*	B4SOW	add text

E 'Transcripts'

	A	B	C	D	E	F
1	Name	Identifier	Translational Efficiency	Encoded Enzyme	Description	Gene Identifier
2	gene of enzyme mKO	gene_mKO	2	mKO	Add text	RR1M4
3	gene of enzyme GFP	gene_GFP	2	GFP	Add text	B4SOW
4	gene of enzyme araC	gene_AraC	2	AraC	Add text	Bba_C0051

● ● ●

G	H
Source Organism	Source
V. concinna	Article or database
A. victoria	Article or database
E. coli	Article or database

F 'Enzymes&Proteins'

	A	B	C	D	E	F	G
1	Name	Identifier	Normal State	Expression Threshold	Half-Life	Inducer (Strength)	Repressor (Strength)
2	mKO	mKO	1	5	2		
3	GFP	GFP	1	5	2		

● ● ●

H	I	J	K
Description	Gene Identifier	Source Organism	Source
Add text	RR1M4	*V. concinna*	Article or database
Add text	B4SOW	*A. victoria*	Article or database

G 'Terminators'

	A	B	C	D	E	F
1	Name	Identifier	Half-life	Description	Source Organism	Source
2	NOS terminator	NOSt	1	Add text	*A. tumefaciens*	Article or database
3	CaMV35 terminator	CaMV35St	2	Add text	Califlower Mosaic Virus	Article or database
4	HSP terminator	HSPt	3	Add text	*A. thaliana*	Article or database

Figure 3. Example unified input sheets
This figure recreates various sheets in the unified input file, an Excel Workbook which is used in the workflow to describe the database and conceptualization in a single place. Shown here are all of the required sheets of that workbook, along with example entries from Schroeder, Baber, and Saha (2021). All non-empty columns are shown in these examples, though not all cells will necessarily be filled (such as repressor or inducer cells if that part has no such effectors). Next to each example table is the name given to the sheet. The names of sheets should be reproduced exactly as shown, otherwise portions of this protocol may not work.
(A) An example of a logic table conceptualization, of a Cd NIMPLY Cu logic circuit with a green fluorescent protein (GFP) output as a reporter of circuit state. This example shows that identifiers, defined in other sheets, are used as headings in the logic table.
(B) This shows what a typical "Ligands" sheet will look like, listing all ligands with both a name and a globally unique identifier.
(C) This shows all that is required in an "Other" sheet, namely the label "max time" and a whole number value representing the largest relative time point to model using EuGeneCiM (where the smallest time point modeled is always 0).
(D) Shows non-empty cells of a typical "Promoters" sheet along with two example promoters. The order and type of information reported in each column should be preserved.
(E). Shows non-empty cells of a typical "Transcripts" sheet with two example transcripts. In using this protocol, the order and type of information reported in each column should be preserved.
(F) Shows non-empty cells of a typical "Enzymes&Proteins" sheet with two example enzymes/proteins. In using this protocol, the order and type of information reported in each column should be preserved.
(G) Shows non-empty cells of a typical "Terminators" sheet with three example terminators. In using this protocol, the order and type of information reported in each column should be preserved.

NIMPLY Cu circuit conceptualization and will often contain the string "Cd_nimply_Cu" in the file names. This will then be replaced by the user's "descriptor".

i. **The logic table sheet.** Create a sheet named "LogicTable". This sheet is the human-readable version of the binary logic table which represents the conceptualization. The first row is the header, which begins by listing the ligands to which the circuit must respond (one per cell), followed by a list of one or more enzymes whose activity is the desired output signal of the conceptualized circuit. An example "LogicTable" is shown in Figure 3A.

⚠ CRITICAL: Since programs which read and convert the unified input file are case-sensitive, sheet names should be reproduced exactly as shown.

⚠ CRITICAL: All ligands in the logic table must match a ligand identifier in the "Ligands" sheet (created in step 8.b.i); otherwise an error will be returned when attempting to solve EuGeneCiD.

⚠ CRITICAL: All enzymes in the logic table must match an enzyme identifier in the "Enzymes&Proteins" sheet (created in "selecting, quantifying, and creating the bioparts library" section, step 12); otherwise, an error will be returned when attempting to solve EuGeneCiD.

Note: The order in which the sheets are arranged in the Excel workbook does not matter. Further, any additional sheets will not affect downstream steps if users wish to include additional documentation.

ii. **The ligand sheet.** Create a sheet named "Ligands". This sheet will have two columns, "Ligand name" and "Ligand identifier". In the first column, provide a human-readable name for each ligand which will serve as a signal to the genetic circuit by its presence or absence (one row per ligand). In the second column, provide a unique short name or shorthand for the ligands which downstream programs in this workflow will use to identify these ligands. An example "Ligands" sheet is shown in Figure 3B.

Note: Any number of ligands may be listed here and in any order. This sheet defines the set of ligands, of which the ligands to which the conceptualized circuit responds is a subset.

Note: The ligand sheet header text is not read by any downstream program, so header text can be customized to whatever the user feels most appropriate. The data which should be included in that column, however, should remain constant (e.g., do not rearrange column order). This applies to all sheets of the unified input except the logic table.

⚠ CRITICAL: Ligand identifiers should be unique strings, without spaces or special characters (in other words, consisting of letters and numbers only). This rule applies to all identifiers. Should the string not be unique, this will cause errors later in the workflow. The same applies to all ''identifiers'' of all bioparts.

iii. **The other sheet.** This sheet should be named ''Other''. At present, there is only two cells which need be written. Cell A1 should contain the text ''max time'', and cell B1 should contain a whole number (i.e., 0, 1, 2, 3, etc.) which is the last time point for EuGeneCiM to model (EuGeneCiM will start a time 0). An example ''other'' sheet is shown in Figure 3C.

Note: As the EuGeneCiD and EuGeneCiM tools use relative values for all bioparts characteristics (promoter strength, enzyme concentration threshold for activity, translational efficiency, biopart half-life, etc.), these time points are relative (as will be enzyme concentration and transcript abundance results from using these tools). It is recommended that the max time point investigated be at least a value of 10 (as potentially problematic dynamic circuit behavior often manifests before this point for simpler circuits like BUFFER, AND, XOR, NOR, etc. as studied in Schroeder, Baber, and Saha (2021)). Should more complex logic circuits be desired, this maximum time point should be increased. As EuGeneCiM is not particularly computationally expensive (in comparison to EuGeneCiD), this time point can generally be increased without major increases to workflow time.

Selecting, quantifying, and creating the bioparts library

⏲ Timing: days to weeks

Literature and database searches must be undertaken to identify useful bioparts (promoters, genes, transcripts, and enzymes) to comprise the input bioparts library. This library both contains all the parts which might be used in creating a genetic circuit as well as quantifies the qualities of these parts. A well-designed biopart library is important to having a viable solution space in the EuGeneCiD problem, and thus identifying potential design solutions.

9. **Using naturally occurring signaling mechanisms (when possible).** Signaling pathways often are comprised of several enzymes, proteins, and/or metabolic reactions necessary to pass the signal from the cell's external environment to the nucleus, for which it may be impractical or difficult to transform into a chassis organism. Therefore, using bioparts which are affected by these pathways may be more viable or simple to implement, while simplifying these signaling pathways to activation or repression by that which triggers the signaling cascade.
   a. **Examples of using naturally occurring mechanisms.** In Schroeder, Baber, and Saha (2021), several promoters are listed as cadmium activated, though this activation is indirect and is rather achieved by kinases acting upon transcription factors, or the indirect metabolic actions of cadmium ions inside the cell which create reactive oxygen species (Chmielowska-Bąk et al., 2014).
   b. **Considerations of the simplification.** This simplification is not a problem for EuGeneCiD and EuGeneCiM since design and modeling parameters associated with bioparts are relative, rather than absolute characterizations which would be complicated by the multiple steps and mechanisms between cadmium ions and the affected promoters.

10. **Identifying and parameterizing promoters.** Promoters are, in some instances, the most difficult of bioparts to identify and characterize, since they are generally closely associated with their related transcript and are often not treated as independent, highlighted by the fact that many identifier systems lack individual identifiers for promoters.

a. **Promotor identifiers and recommended parameter ranges.** Discussed in the following sub steps is the identification and recommended parameter ranges for promotor characterization.
   i. **Promotor names.** In Schroeder, Baber, and Saha (2021), most promoters have names such as "CdI3-associated promoter" and use the identifier of the promoter's associated gene as one method of identifying the biopart of interest. Generally, the strength of a promoter will be determined by transcript abundance or gene expression. Two example entries for characterized promoters are shown in Figure 3D. The name assigned can be anything, as the name is not used to construct input file but exists solely as a human-readable identifier.
   ii. **Promotor identifier.** The identifier should be a unique string which the workflow will use to uniquely identify the promotor in question. In the case of the "CdI3-associated promotor", the identifier given is "P_CdI3.
   iii. **Promotor normal state parameter range.** The normal state is a binary parameter which notes whether a promoter is normally on (value 1) or off (value 0) in what will be defined as the "no signals" state for a genetic circuit conceptualization.
   iv. **Promotor strength suggested parameter range.** The promoter strength can be any non-negative value, though it is suggested to parameterize it using an integer between 1 (very weak) and 5 (very strong).
   v. **Promotor leakiness suggested parameter range.** Leakiness can also take any non-negative value, though it is suggested to parameterized it on a scale of 0 (no transcript production when promoter is off) to 3 (very leaky).

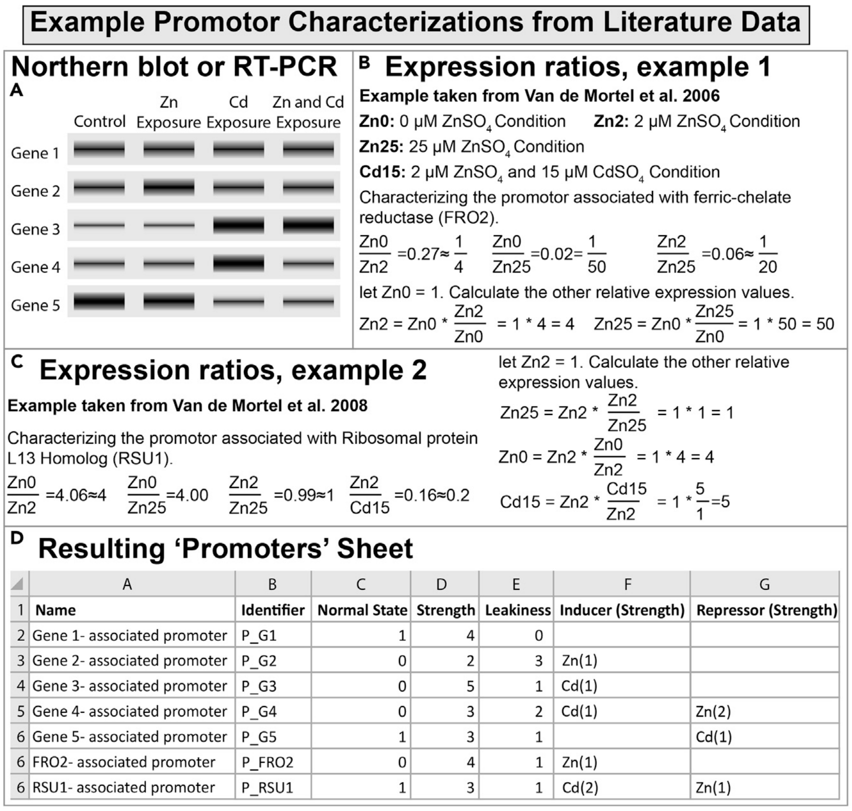
Note: Note that the ranges for strength and leakiness (steps 10.a.iv and 10.a.v) are recommended, not hard-coded, since what are important here are relative values and changing these values will influence downstream characterizations. For instance, if the scale of integer characterizations for a promoter are increased (say, to be on a scale from 2 to 10), then the concentration threshold for protein expression would have to be similarly increased (from 5 to 10 or higher, for instance).

b. **Formatting effectors of the promotor.** Inducers and repressors are reported in the same format of "XX(Y)".
   i. **Effector identifier.** In the format of "XX(Y)", XX denotes the identifier associated with a ligand or enzyme which effects the activity of the promoter. The identifier given must exactly match that of a ligands or enzyme.
   ii. **Effector strength.** In the format of "XX(Y)", and Y denotes the strength of that interaction. The interaction strength is used to denote the primacy of various effectors, and the baseline (weakest) effector strength is generally quantified as having a strength of 1. Should data exist to suggest that a particular effector has a stronger effect than the weakest effector, its strength is assigned the next integer value larger than that of the weakest effector. This provides a relative ranking of effectors under consideration.

Note: An example is Gene 4, shown in Figure 4A,is activated by exposure to Zn (with effector strength of 1 is assigned), yet when exposed to Zn and Cd is inactive, suggesting that Cd is an inhibitor and a stronger effector than Zn (with effector strength of 2 is assigned).

c. **Documentation.** The remaining columns exist for the documentation of the part, including its description, literature source, source organism, identifier of the associated genes, and other pertinent notes to its parameterization and/or inclusion.

Note: Example parameterizations are shown in Figure 3D as well as in Figure 4 which shows the parameterization of promoters from various data sources.



B Expression ratios, example 1

Example taken from Van de Mortel et al. 2006

Zn0: 0 μM $ZnSO_4$ Condition **Zn2:** 2 μM $ZnSO_4$ Condition

Zn25: 25 μM $ZnSO_4$ Condition

Cd15: 2 μM $ZnSO_4$ and 15 μM $CdSO_4$ Condition

Characterizing the promotor associated with ferric-chelate reductase (FRO2).

$\frac{Zn0}{Zn2} = 0.27 \approx \frac{1}{4}$ $\frac{Zn0}{Zn25} = 0.02 = \frac{1}{50}$ $\frac{Zn2}{Zn25} = 0.06 \approx \frac{1}{20}$

let Zn0 = 1. Calculate the other relative expression values.

$Zn2 = Zn0 * \frac{Zn2}{Zn0} = 1 * 4 = 4$ $Zn25 = Zn0 * \frac{Zn25}{Zn0} = 1 * 50 = 50$

C Expression ratios, example 2

Example taken from Van de Mortel et al. 2008

Characterizing the promotor associated with Ribosomal protein L13 Homolog (RSU1).

$\frac{Zn0}{Zn2} = 4.06 \approx 4$ $\frac{Zn0}{Zn25} = 4.00$ $\frac{Zn2}{Zn25} = 0.99 \approx 1$ $\frac{Zn2}{Cd15} = 0.16 \approx 0.2$

let Zn2 = 1. Calculate the other relative expression values.

$Zn25 = Zn2 * \frac{Zn2}{Zn25} = 1 * 1 = 1$

$Zn0 = Zn2 * \frac{Zn0}{Zn2} = 1 * 4 = 4$

$Cd15 = Zn2 * \frac{Cd15}{Zn2} = 1 * \frac{5}{1} = 5$

D Resulting 'Promoters' Sheet

	A	B	C	D	E	F	G
1	Name	Identifier	Normal State	Strength	Leakiness	Inducer (Strength)	Repressor (Strength)
2	Gene 1- associated promoter	P_G1	1	4	0		
3	Gene 2- associated promoter	P_G2	0	2	3	Zn(1)	
4	Gene 3- associated promoter	P_G3	0	5	1	Cd(1)	
5	Gene 4- associated promoter	P_G4	0	3	2	Cd(1)	Zn(2)
6	Gene 5- associated promoter	P_G5	1	3	1		Cd(1)
6	FRO2- associated promoter	P_FRO2	0	4	1	Zn(1)	
6	RSU1- associated promoter	P_RSU1	1	3	1	Cd(2)	Zn(1)

Figure 4. Example promoter characterizations from literature data

As the characterization of promoter bioparts is not a precise quantification, shown here are three example data source (one fictional, two real) and an example resulting "Promoters" sheet.

(A) Shows an example Northern blot or RT-PCR analysis that may be used for promoter characterization.

(B) Shows an example of relative expression data that may be used to quantify promoters for use with EuGeneCiD and EuGeneCiM.

(C) Gives a second example of the same type of data as B.

(D) Shows the resulting promoter characterizations for EuGeneCiD and EuGeneCiM by showing the first several columns of a "Promoters" sheet that might result from this data.

d. **Various possible datasets and their use for the parameterization of promotors.** The following steps list some sources of characterization information that users might commonly encounter and how to characterize this information for EuGeneCiD and EuGeneCiM. This list is not intended to be exhaustive, but rather illustrative of possible sources of promoter information which needs to be characterized in this tool, and some methods of how to perform that characterization.

   i. **Northern blot, RT-PCR, or other semi-quantitative datasets.** Semi-quantitative data can be used with the EuGeneCiD and EuGeneCiM to define relative promoter characteristics though the lens of relative RNA abundance (Reue, 1998).

Note: Examples of how the authors have quantified faux Northern blot or RT-PCR data are shown in Figure 4A, with each example gene highlighting important parameterization examples, with the resulting parameterizations shown in Figure 4D. Fictitious genes are used here to highlight more possible parameterization scenarios. Note that with this type of data, unless time-course data is available, it is assumed that all terminators are the same strength; therefore, the differences in expression are controlled by the promoters.

- **Characterizing constitutive gene expression.** Constitutively expressed genes should be assigned a leakiness of zero and a strength corresponding to their relative expression strength.

Note: In Figure 4 gene 1 is given as an example constitutive gene identified in Northern blot, RT-PCR, or other semi-quantitative manner. For this gene, a leakiness of 0 is assigned as the promoter is never off. For the strength of expression (as shown by relative band intensity compared to other bands) the blot or analysis is used then to determine the strength of the gene's expression due to the promoter.

- **Characterizing leaky gene expression.** Very leaky genes will generally be assigned a high leakiness value in the defined range of values (see step 10.a.iv) and a low strength (see step 10.a.v).

Note: In Figure 4, gene 2 is included as an example of a leaky gene. Gene 2 show a very leaky expression (leakiness assigned as 3) which is activated by exposure to Zn, yet has only a slight increase in expression as a result of activation. This gene then is characterized as normally off with a low strength (assigned as 2, since expression when active is based on the sum of leakiness and strength).

- **Characterizing non-leaky gene expression.** Genes which are non-leaky are assigned a low leakiness value in the defined range (see step 10.a.iv) and a strength corresponding to its relative strength compared to other genes (see step 10.a.v).

Note: Gene 3 is included in Figure 4 as a gene with low leakiness (assigned to a value of 1) which is strongly activated by exposure to Cd. It is therefore assigned a strength of 5 and a leakiness of 1. Since there is only one effector, this effector is assigned a strength of 1 as the baseline.

- **Characterizing gene expression with multiple effectors.** Semi-quantitative data can show genes with multiple effectors. Effector strength should be assigned as detailed in step 10.b.ii.

Note: Gene 4 is included in Figure 4 as an example of a moderately leaky (assigned a leakiness of 2) normally "off" gene with multiple effectors. Gene 4 is activated by Cd yet is not "on" when exposed to both Zn and Cd. This suggests that both Zn and Cd are effectors, and that Zn is a stronger effector than Cd since it can repress gene expression when Cd (an activator) is present. Therefore, the strength of the interaction with Cd is assigned a value of 1 (as the weakest effector), whereas the interaction strength with Zn is assigned a value of 2 (as a stronger effector). Gene 5 is a normally "on" gene showing moderate leakiness. It also shows a slight repression of expression with respect to Zn, and a strong repression with respect to Cd. Since this tool uses binary, rather than continuous, indicators of activity, in this example, the authors have decided that the repression shown by exposure to Zn is not significant enough to parameterize this gene as being repressed by Zn. Therefore, this promoter is only parameterized as being repressed by Cd. In Schroeder, Baber, and Saha (2021), the work of Suzuki et al. (2001) which included Northern blot and RT-PCR data was used to quantify some Zinc-, Copper-, and Cadmium- responsive promoters.

Note: Ideally, when working with several semi-quantitative datasets across different sources, the sources will share a housekeeping gene as a common benchmark for quantifying relative expression. If such a benchmark exists, use this as a standard against which to quantify promoter strength and leakiness of each other promoter against. Note that as a housekeeping gene is generally constitutive, the promoter associated with such a gene would be classified with a leakiness of 0 (since it is never off) and all expression of such a constitutive promoter is assigned to promoter strength (relative to the other bands, and as in contrast to dividing expression strength between leakiness and strength parameters).

Note: Should the ratio of expression in ''on'' compared to ''off'' states be high (e.g., 10 or more), check against a housekeeping gene to confirm if the relative expression is truly extraordinary (as evaluated by comparison to a housekeeping gene), and that a high promoter strength (e.g., greater than 5) is warranted. Should the relative expression compared to a housekeeping gene not be extraordinary (or a comparison not be found) it is recommended to ensure characterizations fall within ranges presented here. One approach which might be taken to achieve characterizations within the usual ranges presented here is to decreased promoter leakiness. For instance, if the leakiness is characterized as 0.25 and the strength as 4.75, then the ratio of expression when on to when off is 5:0.25, or a 20-fold increase in expression without using an unusual strength or leakiness characterization value.

e. **RNA-seq, qRT-PCR, and other quantitative techniques.** Data from quantitative techniques for transcript expression, such as RNA-seq and qRT-PCR may also be used for quantifying promoters for EuGeneCiD and EuGeneCiM.
   i. **Significance of differential expressions.** Using such data, for instance RNA-seq, it must first be determined if expression differences are statistically significant between two or more conditions. Should the difference between two conditions be statistically insignificant, then that condition does not affect the promoter. Should the difference between the two conditions be statistically significant, the condition with the lower expression should be used as the basis for calculation (the leakiness of the promoter in the ''off'' state). The ratio between this basis and the highest expression should be the sum of the leakiness and strength of the promoter, as described in step 10.a.

Note: Methods of statistical analysis for quantitative methods will not be discussed here, as it should be performed by the authors of the study from which the quantitative data is taken. If the users of this protocol are performing the quantitative techniques or the original study did not perform a statistical analysis, the authors would refer users to the work of Fang et al. (2012) (for RNA-seq data), Guénin et al., 2009 (for qRT-PCR), or other relevant works on best statistical practices.

f. **Expression ratios.** Some sources (literature or database) will not report raw expression data, but rather fold change (or log fold change) data comparing two states as ratios of expression. The following substeps will describe the general procedure for how th characterize such data.
   i .**Identify a basis of calculation.** Identify the condition with the lowest expression and assign it a relative expression value of 1.
   ii .**Determine relative expression between conditions.** Then, use the ratios provided to convert to the relative expression value for other experimental conditions, and use these relative values to characterize the promoters.

Note: Two examples of data used by Schroeder, Baber, and Saha (2021) from Van De Mortel et al., 2006 and Van De Mortel et al., 2008 are shown in Figures 4B and 4C, with the resulting parameterizations shown in Figure 4D. These examples are included to help readers better understand parameterization of this type of data. Four transcript expression ratios are given by Van De Mortel et al. relating Zinc deficient (Zn0), Zinc sufficient (Zn2), Zinc excess (Zn25), and Zinc sufficient with Cadmium (Cd15) conditions.
Example 1: Since EuGeneCiD and EuGeneCiM are relative tools, the ratios can be rounded to values which make downstream calculations easier, as shown in Figure 4B. Generally, this will be either rounding to whole numbers or to simple fractions as is the case for these. From these ratios, Zn0 can be identified as the condition with the lowest gene expression and will

be assigned to a relative expression value of 1. From this, relative expressions in conditions Zn2 and Zn25 can be calculated as 4 and 50, respectively. Using the Zn sufficient condition (Zn2) compared to the Zn deficient conditions (Zn0) as a baseline, the promoter associated with FRO2 (ferric-chelate reductase 2), P_FRO2, is first defined as normally off, since expression is relatively very low in the Zn0 condition). P_FRO2 is then defined as induced by Zn, since expression is greatly increased when Zn is present. The strength of this interaction is given the default value of 1. Next, P_FRO2 is assigned a leakiness of 1, as a value of 1 is our default leakiness value based on assigning the lowest expression (Zn0) as having a relative expression of 0. Finally, P_FRO2 is assigned a strength of 3, because the effective strength of a promoter when it is ''on'' is the value of the leakiness (1) plus the assigned strength (3) for the effective strength of the promoter when on (4 total when ''on'', compared to 1 when ''off''). This corresponds to the relative expression value of 4 seen for the Zn2 condition. However, considering the high relative strength of the promoter at higher Zinc concentrations, the strength estimate was later revised up to 4 so that total expression when the promoter is ''on'' is now 5.

Example 2: Unlike the previous example, this one contains statistically non-significant effects (namely the Zn2/Zn25 ratio in Figure 4C), which should be rounded to 1 before doing other rounding. As before, this will be either rounding to whole numbers, as is the case for Zn0/Zn2 ratio, or to simple fractions as is the case of Zn2/Cd15. Since the Cd15 condition has both Zinc and Cadmium present, it will be ignored when determining base relative expression, but will be revisited when determining effectors and effector interaction strengths since the effects of each cannot be separated immediately. Amongst the Zinc conditions, using logic, Zn2 and Zn25 are identified as having the lowest relative expression. For this example then, we let Zn2 expression be 1. From this, using the above ratios, we can calculate that relative expression under the Zn0 condition is 4, and under the Zn25 condition is 1. Therefore, we can call the promoter associated with RSU1 (ribosomal protein L13 homolog), P_RSU1, to be normally on, as it is ''on'' in the absence of the signal ions. As with P_FRO2, we assign a leakiness of 1, as an expression level of 1 is the basis of our calculations (with Zzn2 condition having the minimum expression). Next, from our results we see a four times greater expression in Zn0 compared to Zn2 conditions, therefore we assign a strength of 3 (for a total of four times less expression when Zn is present) and a repressor of Zinc given the default interaction strength of 1. Returning to the Zn2/Cd15 ratio, this shows that Cadmium presence, while also in the presence of Zinc, results in a recovery of expression. This is shown in the relative condition expression levels, where Zn0 relative expression is 4, Zn2 relative expression is 1, and Cd15 (a condition including Zn and Cd presence) relative expression is 5. Therefore, Cd is an activator of the P_RSU1 promoter which is more powerful than the inhibition caused by Zn; therefore, this interaction is assigned a strength of 2.

g. **Descriptions in article text.** Occasionally, the main text of an article, particularly one focusing on a single important promoter related to the expression of an important gene, may detail characteristics of the promoter in question. Users of this protocol will have to use best judgment in characterizations as each reference would likely be idiosyncratic.
h. **Previous characterizations.** Characterizations of promoters from other optimization-based circuit design tools of similar structures can be used. Examples at the time of publication include Schroeder, Baber, and Saha (2021), Dasika and Maranas (2008), and Zomorrodi and Maranas (2014).
i. **Other sources.** Other sources of information exist from which useful characterizations can be drawn, such as data provided by vendors, in-house data on promoter activity, and expert knowledge of a given system.

11. **Include identified promoters in the unified input.** Identified promoters should be included in the unified input file in a new sheet with the name ''Promoters'', the headings of which are shown in Figure 3D, along with two example entries from Schroeder, Baber, and Saha (2021). Further

example entries are shown in Figure 4D related to the characterization of illustrative promoter data discussed in notes to step 10.

Note: As with other sheets, all identifiers must be globally unique strings and the header text may be adjusted to suit user preferences.

12. **Identify, quantify, and include transcripts and enzymes/proteins.** Transcripts and enzymes/proteins are inextricable linked through the central dogma of biology, and thus will be addressed in this unified step, though we will begin with transcripts.
    a. **Transcripts.** The parameterization required for transcripts is shown in Figure 3E. The "Transcripts" sheet should be created in the unified input, populated with this data, as shown in Figure 3E.
        i. **Transcript name.** The "name" given should be a human-readable name, and can be any string, as it is not used by the subsequent programs, but rather is a matter of documentation.
        ii. **Transcript identifier.** The provided identifier should be a unique string and will be used by subsequent programs to identify the biopart.
        iii. **Suggested translational efficiency range.** The translational efficiency parameter allows for accounting of various factors which can influence gene expression such as codon optimization (Kudla et al., 2009)(Ward et al., 2011), speed of transcription, or, presumably, codon deoptimization. Translational efficiency is suggested to be a parameter on a scale of 1–3 which indicates how efficiently mRNA sequences are expressed as proteins.
        iv. **Defining translational efficiency.** In Schroeder, Baber, and Saha (2021), since no such techniques were applied, transcriptional efficiency of all transcripts is set at 2 (used here as the default for a non-optimized efficiency).
            - **Alternative splicing.** If a transcript does encode multiple isoforms, it may be more accurate to decrease transcriptional efficiency to model that some transcript produces each enzyme, but not both (due to post-transcriptional modifications).
        v. **Mapping transcript to enzyme.** Write the identifier of the enzyme which is encoded by the transcript. If there are multiple enzymes, list them separated by semi-colons (so that the convert script recognizes them).
        vi. **Documentation.** The remaining columns of "description", "gene identifier", "source organism", and "source" are for annotation of the database, including describing decisions made during parameterization, giving an identifier which can be used to uniquely identify the transcript, identifying its source, and citing where the data is from.
    b. **Enzymes/Proteins.** The selected enzymes will follow directly from the selected transcripts. All enzymes which are indicated as encoded in the "Transcripts" sheet must be parameterized here.
        i. **Enzyme name and identifier.** As with other sheets, the name is a human-readable name, and the identifier is used by programs throughout the workflow.
        ii. **Enzyme parameter recommended ranges.** Detailed in the following substeps are suggested parameter ranges for characteristics associated with enzymes in the EuGeneCiD and EuGeneCiM tools.
            - **Enzyme normal state.** As with the "Promoters" sheet, a normal state of 1 indicates a protein or enzyme that is "on" by default, and 0 indicates a default "off" state.
            - **Enzyme relative expression threshold suggested range.** Expression threshold denotes a relative value (on a scale of 1–10) of enzyme/protein concentration that is necessary for a cell to express the phenotype associated with that enzyme. The default value is 5 which can be adjusted based on the characteristics of the protein/enzyme in question.
            - **Enzyme half-life parameter suggested range.** Half-life values assigned are generally between 0 and 3, depending on how quickly (relatively) the given enzyme or protein degrades. The default value used is 2.

iii. **Defining enzyme effectors.** Finally, activators and repressors are noted. These entries are to be formatted the same as in the "Promoters" sheet (see step 10b), with the identifier, followed by relative interaction strength in parentheses, and multiple entries separated by semi-colons.
iv. **Documentation.** The final four columns, "description", "gene identifier", "source organism", and "source" exist for the purposes of documentation. An example of an enzyme sheet is shown in Figure 3F. This sheet should be named "Enzymes&Proteins"

13. **Identify, quantify, and include terminators.** An example terminator sheet is shown in Figure 3F. As with other sheets, the name is meant to be a human-readable name which is used for documentation and the identifier is a unique string identifier. The only parameterization associated with terminators is half-life, generally on a scale of 1 (shortest) to 3 (longest). Finally, the "description", "source organism", and "source" columns are for adding documentation about the terminators used.
14. **Review formatting.** By this step, the unified input file should be complete. Please review formatting of the unified input file. Particularly the identifiers used (each should be globally unique) and the order in which the columns are placed (should be the same as in Figure 3), as a common error in the workflow results from repeat identifier and wrong types of data in cells (see troubleshooting 1 and 2).

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
GitHub	www.github.com	RRID:SCR_002630
Software and algorithms		
Perl Programming Language (version 5.26 for Unix)	Perl www.perl.org	RRID:SCR_018313
Strawberry Perl version 5.24.0.1 (for Windows)	Strawberry Perl Strawberryperl.com	RRID:SCR_018313
The world-wide-web library for Perl, module 6.39	LWP Meta CPAN https://metacpan.org/pod/LWP	N/A
Comprehensive Perl Archive Network (CPAN)	https://metacpan.org/	RRID:SCR_007253
Generalized Algebraic Modeling System (GAMS) version 24.7.4	GAMS Products and Downloads www.gams.com/products/buy-gams/	RRID:SCR_018312
CPLEX solver version 12.6	GAMS Products and Downloads www.gams.com/products/buy-gams/	N/A
Other		
Holland Computing Center: Crane Computing Cluster (64 GB RAM, Intel Xenon E5-2670 2.60 GHz processor, 2 CPUs per node)	Holland Computing Center https://hcc.unl.edu/	N/A
ASUSTeK Zyphyrus G model laptop computer with Microsoft Windows 10.	Any reasonably up-to-date computer, and alternative OSs, will work for this protocol.	N/A
Dell OptiPlex 790 desktop computer with Microsoft Windows 10 Enterprise	Any reasonably up-to-date computer, and alternative OSs, will work for this protocol.	N/A

MATERIALS AND EQUIPMENT

Throughout this work, two different computers were used interchangeably, both of which are included in the key resources table. Uses of these computers include directly running program codes and for Secure Shell (SSH) access to the Crane computing cluster at the Holland Computing Center (HCC) of the University of Nebraska – Lincoln. A description of the Crane cluster is provided in the key resources table. The Crane computing cluster is used primarily for running GAMS codes in a timely manner, as these codes are used to solve large linear algebra problems with matrix dimensions in the order of hundreds or thousands of elements in multiple dimensions. Such problems are either very straining on a personal computer or even impossible in a reasonable amount of

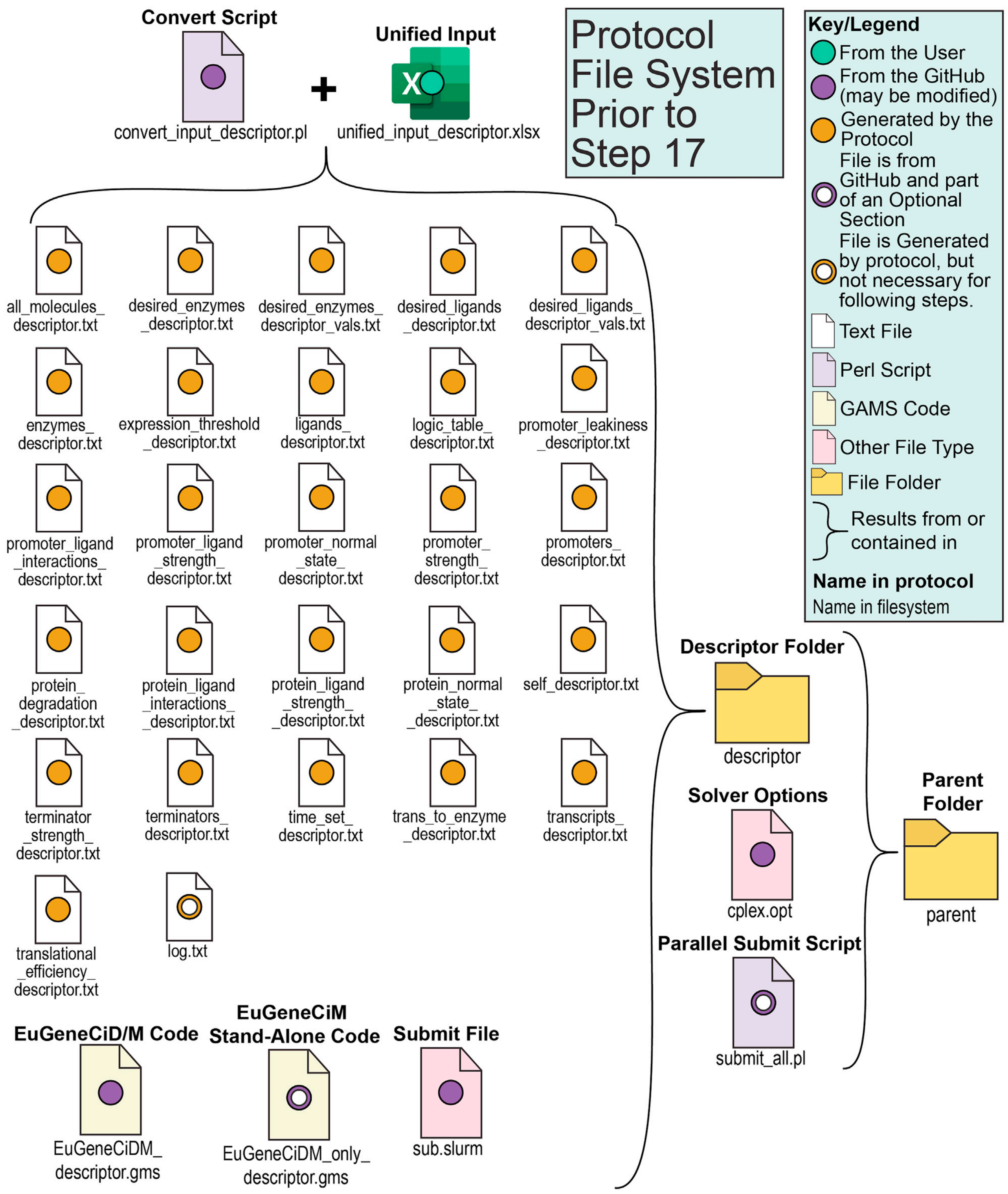

Convert Script
convert_input_descriptor.pl
+
Unified Input
unified_input_descriptor.xlsx
Protocol File System Prior to Step 17
Key/Legend
From the User
From the GitHub (may be modified)
Generated by the Protocol
File is from GitHub and part of an Optional Section
File is Generated by protocol, but not necessary for following steps.
Text File
Perl Script
GAMS Code
Other File Type
File Folder
Results from or contained in
Name in protocol
Name in filesystem
all_molecules_descriptor.txt
desired_enzymes_descriptor.txt
desired_enzymes_descriptor_vals.txt
desired_ligands_descriptor.txt
desired_ligands_descriptor_vals.txt
enzymes_descriptor.txt
expression_threshold_descriptor.txt
ligands_descriptor.txt
logic_table_descriptor.txt
promoter_leakiness_descriptor.txt
promoter_ligand_interactions_descriptor.txt
promoter_ligand_strength_descriptor.txt
promoter_normal_state_descriptor.txt
promoter_strength_descriptor.txt
promoters_descriptor.txt
protein_degradation_descriptor.txt
protein_ligand_interactions_descriptor.txt
protein_ligand_strength_descriptor.txt
protein_normal_state_descriptor.txt
self_descriptor.txt
terminator_strength_descriptor.txt
terminators_descriptor.txt
time_set_descriptor.txt
trans_to_enzyme_descriptor.txt
transcripts_descriptor.txt
translational_efficiency_descriptor.txt
log.txt
EuGeneCiD/M Code
EuGeneCiDM_descriptor.gms
EuGeneCiM Stand-Alone Code
EuGeneCiDM_only_descriptor.gms
Submit File
sub.slurm
Descriptor Folder
descriptor
Solver Options
cplex.opt
Parallel Submit Script
submit_all.pl
Parent Folder
parent

Figure 5. Protocol file system prior to step 17

This figure shows the file system structure necessary to run the GAMS code in step 17. This figure begins by showing the 27 text files that result from applying the convert script to the unified input. Note that all of these files are required, save log.txt which gives run information related to the convert script which is not use by subsequent steps. In the bottom left is three files which may need to be downloaded from GitHub, including the combined

Figure 5. *Continued*
EuGeneCiD and EuGeneCiM workflow, the EuGeneCiM stand-alone code (optional), and the submit file (for batch runs). All of these files are to be placed inside the ''descriptor'' folder, which itself is contained inside the parent folder. Also contained in the parent folder are the required solver options file and the optimal parallel submit script (used in the parallelization section). Shown in bold are names given to the files or folder throughout the protocol, and shown in narrow font are file names as might appear in the user's file system.

time. Therefore, it is highly suggested that followers of this protocol have access to some advanced computing resource.

Alternatives: For running all codes that do not use the Generalized Algebraic Modeling System (GAMS) programming language, any reasonably up-to-date computer may be used whether desktop or laptop with any operating system - Windows, Mac OS, or Unix/Linux. The ''before you begin'' section details some software tools which may be used, and indeed are used by other members of the research laboratory to which the authors belong.

For running GAMS software, access to supercomputing resources is preferable for considerations of time. High research institutions (R1 and R2 institutions by the Carnegie Classification of Institutions of Higher Education) often have either supercomputing facilities of their own or collaborations with other institutions which give them access to supercomputing facilities. For those without such resources, the United State of America, specifically the National Aeronautics and Space Administration (NASA) and the Department of Energy (DOE) have supercomputing facilities which may be used by researchers. The former has several CPUs as part of their High-End Computing Capability (HECC) resource. Use of these resources can be requested by researchers by submitting a request. DOE supercomputing resources are through the supercomputing facilities at the Oak Ridge National Laboratory (ORNL), allows for researchers to apply to use their available supercomputing resources such as Summit, Rhea, and HPSS. ORNL supercomputing resources are used by systems biology researcher nationwide, including by the U.S. Department of Energy Systems Biology Knowledgebase (KBase available at kbase.us).

STEP-BY-STEP METHOD DETAILS

Generating and preparing the necessary input files and file system structure

Timing: minutes

The EuGeneCiD and EuGeneCiM codes have required input files which need to be arranged in the correct manner in the file system from which these codes are to be run. This section details how to create the necessary input files, the requisite file system structure, and other necessary pieces to use the EuGeneCiD and EuGeneCiM tools. This step is separated from running the EuGeneCiDM code to highlight that the preparation to run can be done quickly, whereas the actual running of the code might be slow.

1. **Retrieve the input conversion code.** 27 versions of a Perl script are provided in the associated GitHub repository (located at github.com/ssbio/EuGeneCiDM/inputs) which will convert the unified input file into all the requisite input files for the unified EuGeneCiD/EuGeneCiM workflow. Download a copy of one of the Perl scripts (any one will do). Steps from this point will assume that the selected script is named ''convert_input_Cd_nimply_Cu.pl'' (though any of the other convert files would work equally well).
2. **Place file in the required relationship.** The unified input file and the convert script should be placed in the same directory location.
3. **Update the unified input reference in the convert script.** The convert script reads a specific unified input file which is distinguished by the synthetic biology concept or database. This reference must be updated to point to your desired unified input. On line 19 of the convert script, replace the text in quotes with whatever ''descriptor'' was applied to the naming of the unified input in step 3.

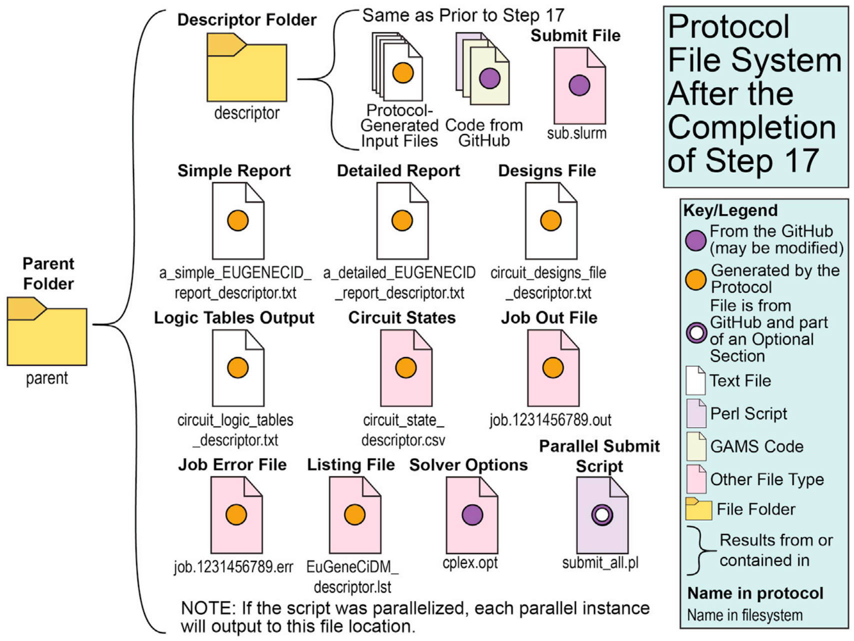

Protocol File System After the Completion of Step 17
Descriptor Folder
descriptor
Same as Prior to Step 17
Protocol-Generated Input Files
Code from GitHub
Submit File
sub.slurm
Parent Folder
parent
Simple Report
a_simple_EUGENECID_report_descriptor.txt
Detailed Report
a_detailed_EUGENECID_report_descriptor.txt
Designs File
circuit_designs_file_descriptor.txt
Logic Tables Output
circuit_logic_tables_descriptor.txt
Circuit States
circuit_state_descriptor.csv
Job Out File
job.1231456789.out
Job Error File
job.1231456789.err
Listing File
EuGeneCiDM_descriptor.lst
Solver Options
cplex.opt
Parallel Submit Script
submit_all.pl
NOTE: If the script was parallelized, each parallel instance will output to this file location.
Key/Legend
From the GitHub (may be modified)
Generated by the Protocol
File is from GitHub and part of an Optional Section
Text File
Perl Script
GAMS Code
Other File Type
File Folder
Results from or contained in
Name in protocol
Name in filesystem

Figure 6. Protocol file system after step 17

This figure shows the same file system as in Figure 4, but after step 17 (where the combined EuGeneCiD and EuGeneCiM code is run), highlighting where workflow results might be found. While the descriptor folder remains unchanged, now created will be several output files from the workflow which will appear in the parent director. This includes the simple report, detailed report, designs file, logic table output, circuit states, job out file, job error file, and listing file. As with Figure 3, shown in bold are names given to the files or folder throughout the protocol, and shown in narrow font are file names as might appear in the user's file system.

a. **Update script name.** Update the name of the convert script from "convert_inputs_Cd_nimply_Cu.pl" to "convert_inputs_descriptor.pl" to avoid confusion.
b. **Update comments.** If versioning of the code is important to the reader, lines 3 and 4 should be updated to indicate who last edited the convert script and on what date, respectively.

4. **Run the code to convert the Excel version of the input.** The convert code may then be run in one of two ways.
   a. **Run as an executable file.** The convert script may be run as an executable (for instance, double-clicking on the file in Windows) which will quickly run the code. The disadvantage to this approach is that the executable window will not remain open, so errors will be hard to identify and debug.
   b. **Run through the command line.** In the command prompt (Windows) or terminal (Mac OS or Linux) navigate to the file system that contains both the unified input and convert script, and use the command "perl convert_input_descriptor.pl" to run the script and retain the command line output.
   c. **Run result.** In either method, this should run very quickly, and create a directory named "descriptor_inputs" which should contain all 26 input files necessary for the combined EuGeneCiD/EuGeneCiM workflow, as well as a file named "log.txt" which describes certain statistics of the database and logic table of the unified input file. This log file can be useful for debugging later, see the troubleshooting 3. Figure 5 shows all 27 filenames and types which result from applying the convert script to the unified input.

Note: When running the convert script, multiple warnings will appear on the command line such as "Smartmatch is experimental at <pathway> line <number>". This is normal and is to be expected. The experimental nature of Smartmatch has not, as of yet, caused any issues in code written by the authors which use this tool.

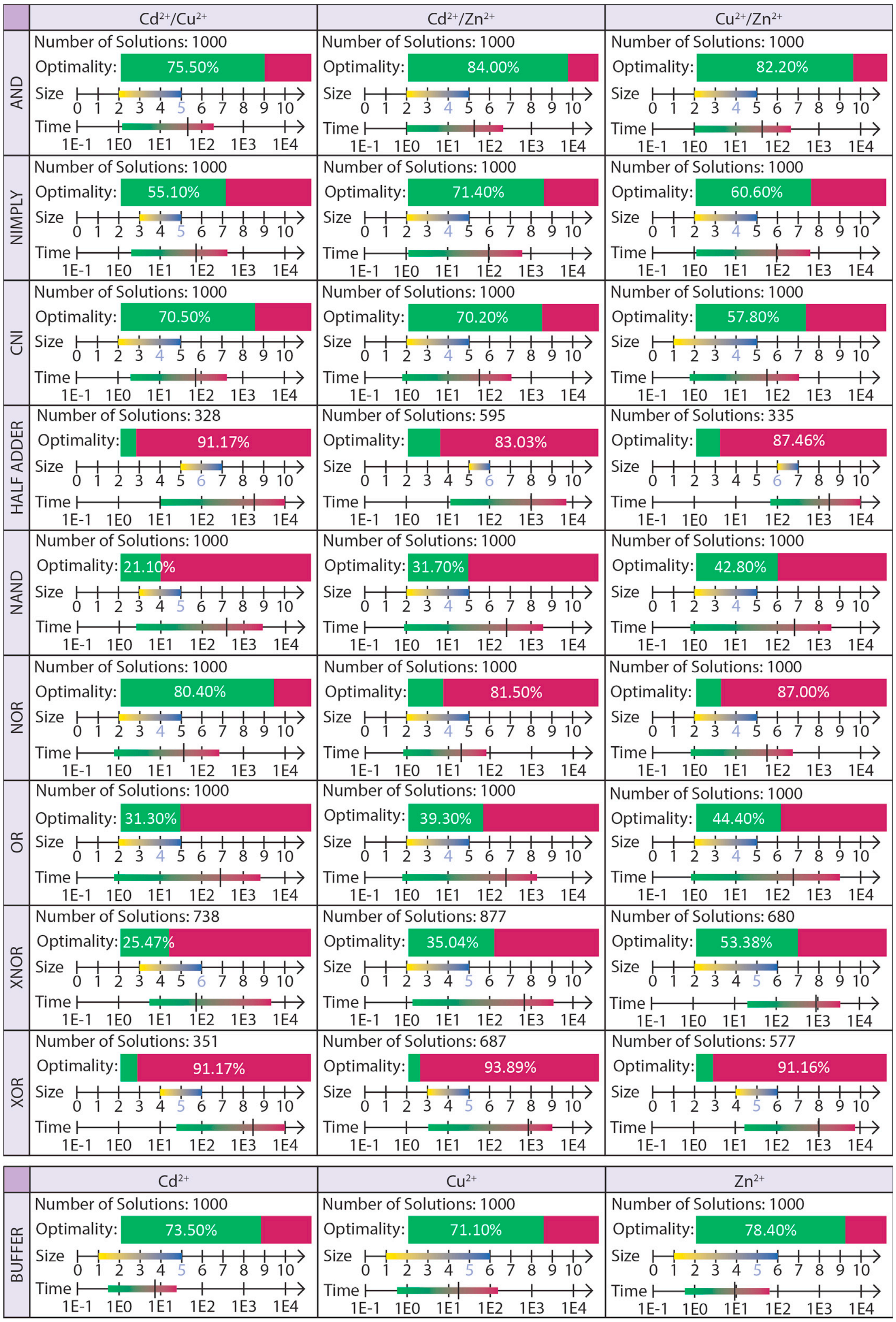

Cd²⁺/Cu²⁺
Cd²⁺/Zn²⁺
Cu²⁺/Zn²⁺
AND
Number of Solutions: 1000
Optimality: 75.50%
Number of Solutions: 1000
Optimality: 84.00%
Number of Solutions: 1000
Optimality: 82.20%
NIMPLY
Number of Solutions: 1000
Optimality: 55.10%
Number of Solutions: 1000
Optimality: 71.40%
Number of Solutions: 1000
Optimality: 60.60%
CNI
Number of Solutions: 1000
Optimality: 70.50%
Number of Solutions: 1000
Optimality: 70.20%
Number of Solutions: 1000
Optimality: 57.80%
HALF ADDER
Number of Solutions: 328
Optimality: 91.17%
Number of Solutions: 595
Optimality: 83.03%
Number of Solutions: 335
Optimality: 87.46%
NAND
Number of Solutions: 1000
Optimality: 21.10%
Number of Solutions: 1000
Optimality: 31.70%
Number of Solutions: 1000
Optimality: 42.80%
NOR
Number of Solutions: 1000
Optimality: 80.40%
Number of Solutions: 1000
Optimality: 81.50%
Number of Solutions: 1000
Optimality: 87.00%
OR
Number of Solutions: 1000
Optimality: 31.30%
Number of Solutions: 1000
Optimality: 39.30%
Number of Solutions: 1000
Optimality: 44.40%
XNOR
Number of Solutions: 738
Optimality: 25.47%
Number of Solutions: 877
Optimality: 35.04%
Number of Solutions: 680
Optimality: 53.38%
XOR
Number of Solutions: 351
Optimality: 91.17%
Number of Solutions: 687
Optimality: 93.89%
Number of Solutions: 577
Optimality: 91.16%
Cd²⁺
Cu²⁺
Zn²⁺
BUFFER
Number of Solutions: 1000
Optimality: 73.50%
Number of Solutions: 1000
Optimality: 71.10%
Number of Solutions: 1000
Optimality: 78.40%
Size
0 1 2 3 4 5 6 7 8 9 10
Time
1E-1 1E0 1E1 1E2 1E3 1E4

Figure 7. Holistic solution statistics for EuGeneCiD applied to 30 conceptualizations

This three by ten grid reports on the general characteristics of the set of EuGeneCiD results for each of the 30 unique circuit conceptualizations. From the top to bottom of each grid, four items describing the results set are shown. First, is the number of solutions in that set. Second, is the percentage of results which are optimal (if this value is above 20%, green bar) or the percentage of results that are suboptimal (red, if the value of this is above 80%). Third is a number line, which indicates the solution set minimum and maximum sizes (in the number of triads in the design) and the mode size (the number is shaded blue). This number line is extended from zero to ten as ten is the maximum allowed circuit size (though no solution was created of this size). Finally, another number line shows the minimum and maximum solution times (in seconds) on a logarithmic scale. A large black line on the solution time range indicates the mean solution time.

5. **Download additional necessary files from GitHub.** Three files need be downloaded from GitHub to perform the combined EuGeneCiD and EuGeneCiM workflow. These files have multiple copies in the GitHub repository, but the files associated with the Cadmium NIMPLY Copper (descriptor of "Cd_nimply_Cu") conceptualization from Schroeder, Baber, and Saha (2021) will be used here (although any of the other 26 conceptualization will work equally as well). In this example, these files can be found in the "Cd_nimply_Cu" folder.
   a. **CPLEX option file.** The first file needed is named "cplex.opt" which specifies the options to use for the CPLEX solver.
   b. **GAMS code file.** The next file need is the EuGeneCiD and EuGeneCiM combined workflow script file, named "EuGeneCiDM_Cd_nimply_Cu.gms".
   c. **Batch run file.** Required next is the file which allows us to submit the combined workflow in a batch run, named "sub.slurm". This file may not be strictly necessary depending on the procedure used by the HPC resource available to the user, please consult the documentation of that resource's documentations for details. These files are shown in the bottom half of Figure 5.
6. **Creating the requisite file system structure.** The input files generated in step 4 and those downloaded in step 5 should next be moved into the file system where EuGeneCiD and EuGeneCiM will be performed. Figure 5 provides a visualization of the required file system which will be described here.
   a. **Child and parent directories.** All these files (29 in total, 30 if including the log file) should be placed in a single sub-folder where the user has editing permissions for its parent folder (output will be placed in the parent folder, as shown in Figure 6, and this file system structure facilitates parallelization, see "Parallelizing the EuGeneCiD and EugeneCiM Workflow"). Name the child folder "descriptor" (that is whatever descriptor you used in step 3).
   b. **File transfer protocols.** It is highly recommended that, if using a High Performance Computing (HPC) resource, an FTP or data management system other than nppFTP is used, as nppFTP transfers files singly. Various tools, mentioned in "before you begin", are available for this task, including Cyberduck and the Globus research data management system.

   Note: In future steps, for convenience, the folder in which the files generated in step 12 and downloaded in step 13 are contained will be referred to as the "descriptor folder", whereas the folder in which it is contained will be referred to as the "parent folder".

   Note: Depending on the HPC resource used, there may be limits as to from which file system a batch job may be run. Consult the documentation of the resource which you are using and place the parent directory appropriately.

7. **Modify the files from GitHub.** The files downloaded in step 5 will have to be modified for the user's application and conceptualization.
   a. **Modify the GAMS code.** Several edits to the GAMS code should or can be made to both point to the correct input files and customize restrictions placed on the runs.
      i. **Update file name.** Begin by renaming the "EuGeneCiDM_Cd_nimply_Cu.gms" file to "EuGeneCiDM_descriptor.gms".

   ii. **Update internal reference to other files.** In the file renamed in step 7.a.i, edit the reference to the descriptor folder and the input files. This is most easily accomplished by performing a find-and-replace operation for the strings "Cd_nimply_Cu" and replacing all instances with "descriptor" (check, there should be 57 replacements made).
   iii. **Update internal documentation.** If versioning and author tracking is important, please updates lines 22 (code authors) and 24 (date).
   iv. **Define the number of solutions sought.** To adjust the maximum number of solutions sought, adjust the value of the parameter "max_solns" defined on line 184 of the file "EuGeneCiDM_descriptor.gms". The value is currently set to 1000, but as noted in Schroeder, Baber, and Saha (2021), for most applications this value will likely not need to be greater than 100 as the globally optimal solution (that is the solution with the strongest response, irrespective of the circuit size) generally is found within the first 100 solution.
   v. **Define the maximum number of repeated parts usages in a single solution.** Parameters are defined on lines 199 to 201 of the file "EuGeneCiDM_descriptor.gms" which restrict the maximum number of times are part can be used in a given circuit. These lines restrict the maximum number of times a particular promoter, transcript, or terminator respectively, can appear in any solution. At present, these are set at the default values of 3, 3, and 100, respectively, though the user may define these to whatever meets their design criteria. Note that if the number of allowed repeats is greater than the number of allowed triads in the solution, then there is effectively no limit (as in the case with the default terminator repetition limit, for example).
   vi. **Define the minimum and maximum allowed design sizes.** Lines 204 and 205 of "EuGeneCiDM_descriptor.gms" define the minimum and maximum allowed design sizes, respectively. The default values are 1 and 10, respectively. Depending on the complexity of the database and/or conceptualization, these may be adjusted to save time or increase the allowed solution space.

b. **Modify the SLURM batch file.** The reference in the SLURM batch file ("sub.slurm") must be updated to point to the correct file. Line 9 should be rewritten to read "gams descriptor/EuGeneCiDM_descriptor.gms". This file also defines various settings of the batch run including memory limit (line 4, given in MBs), and time limit (line 5, given in hours:minutes:-seconds). Please adjust these lines accordingly for the HPC resource used, expected memory requirements, and for expected runtime. For the latter, example runtimes are provided in Figure 7 and S1, though depending on the resources used, bioparts database, and sought logic, runtime may vary considerably.

c. **Modify the solver options file.** The solver options file ("cplex.opt") can be adjusted to result in differences in solution accuracy, speed, and feasibility. See GAMS documentation for CPLEX options to learn more about what each specified option does (https://www.gams.com/35/docs/S_CPLEX.html). Due to potential numerical issues, there is a required relationship between the values of one CPLEX option and arbitrary values used in the EuGeneCiD and EuGeneCiM formulations (e.g., our very small and very large values). See troubleshooting 4 for details.

⚠ CRITICAL: It should be noted that, due to the use of an arbitrary small number to force some infeasibilities, feasibility threshold options, such as "epint" or "eprhs" should not be increased to that number or greater, as this will result in nonsense solutions and render the EuGeneCiD and EuGeneCiM tools useless. The feasibility thresholds, such as "epint" and "eprhs" are options which allow the solver to use an infeasible solution so long as that infeasibility falls within certain bounds. For "epint", binary and integer variables are allowed to vary from true binary or integer values by some small amount, whereas "eprhs" allows a constraint equation to be infeasible by some small amount. See troubleshooting 4 for details.

(If Required) Adjusting EuGeneCiD and EuGeneCiM for other than two input signals

⏲ Timing: minutes to hours

The EuGeneCiD and EuGeneCiM codes made available are for two input signals and any number of output signals. Obviously, circuits may be conceptualized which use other than two signals. These conceptualizations can be designed for and modeled by EuGeneCiD and EuGeneCiM so long as the mathematics and GAMS code is carefully updated. Detailed in this section are the steps to generalize the mathematics and code of EuGeneCiD and EuGeneCiM to accommodate fewer or more input signals for a desired conceptualization.

8. **Change the number of ligand set copies.** There should be one set of all ligands and one set of desired ligands (the signals to which the circuit will respond) for each element of the desired ligands set. The latter set is defined in the logic table, see before you begin step 2. In the example code, the sets of all ligands are labeled "L1" and "L2", where the sets of desired ligands are labeled "Ld" and "Ld1". Duplicate (or remove) lines 79 through 82 to achieve the requisite number of set copies with unique handles.

9. **Update references to ligand sets in parameter, variable, and equation argument.** For many parameters, variables, and equations, the two-ligands input is hardcoded, and will need to be changed. The following sub steps describe changes to the "EuGeneCiDM_descriptor.gms" code which need be made to change the number of signal inputs for which EuGeneCiD and EuGeneCiM design or model, respectively.

   ⚠ CRITICAL: Order of arguments is important. When updating references in parameters, variables, or equations, the order of arguments must stay constant. One easy way to ensure this is to use find and replace tools inherent in the user's text editor of choice.

   a. **Parameters to update.** Below is a list of parameters (in GAMS, this is the object type which does not change value when solving an optimization problem) whose arguments must be updated to the appropriate number of signals. They are written below as they appear in "EuGeneCiDM_descriptor.gms". The two-input variation of the arguments is given on the LHS below, followed an example of how these arguments might appear in a three-input system, followed by the number of instances which should be replaced.

   Note: Sets with labels beginning with "Ld" (and similar such as Ld1) are the desired ligands set, which itself is a subset of the sets of all ligands (L1, L2, and similar).

   Note: Due to the number of references within the code, and that this procedure will alter line numbers in later steps, an exhaustive list of line numbers to change will not be included. Instead, the number of instances of this particular parameter in "EuGeneCiDM_descriptor.gms" will be listed alongside as square brackets. Also note that the number of instances may change with further versioning of the EuGeneCiD/EuGeneCiM algorithm and combined workflow, but this should help users determine if the find-and-replace tools are operating on approximately the correct number of instances. The specified changes are shown in Table 1 below find-and-replace instances.

   b. **Variables to update.** Below is a list of variables (in GAMS, this is the object type which changes value when solving and optimization problem) whose arguments must be updated to the appropriate number of signals. They are written below as they appear in "EuGeneCiDM_descriptor.gms". This section is formatted the same as step 11a.

Table 1. Find-and-Replace type changes to make to EuGeneCiDM_descriptor.gms" for step 9a

Find text	Replace with	Instances
`Lambda(Ld,Ld1,Ed)`	`Lambda(Ld,Ld1,Ld2,Ed)`	5
`ECTrans(E,L1,L2)`	`ECTrans(E,L1,L2,L3)`	13
`RNATrans(J,T,L1,L2)`	`RNATrans(J,T,L1,L2,L3)`	10

Note: When variables are instructed to be written in input files, generally they must be reported using an attribute label. For instance, when the value of "`alpha(P,L1,L2)`" in a particular solution is to be reported, it is written as "`alpha.l(P,L1,L2)`". Thus, for the required changes are shown in Table 2 below, along with the number of direct find-and-replace instances (unattributed instances), the number of instances where an attributed variable reference must also be updated is also given (attributed instances).

c. **Equations to update.** Below is a list of equations (the GAMS object type which enforces optimization constraints) whose arguments must be updated to the appropriate number of signals. They are written below as they appear in "EuGeneCiDM_descriptor.gms", followed by how they might be modified in a three-input case and the number of replacement instances. This section is formatted the same as step 9a. Table 3 below shows the find-and-replace type changes needed, and the number of instances which need be replaced.

10. **Update nested loops over desired ligand elements.** Throughout the "EuGeneCiDM_descriptor.gms" code are several instances where looping occurs over each element of the desired ligands set at each argument position for the desired ligands. When changing the number of ligands then, the number of these nested loops must be appropriately adjusted. The easiest method by which to find these nested loops is to search for the string "`LOOP(Ld1`" (this will find the innermost nested loop). The number of nested loops should be made equal to the number of elements in the desired ligand sets (e.g., if it is desired to have a three-input responsive circuit, then there should be three controlling loops over the desired ligands set). An example is given below on what the conversion would look like for the nested loops converting from the two-input signal GAMS code to the one-input and three-input versions.

Code format in the two-input signal GAMS code (original):

```
>LOOP(Ld,
>
>    LOOP(Ld1,
>
>          /*code*/
>
>    );
>
>);
```

Table 2. Find-and-Replace type changes to make to EuGeneCiDM_descriptor.gms" for step 9b

Find text	Replace with	Unattributed instances	Attributed instances
`alpha(P,L1,L2)`	`alpha(P,L1,L2,L3)`	4	2
`alpha_plus(P,L1,L2)`	`alpha_plus(P,L1,L2,L3)`	7	2
`gamma(E,L1,L2)`	`gamma(E,L1,L2,L3)`	4	2
`gamma_plus(E,L1,L2)`	`gamma_plus(E,L1,L2,L3)`	6	1
`C(E,L1,L2)`	`C(E,L1,L2,L3)`	11	8
`phi(J,T,L1,L2)`	`phi(J,T,L1,L2,L3)`	4	5
`xi(P,J,T,L1,L2)`	`xi(P,J,T,L1,L2,L3)`	5	6
`omega(E,L1,L2)`	`omega(E,L1,L2,L3)`	8	2
`kappa(E,L1,L2)`	`kappa(E,L1,L2,L3)`	7	2

Code format in the one-input signal GAMS code (modified):

```
>LOOP(Ld,

>

>    /*code*/

>

>);
```

Code format in the three-input signal GAMS code (modified):

```
>LOOP(Ld,

>

>     LOOP(Ld1,

>

>           LOOP(Ld2,

>

>                 /*code*/

>

>           );

>

>     );

>

>);
```

11. **Update mathematics related to ligand interactions.** Some equations in EuGeneCiD and EuGeneCiM are written in a way unique to the number of input ligands, and therefore need to be adjusted to apply EuGeneCiD and EuGeneCiM to circuits with other than two inputs. This is done by increasing or decreasing the number of terms which account for the individual effects of ligands upon the activity of a promotor or enzyme in question and which prevent "double counting" of those effects. The number of such terms is dependent upon the number of elements in the desired ligands sets. In an effort to help the reader understand this statement, an example is provided in the note below.

Table 3. Find-and-Replace type changes to make to EuGeneCiDM_descriptor.gms" for step 9c

Find Text	Replace with	Instances
act_threshold_0(E,Ld,Ld1)	act_threshold_0(E,Ld,Ld1,Ld2)	2
act_threshold_1(E,Ld,Ld1)	act_threshold_1(E,Ld,Ld1,Ld2)	2
prot_sum(E,Ld,Ld1)	prot_sum(E,Ld,Ld1,Ld2)	2
trans_sum(J,T,Ld,Ld1)	trans_sum(J,T,Ld,Ld1,Ld2)	2
findalpha(P,Ld,Ld1)	findalpha(P,Ld,Ld1,Ld2)	2
findalpha_sign_0(P,Ld,Ld1)	findalpha_sign_0(P,Ld,Ld1,Ld2)	2
findalpha_sign_1(P,Ld,Ld1)	findalpha_sign_1(P,Ld,Ld1,Ld2)	2
transcribed0(P,J,T,Ld,Ld1)	transcribed0(P,J,T,Ld,Ld1,Ld2)	2
transcribed1(P,J,T,Ld,Ld1)	transcribed1(P,J,T,Ld,Ld1,Ld2)	2
transcribed2(P,J,T,Ld,Ld1)	transcribed2(P,J,T,Ld,Ld1,Ld2)	2
produced0(E,Ld,Ld1)	produced0(E,Ld,Ld1,Ld2)	2
produced1(E,Ld,Ld1)	produced1(E,Ld,Ld1,Ld2)	2
findgamma(E,Ld,Ld1)	findgamma(E,Ld,Ld1,Ld2)	2
findgamma_sign0(E,Ld,Ld1)	findgamma_sign0(E,Ld,Ld1,Ld2)	2
prot_could_act0(E,Ld,Ld1)	prot_could_act0(E,Ld,Ld1,Ld2)	2
prot_could_act1(E,Ld,Ld1)	prot_could_act1(E,Ld,Ld1,Ld2)	2
prot_could_act2(E,Ld,Ld1)	prot_could_act2(E,Ld,Ld1,Ld2)	2
prot_is_act0(E,Ld,Ld1)	prot_is_act0(E,Ld,Ld1,Ld2)	2
prot_is_act1(E,Ld,Ld1)	prot_is_act1(E,Ld,Ld1,Ld2)	2
prot_is_act2(E,Ld,Ld1)	prot_is_act2(E,Ld,Ld1,Ld2)	2
sat_lambda(Ld,Ld1,Ed)	sat_lambda(Ld,Ld1,Ed,Ld2)	2
explicit_1(E,Ld,Ld1)	explicit_1(E,Ld,Ld1,Ld2)	2
explicit_2(E,Ld,Ld1)	explicit_2(E,Ld,Ld1,Ld2)	2
explicit_3(E,Ld,Ld1)	explicit_3(E,Ld,Ld1,Ld2)	2
explicit_4(E,Ld,Ld1)	explicit_4(E,Ld,Ld1,Ld2)	2
explicit_5(E,Ld,Ld1)	explicit_5(E,Ld,Ld1,Ld2)	2
prot_conc_M(E,Ld,Ld1)	prot_conc_M(E,Ld,Ld1,Ld2)	2
trans_level_M(J,T,Ld,Ld1)	trans_level_M(J,T,Ld,Ld1,Ld2)	2
produced0_M(E,Ld,Ld1)	produced0_M(E,Ld,Ld1,Ld2)	2
Produced1_M(E,Ld,Ld1)	Produced1_M(E,Ld,Ld1,Ld2)	2
act_threshold_0_M(E,Ld,Ld1)	act_threshold_0_M(E,Ld,Ld1,Ld2)	2
act_threshold_1_M(E,Ld,Ld1)	act_threshold_1_M(E,Ld,Ld1,Ld2)	2

Note: Below is an example of one such equation which needs to be updated: "findalpha".

Equation:

```
>findalpha(P,Ld,Ld1)
```

GAMS Statement:

```
>alpha(P,Ld,Ld1) =e= Z(P) + sum(E, (W(E,Ld,Ld1) * I(P,E) * H(P,E))) + I(P,Ld) * H(P,Ld) +
I(P,Ld1) * H(P,Ld1) – (I(P,Ld1) * H(P,Ld1) * sigma(Ld,Ld1))
```

Mathematical Statement:

$$\begin{aligned}\alpha_{pL_dL_{d1}} &= Z_p + \sum_{e\in E}\left[W_{eL_dL_{d1}} I_{pe} H_{pe}\right] + I_{pL_d} H_{pL_d} \\ &\quad + I_{pL_{d1}} H_{pL_{d1}} - I_{pL_{d1}} H_{pL_{d1}} \sigma_{L_dL_{d1}} \qquad \forall p\in P;\ L_d, L_{d1}\in L^D\end{aligned} \quad \text{(Equation 1)}$$

Note that the last three terms are specific to the indices or arguments of α (both in number and identity thereof) and that necessarily the numbers of these terms will change with the number of ligands. The first two of these three terms add the contribution of each individual ligand upon the activity of the promoter under the given ligand conditions. The final term ensures that the effects of ligands are not "double counted" if the current element of L_d is the same as that of L_{d1}. The addition of the effect of each ligand to the RHS of Equation (1) and the subtraction of each unique combination of ligands should persist for a EuGeneCiD or EuGeneCiM formulation which is generalized to any number of input ligand signals. For instance, if the circuit was to respond to three input signals, then the equation "findalpha" should be updated to appear as follows:

Equation:

```
>findalpha(P,Ld,Ld1,Ld2)
```

GAMS Statement:

```
> alpha(P,Ld,Ld1) =e= Z(P) + sum(E, (W(E,Ld,Ld1) * I(P,E) * H(P,E))) + I(P,Ld) * H(P,Ld) +
I(P,Ld1) * H(P,Ld1) + I(P,Ld2) * H(P,Ld2) - (I(P,Ld1) * H(P,Ld1) * sigma(Ld,Ld1)) - (I(P,Ld1)
* H(P,Ld1) * sigma(Ld1,Ld2)) - (I(P,Ld) * H(P,Ld) * sigma(Ld,Ld2))
```

Mathematical Statement:

$$\begin{aligned}\alpha_{pL_dL_{d2}} \quad &= Z_p + \sum_{e\in E}\left[W_{eL_dL_{d1}} I_{pe} H_{pe}\right] + I_{pL_d} H_{pL_d} \\ &+ I_{pL_{d1}} H_{pL_{d1}} + I_{pL_{d2}} H_{pL_{d2}} - I_{pL_d} H_{pL_d} \sigma_{L_dL_{d1}} \\ &- I_{pL_{d1}} H_{pL_{d1}} \sigma_{L_{d1}L_{d2}} - I_{pL_d} H_{pL_d} \sigma_{L_dL_{d2}}\end{aligned} \qquad \forall p\in P;\, L_d, L_{d1}\in L^D \qquad \text{(Equation 2)}$$

This same mathematical update should be applied to each of the following equations:

```
>findalpha(P,Ld,Ld1,Ld2)

>findgamma(E,Ld,Ld1,Ld2)
```

(Optional) Parallelizing the EuGeneCiD and EuGeneCiM workflow

Timing: Minutes to Hours

The EugeneCiD and EuGeneCiM workflow can be easily parallelized for designing and modeling multiple circuit conceptualizations, particularly if the same bioparts database is to be used. In fact, this is the reason for a handful of oddities in the file structure and instructions, such as those specified in step 6. Detailed in this section are the additional steps which must be taken to run parallel instances of combined EuGeneCiD and EuGeneCiM workflow simultaneously. This procedure was used in Schroeder, Baber, and Saha (2021) to run batches for all 30 circuit conceptualizations in parallel.

12. **Build a new unified input file and file system.** Depending on how different the circuit conceptualizations are, there are two different approaches to the first few steps of parallelizing the EuGeneCiD and EuGeneCiM workflow. If the same bioparts library or the same conceptualization is to be used between the two conceptualizations, then great savings in time and effort are possible. Given that "descriptor" has been used up to this point to describe the user's first

conceptualization, "descriptor_1", "descriptor_2", etc. will be used for further conceptualizations in an effort to avoid confusion.

a. **Different library and conceptualization.** If both the database of bioparts and the conceptualization are different, the protocol must be largely repeated up to this point.
   i. **Create the new unified input.** If the application is to use different bioparts, then before you begin steps 1–9 of should be repeated for the new conceptualization and bioparts library.
   ii. **Create the necessary files and file system.** Similarly, steps 1–11 should be repeated (as necessary for the optional sections) to create all necessary files and filesystem using "descriptor_1" in place of "descriptor".

Note: If both the bioparts library and conceptualization are different, little in the way of time savings will be noticed for the parallelization procedure.

b. **Different conceptualization, same library.** If this is the case, follow this modified procedure which assumes all previous steps have been completed:
   i. **Make the new unified input.** The unified input for the first conceptualization ("descriptor") can be copied and renamed to "unified_input_descriptor_1.xlsx". Update the "Ligands", "LogicTable", and "other" file with the new conceptualization.
   ii. **Make the new convert code.** Copy the convert script ("convert_input_descriptor.pl") to the file "convert_input_descriptor_1.pl". Following step 3, edit the file references in the code.
   iii. **Run the new convert code.** Run the convert script for "descriptor_1" as described in step 4.
   iv. **Copy the "descriptor folder".** Copy the "descriptor" folder inside the parent folder, and rename the result to "descriptor_1".
   v. **Remove unnecessary files.** Remove all files except the EuGeneCiD/M code, the EuGeneCiD stand-alone code (if present), and the submit file.
   vi. **Place the new input files.** Copy and paste the files generated in step 12.b.iii. into "descriptor_1".
   vii. **Update file pointers.** Following the procedure outlined in step 7, update the file names and references in the EuGeneCiD/M code.

Note: If "descriptor" and "descriptor_1" have the same number of input signals for the circuit to respond to, then steps 8–11 do not need to be repeated.

c. **Different library, same conceptualization.** This instance can follow the same procedure as step 12.b., save for step 12.b.i., which would involve editing the "Promoters", "Transcripts", "Enzymes&Protiens", and "Terminators" sheets of the unified input instead of those related to the conceptualization.

13. **Repeat step 12 for each parallel instance desired.** Repeat step 12 for each parallel instance of the EuGeneCiD and EuGeneCiM combined workflow which the user desires to create for "descriptor_2" through "descriptor_n" (using whichever descriptor text is appropriate).
14. **Get the parallel submit script file.** By this step, the requisite files and file systems should be in place for running parallel instances of the combined EuGeneCiD and EuGeneCiM workflow. From the GitHub (at http://github.com/ssbio/EuGeneCiDM) copy the file "submit_all.pl" into the parent folder (see Figure 5).
15. **Update the parallel submit script file.** The parallel submit script should be updated to point to each submit file of each parallel EuGeneCiD/EuGeneCiM instance. Begin by deleting all lines from line 8 to the end of the file. Then, write the appropriate lines to submit batch runs for all

parallel runs. This can be done by creating two lines for each descriptor (“descriptor_1” is used as an example below):

```
>printf ''descriptor_1: '';

>system(''sbatch descriptor_1/sub.slurm'');
```

The first line will print “descriptor_1” to the terminal so that when the system produces a response to the command to submit a batch for that conceptualization, that output is labeled. The second line submits the actual batch run using the SLURM job management system, as used by the authors.

Note: The parallel submit script may need to be structured or written differently depending on the job management system used by the user’s choice of HPC resource(s).

Running the EuGeneCiD and EuGeneCiM combined workflow

Timing: minutes to days

This section specifies how to run the combined EuGeneCiD/EuGeneCiM workflow using a SLURM batch management system as the authors have used for Schroeder, Baber, and Saha (2021).

16. **Running from the correct file position.** On the command line interface, navigate to the parent folder which contains the subfolder. The batch files will be run from this location, as it helps in parallelization if you wish to design for multiple conceptualizations simultaneously (see “Parallelizing the EuGeneCiD andEuGeneCiM Workflow”).
17. **Submit the batch run.** The submission procedure for the batch runs of the combined workflow are different for single runs and parallelized runs, as will be discussed in the following sub steps.
    a. **Single run.** Submit the batch run with the command “sbatch descriptor/sub.slurm” (if not parallelizing runs).
    b. **Parallelized runs.** If parallelizing runs, submit all batch runs with the command “perl submit_all.pl” (copied in step 14 and edited in step 15).
    c. **Allow time for the runs to complete.** Provided that the batch run (and parallel submit script, if needed) has been set up correctly, results will be available in as little as a few minutes, or as long as several days (the upper limit of the time allowed for the job is defined in step 7b). The speed of solution will depend on several factors including solver options (step 7c), parameterization of the solution space (step 7a), bioparts library (“before you begin” section), and power of the HPC resource used. Figures 7 and S1 shows statistics on solution number, speed, size, and optimality for the application of the EuGeneCiD and EuGeneCiM workflow to 30 unique conceptualizations.

Note: Generally, there will be some method, often command line, which will allow for checking on the status of the run. For this, the authors used the “squeue” command.

(Optional) Running only EuGeneCiM for a pre-defined circuit design

Timing: seconds to minutes

Sometimes, a particularly interesting solution will be put forth by EuGeneCiD which the user may wish to model out to a greater number of time points in order to better study its long-term behavior, or perhaps it is desired to model a dynamic circuit which EuGeneCiD could not have designed in the first place. For instance, the section “Adjusting EuGeneCiD and EuGeneCiM for Other than Two Inputs” and this one are combined in Schroeder, Baber, and Saha (2021) to model a repressilator circuit using only EuGeneCiM (which responds to the single ligand “none” which is a placeholder having no interactions with any biopart). Detailed here are the steps to run EuGeneCiM independently to model the behavior of a pre-defined circuit.

18. **Download, place, and rename the stand-alone EuGeneCiM code.** Stand-alone EuGeneCiM code can be downloaded from the GitHub repository associated with this work at https://github.com/ssbio/EuGeneCiDM/Cd_nimply_Cu/.
    a. **Download the file.** The filename will be "EuGeneCiM_only_Cd_nimply_Cu.gms" (note that as with other steps, other copies of this code exist in the repository, the "Cd_nimply_Cu" case is chosen for consistency and convenience).
    b. **Place the file.** This file should be placed in the same directory as the "EuGeneCiDM_descriptor.gms".
    c. **Rename the file.** The file should be renamed to "EuGeneCiM_only_descriptor.gms" (where "descriptor" is the users description used elsewhere in this protocol).
19. **Modifying the GAMS code for the user's input files and file system.** File references should be updated in this code in the same manner as detailed in step 7a. The only substantive difference is that, in step 7.a.i., instead of 57 replacements, 54 replacements should be made instead.
20. **Provide the GAMS code with the pre-defined circuit.** Since EugeneCiM is a modeling tool, the circuit design must be provided to it in the form of design triads. This involves defining the parameter "`M_design(P,J,T)`" as 1 for each triad of a promoter, transcript, terminator. An example is provided for a three triad design between lines 240 and 255 of the file which should now be named "EuGeneCiM_only_descriptor.gms". Another example can be found in the file "EuGeneCiM_rep.gms" at https://github.com/ssbio/EuGeneCiDM/Cd_nimply_Cu/. Following these examples, define the circuit design to be modeled.
21. **Set the number of time points to model.** If using this part of the protocol, it is often desired to model the given circuit for a longer period of time than done in a previous run of the combined EuGeneCiD and EuGeneCiM workflow. Edit the file "time_set_descriptor.txt" in the subfolder "descriptor" (which should be in the same folder as "EuGeneCiM_only_descriptor.gms") to reflect the time points desired to be modeled.

Note: The file "time_set_descriptor.txt" will accept any globally unique string as its set elements; however, for automatic plotting of results, these strings should be evenly spaced numbers beginning at time point zero (otherwise the plotting will not work well). Generally, this will be whole numbers (e.g., 0, 1, 2, etc.). Therefore, for instance, for modeling 100 time points, from 0 to 99 in intervals of 1, 100 unique elements must be defined like '0', '1', etc. up to '99'.

22. **Run from the correct file position.** On the command line interface, navigate to the parent folder which contains the subfolder. The batch files will be run from this location, as it keeps the workflow consistent with the combined EuGeneCiD/EuGeneCiM workflow.
23. **Run the stand-alone EuGeneCiM.** As a stand-alone tool, EuGeneCiM is very quick to run, and therefore submitting a batch run is unlikely to be necessary; therefore, simply run EuGeneCiM with the command "gams descriptor/EuGeneCiM_only_descriptor.gms". EuGeneCiM only takes a few seconds to run for as many as 500 time points (note that it will be quick for larger sets of time points as well, this is simply the largest set so far attempted to be modeled by the authors).

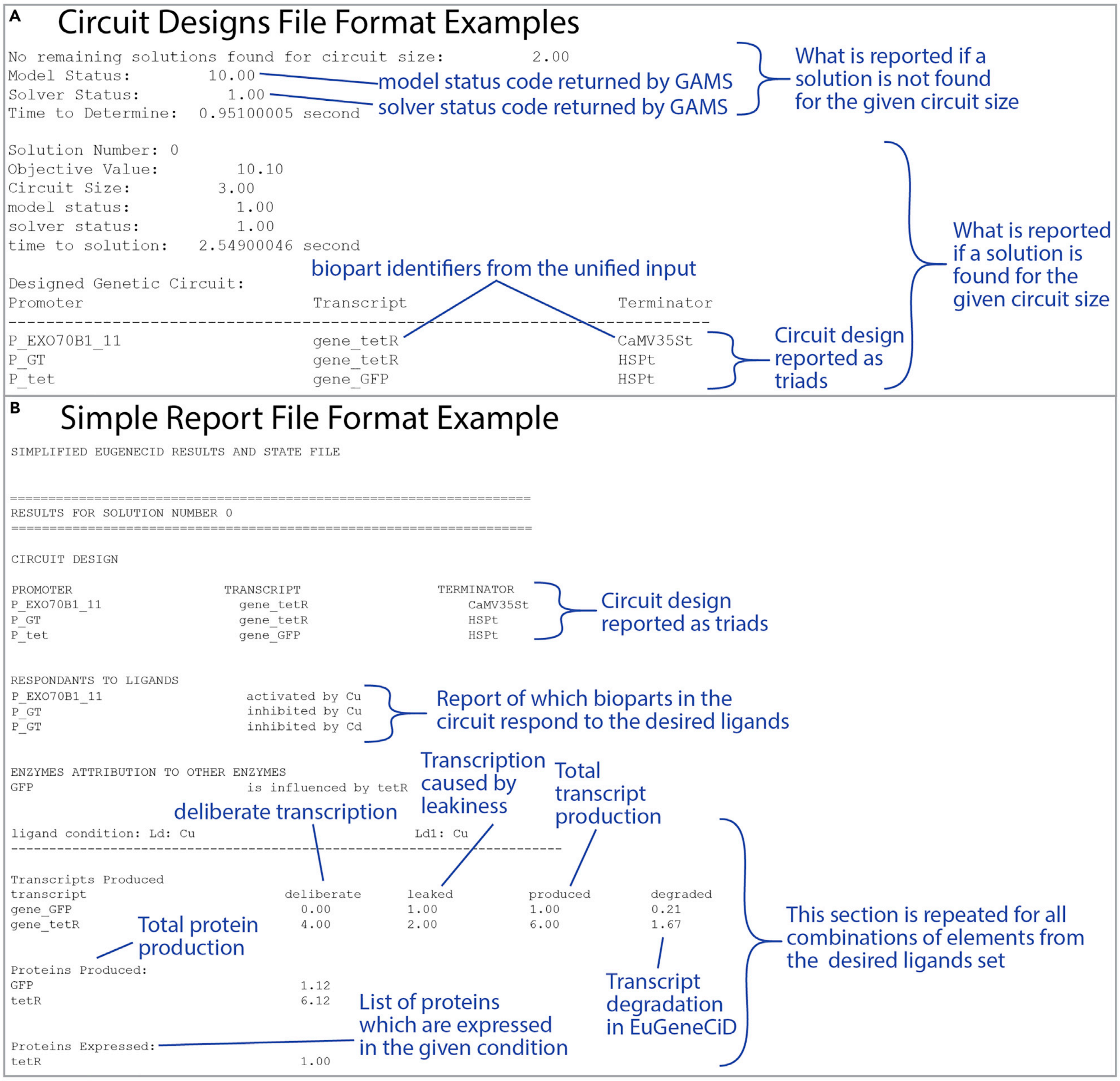
Reading results and processing results into more human-readable formats

Timing: seconds to minutes

The output files which are produced by the combined EuGeneCiD and EuGeneCiM workflow are designed to some extent to be human-readable with the limited text-formatting abilities of GAMS. The first steps of this section (steps 24–26) will detail how to interpret some raw output files. Other files, particularly for hundreds of solutions, become too unwieldy for humans to read or process the amount of data. These files generally either exist for the purposes of troubleshooting or for processing by certain Perl scripts into more human-friendly formats. The latter steps of this section (steps 27 and 28) will focus on the use of Perl scripts which help summarize or convert certain results to make them more human-readable or user friendly.

24. **Read the circuit design file.** The circuit design file is one of several outputs used to report the results of the EuGeneCiD and EuGeneCiM workflow which is focused on reporting the results of EuGeneCiD, and will be named "circuit_designs_file_descriptor.txt". Two portions of an example circuit design file (results are for a Cadmium NIMPLY Copper circuit) are shown in Figure 8A.
    a. **When no solution is found for a given circuit size.** The first portion (the first four lines) show what is written if no further solutions can be found at the given time point. Reported here is the failure (line 1), model status (line 2), solver status (line 3), and time it took for the solver to decide that there was no further solutions (line 4). Model and solver status code meaning can be found in GAMS documentation, though the authors have found the succinct summary at www.gamsworld.org/performance/status_codes.htm a useful reference.
    b. **When a solution is found for the given circuit size.** The remaining lines of output shown in Figure 8A are the output to this file when a solution is found. Several solution metrics are reported here including objective value, circuit size, model status, solver status, and time to find the solution. Below this block of text is a table representing the designed circuit, with each row being one triad of the design (promoter, transcript, and terminator grouping).
25. **Read the simple report.** The simple report is another of several outputs used to report the results of the combined EuGeneCiD and EuGeneCiM workflow which is focused on reporting the results of EuGeneCiD, and will be named "a_simple_EUGENECID_report_descriptor.txt". A portion of a simple report for a single solution is shown in Figure 8B.
    a. **Circuit design report.** Each simple report begins with a header indicating the solution number, followed by the triads of the circuit design (with each row representing one triad).
    b. **Ligand responsiveness report.** The design report is followed by a "respondents to ligands" table, which specifies which bioparts respond to the desired ligands (ligands will be reported using their identifiers from the unified input).
    c. **Enzyme responsiveness report.** Similarly, a table of which enzymes are affected by which other enzymes (as determined through the attribution equations) is made.
    d. **Ligand condition reports.** Following the three previously described reports, each possible combination of desired ligands is listed with three tables reporting on the EuGeneCiD results.
        i. **Transcript production table.** The first table reports transcript production and breaks down production into those that are deliberate and those caused by expression leakiness, in addition to reporting the amount predicted to be degraded.
        ii. **Protein production table.** The next table lists the total protein production. A report is only made of a protein here if the protein concentration is non-zero.
        iii. **Protein expression table.** The final table lists all proteins which are expressed and the value of the expression binary variable ($W_{EL_d L_{d1}}$). If a protein fails to be expressed (by being inhibited or not meeting the concentration threshold), it will not be reported in this table.
26. **Read the logic tables output.** The logic table file is another of several outputs used to report the results of the EuGeneCiD and EuGeneCiM workflow which is focused on reporting the results of EuGeneCiD, and will be named "circuit_designs_file_descriptor.txt". Figure 9 shows an example of the logic table output file for one solution and the first time point when modeling that solution using EuGeneCiM.

Note: Other output files are generally quite large and useful primarily for debugging purposes, and therefore will not be discussed here. Please see the troubleshooting section.


A
Circuit Designs File Format Examples
model status code returned by GAMS
solver status code returned by GAMS
What is reported if a solution is not found for the given circuit size
What is reported if a solution is found for the given circuit size
biopart identifiers from the unified input
Circuit design reported as triads
B
Simple Report File Format Example
Circuit design reported as triads
Report of which bioparts in the circuit respond to the desired ligands
deliberate transcription
Transcription caused by leakiness
Total transcript production
This section is repeated for all combinations of elements from the desired ligands set
Total protein production
Transcript degradation in EuGeneCiD
List of proteins which are expressed in the given condition

```
No remaining solutions found for circuit size:          2.00
Model Status:         10.00
Solver Status:         1.00
Time to Determine:   0.95100005 second

Solution Number: 0
Objective Value:         10.10
Circuit Size:         3.00
model status:            1.00
solver status:           1.00
time to solution:   2.54900046 second

Designed Genetic Circuit:
Promoter                        Transcript                      Terminator
-------------------------------------------------------------------------
P_EXO70B1_11                    gene_tetR                       CaMV35St
P_GT                            gene_tetR                       HSPt
P_tet                           gene_GFP                        HSPt
```

```
SIMPLIFIED EUGENECID RESULTS AND STATE FILE


====================================================================
RESULTS FOR SOLUTION NUMBER 0
====================================================================

CIRCUIT DESIGN

PROMOTER                    TRANSCRIPT                  TERMINATOR
P_EXO70B1_11                  gene_tetR                     CaMV35St
P_GT                          gene_tetR                     HSPt
P_tet                         gene_GFP                      HSPt


RESPONDANTS TO LIGANDS
P_EXO70B1_11                    activated by Cu
P_GT                            inhibited by Cu
P_GT                            inhibited by Cd


ENZYMES ATTRIBUTION TO OTHER ENZYMES
GFP                             is influenced by tetR


ligand condition: Ld: Cu                             Ld1: Cu
-----------------------------------------------------------------------

Transcripts Produced
transcript                          deliberate      leaked      produced      degraded
gene_GFP                              0.00           1.00        1.00          0.21
gene_tetR                             4.00           2.00        6.00          1.67


Proteins Produced:
GFP                                   1.12
tetR                                  6.12


Proteins Expressed:
tetR                                  1.00
```

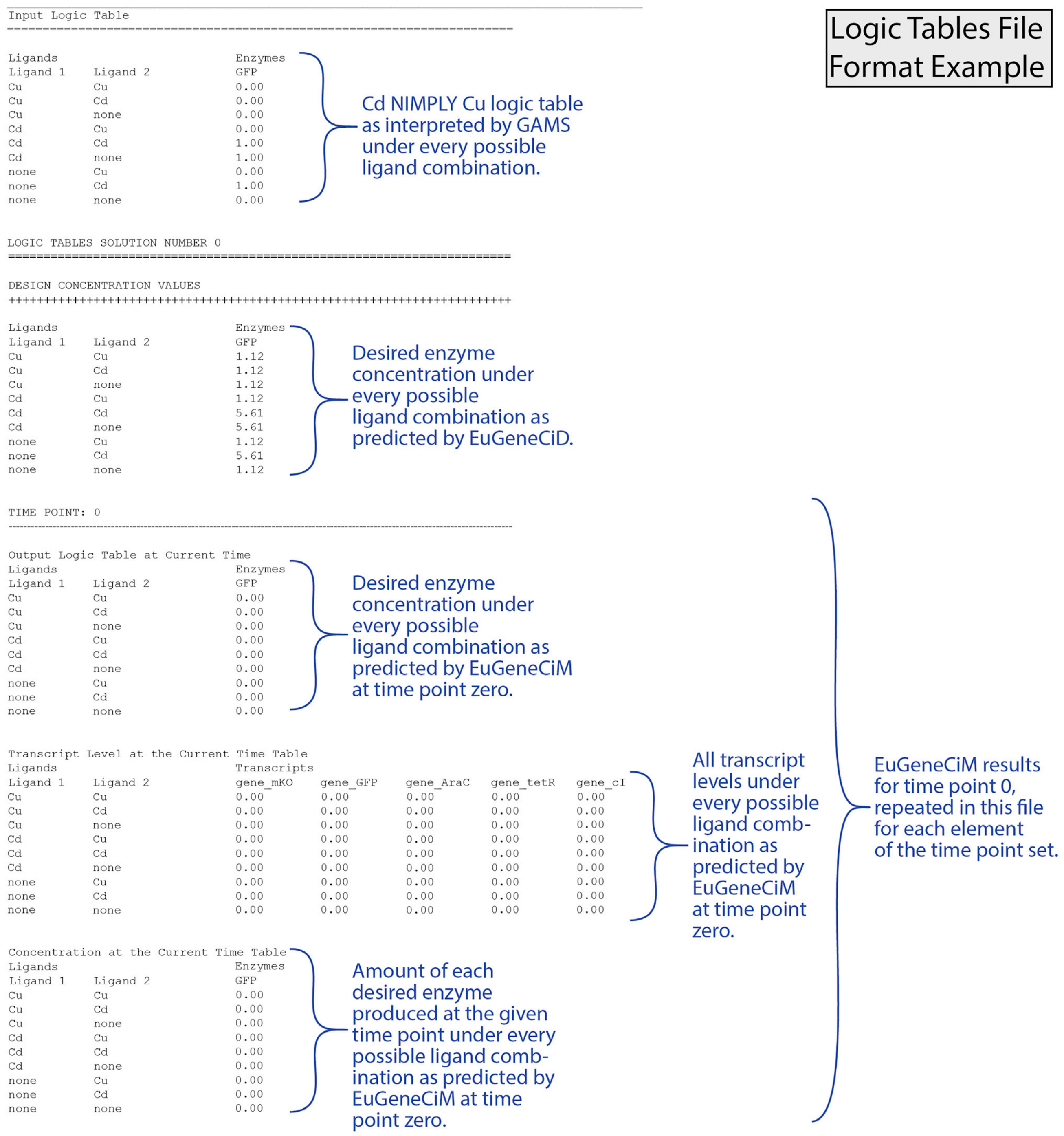
Figure 8. Examples from the circuit design and simple report output files

Shown in this figure are pieces of two example output files (the circuit design and simple report files) of the EuGeneCiD and EuGeneCiM combined output (black text). Feature of note or interest are pointed to by blue open brackets or lines and comments are made about these features in blue text.

(A) Shows both a successful and unsuccessful solution output to the circuit designs file for a Cadmium NIMPLY Copper circuit conceptualization.

(B) Shows a portion of the simple report for the zeroth solution to the Cadmium NIMPLY Copper circuit conceptualization.

27. **Use the statistics script to get holistic results.** As already noted, many of the files are unwieldy too large, or not optimized for human readability. This issue is addressed by two Perl scripts that can help users understand general and specific trends in the solutions. The first is the statistics script, which reports on the general statistics of the set of solutions produced by EuGeneCiD. The following sub steps address how to use this script.
    a. **Download the statistics script from GitHub.** The statistics script is named "get_results_statistics.pl", and can be found in GitHub at http://github.com/ssbio/EuGeneCiDM/EuGeneCiDM_results_final.



Logic Tables File Format Example

```
Input Logic Table
======================================================================

Ligands                        Enzymes
Ligand 1     Ligand 2          GFP
Cu           Cu                0.00
Cu           Cd                0.00
Cu           none              0.00
Cd           Cu                0.00
Cd           Cd                1.00
Cd           none              1.00
none         Cu                0.00
none         Cd                1.00
none         none              0.00

LOGIC TABLES SOLUTION NUMBER 0
======================================================================

DESIGN CONCENTRATION VALUES
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Ligands                        Enzymes
Ligand 1     Ligand 2          GFP
Cu           Cu                1.12
Cu           Cd                1.12
Cu           none              1.12
Cd           Cu                1.12
Cd           Cd                5.61
Cd           none              5.61
none         Cu                1.12
none         Cd                5.61
none         none              1.12

TIME POINT: 0
----------------------------------------------------------------------

Output Logic Table at Current Time
Ligands                        Enzymes
Ligand 1     Ligand 2          GFP
Cu           Cu                0.00
Cu           Cd                0.00
Cu           none              0.00
Cd           Cu                0.00
Cd           Cd                0.00
Cd           none              0.00
none         Cu                0.00
none         Cd                0.00
none         none              0.00

Transcript Level at the Current Time Table
Ligands                        Transcripts
Ligand 1     Ligand 2          gene_mKO     gene_GFP     gene_AraC    gene_tetR    gene_cI
Cu           Cu                0.00         0.00         0.00         0.00         0.00
Cu           Cd                0.00         0.00         0.00         0.00         0.00
Cu           none              0.00         0.00         0.00         0.00         0.00
Cd           Cu                0.00         0.00         0.00         0.00         0.00
Cd           Cd                0.00         0.00         0.00         0.00         0.00
Cd           none              0.00         0.00         0.00         0.00         0.00
none         Cu                0.00         0.00         0.00         0.00         0.00
none         Cd                0.00         0.00         0.00         0.00         0.00
none         none              0.00         0.00         0.00         0.00         0.00

Concentration at the Current Time Table
Ligands                        Enzymes
Ligand 1     Ligand 2          GFP
Cu           Cu                0.00
Cu           Cd                0.00
Cu           none              0.00
Cd           Cu                0.00
Cd           Cd                0.00
Cd           none              0.00
none         Cu                0.00
none         Cd                0.00
none         none              0.00
```

Cd NIMPLY Cu logic table as interpreted by GAMS under every possible ligand combination.

Desired enzyme concentration under every possible ligand combination as predicted by EuGeneCiD.

Desired enzyme concentration under every possible ligand combination as predicted by EuGeneCiM at time point zero.

All transcript levels under every possible ligand comb-ination as predicted by EuGeneCiM at time point zero.

Amount of each desired enzyme produced at the given time point under every possible ligand comb-ination as predicted by EuGeneCiM at time point zero.

EuGeneCiM results for time point 0, repeated in this file for each element of the time point set.

Figure 9. Example from the logic table output file

Shown in this figure is a portion of the logic tables output file of the EuGeneCiD and EuGeneCiM combined output (black text). Feature of note or interest are pointed to by blue open brackets or lines and comments are made about these features in blue text.

b. **Correctly place the code.** The statistics script should be placed in the parent directory.
c. **Update the descriptor reference.** On line 12 of the script, replace the string inside the quota-tion marks with the user's "descriptor".

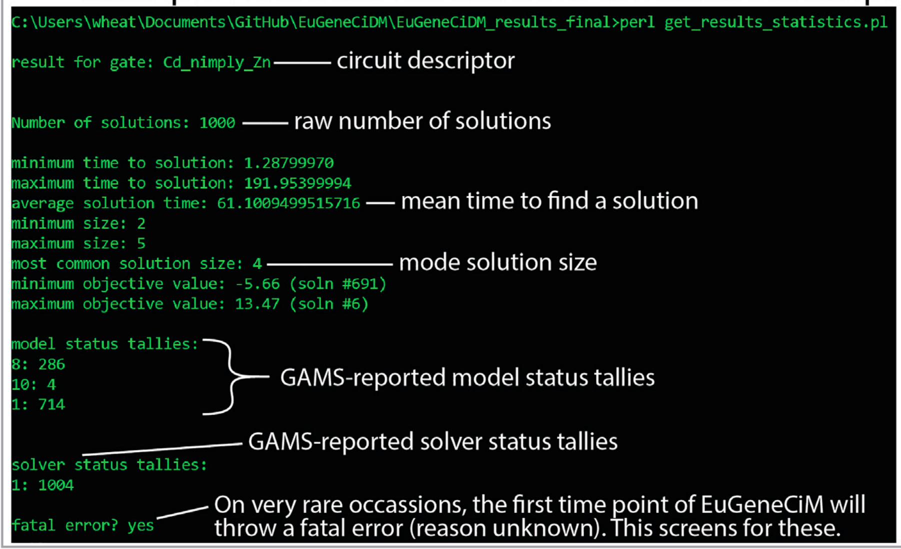
⚠ CRITICAL: If the descriptor does not match that used to name the output files of the combined EuGeneCiD and EuGeneCiM workflow, then the script will throw an error when the user attempts to run it.

d. **Run the statistics script.** From the parent directory, run the statistics script using the command "perl get_results_statistics.pl" as shown in Figure 10A.
e. **Read the results of the statistics script.** Figure 10A shows an example result of running the statistics script for the Cadmium NIMPLY Copper circuit. This reports important metrics about the set of solutions produced by EuGeneCiD including the number of solutions; minimum, maximum, and mean solution times; minimum, maximum, and mode solution sizes (in number of triads); minimum and maximum objective values with the corresponding solution number; a tally of model statuses returned; a tally of solver statuses returned; and whether a fatal error has occurred.
   i. **Fatal error.** On very rare occasions, a fatal error will occur when attempting to solve the first time point of EuGeneCiM, which will be reported here if one has occurred. These fatal errors are very rare and inconsistent in occurrence, resulting in no clear reason as to why these occur. As they only occur when implementing EuGeneCiM, it is possible to re-create the lost data using the solution design and steps 18–23 for a particular solution.

28. **Use the plot a solution script to get specific results in Excel format.** As the opposite of the statistics script, the plot a solution script reports on the EuGeneCiM results of a single indicated solution. This report takes the form of a Microsoft Excel file with four sheets reporting on transcript production ("Trans_Prod" sheet), transcript level ("Trans_Level" sheet), enzyme production ("Enz_Prod" sheet), and enzyme level ("Enz_Level" sheet). The following sub steps address how to use this script.
   a. **Download the statistics script from GitHub.** The statistics script is named "plot_a_solution.pl", and can be found in GitHub at http://github.com/ssbio/EuGeneCiDM/EuGeneCiDM_results_final.
   b. **Correctly place the code.** The plot a solution script should be placed in the parent directory.
   c. **Update the descriptor reference.** On line 16 of the script, replace the string inside the quotation marks with the user's "descriptor" (assigning the string as the value of the "$gate" variable). On line 17 of the script, the numerical value assigned to the variable "$soln_num" should be replaced with the solution number that the user wishes to examine more closely. Note that this value should be less than or equal to the total number of solutions as determined in step 27.
   d. **Run the statistics script.** From the parent directory, run the statistics script using the command "perl plot_a_solution.pl".
   e. **Read the results of the statistics script.** Figure 10B shows an example result of running the statistics script for the Cadmium NIMPLY Copper circuit. This results in an excel workbook with four distinct sheets and 16 scatterplots showing different aspects of the modeled behavior of the chosen solution.
      i. **Enzyme level results.** Shown in Figure 10B is an example "Enz_Level" sheet for the zeroth solution of the Cadmium NIMPLY Copper circuit conceptualization. This sheet contains tables for enzyme level at all time points across each of the four unique condition combinations. These values are then plotted in a scatter plot (one set of data per enzyme). Shown here are plots of tetR (red squares) and GFP (blue diamonds). Note that normally the generated graphs are overlapping, and they have been rearranged in this figure to be easier to see and read. Each sheet is similarly formatted, simply reporting different datasets.
      ii. **Enzyme production results.** Similar in layout to the "Enz_Level" sheet, the "Enz_Prod" sheet reports on the rate of enzyme production at each time point under each ligand condition, both in table and scatterplot forms.

Results of Perl Scripts Designed to aid in Understanding of Results

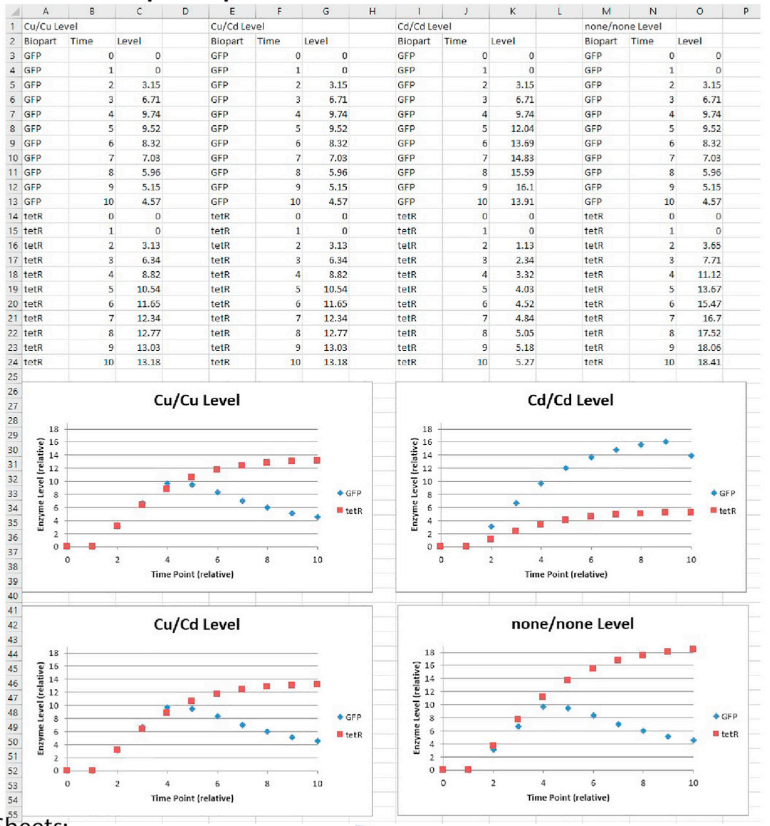
A Example Heuristics Results from the Statistics Script



```
C:\Users\wheat\Documents\GitHub\EuGeneCiDM\EuGeneCiDM_results_final>perl get_results_statistics.pl

result for gate: Cd_nimply_Zn

Number of solutions: 1000

minimum time to solution: 1.28799970
maximum time to solution: 191.95399994
average solution time: 61.1009499515716
minimum size: 2
maximum size: 5
most common solution size: 4
minimum objective value: -5.66 (soln #691)
maximum objective value: 13.47 (soln #6)

model status tallies:
8: 286
10: 4
1: 714

solver status tallies:
1: 1004

fatal error? yes
```

circuit descriptor
raw number of solutions
mean time to find a solution
mode solution size
GAMS-reported model status tallies
GAMS-reported solver status tallies
On very rare occassions, the first time point of EuGeneCiM will throw a fatal error (reason unknown). This screens for these.

B Example Specific Results from the Plot a Solution Script (Enz_Level sheet)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Cu/Cu Level				Cu/Cd Level				Cd/Cd Level				none/none Level			
2	Biopart	Time	Level		Biopart	Time	Level		Biopart	Time	Level		Biopart	Time	Level	
3	GFP	0	0		GFP	0	0		GFP	0	0		GFP	0	0	
4	GFP	1	0		GFP	1	0		GFP	1	0		GFP	1	0	
5	GFP	2	3.15		GFP	2	3.15		GFP	2	3.15		GFP	2	3.15	
6	GFP	3	6.71		GFP	3	6.71		GFP	3	6.71		GFP	3	6.71	
7	GFP	4	9.74		GFP	4	9.74		GFP	4	9.74		GFP	4	9.74	
8	GFP	5	9.52		GFP	5	9.52		GFP	5	12.04		GFP	5	9.52	
9	GFP	6	8.32		GFP	6	8.32		GFP	6	13.69		GFP	6	8.32	
10	GFP	7	7.03		GFP	7	7.03		GFP	7	14.83		GFP	7	7.03	
11	GFP	8	5.96		GFP	8	5.96		GFP	8	15.59		GFP	8	5.96	
12	GFP	9	5.15		GFP	9	5.15		GFP	9	16.1		GFP	9	5.15	
13	GFP	10	4.57		GFP	10	4.57		GFP	10	13.91		GFP	10	4.57	
14	tetR	0	0		tetR	0	0		tetR	0	0		tetR	0	0	
15	tetR	1	0		tetR	1	0		tetR	1	0		tetR	1	0	
16	tetR	2	3.13		tetR	2	3.13		tetR	2	1.13		tetR	2	3.65	
17	tetR	3	6.34		tetR	3	6.34		tetR	3	2.34		tetR	3	7.71	
18	tetR	4	8.82		tetR	4	8.82		tetR	4	3.32		tetR	4	11.12	
19	tetR	5	10.54		tetR	5	10.54		tetR	5	4.03		tetR	5	13.67	
20	tetR	6	11.65		tetR	6	11.65		tetR	6	4.52		tetR	6	15.47	
21	tetR	7	12.34		tetR	7	12.34		tetR	7	4.84		tetR	7	16.7	
22	tetR	8	12.77		tetR	8	12.77		tetR	8	5.05		tetR	8	17.52	
23	tetR	9	13.03		tetR	9	13.03		tetR	9	5.18		tetR	9	18.06	
24	tetR	10	13.18		tetR	10	13.18		tetR	10	5.27		tetR	10	18.41	

Logic Gate Conceptualization: Cd NIMPLY Zn

Enzyme level for GFP under a given condition for all EuGeneCiM-modeled time points.

New datasets added for each enzyme with non-zero concentration at at least one time point.

Repeated for each condition

Scatterplots reporting enzyme levels with respect to time to visualize the results. (note, raw is stacked and staggered by 4 columns, these were unstacked manually).

Sheets:
Trans_Prod - Transcript Production
Trans_Level - Transcript Level
Enz_Prod - Enzyme Production
Enz_Level - Enzyme Level

The above formatted output is produced and similarly formatted for each sheet.

Figure 10. Results of two perl scripts created to analyze the results of the combined EuGeneCiD and EuGeneCiM workflow
(A) Example command line output of Perl scripts to aid in analysis of results (green text). Features of particular interest are pointed to by white lines or brackets with white descriptive text.
(B) Shown here is a portion of the logic tables output file of the EuGeneCiD and EuGeneCiM combined output (black text). Feature of note or interest are pointed to by blue open brackets or lines and comments are made about these features in blue text.

iii. **Transcript level results.** Similar in layout to the "Enz_Level" sheet, the "Trans_Level" sheet reports on the rate of enzyme production at each time point under each ligand condition, both in table and scatterplot forms.
iv. **Transcript production results.** Similar in layout to the "Enz_Level" sheet, the "Trans_Prod" sheet reports on the rate of transcript production at each time point under each ligand condition, both in table and scatterplot forms.

Note: This code can be applied to the output of running only the EuGeneCiM code as well. The variable "$gate" in step 28.c. should be assigned the value "descriptor_CiM_only", and the variable "$soln_num" should be set to 0 in this case.

EXPECTED OUTCOMES

Presented in Figures 6, 8, 9, and 10 are partial examples of expected outcomes, showing the resulting files, their contents, and outputs from the Perl scripts to make the outcomes easier to understand. In most applications of this protocol, there will be dozens if not hundreds of outcomes. Here, an "outcome" is defined as having two parts: 1) a genetic circuit design using pieces from the bioparts library, and 2) a model of how that circuit will behave with respect to time. Designs of particular interest can potentially be identified through steps 24–27 and further analyzed in step 28 for desired dynamic behavior. From the set of solutions (however large the user has limited it to be in step 7), the user can analyze and select (a) desired design(s) with which to proceed for implementation and testing *in vivo*.

LIMITATIONS

The EuGeneCiD and EuGeneCiM systems are essentially applied to a small single cell, as there is no explicit inclusion of transport mechanisms, diffusion, or cell differentiation. Cell differentiation, and the resulting differential expression of genes must be considered when defining the bioparts database. This may be problematic when attempting to model behavior in a multi-cellular organism) and has limited ability to account for individual variations between cells. In contrast to other techniques, EuGeneCiD and EuGeneCiM produce relative concentration predictions, rather than exact levels. Additionally, as already discussed, while *in vivo* repressilators have sinusoidal behavior, EuGeneCiM-modeled repressilators do not due to their underlying binary mathematics, though their shape is similar as already discussed. Further, some current tools (with a more biophysical focus) include considerations of copy number and phenotypic ranges, which are not accounted for in the EuGeneCiD and EuGeneCiM tools. Finally, as shown in Schroeder, Baber, and Saha (2021), many returned solutions are non-optimal, particularly for circuit logic which are more difficult to construct such as HALF ADDER, XOR, and others.

TROUBLESHOOTING

Problem 1: Error occurs when running a code or script

Since this workflow is entirely *in silico* and relies on codes downloaded from GitHub, some of which require manual editing, there may be occasional issues of syntax, versioning, or typographical errors. This problem would most likely be noticed when attempting steps 4, 17, 23, 27, and 28.

Potential solution

All provided code will be in either Perl or GAMS programming languages, and as such, there will be two different approaches for common debugging. For Perl scripts, errors will be output to the command line, and, generally, error codes are descriptive enough (including line numbers) for the error to be easily tracked down and corrected. In the case of Perl scripts, the most likely error will be misspellings of the "descriptor", leading to references for files which do not exist. GAMS code is somewhat more difficult to debug. Should a single mistake be made, generally a few dozen errors will occur, and the command line output will not be sufficiently descriptive. The most effective procedure, from the point of view of the authors, is to open the listing file (see Figure 6) and search for a dollar sign ($) followed by a number. If the selected text editor allows regular expressions (regex) searching of the document (such as Notepad++), it is most effective to search using the regular expression "\$\d". This will take the user to the first error, with the dollar sign occurring under the first character which caused the error. This error is likely the (or one of) the root error(s) which, when corrected, will significantly reduce the total number of errors. To determine what the particular error is, search for the string which matched the regular expression (for example, error code "$56" This method is particularly effective for compilation errors which could be caused by typographical errors which occur when users edit code downloaded from GitHub such as in steps 7 and 19 wherein GAMS code is modified. Unfortunately, this procedure sometimes cannot be used effectively to trace error which occur a program has been running for a very long time (for instance, several days), because the listing file becomes so large that some text editors will not be able to open it. In such cases, errors may go unresolved (such as the fatal error issue for EuGeneCiM checked for in step 27) and may have to be worked around in some manner.

Problem 2: Errors such as "element redefined" occur in GAMS

One of the more likely GAMS errors, in the authors' opinion, will be to find an error from troubleshooting problem 1 which refers to redefining an element. These errors are likely to be found when addressing troubleshooting 1, or in steps 4, 17, 23, 27, and 28.

Potential solution

This particular error will occur when a symbol has a non-unique global identifier. The solution to this problem is to ensure that each element of the bioparts library has a globally unique identifier. An identifier is valid and globally unique if:

1. It is a single string (e.g., no spaces).
2. It contains no special characters (e.g., !, #, @, etc.).
3. No other biopart (promoter, transcript, terminator, or enzyme) or ligand shares the same identifier.

Generally, a way to increase the chances that an identifier is globally unique is to put some type-specific reference in the identifier such as a "p" for promoter, "gene" for transcript, or a "t" for a terminator as done in Schroeder, Baber, and Saha (2021).

Problem 3: Code and similar items are out of date

Inspecting the log file from running convert_input_descriptor.pl. This problem is associated with step 4.

Potential solution

This file should, briefly, discuss what was found by the convert script and will provide basic information about the database size, logic table, and time points to be modeled. The first few rows of the output should read as follows (where XX stands in for some number):

```
Number of promoters: XX

Number of transcripts: XX

Number of terminators: XX

Number of enzymes: XX

Number of ligands: XX
```

Should the number of elements in any of the above set appear different than expected, it may indicate that the formatting of that particular sheet is incorrect, and that Figure 3 and the "before you begin" section of this protocol should be consulted to ensure formatting is correct. The next section will relate to how the convert code unpacks the logic table into something readable by GAMS. Each row will be reported on from the logic table. So, a logic table of the form shown in Figure 3 will produce the following output in the log file. In short, the ligands indicated as "present" in the given condition by the binary variables in the logic table will be listed, followed by what (if any) enzyme response to that condition is coded as desired. If an enzyme response is desired, the log file will then have written to it the condition strings which define the logic table in GAMS. The condition strings iterate through combinations of 'none' and the desired ligand(s), not all of which will be written to the "logic_table.txt" input file.

```
Row #1: present ligands: Cu Cd

Desired to have no enzyme response

Row #2: present ligands: Cu Cd

Desired to have no enzyme response

Row #3: present ligands: Cd

Enzyme Response: GFP

Condition string: 'Cd'.'Cd'.'GFP' 1

Found 'Cd'

Condition string: 'Cd'.'none'.'GFP' 1

Found 'Cd'

Condition string: 'Cd'.'Cd'.'GFP' 1

Found 'Cd'

Condition string: 'Cd'.'none'.'GFP' 1

Found 'Cd'

Condition string: 'none'.'Cd'.'GFP' 1

Found 'Cd'

Condition string: 'none'.'Cd'.'GFP' 1

Found 'Cd'

Row #4: present ligands: none

Desired to have no enzyme response
```

If this section reports an enzyme response to a condition which is not desired, then the input logic table may be incorrectly formatted. Finally, the log file reports the read maximum time.

```
max time: 10
```

Note if the reported time is not as expected, the "other" sheet (as shown in Figure 3C) may need to be corrected.

Problem 4: A solution occurs which should be infeasible

One problem which might occur is that when, in double-checking some of the EuGeneCiD solutions, a solution is discovered which should, given the formulation of this tool given in Schroeder, Baber, and Saha (2021) be infeasible. This problem is most likely to occur if the solver settings have been adjusted (for example, if the user decides to adjust the solver settings in the solver options file), or lines of the GAMS code have been edited other than those explicitly references in steps 7 and 19.

Potential solution

Generally, this will result from a mismatch between solver settings (particularly those with "ep" in the name such as "epint" or "eprhs") and the value of the parameter "epsilon" in the combined EuGeneCiD and EuGeneCiM workflow code. The epsilon parameter generally exists to cause slight infeasibilities in constraint equations under certain variable state conditions. This is generally caused by combinations of binary variables diverging too much and too much infeasibility being allowed between the two sides of a constraint equation. If the value of binary variables is allowed too much divergence from exactly 0 or 1 and the allowed infeasibility between sides of a constraint is too large, then the effects of epsilon can be negated, causing numerical issues. Generally, constraints related to the allowed value divergence from strict binary values (such as "epint") and those which allow infeasibility between constraint sides (such as "eprhs") should be anything less than or equal to three orders of magnitude smaller than epsilon.

One specific reason for this issue might result from the determination of whether enzyme concentration reaches a set level. These are the two constraints which make this determination, which are also sensitive to the interrelation between the arbitrarily large number (V), arbitrary small number (ϵ), and the solver settings (particularly, "eprhs" = ϵ_{rhs}). These equations are listed below.

$$(\theta_e + \epsilon)C^+_{e,l_d,l_{d1}} \leq C_{e,l_d,l_{d1}} \quad \forall e \in E, l_d, l_{d1} \in L_d \tag{Equation 1}$$

$$C_{e,l_d,l_{d1}} \leq (V - (\theta_e - \epsilon))C^+_{e,l_d,l_{d1}} + (\theta_e - \epsilon) \quad \forall e \in E, l_d, l_{d1} \in L_d \tag{Equation 2}$$

Where E is the set of enzymes and proteins; L_d is the set of ligands to which the system responds as defined by the logic table; $C_{e,l_d,l_{d1}}$ is the concentration of enzyme e under ligand signal conditions l_d and l_{d1}; $C^+_{e,l_d,l_{d1}}$ is the binary variable which stores if the enzyme concentration is sufficient for the phenotype to be expressed (1 if sufficient, 0 otherwise); and θ_e is the concentration for enzyme e which defines sufficiency for concentration. The above equations can be solved for $C^+_{e,l_d,l_{d1}}$ as follows.

$$C^+_{e,l_d,l_{d1}} \leq \frac{C_{e,l_d,l_{d1}}}{\theta_e + \epsilon} \quad \forall e \in E, l_d, l_{d1} \in L_d \tag{Equation 3}$$

$$\frac{C_{e,l_d,l_{d1}} - \theta_e + \epsilon}{V - \theta_e + \epsilon} \leq C^+_{e,l_d,l_{d1}} \quad \forall e \in E, l_d, l_{d1} \in L_d \tag{Equation 4}$$

The success of these constraints to produce a robust solution for $C^+_{e,l_d,l_{d1}}$ depend somewhat on the values of V, ϵ, ϵ_{rhs}, θ_e, and $C_{e,l_d,l_{d1}}$ as can be demonstrated through the following example where $V = 1E4$, $\epsilon = 1E-4$, $\epsilon_{rhs} = 1E-6$, $\theta_e = 5$, and $C_{e,l_d,l_{d1}} = 5.0007$. Substituting these values into Equations (3) and (4) gives the following.

$$\frac{C_{e,l_d,l_{d1}} - \theta_e + \epsilon}{V - \theta_e + \epsilon} \approx 8E-8 \leq C^+_{e,l_d,l_{d1}} \leq \frac{C_{e,l_d,l_{d1}}}{\theta_e + \epsilon} = 1.0012 \quad \forall e \in E, l_d, l_{d1} \in L_d \tag{Equation 5}$$

Rigorously, since $C^+_{e,l_d,l_{d1}}$ is a binary variable, it should take the value of 1; however, since numerical optimization solvers allow for some infeasibility in solutions, $C^+_{e,l_d,l_{d1}}$ is allowed to be either 0 or 1. This is because $C^+_{e,l_d,l_{d1}} = 0$ causes an infeasibility of approximately $8E-8$, which is less than the solver setting, $\epsilon_{rhs} = 1E-6$, and is therefore allowed. For convenience, this will be called a Threshold Ambiguity Error (TEA). A TEA allows an enzyme in a specific concentration range to be either "on" or "off" at the discretion of the solver. The equations related to this numerical error can be explored to identify a range for the concentration variable $C_{e,l_d,l_{d1}}$ in which these errors can occur.

For Equation (3) to allow both values of $C^+_{e,l_d,l_{d1}}$, the following must be true (for some $e \in E$ and $l_d, l_{d1} \in L_d$):

$$\frac{C_{e,l_d,l_{d1}}}{\theta_e + \epsilon} \geq 1 \tag{Equation 6}$$

Therefore, these problematic solutions can occur if:

$$C_{e,l_d,l_{d1}} \geq \theta_e + \epsilon \qquad \text{(Equation 7)}$$

The second necessary condition, Equation (4), must also allow for both values of $C^+_{e,l_d,l_{d1}}$. This occurs by the lower bound being less than ϵ_{rhs}, so that the infeasibility between that number and 0 is less than ϵ_{rhs}. For this to occur then, the following must be true:

$$\frac{C_{e,l_d,l_{d1}} - \theta_e + \epsilon}{V - \theta_e + \epsilon} \leq \epsilon_{rhs} \qquad \text{(Equation 8)}$$

Solving for the concentration range at which the above can be true:

$$C_{e,L_d,L_{d1}} - (\theta_e - \epsilon) \leq \epsilon_{rhs}(V - (\theta_e - \epsilon)) \qquad \text{(Equation 9)}$$

$$C_{e,L_d,L_{d1}} \leq \epsilon_{rhs}(V - (\theta_e - \epsilon)) + (\theta_e - \epsilon) \qquad \text{(Equation 10)}$$

$$C_{e,L_d,L_{d1}} \leq \epsilon_{rhs}V - \epsilon_{rhs}(\theta_e - \epsilon) + (\theta_e - \epsilon) \qquad \text{(Equation 11)}$$

$$C_{e,L_d,L_{d1}} \leq \epsilon_{rhs}V - (\epsilon_{rhs}\theta_e - \epsilon_{rhs}\epsilon) + (\theta_e - \epsilon) \qquad \text{(Equation 12)}$$

$$C_{e,L_d,L_{d1}} \leq \epsilon_{rhs}V - \epsilon_{rhs}\theta_e + \epsilon_{rhs}\epsilon + \theta_e - \epsilon \qquad \text{(Equation 13)}$$

$$C_{e,L_d,L_{d1}} \leq (1 - \epsilon_{rhs})\theta_e + \epsilon_{rhs}V + \epsilon_{rhs}\epsilon - \epsilon \qquad \text{(Equation 14)}$$

Therefore, the range of concentrations at which a single concentration could return both sufficient and deficient concentration state variables ($C^+_{e,l_d,l_{d1}}$) is:

$$\theta_e + \epsilon \leq C_{e,L_d,L_{d1}} \leq (1 - \epsilon_{rhs})\theta_e + \epsilon_{rhs}V + \epsilon_{rhs}\epsilon - \epsilon \qquad \text{(Equation 15)}$$

Returning to the example TEA above , Equation (15) appears as:

$$5.0001 \leq 5.0007 \leq 5.009895 \qquad \text{(Equation 16)}$$

Therefore, the particular combination of arbitrary values, solver settings, and calculated concentration resulted in a TEA. To force the range in Equation (15) to be non-existent and effectively prevent TEAs, the following must be true:

$$\theta_e + \epsilon \geq (1 - \epsilon_{rhs})\theta_e + \epsilon_{rhs}V + \epsilon_{rhs}\epsilon - \epsilon \qquad \text{(Equation 17)}$$

This can be solved to find an inequality governing the ideal relationship between θ_e, ϵ, ϵ_{rhs}, and V yields the following:

$$\theta_e + 2\epsilon \geq (1 - \epsilon_{rhs})\theta_e + \epsilon_{rhs}V + \epsilon_{rhs}\epsilon \qquad \text{(Equation 18)}$$

$$\epsilon_{rhs}\theta_e + 2\epsilon \geq \epsilon_{rhs}V + \epsilon_{rhs}\epsilon \qquad \text{(Equation 19)}$$

$$\theta_e + \frac{2\epsilon}{\epsilon_{rhs}} \geq V + \epsilon \qquad \text{(Equation 20)}$$

$$\theta_e + \frac{2\epsilon}{\epsilon_{rhs}} - \epsilon \geq V \qquad \text{(Equation 21)}$$

$$\theta_e + \frac{2\epsilon}{\epsilon_{rhs}} - \frac{\epsilon_{rhs}\epsilon}{\epsilon_{rhs}} \geq V \qquad \text{(Equation 22)}$$

$$\theta_e + \frac{2\epsilon - \epsilon_{rhs}\epsilon}{\epsilon_{rhs}} \geq V \qquad \text{(Equation 23)}$$

$$\theta_e + \frac{(2 - \epsilon_{rhs})}{\epsilon_{rhs}}\epsilon \geq V \qquad \text{(Equation 24)}$$

The value of $C^{+}_{e,l_d,l_{d1}}$ can be ambiguous when the above relationship does not hold. An example is provided in Equation (25), with the example values used in Equation (5) are substituted into Equation (24).

$$5 + \frac{(2-(1E-6))}{(1E-6)}(1E-4) = 204.9999 \geq 1E4 \qquad \text{(Equation 25)}$$

Equation (24) has a few notable points for setting arbitrary values and defining solver settings:

1) The influence of the threshold value is largely irrelevant at the magnitude used throughout this protocol.
2) The arbitrarily large number V should be less than or equal to twice the ratio of ϵ to ϵ_{rhs}. For instance, in Equation (24), the ratio between ϵ and ϵ_{rhs} is $1E2$; therefore, to not have this numerical issue, V should be less than $2E2$. Consider this example where the relationships are correct ($\epsilon_{rhs} = 1E-6$, $\epsilon = 1E-3$, $\theta_e = 5$, $V = 1E3$):

$$5 + \frac{(2-(1E-6))}{(1E-6)}(1E-3) = 2004.9999 \geq 1E3 \qquad \text{(Equation 26)}$$

Substituting these values back into Equation (15), we then see that Equation (15) then results in a non-viable range at which this error can occur:

$$5.001 \leq C_{e,L_d,L_{d1}} \leq 4.995000001 \qquad \text{(Equation 27)}$$

Should solutions which are infeasible occur, it is therefore recommended to check the provided values of ϵ, V, and ϵ_{rhs} defined in the code and "cplex.opt" file to ensure that the relation shown in Equation (24) holds.

Problem 5: Code and similar items are out of date

This work is out-of-date and the time necessary to update it is not considered worthwhile or the workflow has been significantly changed.

Potential solution

The EuGeneCiD and EuGeneCiM tools are created as the basis for the future development of a synthetic biology application development pipeline. Therefore this tool will be supported for some time, and the workflow of the tool may evolve. See the associated GitHub page (github.com/ssbio/EuGeneCiDM) for updates, including updated documentation. Given that this workflow utilizes programming languages which are constantly evolving, it is not inconceivable that codes in this workflow may eventually become out-of-date. The authors do not intend this to be a monolithic procedure that can be forever duplicated. Rather, in future, it is hoped that this protocol outlines a procedure which may be a useful guideline in automating the design and modeling of synthetic biology circuits. It is particularly hoped that this procedure will see an increase in the use of the EuGeneCiD and EuGeneCiM tools for these tasks, or could be a guide for wholescale automation for the development of a synthetic biology circuit applications pipeline.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Rajib Saha (rsaha2@unl.edu).

Materials availability

This study did not generate new unique reagents.

Data and code availability

The published article does not include all data sets and code generated or analyzed during this study. All data sets and code generated during this study are available at GitHub in the ssbio/EuGeneCiDM repository [https://doi.org/10.5281/zenodo.4762590] or at the following URL github.com/ssbio/EuGeneCiDM. In addition, the data and code can be found in Mendeley Data at https://data.mendeley.com/datasets/n5zsmxgd4h/1.

SUPPLEMENTAL INFORMATION

Supplemental information can be found online at https://doi.org/10.1016/j.xpro.2021.100820.

ACKNOWLEDGMENTS

This work has been completed utilizing the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative. The authors gratefully acknowledge funding from the NSF EPSCoR Center for Root and Rhizobiome Innovation Grant 25-1215-0139-025 at the University of Nebraska – Lincoln. The authors would also like to acknowledge the contribution of participants of the Young Nebraska Scientists Program (YNSP) who participated in this research including (listing in alphabetical order by family name): Bree Brunsman, Evan Fulton, Ali Keshk, Kareem Keshk, and Molly Nora.

AUTHOR CONTRIBUTIONS

Conceptualization, W.L.S. and R.S.; data curation, investigation, formal analysis, and software, W.L.S. and A.S.B.; funding acquisition, R.S.; methodology, W.L.S.; project administration, resources, and supervision, R.S.; validation and visualization, W.L.S.; writing – original draft, W.L.S. and R.S.; writing – reviewing & editing – W.L.S., A.S.B., and R.S.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Chmielowska-Bąk, J., Gzyl, J., Rucinska-Sobkowiak, R., Arasimowicz-Jelonek, M., and Deckert, J. (2014). The new insights into cadmium sensing. Front. Plant Sci. *5*, 1–13. https://doi.org/10.3389/fpls.2014.00245.

Dasika, M.S., and Maranas, C.D. (2008). OptCircuit: an optimization based method for computational design of genetic circuits. BMC Syst. Biol. *2*, 1–19. https://doi.org/10.1186/1752-0509-2-24.

Fang, Z., Martin, J., and Wang, Z. (2012). Statistical methods for identifying differentially expressed genes in RNA-Seq experiments. Cell Biosci. *2*, 1–8. https://doi.org/10.1186/2045-3701-2-26.

Guénin, Stephanie, Mauriat, Melanie, Pelloux, Jerome, Van Wuytswinkel, Olivier, Bellini, Catherine, and Gutierrez, Laurent (2009). Normalization of qRT-PCR data: the necessity of adopting a systematic, experimental conditions-specific, validation of references. J. Exp. Bot. *60*, 487–493. https://doi.org/10.1093/jxb/ern305.

Kudla, G., Murray, A.W., Tollervey, D., and Plotkin, J.B. (2009). Coding-sequence determinants of gene expression in Escherichia coli. Science, 255–259.

Reue, K. (1998). 'Issues and opinions in nutrition mRNA quantitation Techniques: considerations for experimental'. J. Nutr. 2038–2044.

Schroeder, W.L., Baber, W.S., and Saha, R. (2021). Optimization-based Eukaryotic Genetic Circuit Design (EuGeneCiD) and modeling (EuGeneCiM) tools: Computational approach to synthetic biology. iScience103000. https://doi.org/10.1016/j.isci.2021.103000.

Schroeder, W.L., and Saha, R. (2020a). OptFill: a tool for infeasible cycle-free gapfilling of stoichiometric metabolic models. iScience *23*, 100783. https://doi.org/10.1016/j.isci.2019.100783.

Schroeder, W.L., and Saha, R. (2020b). Protocol for genome-scale reconstruction and melanogenesis analysis of exophiala dermatitidis protocol for genome-scale reconstruction and melanogenesis analysis of exophiala dermatitidis. STAR Protoc. *1*. https://doi.org/10.1016/j.xpro.2020.100105.

Suzuki, N., Koizumi, N., and Sano, H. (2001). Screening of cadmium-responsive genes in Arabidopsis thaliana. Plant Cell Environ. *24*, 1177–1188. https://doi.org/10.1046/j.1365-3040.2001.00773.x.

Van De Mortel, J.E., Schat, H., Moerland, P.D., Van Thermaat, E.V.L., Van Der Ent, S., Blankestijn, H., Ghandilyan, A., Tsiatsiani, S., and Aarts, M.G.M. (2008). Expression differences for genes involved in lignin, glutathione and sulphate metabolism in response to cadmium in Arabidopsis thaliana and the related Zn/Cd-hyperaccumulator Thlaspi caerulescens. Plant Cell Environ. *31*, 301–324. https://doi.org/10.1111/j.1365-3040.2007.01764.x.

Van De Mortel, Judith E., Villanueva, Laia Almar, Schat, Henk, Kwekkeboom, Jeroen, Coughlan, Sean, Moerland, Perry D., van Themaat, Emiel Ver Loren, Koornneef, Maarten, and Aarts, Mark G.M. (2006). Large expression differences in genes for iron and zinc homeostasis, stress response, and lignin biosynthesis distinguish roots of Arabidopsis thaliana and the related metal hyperaccumulator Thlaspi caerulescens. Plant Physiol. *142*, 1127–1147. https://doi.org/10.1104/pp.106.082073.

Ward, N.J., Buckley, S.M.K., Waddington, S.N., VandenDriessche, T., Chuah, M.K.L., Nathwani, A.C., McIntosh, J., Tuddenham, E.G.D., Kinnon, C., Thrasher, A.J., and McVey, J.H. (2011). Codon optimization of human factor VIII cDNAs leads to high-level expression. Blood *117*, 798–807. https://doi.org/10.1182/blood-2010-05-282707.

Zomorrodi, A.R., and Maranas, C.D. (2014). Coarse-grained optimization-driven design and piecewise linear modeling of synthetic genetic circuits. Eur. J. Oper. Res. *237*, 665–676. https://doi.org/10.1016/j.ejor.2014.01.054.