Inherent Dynamics Visualizer, an Interactive Application for Evaluating and Visualizing Outputs from a Gene Regulatory Network Inference Pipeline

Robert C. Moseley[1], **Sophia Campione**[1], **Bree Cummins**[2], **Francis Motta**[3], **Steven B. Haase**[1]

[1]Department of Biology, Duke University

[2]Department of Mathematical Sciences, Montana State University

[3]Department of Mathematical Sciences, Florida Atlantic University

Abstract

Developing gene regulatory network models is a major challenge in systems biology. Several computational tools and pipelines have been developed to tackle this challenge, including the newly developed Inherent Dynamics Pipeline. The Inherent Dynamics Pipeline consists of several previously published tools that work synergistically and are connected in a linear fashion, where the output of one tool is then used as input for the following tool. As with most computational techniques, each step of the Inherent Dynamics Pipeline requires the user to make choices about parameters that don't have a precise biological definition. These choices can substantially impact gene regulatory network models produced by the analysis. For this reason, the ability to visualize and explore the consequences of various parameter choices at each step can help increase confidence in the choices and the results.The Inherent Dynamics Visualizer is a comprehensive visualization package that streamlines the process of evaluating parameter choices through an interactive interface within a web browser. The user can separately examine the output of each step of the pipeline, make intuitive changes based on visual information, and benefit from the automatic production of necessary input files for the Inherent Dynamics Pipeline. The Inherent Dynamics Visualizer provides an unparalleled level of access to a highly intricate tool for the discovery of gene regulatory networks from time series transcriptomic data.

Introduction

Many important biological processes, such as cell differentiation and environmental response, are governed by sets of genes that interact with each other in a gene regulatory network (GRN). These GRNs produce the transcriptional dynamics needed for activating and maintaining the phenotype they control, so identifying the components and topological structure of the GRN is key to understanding many biological processes and functions. A GRN may be modeled as a set of interacting genes and/or gene products described by a

Corresponding Author: Robert C. Moseley, robert.moseley@duke.edu.

A complete version of this article that includes the video component is available at http://dx.doi.org/10.3791/63084.

Disclosures
The authors have nothing to disclose.

network whose nodes are the genes and whose edges describe the direction and form of interaction (e.g., activation/repression of transcription, post-translational modification, etc.)[1]. Interactions can then be expressed as parameterized mathematical models describing the impact a regulating gene has on the production of its target(s)[2,3,4]. Inference of a GRN model requires both an inference of the structure of the interaction network and estimation of the underlying interaction parameters. A variety of computational inference methods have been developed that ingest time series gene expression data and output GRN models[5]. Recently, a new GRN inference method was developed, called the Inherent Dynamics Pipeline (IDP), that utilizes time series gene expression data to produce GRN models with labeled regulator-target interactions that are capable of producing dynamics that match the observed dynamics in the gene expression data[6]. The IDP is a suite of tools connected linearly into a pipeline and can be broken down into three steps: a Node Finding step that ranks genes based on gene expression characteristics known or suspected to be related to the function of the GRN[7,8], an Edge Finding step that ranks pairwise regulatory relationships[8,9], and a Network Finding step that produces GRN models that are capable of producing the observed dynamics[10,11,12,13,14,15].

Like most computational methods, the IDP requires a set of user-specified arguments that dictate how the input data is analyzed, and different sets of arguments can produce different results on the same data. For example, several methods, including the IDP, contain arguments that apply some threshold on the data, and increasing/decreasing this threshold between successive runs of the particular method can result in dissimilar results between runs (see Supplement Note 10: Network inference methods of[5]). Understanding how each argument may impact the analysis and subsequent results is important for achieving high confidence in the results. Unlike most GRN inference methods, the IDP consists of multiple computational tools, each having its own set of arguments that a user must specify and each having its own results. While the IDP provides extensive documentation on how to parameterize each tool, the interdependency of each tool on the output of the previous step makes parameterizing the entire pipeline without intermediate analyses challenging. For instance, arguments in the Edge and Network Finding steps are likely to be informed by prior biological knowledge, and so will depend on the dataset and/or organism. To interrogate intermediate results, a basic understanding of programming, as well as a deep understanding of all the result files and their contents from the IDP, would be needed.

The Inherent Dynamics Visualizer (IDV) is an interactive visualization package that runs in a user's browser window and provides a way for users of the IDP to assess the impact of their argument choices on results from any step in the IDP. The IDV navigates a complicated directory structure produced by the IDP and gathers the necessary data for each step and presents the data in intuitive and interactive figures and tables for the user to explore. After exploring these interactive displays, the user can produce new data from an IDP step that can be based on more informed decisions. These new data can then be immediately used in the next respective step of the IDP. Additionally, exploration of the data can help determine whether an IDP step should be rerun with adjusted parameters. The IDV can enhance the use of the IDP, as well as make the use of the IDP more intuitive and approachable, as demonstrated by investigating the core oscillator GRN of the yeast cell-cycle. The following protocol includes IDP results from a fully parameterized IDP run versus an approach that incorporates the IDV after runs of each IDP step, i.e., Node, Edge, and Network Finding.

Protocol

1. Install the IDP and IDV

NOTE: This section assumes that docker, conda, pip, and git are installed already (Table of materials).

1. In a terminal, enter the command: *git clone* https://gitlab.com/biochron/inherent_dynamics_pipeline.git.

2. Follow the install instructions in the IDP's README file.

3. In a terminal, enter the command: *git clone* https://gitlab.com/bertfordley/inherent_dynamics_visualizer.git.

   NOTE: Cloning of the IDV should happen outside of the IDP's top-level directory.

4. Follow the install instructions in the IDV's README file.

2. Node finding

1. Create a new IDP configuration file that parametrizes the Node Finding step.

   NOTE: All quotation marks in the following steps should not be typed out. The quotation marks are only used here as a delimiter between the protocol text and what is to be typed out.

   1. Add the main IDP arguments to the configuration file.

   2. Open a new text file in a text editor and type "data_file =", "annotation_file =", "output_dir =", "num_proc =", and "IDVconnection = True" on individual lines.

   3. For "data_file", after the equal to sign, type the path to and name of the respective time series file and type a comma after the name. Separate each data by a comma, if more than one time series data set is being used. See Supplemental File 1 and Supplemental File 2 for an example of time series gene expression files.

   4. Type the path to and name of the annotation file for "annotation_file", after the equal to sign. See Supplemental File 3 for an example of an annotation file.

   5. For "output_file", after the equal to sign, type the path to and name of the folder where results will be saved.

   6. After the equal to sign, for "num_proc", type the number of processes the IDP should use.

   7. Add Node Finding arguments to the configuration file.

8. In the same text file as in step 2.1.1, type in the order presented "[dlxjtk_arguments]", "periods =", and "dlxjtk_cutoff =" on individual lines. Place these after the main arguments.

9. For "periods", after the equal to sign, if one-time series data set is being used, type each period length separated by commas. For more than one time series data set, type each set of period lengths as before but place square brackets around each set and place a comma between the sets.

10. After the equal to sign, for "dlxjtk_cutoff", type an integer specifying the maximum number of genes to retain in the gene_list_file output by de Lichtenberg by JTK_CYCLE (DLxJTK) (Table 1).

    NOTE: It is highly recommended to review the dlxjtk_arguments sections in the IDP README to get a better understanding of each argument. See Supplemental File 4 for an example of a configuration file with the Node Finding arguments specified.

2. In the terminal, move into the IDP directory, named inherent_dynamics_pipeline.

3. In the terminal, enter the command: *conda activate dat2net*

4. Run the IDP using the configuration file created in step 2.1 by running this command in the terminal, where <config file name> is the name of the file: *python src/dat2net.py <config file name>*

5. In the terminal, move to the directory named inherent_dynamics_visualizer and enter the command: *./viz_results.sh <results_directory>*

   NOTE: <results_directory> will point to the directory used as the output directory for the IDP.

6. In a web browser, enter http://localhost:8050/ as the URL.

7. With the IDV now open in the browser, click on the **Node Finding** tab and select the node finding folder of interest from the dropdown menu.

8. Manually curate a new gene list from the gene list table in the IDV to be used for subsequent IDP steps.

   1. To extend or shorten the gene list table, click on the up or down arrows or manually enter in an integer between 1 and 50 in the box next to **Gene expression of DLxJTK-ranked genes. Top:**.

   2. In the gene list table, click on the box beside a gene to view its gene expression profile in a line graph. Multiple genes can be added.

   3. Optionally specify the number of equally sized bins to compute and order genes by the time interval containing their peak expression, by inputting an integer into the input box above the gene list table labeled **Input integer to divide the first cycle into bins:**.

NOTE: This option is specific to oscillatory dynamics and might not be applicable to other types of dynamics.

4. Select a heatmap viewing preference by clicking on an option under **Order Genes By: First Cycle Max Expression** (Table 1) which orders genes based on the time of the gene-expression peak in the first cycle.

   NOTE: **DLxJTK Rank** orders genes based on the periodicity ranking from the DLxJTK algorithm of the IDP.

5. Click on the **Download Gene List** button to download the gene list into the file format needed for the Edge Finding step. See Supplemental File 5 for an example of a gene list file.

9. In the **Editable Gene Annotation Table**, label a gene as a target, a regulator, or both in the annotation file for the Edge Finding step in a new Edge Finding run. If a gene is a regulator, label the gene as an activator, repressor, or both.

   1. To label a gene as an activator, click on the cell in the tf_act column and change the value to 1. To label a gene as a repressor, change the value in the tf_rep column to 1. A gene will be allowed to act as both an activator and a repressor in the Edge Finding step by setting the values in both the tf_act and tf_rep columns to 1.

   2. To label a gene as a target, click on the cell in the target column and change the value to 1.

10. Click on the **Download Annot. File** button to download the annotation file into the file format needed for the Edge Finding step.

3. Edge finding

1. Create a new IDP configuration file that parametrizes the Edge Finding step.

   1. Add the main IDP arguments to the configuration file. Open a new text file in a text editor and repeat step 2.1.1.

   2. Add Edge Finding arguments to the configuration file.

   3. In the same text file as in step 3.1.1, type in the order presented “[lempy_arguments]”, “gene_list_file =“, “[netgen_arguments]”, “edge_score_column =“, “edge_score_thresho =“, “num_edges_for_list =“, “seed_threshold =“, and “num_edges_for_seed =“ on individual lines. These should go below the main arguments.

   4. For “gene_list_file”, after the equal to sign, enter the path to and name of the gene list file generated in step 2.8.5.

   5. For “edge_score_column”, after the equal to sign, enter either “pld” or “norm_loss” to specify which data frame column from the lempy output is used to filter the edges.

6. Select either “edge_score_threshold” or “num_edges_for_list”, and delete the other. If “edge_score_threshold” was selected, enter a number between 0 and 1. This number will be used to filter edges based on the column specified in step 3.1.5.

    1. If “num_edges_for_list” was selected, enter a value equal to or less than the number of possible edges. This number will be used to filter the edges based on how they are ranked in the column specified in step 3.1.5. The edges left over will be used to build networks in Network Finding.

7. Select either “seed_threshold” or “num_edges_for_seed” and delete the other. If “seed_threshold” was selected, enter a number between 0 and 1. This number will be used to filter edges based on the column specified in step 3.1.5.

    1. If “num_edges_for_seed” was selected, enter a value equal to or less than the number of possible edges. This number will be used to filter the edges based on how they are ranked in the column specified in step 3.1.5. The edges left over will be used to build the seed network (Table 1) used in Network Finding.

       NOTE: It is highly recommended to review the lempy_arguments and netgen_arguments sections in the IDP README to get a better understanding of each argument. See Supplemental File 7 for an example of a configuration file with the Edge finding arguments specified.

2. Repeat steps 2.2 and 2.3.

3. Run the IDP using the configuration file created in step 3.1 by running this command in the terminal, where <config file name> is the name of the file: *python src/dat2net.py <config file name>*

4. If the IDV is still running, stop it by pressing **Control C** in the terminal window to stop the program. Repeat steps 2.5 and 2.6.

5. With the IDV open in the browser, click on the **Edge Finding** tab and select the edge finding folder of interest from the drop-down menu.

   NOTE: If multiple datasets are used in Edge Finding, then make sure to select the last dataset that was used in the Local Edge Machine (LEM) analysis (Table 1). It is important when selecting edges for the seed network or edge list based on LEM results to look at the last time series data listed in the configuration file as this output incorporates all preceding datafiles in its inference of regulatory relationships between nodes.

6. To extend or shorten the edge table, manually enter an integer in the input box under **Number of Edges:**.

7. Optionally filter edges on the LEM ODE parameters. Click and drag to move either the left side or the right side of each parameter's slider to remove edges from the edge table that have parameters outside of their new allowed parameter bounds.

8. Optionally create a new seed network if a different seed network is wanted than the one proposed by the IDP. See Supplemental File 8 for an example of a seed network file.

    1. Select either **From Seed** to select the seed network or **From Selection** from the dropdown menu under **Network:**.

    2. Deselect/select edges from the edge table by clicking the corresponding checkboxes adjacent to each edge to remove/add edges from the seed network.

9. Click on the **Download DSGRN NetSpec** button to download the seed network in the Dynamic Signatures Generated by Regulatory Networks (DSGRN) (Table 1) network specification format.

10. Select additional nodes and edges to be used in the Network Finding step.

    1. Select edges from the edge table by clicking the corresponding checkboxes to include in the edge list file used in Network Finding.

    2. Click on **Download Node and Edge Lists** to download the node list and edge list files in the format required for their use in Network Finding. See Supplemental File 9 and Supplemental File 10 for examples of edge and node list files, respectively.

       NOTE: The node list must contain all the nodes in the edge list file, so the IDV automatically creates the node list file based on the selected edges. Two options are available for viewing the edges in Edge Finding. The **LEM Summary Table** option presents the edges as a ranked list of the top 25 edges. **Top-Line LEM Table** presents the edges in a concatenated list of the top three ranked edges for each possible regulator. The number of edges viewed for each option can be adjusted by the user by changing the number in the **Number of Edges** input box.

4. Network finding

1. Create a new IDP configuration file that parametrizes the Network Finding step.

    1. Add the main IDP arguments to the configuration file. Open a new text file in a text editor and repeat step 2.1.1.

    2. Add Network Finding arguments to the configuration file.

    3. In the same text file as in step 4.1.1, type in the order presented "[netper_arguments]", "edge_list_file ="", "node_list_file ="", "seed_net_file ="", "range_operations ="", "numneighbors ="", "maxparams ="", "[[probabilities]]", "addNode ="", "addEdge ="",

"removeNode =", and "removeEdge =" on individual lines, below the main arguments.

4. For "seed_net_file", "edge_list_file" and "node_list_file", after the equal sign, enter the path to and name of the seed network file and the edge and node list files generated in steps 3.9 and 3.10.2.

5. After the equal to sign, for "range_operations", type two numbers separated by a comma. The first and second numbers are the minimum and the maximum number of addition or removal of nodes or edges per network made, respectively.

6. For "numneighbors", after the equal to sign, enter a number that represents how many networks to find in Network Finding.

7. For "maxparams", after the equal to sign, enter a number that represents the maximum number of DSGRN parameters to allow for a network.

8. Enter values between 0 and 1 for each of these arguments: "addNode", "addEdge", "removeNode", and "removeEdge", after the equal to sign. The numbers must sum to 1.

   NOTE: It is highly recommended to review the netper_arguments and netquery_arguments sections in the IDP README to get a better understanding of each argument. See Supplemental File 11 and Supplemental File 12 for examples of a configuration file with the Network Finding arguments specified.

2. Repeat steps 2.2 and 2.3.

3. Run the IDP using the configuration file created in step 4.1 by running this command in the terminal, where <config file name> is the name of the file: *python src/dat2net.py <config file name>*

4. If the IDV is still running, stop it by pressing **Control C** in the terminal window to stop the program. Repeat steps 2.5 and 2.6.

5. With the IDV open in the browser, click on the **Network Finding** tab and select the network finding folder of interest.

6. Select a network or set of networks to generate an edge prevalence table (Table 1) and to view the networks along with their respective query results.

   1. Two options are available for selecting networks: Option 1 - Input lower and upper bounds on query results by inputting minimum and maximum values in the input boxes corresponding to the x-axis and y-axis of the plot. Option 2 - Click and drag over the scatterplot to draw a box around the networks to be included. After selection or input bounds are entered, press the **Get Edge Prevalence From Selected Networks** button.

NOTE: If more than one DSGRN query was specified, use the radio buttons labeled with the query type to switch between each query's results. The same applies if more than one epsilon (noise level) was specified.

7. Click the arrows beneath the edge prevalence table to move to the next page of the table. Press **Download Table** to download the edge prevalence table.

8. Input an integer in the Network Index input box to display a single network from the selection made in step 4.6. Click on **Download DSGRN NetSpec** to download the displayed network in the DSGRN network specification format.

9. Search networks for similarity to a specified motif or network of interest.

   1. Use the checkboxes corresponding to each edge to select edges to be included in the network or motif used for the similarity analysis. Click on **Submit** to create the similarity scatterplot for the selected motif or network.

      NOTE: Use the arrows in the edge list to sort alphabetically and the arrows beneath the table to move to the next page of the table.

   2. Click and drag over the scatterplot to draw a box around the networks to be included to select a network or set of networks to generate an edge prevalence table and to view the networks along with their respective query results.

      NOTE: If more than one DSGRN query was specified, use the radio buttons labeled with the query type to switch between each query's results. The same applies if more than one epsilon (noise level) was specified.

   3. Repeat steps 4.7 and 4.8 to download the edge prevalence table and the displayed network for the similarity analysis, respectively.

Representative Results

The steps described textually above and graphically in Figure 1 were applied to the core oscillating GRN of the yeast cell-cycle to see if it is possible to discover functional GRN models that are capable of producing the dynamics observed in time series gene expression data collected in a yeast cell-cycle study[16]. To illustrate how the IDV can clarify and improve IDP output, the results, after performing this analysis in two ways, were compared: 1) running all steps of the IDP in one pass without the IDV and 2) stepping through the IDP with the aid of the IDV, which allows the adjustment of intermediate results both by incorporating prior biological knowledge and by making refined choices based on IDP outputs. The well-studied yeast cell-cycle GRN used as an example has many of its regulatory relationships experimentally verified. If a different and/or less annotated organism or biological process is being studied, the choices on how intermediate results or parameters are adjusted might be different. To illustrate one type of query that can be used to assess networks, the robustness of each network was measured to support stable oscillations and match the observed transcriptional dynamics of their nodes across model parameters.

Gene expression time series data of two replicate series were taken from Orlando 2008[16] and preprocessed to remove any gene expression associated with the cell-cycle synchronization method applied in the original experiment (Supplemental File 1 and Supplemental File 2). An annotation file was created containing all the genes in the time series data that are supported by both DNA binding and expression evidence found in Yeastract[17] and thus could function as a regulator in a GRN. *TOS4, PLM2,* and *NRM1* were also included as regulators, even though they were not found in Yeastract to have both types of evidence, because they are believed to be important for the yeast core GRN based on evidence in the literature[18,19] (Supplemental File 3). All regulators were labeled as both an activator and repressor as well as targets.

The IDP was first parameterized to run through all steps of the IDP, that is Node, Edge, and Network Finding. A set of arguments was selected that appeared appropriate based on the current understanding of the yeast cell-cycle GRN, a small set of genes participating in a strongly connected network (Supplemental File 4). This understanding mostly influenced the Node and Edge Finding choices. The probabilities parameters in Network Finding were based on the assumption that only true genes and regulatory interactions would be passed onto Network Finding. This fully parameterized run of the IDP produced results for Node and Edge Finding (Figure 2B,C), yet in Network Finding no model-admissible networks were discovered (Figure 2A,D). Model admissibility is explained in the code documentation of the python module dsgrn_net_gen[14], a dependency of the IDP. Briefly, networks that contain self-repressing edges or have too many inputs or outputs at a single node are not queryable by the DSGRN software (Table 1). The IDP gives many reasons why model-admissible networks may not be found and describes troubleshooting steps to resolve the issue(s). Essentially, this involves changing parameters and/or input files and rerunning the respective IDP step, and examining results. The IDV was used to make this process less tedious and time-consuming.

The Node Finding results were loaded into the IDV to examine the genes being passed to the Edge Finding step of the IDP. The nodes given by IDP are the top N genes as ranked by DLxJTK (Table 1), N being specified by the user, however, this gene list may not be appropriate for the goal of the analysis. Without prior biological knowledge, automatic selection of nodes using only DLxJTK scores returned a gene with limited evidence of a role in the yeast cell-cycle (RME1), while a few known cell-cycle transcriptional regulators were not highly ranked (Figure 2B). Yeastract experimental evidence was used to select from among the highest-ranking genes by DLxJTK those with cell-cycle annotation. These genes are *SWI4, YOX1, YHP1, HCM1, FKH2, NDD1,* and *SWI5*. Their known regulatory relationships can be seen in Figure 3. *FKH2* does not appear in the top ten genes (dlxjtk_cutoff was set to ten in Supplemental File 4) as ranked by DLxJTK, so the gene list was extended using the IDV until *FKH2* was found (Figure 4). Several of the additional genes in the extended gene list are known core genes and would have been missed without investigating the Node Finding results. While more known core genes have been found by extending the gene list down the DLxJTK ranked list, the focus was kept on the genes of interest. Therefore, some high-ranking genes were deselected, resulting in a gene list (Supplemental File 5) containing seven genes (Figure 4). A new annotation file was created (Supplemental File 6) based on these seven genes, each gene was labeled as a target, and the regulator type was specified using Yeastract. The new gene list and annotation file were downloaded for subsequent use in the next IDP step, Edge Finding. Without the IDV, the procedure of adding and removing genes from the gene list and annotation file would require modest coding skills.

A new IDP configuration file was parameterized for just the Edge Finding step (Supplemental File 7), with the new gene list and annotation file. After completion of the IDP with the new configuration file, results were loaded into the IDV (Figure 5A). As the Network Finding step searches stochastically around the network space of the seed network supplied to it, providing a good seed network can be important. A good seed network can be thought of as one that contains true edges. With the IDV and using online databases such as Yeastract and the Saccharomyces Genome Database (SGD)[20], the seed network can be viewed and adjusted using the regulatory relationships from LEM (Table 1) that have experimental evidence. As an example, the edge YHP1 = tf_act(HCM1) was deselected because there is no documented evidence of this relationship (Figure 5B) in Yeastract. The edge SWI5 = tf_act(FKH2) was added as there is documented evidence of this relationship[21]. Once the seed network (Table 1) was satisfactory, the DSGRN network specification file for the network was downloaded (Supplemental File 8).

Without the IDV, there is a higher chance of edges for which there is no experimental evidence being used to construct the seed network. As can be seen in Figure 2C, the seed network generated in the Edge Finding step from running the IDP nonstop through each step contains an edge, SWI4 = tf_rep(NDD1), that is not supported by experimental evidence in Yeastract, likely because *NDD1* is known to be a transcriptional activator[22]. This information was not encoded in the annotation file in the non-stop run, which allowed all regulators to be both activators and repressors.

Using the IDV, a seed network was manually curated that is a subnetwork of Figure 3, and the remaining four edges were placed in the edge list used for sampling network space (YHP1 = tf_act(SWI4), YOX1 = tf_act(SWI4), SWI4 = tf_rep(YOX1), SWI5 = tf_act(NDD1)). Selecting edges based on prior biological knowledge can also be used to build the edge list; however, in this case, the top 20 edges from the LEM Summary Table view were selected (Supplemental File 9). The node list file is created from the selected edges automatically (Supplemental File 10). The ODE parameters from LEM can also be used to filter edges if one believes the inferred parameters in the ODE model are not biologically realistic, but this information was not used here.

Next, a new IDP configuration file was parameterized for the Network Finding step using the three new files. As the seed network was created with edges well-supported by experimental evidence, the inclusion of these edges in all networks was desired. Thus, the Network Finding probabilities were set to allow the addition but not the removal of nodes and edges (Supplemental File 11). The Network Finding parameter numneighbors was set to search for 2,000 networks. After running the IDP, 37 model-admissible networks were found in the Network Finding step, as opposed to the non-stop run that had zero. Loading the Network Finding results into the IDV, 64% (24) of these 37 networks had the capacity to stably oscillate (Figure 6A). Of these 24 networks, the best performers were two networks that matched the data at 50% of their stably oscillating model parameters (Figure 6B).

The Edge Prevalence Table (Table 1) tabulates the number of times an edge occurs in a selected collection of networks, giving an indication of its prevalence in high-performing networks. Examining the Edge Prevalence Table produced by selecting the previous two networks in the scatter plot reveals that all the seed network edges are present in each of the two networks, as expected, along with two non-seed network edges (Figure 6B), SWI4 = tf_act(SWI5) and HCM1 = tf_rep(YHP1). Neither of these two edges had evidence supporting them in Yeastract. As such a small amount of network space was explored so it is difficult to assess the importance of edges and nodes in producing the observed dynamics.

Only 37 model-admissible networks were found in Network Finding even though the parameter numneighbors was set to 2,000, which suggests that the network search may have been unduly limited. As described in the documentation for the dsgrn_net_gen python module in the IDP, the issue could be related to the seed network, edge list, node list, Network Finding parameter choices, or some combination of these. To investigate, the same seed network, edge list, and node list as before were used, but the Network Finding parameters were altered by adding the ability to remove edges during network generation (Supplemental File 12). Loading the new Network Finding results into the IDV shows that 612 networks were found in this step, with 67% (411) of these networks having the capacity to stably oscillate (Figure 7A). Interestingly, 13% (82) of the networks that were capable of stable oscillatory dynamics were not capable of producing dynamics similar to those seen in the data (Figure 7B). Of the 411 networks, 30% (124) exhibited robust matches to data (i.e., more than 50% of their stably oscillating model parameters exhibited a data match) (Figure 7C).

The edge prevalence numbers generated by the second round of Network Finding are now based on a much larger selection of networks and can be more confidently used in assessing the importance of a regulatory relationship in a GRN. For instance, HCM1 = tf_rep(YHP1) is still represented highly in networks that produce robust dynamics, suggesting that this relationship could be worth investigating experimentally (Figure 7C). Further examination of the Edge Prevalence Table (based on the 124 networks mentioned above) revealed that the edges SWI4 = tf_rep(YOX1) and YOX1 = tf_act(SWI4) are not highly ranked yet the edges SWI4 = tf_rep(YHP1) and YHP1 = tf_act(SWI4) are highly ranked (Figure 7C). Negative feedback is important for producing oscillatory dynamics[23] and both of these sets of regulatory relationships provide this function in the GRN in Figure 3. Finding if a network exists that contains all four of these edges could provide some insight into why these do not frequently exist together in the collection of GRN models; however, clicking through individual networks would be tedious. Instead, the Similarity Analysis portion of the Network Finding page was used to search for networks that may contain all four edges (Figure 7D). Examining the scatter plot that displays how similar the 612 networks are to a motif of these four edges versus the percentage of the model parameter space that matches the observed dynamics reveals that only 0.65% (4) of the 612 networks contain all four of these edges (Figure 7D). This suggests a testable hypothesis that only one of the two negative feedback loops is needed for a network of this size to produce the observed dynamics. This hypothesis can be further investigated computationally by reparameterization of IDP steps and a more exhaustive search of network space or experimentally, such as gene knockouts. All results from this analysis can be found in Supplemental File 13.

Discussion

The inference of GRNs is an important challenge in systems biology. The IDP generates model GRNs from gene expression data using a sequence of tools that utilize the data in increasingly complex ways. Each step requires decisions about how to process the data and what elements (genes, functional interactions) will be passed to the next layer of the IDP. The impacts of these decisions on IDP results are not as obvious. To help in this regard, the IDV provides useful interactive visualizations of the outputs from individual steps of the GRN inference tools within the IDP. The IDV streamlines and facilitates the process of evaluating results from these computational inference methods to speed up experimentation and inform analysis choices, which in turn will allow for the accelerated production of high-confidence network models and hypotheses. The IDV also implements features that expand on the functionality of the IDP, including filtering edges by LEM ODE parameter choices, binning of genes by their expression time, and clustering networks based on similarity to a motif or network. Importantly, the IDV allows for manual interventions between each IDP step, which allows the user to easily incorporate human knowledge and prior information from the literature in ways that cannot be easily automated. A naive run of the IDP will not natively incorporate this information, so the use of the IDV will increase the confidence in the results whenever information specific to the experiment is available. Overall, using the IDV in conjunction with the IDP allows users to create network hypotheses for biological processes with greater confidence, even with little or no knowledge of the true GRN.

There are three critical steps in the IDV. The first is evaluating IDP Node Finding results in the IDV. IDV’s Node Finding page can produce a new gene list and, if desired, a gene annotation file. Curating a new gene list is a critical step as it greatly reduces the potential network space by limiting which genes are allowed to be modeled as GRN targets and/or regulators. Additionally, as GRNs are mostly made up of transcription factors, having gene annotations will greatly help in creating coherent GRN models.

The next step is evaluating IDP Edge Finding results in the IDV. Curating a new seed network is a critical step since it localizes the region of network space that will be sampled in the Network Finding step. However, knowing where to start isn’t always obvious, so it is recommended to use edges that have some form of experimental evidence to provide confidence that one is starting in a region of network space that contains high confidence edges. The IDV’s Edge Finding page enables easy assembly of seed networks and generates the associated DSGRN network specification file as well as node and edge lists.

The last step is evaluating IDP Network Finding results in the IDV. IDV’s Network Finding page allows for easy exploration of sampled networks and their associated scores that estimate the capacity of the network to produce the observed dynamics. While Node and Edge Finding will always return results (if at least two genes are passed on from Node Finding), Network Finding can return zero results. Therefore, knowing whether adjustments in parameters are needed will be more obvious in Network Finding than in Node and Edge Finding. Such occurrences of few to no networks found could be a result of constraints placed on what networks can be analyzed. These constraints are: 1) whether or not the networks are always strongly connected, 2) the minimum and the maximum number of input edges to each node, 3) the probabilities of adding and removing nodes and edges, and 4) the number of additions and removals of nodes and edges allowed. If few or no model-admissible networks are found, as in Figure 2, then referring to the IDP documentation is recommended for guidance on reparameterization of any or all steps of the IDP with subsequent evaluation of results in the IDV.

A current limitation of this approach is that the Node Finding page is mostly focused on oscillatory dynamics, such as those seen in the transcriptional programs of the cell-cycle and circadian clock. In particular, the IDP Node Finding step is currently configured to search for genes that exhibit oscillatory dynamics at a specified period. As the IDP expands to include analyses that can quantify different types of transcriptional dynamics, so too will the IDV be updated to support visualization and interrogation of these other behaviors. The size of networks searched for and analyzed in the Network Finding step are currently limited to networks of smaller size, e.g., around 10 genes. This is a necessity as computations in the DSGRN scale occur combinatorially. Another limitation is that exploring model parameter space for a selected network is not possible in the IDV. However, the DSGRN network specification file for a given network can be downloaded and the dynamics associated with each model parameter can be visualized on the DSGRN Visualization website (https://sites.math.rutgers.edu/~gameiro/dsgrn_viz/). Lastly, the IDV has been tested using Linux (Ubuntu) and iOS (Big Sur) systems. The IDV has been tested on Windows 10 using the Windows Subsystem for Linux (WSL), which allows Windows 10 users to run Linux and the IDV without the need for a different computer, a virtual machine, or a dual-boot setup. IDV does not currently run on native Windows.

Studying GRNs is difficult due to their inherent complexity and useful inference tools such as the IDP can be difficult to understand and deploy with confidence. The IDV provides a method to reduce the complexity of studying GRNs inferred using the IDP while facilitating the inclusion of additional information beyond gene expression dynamics. Using the IDV in conjunction with the IDP as described here will empower researchers to develop and analyze functional models of well-studied systems, such as the human cell-cycle. Furthermore, these tools will generate testable hypotheses for less understood processes, such as the malaria intra-erythrocytic development cycle, which is suspected to be controlled by a GRN[24] but for which a model is yet to be proposed.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

This work was funded by the NIH grant R01 GM126555-01 and NSF grant DMS-1839299.

References

1. Karlebach G, Shamir R Modelling and analysis of gene regulatory networks. Nature Reviews Molecular Cell Biology. 9 (10), 770–780 (2008). [PubMed: 18797474]
2. Aijö T, Lähdesmäki HLearning gene regulatory networks from gene expression measurements using non-parametric molecular kinetics. Bioinformatics. 25 (22), 2937–2944 (2009). [PubMed: 19706742]
3. Huynh-Thu VA, Sanguinetti G Combining tree-based and dynamical systems for the inference of gene regulatory networks. Bioinformatics. 31 (10), 1614–1622 (2015). [PubMed: 25573916]
4. Oates CJ et al. Causal network inference using biochemical kinetics. Bioinformatics. 30 (17), i468–i474 (2014). [PubMed: 25161235]
5. Marbach D et al. Wisdom of crowds for robust gene network inference. Nature Methods. 9 (8), 796–804 (2012). [PubMed: 22796662]
6. inherent_dynamics_pipeline. https://gitlab.com/biochron/inherent_dynamics_pipeline. (2021).
7. Motta FC, Moseley RC, Cummins B, Deckard A, Haase SB Conservation of dynamic characteristics of transcriptional regulatory elements in periodic biological processes. bioRxiv. (2020).
8. LEMpy. https://gitlab.com/biochron/lempy. (2021).
9. McGoff KA et al. The local edge machine: inference of dynamic models of gene regulation. Genome Biology. 17, 214 (2016). [PubMed: 27760556]
10. Cummins B, Gedeon T, Harker S, Mischaikow K Model rejection and parameter reduction via time series. SIAM Journal on Applied Dynamical Systems. 17 (2), 1589–1616 (2018). [PubMed: 31762711]
11. Cummins B, Gedeon T, Harker S, Mischaikow K Database of Dynamic Signatures Generated by Regulatory Networks (DSGRN). Lecture Notes in Computer Science. (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 10545 LNBI, 300–308 (2017).
12. Cummins B, Gedeon T, Harker S, Mischaikow K DSGRN: Examining the dynamics of families of logical models. Frontiers in Physiology. 9, 549 (2018). [PubMed: 29875674]
13. DSGRN. https://github.com/marciogameiro/DSGRN. (2021).
14. dsgrn_net_gen. https://github.com/breecummins/dsgrn_net_gen. (2021).
15. dsgrn_net_query. https://github.com/breecummins/dsgrn_net_query. (2021).
16. Orlando DA et al. Global control of cell-cycle transcription by coupled CDK and network oscillators. Nature. 453 (7197), 944–947 (2008). [PubMed: 18463633]
17. Monteiro PT et al. YEASTRACT+: a portal for cross-species comparative genomics of transcription regulation in yeasts. Nucleic Acids Research. 48 (D1), D642–D649 (2020). [PubMed: 31586406]
18. Bruin R. A. M. de et al. Constraining G1-specific transcription to late G1 phase: The MBF-associated corepressor Nrm1 acts via negative feedback. Molecular Cell. 23 (4), 483–496 (2006). [PubMed: 16916637]
19. Horak CE et al. Complex transcriptional circuitry at the G1/S transition in Saccharomyces cerevisiae. Genes & Development. 16 (23), 3017–3033 (2002). [PubMed: 12464632]
20. Cherry JM et al. Saccharomyces genome database: The genomics resource of budding yeast. Nucleic Acids Research. 40 (Database issue), D700–D705 (2012). [PubMed: 22110037]
21. Zhu G et al. Two yeast forkhead genes regulate the cell cycle and pseudohyphal growth. Nature. 406 (6791), 90–94 (2000). [PubMed: 10894548]
22. Loy CJ, Lydall D, Surana U NDD1, a high-dosage suppressor of cdc28–1N, is essential for expression of a subset of late-S-phase-specific genes in saccharomyces cerevisiae. Molecular and Cellular Biology. 19 (5), 3312–3327 (1999). [PubMed: 10207056]

23. Cho CY, Kelliher CM, Hasse SB The cell-cycle transcriptional network generates and transmits a pulse of transcription once each cell cycle. Cell Cycle. 18 (4), 363–378 (2019). [PubMed: 30668223]

24. Smith LM et al. An intrinsic oscillator drives the blood stage cycle of the malaria parasite Plasmodium falciparum. Science. 368 (6492), 754–759 (2020). [PubMed: 32409472]

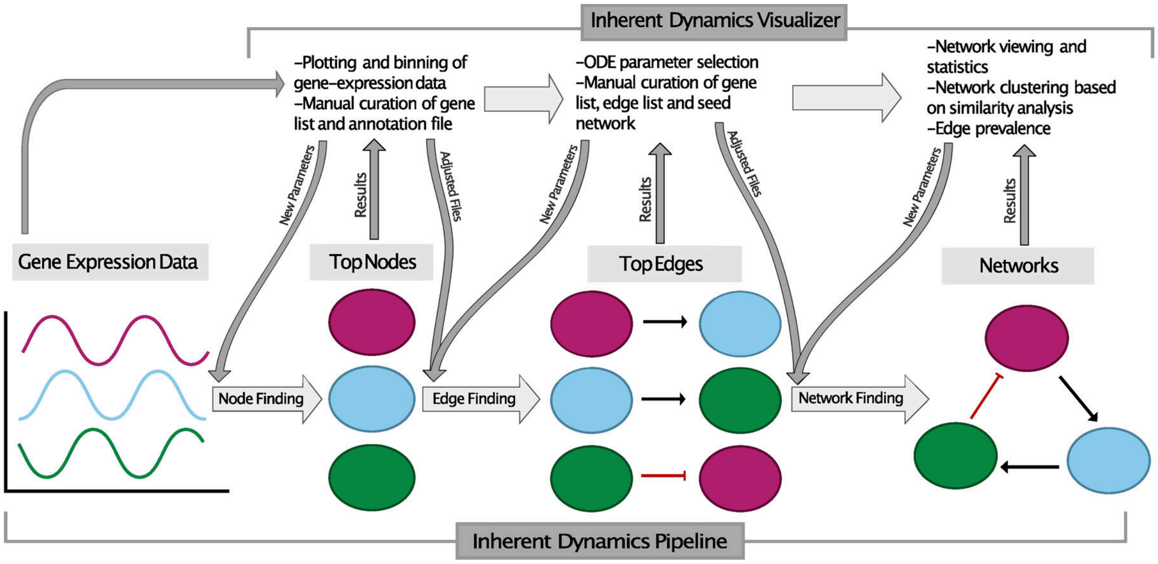
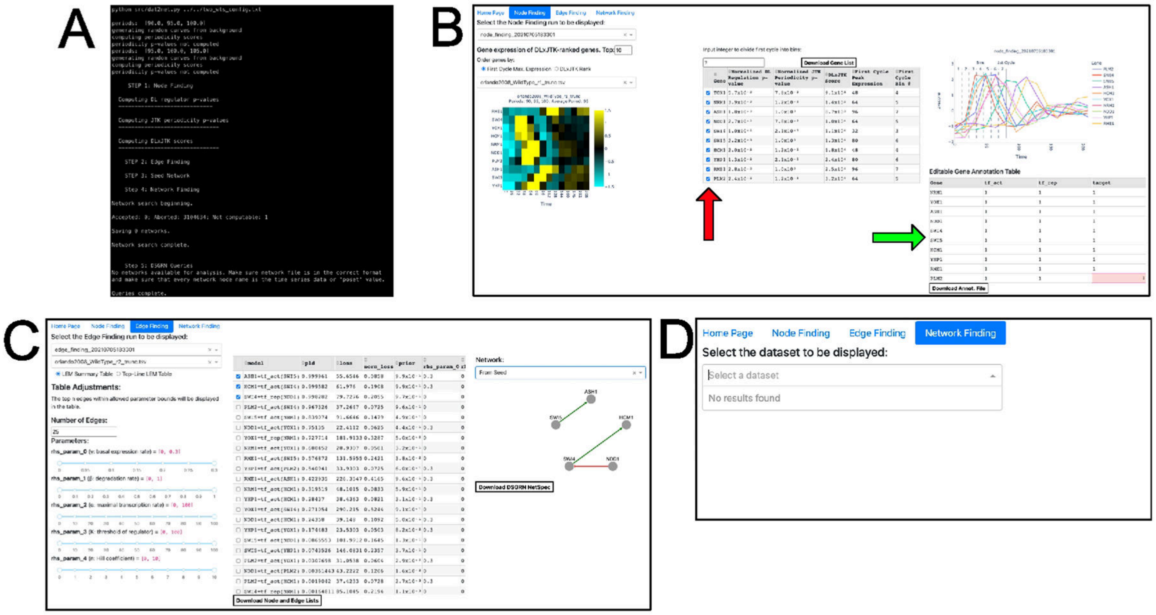

Inherent Dynamics Visualizer
–Plotting and binning of gene–expression data
–Manual curation of gene list and annotation file
–ODE parameter selection
–Manual curation of gene list, edge list and seed network
–Network viewing and statistics
–Network clustering based on similarity analysis
–Edge prevalence
New Parameters
Results
Adjusted Files
New Parameters
Results
Adjusted Files
New Parameters
Results
Gene Expression Data
Top Nodes
Top Edges
Networks
Node Finding
Edge Finding
Network Finding
Inherent Dynamics Pipeline

Figure 1: IDP and IDV workflow overview.

The bottom row depicts the three major steps of the IDP: Node, Edge, and Network Finding. The top row depicts the major steps of the IDV and describes various ways a user can interact with the results. The dark gray arrows between the two depict how the IDV and the IDP can work synergistically to allow users to make informed decisions for each step of the IDP, with individual IDP steps providing results for the visualizations in the IDV, individual IDV steps allowing for the input of new or adjusted parameters and adjusted results and inputs for the subsequent IDP step.



```
python src/dat2net.py ../../two_wts_config.txt

periods:  [90.0, 95.0, 100.0]
generating random curves from background
computing periodicity scores
periodicity p-values not computed
periods:  [95.0, 100.0, 105.0]
generating random curves from background
computing periodicity scores
periodicity p-values not computed

     STEP 1: Node Finding

  Computing DL regulator p-values
  ----------------------------

  Computing JTK periodicity p-values
  ------------------------------

  Computing DLxJTK scores
  ------------------------------

    STEP 2: Edge Finding

    STEP 3: Seed Network

    Step 4: Network Finding

Network search beginning.

Accepted: 0; Aborted: 3104634; Not computable: 1

Saving 0 networks.

Network search complete.


    Step 5: DSGRN Queries
No networks available for analysis. Make sure network file is in the correct format
and make sure that every network node name is the time series data or 'poset' value.

Queries complete.
```

Select the Node Finding run to be displayed:

node_finding_20210705183301

Gene expression of DLxJTK-ranked genes. Top: 10

Order genes by:

● First Cycle Max. Expression ○ DLxJTK Rank

orlando2008_WildType_r1_trunc.tsv

orlando2008_WildType_r1_trunc
Periods: 90, 95, 100. Average Period: 95
RME1
SWI4
YOX1
HCM1
NRM1
NDD1
PLM2
ASH1
SWI5
YHP1
1.5
1
0.5
0
-0.5
-1
-1.5
Time

Input integer to divide first cycle into bins:

7

Download Gene List

	Gene	Normalized DL Regulation p-value	Normalized JTK Periodicity p-value	DLxJTK Score	First Cycle Peak Expression	First Cycle Bin #
☑	YOX1	5.7x10[illegible]	7.8x10[illegible]	9.1x10[illegible]	48	4
☑	NRM1	3.9x10[illegible]	1.2x10[illegible]	1.4x10[illegible]	64	5
☑	ASH1	1.8x10[illegible]	1.0x10[illegible]	8.7x10[illegible]	96	7
☑	NDD1	2.7x10[illegible]	7.8x10[illegible]	1.0x10[illegible]	64	5
☑	SWI4	1.0x10[illegible]	2.1x10[illegible]	1.1x10[illegible]	32	3
☑	SWI5	2.2x10[illegible]	1.0x10[illegible]	1.3x10[illegible]	80	6
☑	HCM1	2.0x10[illegible]	1.2x10[illegible]	1.8x10[illegible]	48	4
☑	YHP1	1.3x10[illegible]	2.1x10[illegible]	2.4x10[illegible]	80	6
☑	RME1	2.8x10[illegible]	1.0x10[illegible]	2.5x10[illegible]	96	7
☑	PLM2	2.4x10[illegible]	1.2x10[illegible]	3.2x10[illegible]	64	5

Editable Gene Annotation Table

Gene	tf_act	tf_rep	target
NRM1	1	1	1
YOX1	1	1	1
ASH1	1	1	1
NDD1	1	1	1
SWI4	1	1	1
SWI5	1	1	1
HCM1	1	1	1
YHP1	1	1	1
RME1	1	1	1
PLM2	1	1	1

Download Annot. File

Select the Edge Finding run to be displayed:

edge_finding_20210705183301

orlando2008_WildType_r2_trunc.tsv

● LEM Summary Table ○ Top-Line LEM Table

Table Adjustments:

The top n edges within allowed parameter bounds will be displayed in the table.

Number of Edges:

25

Parameters:

rhs_param_0 (v: basal expression rate) = [0, 0.3]

rhs_param_1 (β: degradation rate) = [0, 1]

rhs_param_2 (α: maximal transcription rate) = [0, 100]

rhs_param_3 (K: threshold of regulator) = [0, 100]

rhs_param_4 (n: Hill coefficient) = [0, 10]

	model	pld	loss	norm_loss	prior	rhs_param_0	r
☑	ASH1=tf_act(SWI5)	0.999961	55.6546	0.0858	9.9x10[illegible]	0.3	0
☑	HCM1=tf_act(SWI4)	0.999582	61.976	0.1908	9.9x10[illegible]	0.3	0
☑	SWI4=tf_rep(NDD1)	0.998282	79.7236	0.2055	9.7x10[illegible]	0	0
☐	PLM2=tf_act(SWI4)	0.967326	37.2617	0.0725	9.6x10[illegible]	0	0
☐	SWI5=tf_act(NRM1)	0.839074	91.6646	0.1479	4.9x10[illegible]	0	0
☐	NDD1=tf_act(YOX1)	0.75135	22.4112	0.0625	4.4x10[illegible]	0.3	0
☐	YOX1=tf_rep(NRM1)	0.727714	181.9133	0.3287	5.0x10[illegible]	0	0
☐	NRM1=tf_act(YOX1)	0.680452	28.9307	0.0561	3.2x10[illegible]	0	0
☐	RME1=tf_act(SWI5)	0.576872	131.5955	0.2421	3.8x10[illegible]	0	0
☐	YHP1=tf_act(PLM2)	0.540941	33.9303	0.0725	6.0x10[illegible]	0.3	0
☐	RME1=tf_act(ASH1)	0.422935	226.3547	0.4165	9.6x10[illegible]	0.3	0
☐	NRM1=tf_act(HCM1)	0.319519	48.1015	0.0833	5.9x10[illegible]	0	0
☐	YHP1=tf_act(HCM1)	0.28437	38.4363	0.0821	3.1x10[illegible]	0.3	0
☐	YOX1=tf_act(SWI4)	0.271054	290.215	0.5246	9.1x10[illegible]	0	0
☐	NDD1=tf_act(HCM1)	0.24358	39.148	0.1092	5.0x10[illegible]	0.3	0
☐	YHP1=tf_act(YOX1)	0.174483	23.5303	0.0503	8.2x10[illegible]	0.3	0
☐	SWI5=tf_act(NDD1)	0.0865553	101.9912	0.1645	1.3x10[illegible]	0	0
☐	SWI5=tf_act(YHP1)	0.0743526	146.0831	0.2357	3.7x10[illegible]	0	0
☐	PLM2=tf_act(YOX1)	0.0307698	31.0538	0.0606	2.9x10[illegible]	0.3	0
☐	NDD1=tf_act(PLM2)	0.00361443	43.2222	0.1206	1.6x10[illegible]	0	0
☐	PLM2=tf_act(HCM1)	0.0019042	37.4233	0.0728	2.7x10[illegible]	0.3	0
☐	SWI4=tf_rep(NRM1)	0.00164811	85.1045	0.2156	1.1x10[illegible]	0	0

Download Node and Edge Lists

Network:

From Seed

Download DSGRN NetSpec

Select the dataset to be displayed:

Select a dataset

No results found

Figure 2: Example of results from running every step of the IDP consecutively without using the IDV between steps.

(**A**) A screenshot of the terminal output from running every IDP step consecutively. The IDP ran to completion, but zero networks were found during the Network Finding step. (**B**) Node Finding results directory node_finding_20210705183301 (Supplemental File 13) loaded into the IDV. All the genes in the gene list table were selected (red arrow) to show their respective expression profiles in the line graph and to generate an annotation table. The annotation table was filled in to reflect how the genes are labeled in the original annotation file (green arrow). (**C**) Edge Finding results directory edge_finding_20210705183301 (Supplemental File 13) loaded into the IDV. (**D**) Network Finding results directory network_finding_20210705183301 (Supplemental File 13) loaded into the IDV. The Network Finding page shows no results, suggesting either reparameterization of the Network Finding step or reevaluating the Node or Edge Finding step is needed. The IDP documentation contains troubleshooting steps for helping the user determine what they could try next.

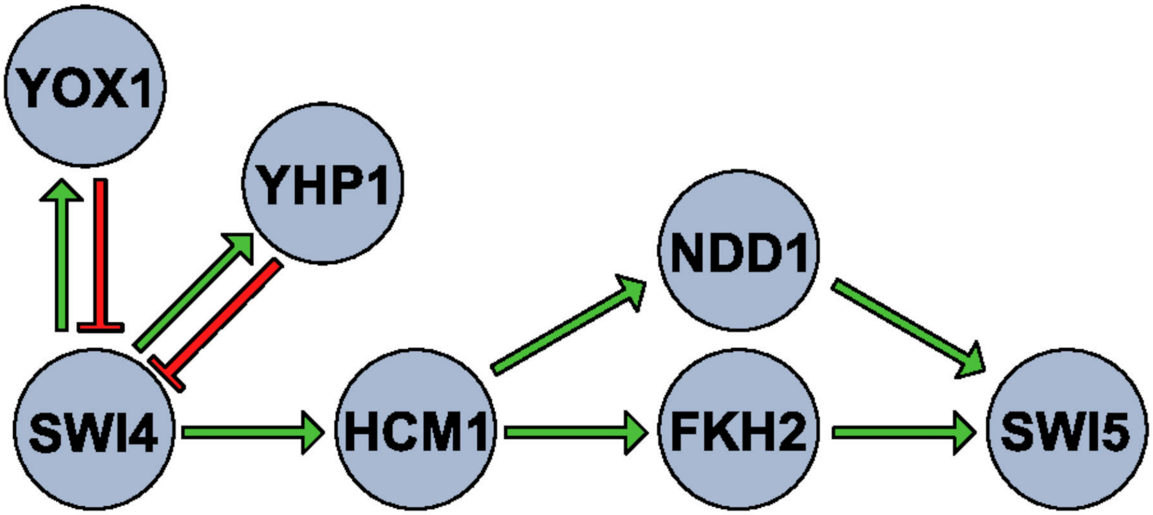
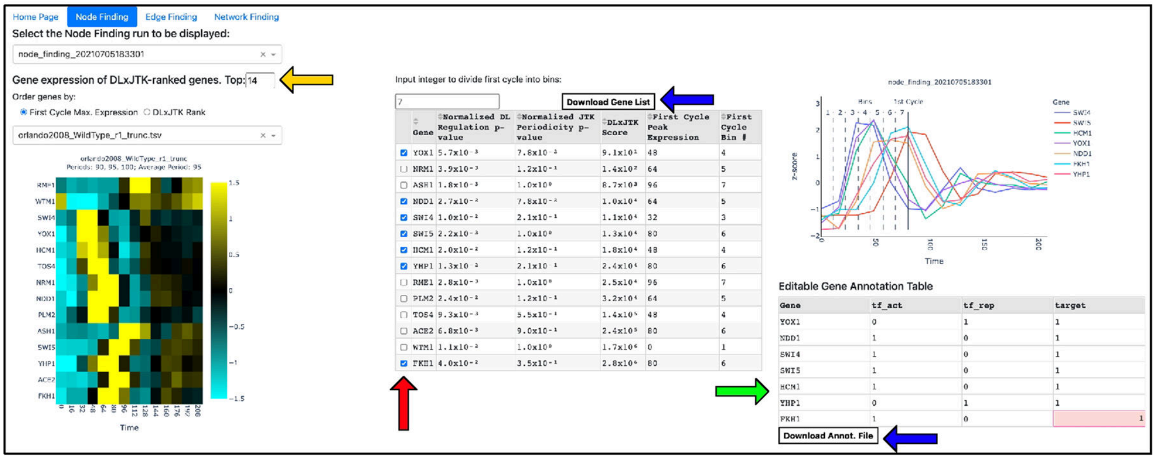


Figure 3: A yeast cell-cycle GRN model.

A set of known yeast cell-cycle regulators were selected from SGD and known regulatory relationships between genes were extracted from Yeastract.



Select the Node Finding run to be displayed:

node_finding_20210705183301

Gene expression of DLxJTK-ranked genes. Top: 14

Order genes by:

● First Cycle Max. Expression ○ DLxJTK Rank

orlando2008_WildType_r1_trunc.tsv

Input integer to divide first cycle into bins:

7

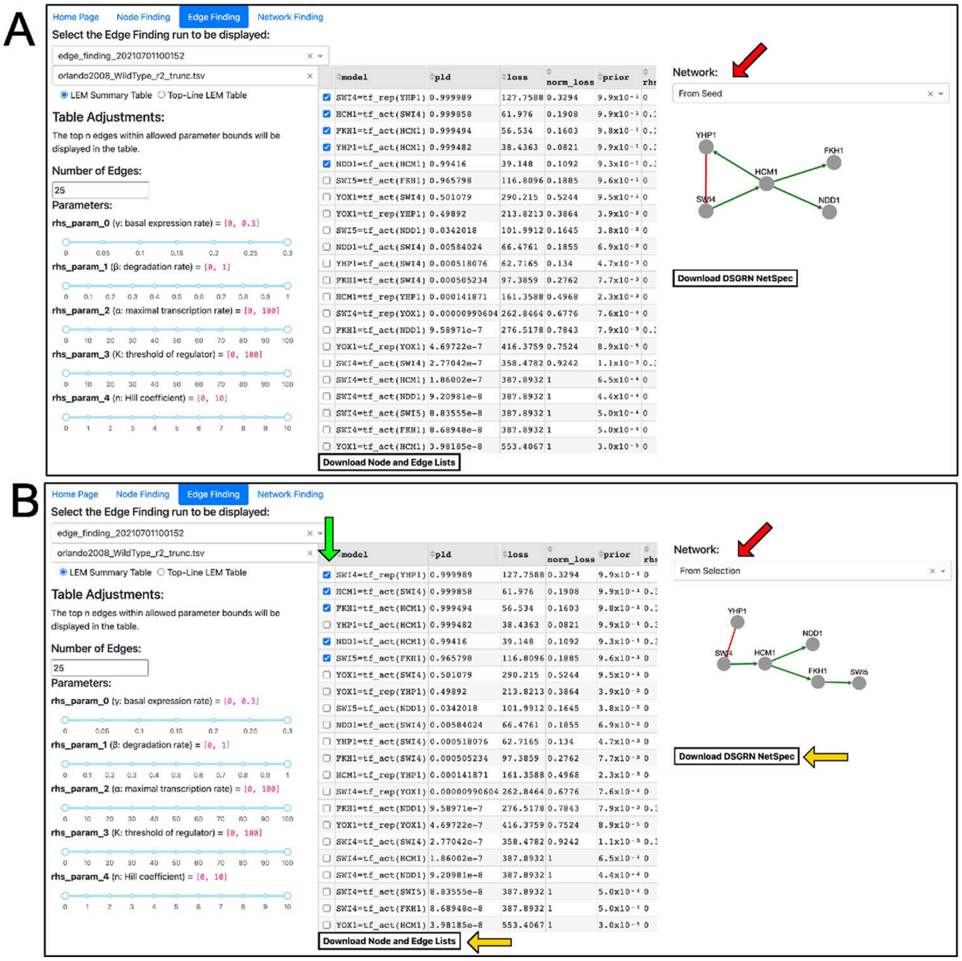
	Gene	Normalized DL Regulation p-value	Normalized JTK Periodicity p-value	DLxJTK Score	First Cycle Peak Expression	First Cycle Bin #
☑	YOX1	$5.7x10^{-3}$	$7.8x10^{-2}$	$9.1x10^{1}$	48	4
□	NRM1	$3.9x10^{-3}$	$1.2x10^{-1}$	$1.4x10^{2}$	64	5
□	ASH1	$1.8x10^{-3}$	$1.0x10^{0}$	$8.7x10^{3}$	96	7
☑	NDD1	$2.7x10^{-2}$	$7.8x10^{-2}$	$1.0x10^{4}$	64	5
☑	SWI4	$1.0x10^{-2}$	$2.1x10^{-1}$	$1.1x10^{4}$	32	3
☑	SWI5	$2.2x10^{-3}$	$1.0x10^{0}$	$1.3x10^{4}$	80	6
☑	HCM1	$2.0x10^{-2}$	$1.2x10^{-1}$	$1.8x10^{4}$	48	4
☑	YHP1	$1.3x10^{-2}$	$2.1x10^{-1}$	$2.4x10^{4}$	80	6
□	RME1	$2.8x10^{-3}$	$1.0x10^{0}$	$2.5x10^{4}$	96	7
□	PLM2	$2.4x10^{-2}$	$1.2x10^{-1}$	$3.2x10^{4}$	64	5
□	TOS4	$9.3x10^{-3}$	$5.5x10^{-1}$	$1.4x10^{5}$	48	4
□	ACE2	$6.8x10^{-3}$	$9.0x10^{-1}$	$2.4x10^{5}$	80	6
□	WTM1	$1.1x10^{-2}$	$1.0x10^{0}$	$1.7x10^{6}$	0	1
☑	FKH1	$4.0x10^{-2}$	$3.5x10^{-1}$	$2.8x10^{4}$	80	6

Editable Gene Annotation Table

Gene	tf_act	tf_rep	target
YOX1	0	1	1
NDD1	1	0	1
SWI4	1	0	1
SWI5	1	0	1
HCM1	1	0	1
YHP1	0	1	1
FKH1	1	0	1

Figure 4: Example of IDP Node Finding results in the IDV.

Loaded into the IDV is the Node Finding results directory node_finding_20210705183301 (Supplemental File 13). The adjusted results after inspecting curated online yeast databases. The gene list table was extended (yellow arrow) to find the remaining gene in the GRN model of Figure 3 and genes were deselected to remove genes not found in the same GRN model (red arrow). The annotation table was filled in based on evidence of regulation for each gene found on Yeastract (green arrow). The new gene list and annotation file were downloaded by selecting their respective download buttons (blue arrows).



A

Select the Edge Finding run to be displayed:

edge_finding_20210701100152

orlando2008_WildType_r2_trunc.tsv

● LEM Summary Table ○ Top-Line LEM Table

Table Adjustments:

The top n edges within allowed parameter bounds will be displayed in the table.

Number of Edges:

25

Parameters:

rhs_param_0 (γ: basal expression rate) = [0, 0.3]

0 0.05 0.1 0.15 0.2 0.25 0.3

rhs_param_1 (β: degradation rate) = [0, 1]

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

rhs_param_2 (α: maximal transcription rate) = [0, 100]

0 10 20 30 40 50 60 70 80 90 100

rhs_param_3 (K: threshold of regulator) = [0, 100]

0 10 20 30 40 50 60 70 80 90 100

rhs_param_4 (n: Hill coefficient) = [0, 10]

0 1 2 3 4 5 6 7 8 9 10

	model	pld	loss	norm_loss	prior	rhs
☑	SWI4=tf_rep(YHP1)	0.999989	127.7588	0.3294	$9.9x10^{-1}$	0
☑	HCM1=tf_act(SWI4)	0.999858	61.976	0.1908	$9.9x10^{-1}$	0.
☑	FKH1=tf_act(HCM1)	0.999494	56.534	0.1603	$9.8x10^{-1}$	0.
☑	YHP1=tf_act(HCM1)	0.999482	38.4363	0.0821	$9.9x10^{-1}$	0.
☑	NDD1=tf_act(HCM1)	0.99416	39.148	0.1092	$9.3x10^{-1}$	0.
☐	SWI5=tf_act(FKH1)	0.965798	116.8096	0.1885	$9.6x10^{-1}$	0
☐	YOX1=tf_act(SWI4)	0.501079	290.215	0.5244	$9.5x10^{-1}$	0
☐	YOX1=tf_rep(YHP1)	0.49892	213.8213	0.3864	$3.9x10^{-2}$	0
☐	SWI5=tf_act(NDD1)	0.0342018	101.9912	0.1645	$3.8x10^{-2}$	0
☐	NDD1=tf_act(SWI4)	0.00584024	66.4761	0.1855	$6.9x10^{-2}$	0
☐	YHP1=tf_act(SWI4)	0.000518076	62.7165	0.134	$4.7x10^{-3}$	0
☐	FKH1=tf_act(SWI4)	0.000505234	97.3859	0.2762	$7.7x10^{-3}$	0
☐	HCM1=tf_rep(YHP1)	0.000141871	161.3588	0.4968	$2.3x10^{-3}$	0
☐	SWI4=tf_rep(YOX1)	0.00000990604	262.8464	0.6776	$7.6x10^{-4}$	0
☐	FKH1=tf_act(NDD1)	9.58971e-7	276.5178	0.7843	$7.9x10^{-3}$	0.
☐	YOX1=tf_rep(YOX1)	4.69722e-7	416.3759	0.7524	$8.9x10^{-5}$	0
☐	SWI4=tf_act(SWI4)	2.77042e-7	358.4782	0.9242	$1.1x10^{-3}$	0.
☐	SWI4=tf_act(HCM1)	1.86002e-7	387.8932	1	$6.5x10^{-4}$	0
☐	SWI4=tf_act(NDD1)	9.20981e-8	387.8932	1	$4.4x10^{-4}$	0
☐	SWI4=tf_act(SWI5)	8.83555e-8	387.8932	1	$5.0x10^{-4}$	0
☐	SWI4=tf_act(FKH1)	8.68948e-8	387.8932	1	$5.0x10^{-4}$	0
☐	YOX1=tf_act(HCM1)	3.98185e-8	553.4067	1	$3.0x10^{-5}$	0

Download Node and Edge Lists

Network:

From Seed

Download DSGRN NetSpec

B

Select the Edge Finding run to be displayed:

edge_finding_20210701100152

orlando2008_WildType_r2_trunc.tsv

● LEM Summary Table ○ Top-Line LEM Table

Table Adjustments:

The top n edges within allowed parameter bounds will be displayed in the table.

Number of Edges:

25

Parameters:

rhs_param_0 (γ: basal expression rate) = [0, 0.3]

0 0.05 0.1 0.15 0.2 0.25 0.3

rhs_param_1 (β: degradation rate) = [0, 1]

0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

rhs_param_2 (α: maximal transcription rate) = [0, 100]

0 10 20 30 40 50 60 70 80 90 100

rhs_param_3 (K: threshold of regulator) = [0, 100]

0 10 20 30 40 50 60 70 80 90 100

rhs_param_4 (n: Hill coefficient) = [0, 10]

0 1 2 3 4 5 6 7 8 9 10

	model	pld	loss	norm_loss	prior	rhs
☑	SWI4=tf_rep(YHP1)	0.999989	127.7588	0.3294	$9.9x10^{-1}$	0
☑	HCM1=tf_act(SWI4)	0.999858	61.976	0.1908	$9.9x10^{-1}$	0.
☑	FKH1=tf_act(HCM1)	0.999494	56.534	0.1603	$9.8x10^{-1}$	0.
☐	YHP1=tf_act(HCM1)	0.999482	38.4363	0.0821	$9.9x10^{-1}$	0.
☑	NDD1=tf_act(HCM1)	0.99416	39.148	0.1092	$9.3x10^{-1}$	0.
☑	SWI5=tf_act(FKH1)	0.965798	116.8096	0.1885	$9.6x10^{-1}$	0
☐	YOX1=tf_act(SWI4)	0.501079	290.215	0.5244	$9.5x10^{-1}$	0
☐	YOX1=tf_rep(YHP1)	0.49892	213.8213	0.3864	$3.9x10^{-2}$	0
☐	SWI5=tf_act(NDD1)	0.0342018	101.9912	0.1645	$3.8x10^{-2}$	0
☐	NDD1=tf_act(SWI4)	0.00584024	66.4761	0.1855	$6.9x10^{-2}$	0
☐	YHP1=tf_act(SWI4)	0.000518076	62.7165	0.134	$4.7x10^{-3}$	0
☐	FKH1=tf_act(SWI4)	0.000505234	97.3859	0.2762	$7.7x10^{-3}$	0
☐	HCM1=tf_rep(YHP1)	0.000141871	161.3588	0.4968	$2.3x10^{-3}$	0
☐	SWI4=tf_rep(YOX1)	0.00000990604	262.8464	0.6776	$7.6x10^{-4}$	0
☐	FKH1=tf_act(NDD1)	9.58971e-7	276.5178	0.7843	$7.9x10^{-3}$	0.
☐	YOX1=tf_rep(YOX1)	4.69722e-7	416.3759	0.7524	$8.9x10^{-5}$	0
☐	SWI4=tf_act(SWI4)	2.77042e-7	358.4782	0.9242	$1.1x10^{-3}$	0.
☐	SWI4=tf_act(HCM1)	1.86002e-7	387.8932	1	$6.5x10^{-4}$	0
☐	SWI4=tf_act(NDD1)	9.20981e-8	387.8932	1	$4.4x10^{-4}$	0
☐	SWI4=tf_act(SWI5)	8.83555e-8	387.8932	1	$5.0x10^{-4}$	0
☐	SWI4=tf_act(FKH1)	8.68948e-8	387.8932	1	$5.0x10^{-4}$	0
☐	YOX1=tf_act(HCM1)	3.98185e-8	553.4067	1	$3.0x10^{-5}$	0

Download Node and Edge Lists

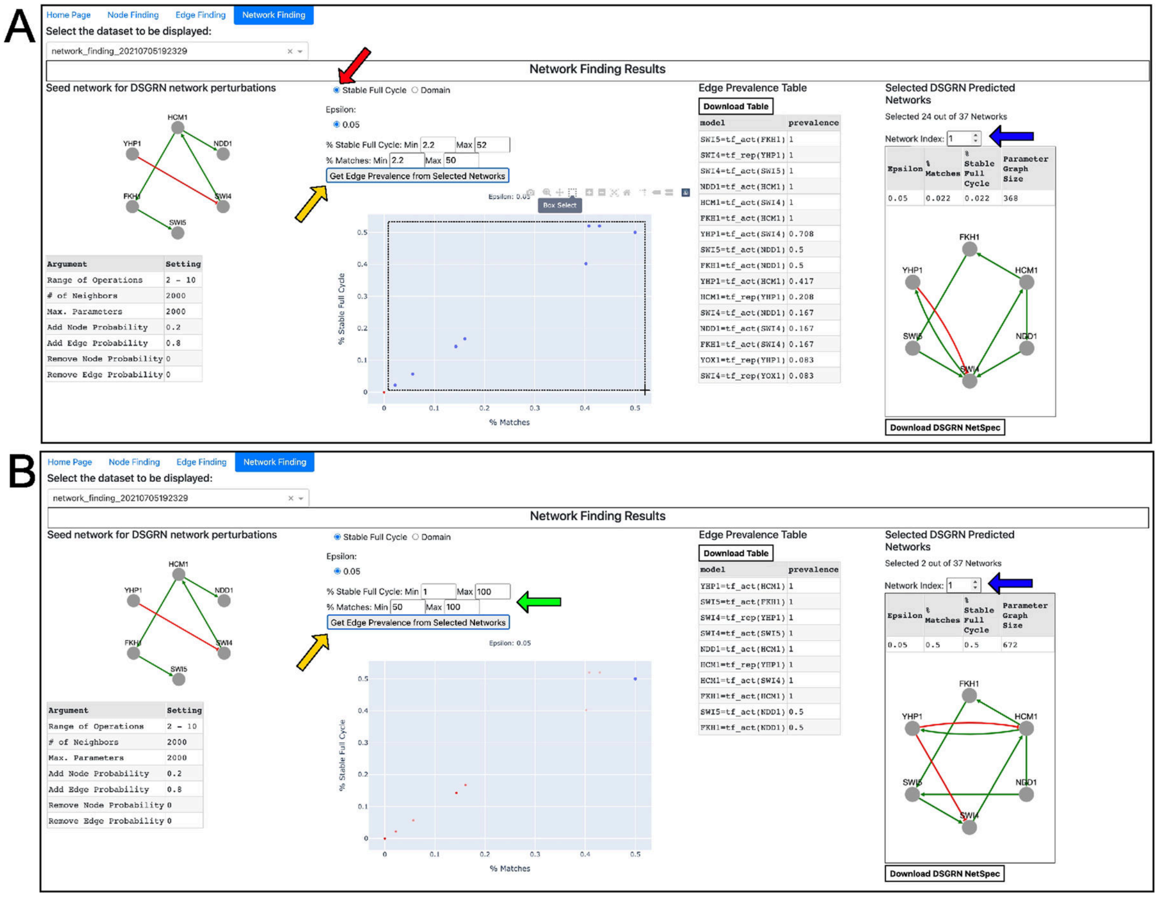
Network:

From Selection

Download DSGRN NetSpec

Figure 5: Example of IDP Edge Finding results in the IDV.

Loaded into the IDV is the Edge Finding results directory edge_finding_20210701100152 (Supplemental File 13). (**A**) The initial result as produced by the IDP. The **Network** dropdown option **From Seed** was selected (red arrow) to view the seed network produced by the IDP based on the arguments in the configuration file used (Supplemental File 7). The selected genes in the edge table are the edges used in the seed network. (**B**) The adjusted results after inspecting the seed network for edges that do not contain experimental evidence. The **Network** dropdown option **From Selection** was selected (red arrow). Edges were selected/deselected from the edge table (green arrow). The seed network, edge list, and node list files were downloaded by clicking their respective buttons (yellow arrows). The edge table shown is for the last time series data as listed in the configuration file two_wts_EdgeFinding_config.txt (Supplemental File 7). It is important when selecting edges for the seed network or edge list based on LEM results to look at the last time series data listed in the configuration file as this output incorporates all preceding datafiles in its inference of regulatory relationships between nodes.



Select the dataset to be displayed:

network_finding_20210705192329

Network Finding Results

Seed network for DSGRN network perturbations

● Stable Full Cycle ○ Domain

Epsilon:

● 0.05

% Stable Full Cycle: Min 2.2 Max 52

% Matches: Min 2.2 Max 50

Get Edge Prevalence from Selected Networks

Epsilon: 0.05

Argument	Setting
Range of Operations	2 – 10
# of Neighbors	2000
Max. Parameters	2000
Add Node Probability	0.2
Add Edge Probability	0.8
Remove Node Probability	0
Remove Edge Probability	0

Edge Prevalence Table

Download Table

model	prevalence
SWI5=tf_act(FKH1)	1
SWI4=tf_rep(YHP1)	1
SWI4=tf_act(SWI5)	1
NDD1=tf_act(HCM1)	1
HCM1=tf_act(SWI4)	1
FKH1=tf_act(HCM1)	1
YHP1=tf_act(SWI4)	0.708
SWI5=tf_act(NDD1)	0.5
FKH1=tf_act(NDD1)	0.5
YHP1=tf_act(HCM1)	0.417
HCM1=tf_rep(YHP1)	0.208
SWI4=tf_act(NDD1)	0.167
NDD1=tf_act(SWI4)	0.167
FKH1=tf_act(SWI4)	0.167
YOX1=tf_rep(YHP1)	0.083
SWI4=tf_rep(YOX1)	0.083

Selected DSGRN Predicted Networks

Selected 24 out of 37 Networks

Network Index: 1

Epsilon	% Matches	% Stable Full Cycle	Parameter Graph Size
0.05	0.022	0.022	368

Download DSGRN NetSpec

Select the dataset to be displayed:

network_finding_20210705192329

Network Finding Results

Seed network for DSGRN network perturbations

● Stable Full Cycle ○ Domain

Epsilon:

● 0.05

% Stable Full Cycle: Min 1 Max 100

% Matches: Min 50 Max 100

Get Edge Prevalence from Selected Networks

Epsilon: 0.05

Argument	Setting
Range of Operations	2 – 10
# of Neighbors	2000
Max. Parameters	2000
Add Node Probability	0.2
Add Edge Probability	0.8
Remove Node Probability	0
Remove Edge Probability	0

Edge Prevalence Table

Download Table

model	prevalence
YHP1=tf_act(HCM1)	1
SWI5=tf_act(FKH1)	1
SWI4=tf_rep(YHP1)	1
SWI4=tf_act(SWI5)	1
NDD1=tf_act(HCM1)	1
HCM1=tf_rep(YHP1)	1
HCM1=tf_act(SWI4)	1
FKH1=tf_act(HCM1)	1
SWI5=tf_act(NDD1)	0.5
FKH1=tf_act(NDD1)	0.5

Selected DSGRN Predicted Networks

Selected 2 out of 37 Networks

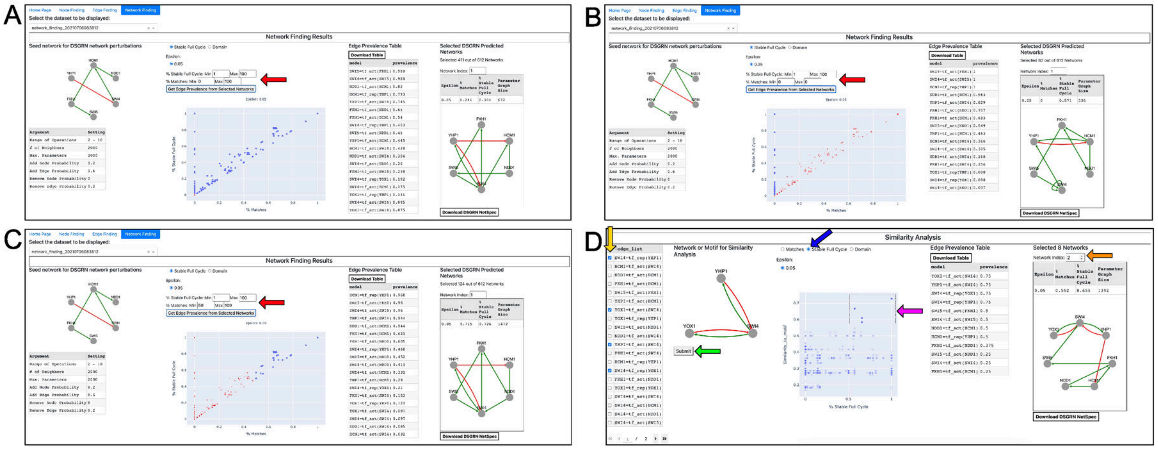
Network Index: 1

Epsilon	% Matches	% Stable Full Cycle	Parameter Graph Size
0.05	0.5	0.5	672

Download DSGRN NetSpec

Figure 6: Example of IDP Network Finding results in the IDV from using the IDP configuration file two_wts_NetFind_rd1_config.txt (Supplemental File 11).

(**A**) The query **Stable Full Cycle** was selected (red arrow) to display the respective data on the y-axis in the scatter plot. Blue dots in the scatter plot represent selected points using the Box Select function for the scatter plot. The dotted selection box was illustrated to show what the box selection looks like. (**B**) The min and max integers for the y-axis and x-axis were manually entered to selected networks within these bounds (green arrow). After each selection, the **Get Edge Prevalence from Selected Networks** button (yellow arrows) was clicked and the Edge Prevalence Table and Selected DSGRN Predicted Networks areas were generated. In the Network Index, up and down arrows can be clicked to browse the selected networks (blue arrows).



A

Select the dataset to be displayed: network_finding_20210706083812

Network Finding Results

Seed network for DSGRN network perturbations

Argument	Setting
Range of Operations	2 - 10
# of Neighbors	2000
Max. Parameters	2000
Add Node Probability	0.2
Add Edge Probability	0.6
Remove Node Probability	0
Remove Edge Probability	0.2

● Stable Full Cycle ○ Domain

Epsilon: ● 0.05

% Stable Full Cycle: Min 1 Max 100

% Matches: Min 0 Max 100

Epsilon: 0.05

Edge Prevalence Table

model	prevalence
SWI5=tf_act(FKH1)	0.988
SWI4=tf_act(SWI5)	0.988
NDD1=tf_act(HCM1)	0.82
HCM1=tf_rep(YHP1)	0.793
YHP1=tf_act(SWI4)	0.745
FKH1=tf_act(NDD1)	0.65
FKH1=tf_act(HCM1)	0.54
SWI4=tf_rep(YHP1)	0.453
SWI5=tf_act(NDD1)	0.45
YHP1=tf_act(HCM1)	0.445
HCM1=tf_act(SWI4)	0.428
NDD1=tf_act(SWI4)	0.304
SWI4=tf_act(NDD1)	0.26
FKH1=tf_act(SWI4)	0.238
SWI4=tf_rep(YOX1)	0.192
SWI4=tf_act(HCM1)	0.175
YOX1=tf_rep(YHP1)	0.131
SWI4=tf_act(SWI4)	0.095
YOX1=tf_act(SWI4)	0.075

Selected DSGRN Predicted Networks

Selected 411 out of 612 Networks

Network Index: 1

Epsilon	% Matches	% Stable Full Cycle	Parameter Graph Size
0.05	0.244	0.304	672

B

Select the dataset to be displayed: network_finding_20210706083812

Network Finding Results

Seed network for DSGRN network perturbations

Argument	Setting
Range of Operations	2 - 10
# of Neighbors	2000
Max. Parameters	2000
Add Node Probability	0.2
Add Edge Probability	0.6
Remove Node Probability	0
Remove Edge Probability	0.2

● Stable Full Cycle ○ Domain

Epsilon: ● 0.05

% Stable Full Cycle: Min 1 Max 100

% Matches: Min 0 Max 0

Epsilon: 0.05

Edge Prevalence Table

model	prevalence
SWI5=tf_act(FKH1)	1
SWI4=tf_act(SWI5)	1
HCM1=tf_rep(YHP1)	1
NDD1=tf_act(HCM1)	0.963
YHP1=tf_act(SWI4)	0.829
FKH1=tf_act(NDD1)	0.707
FKH1=tf_act(HCM1)	0.683
SWI5=tf_act(NDD1)	0.549
YHP1=tf_act(HCM1)	0.463
HCM1=tf_act(SWI4)	0.366
SWI4=tf_act(SWI4)	0.305
NDD1=tf_act(SWI4)	0.268
FKH1=tf_act(SWI4)	0.256
YOX1=tf_rep(YHP1)	0.098
SWI4=tf_rep(YOX1)	0.098
SWI4=tf_act(NDD1)	0.037

Selected DSGRN Predicted Networks

Selected 82 out of 612 Networks

Network Index: 1

Epsilon	% Matches	% Stable Full Cycle	Parameter Graph Size
0.05	0	0.571	336

C

Select the dataset to be displayed: network_finding_20210706083812

Network Finding Results

Seed network for DSGRN network perturbations

Argument	Setting
Range of Operations	2 - 10
# of Neighbors	2000
Max. Parameters	2000
Add Node Probability	0.2
Add Edge Probability	0.6
Remove Node Probability	0
Remove Edge Probability	0.2

● Stable Full Cycle ○ Domain

Epsilon: ● 0.05

% Stable Full Cycle: Min 1 Max 100

% Matches: Min 60 Max 100

Epsilon: 0.05

Edge Prevalence Table

model	prevalence
HCM1=tf_rep(YHP1)	0.968
SWI5=tf_act(FKH1)	0.96
SWI4=tf_act(SWI5)	0.96
YHP1=tf_act(SWI4)	0.944
NDD1=tf_act(HCM1)	0.944
FKH1=tf_act(HCM1)	0.621
FKH1=tf_act(NDD1)	0.605
SWI4=tf_rep(YHP1)	0.468
SWI5=tf_act(NDD1)	0.452
SWI4=tf_act(NDD1)	0.411
SWI4=tf_act(HCM1)	0.331
YHP1=tf_act(HCM1)	0.29
SWI4=tf_rep(YOX1)	0.21
FKH1=tf_act(SWI4)	0.153
YOX1=tf_rep(YHP1)	0.121
YOX1=tf_act(SWI4)	0.097
SWI4=tf_act(SWI4)	0.097
NDD1=tf_act(SWI4)	0.065
HCM1=tf_act(SWI4)	0.032

Selected DSGRN Predicted Networks

Selected 124 out of 612 Networks

Network Index: 1

Epsilon	% Matches	% Stable Full Cycle	Parameter Graph Size
0.05	0.715	0.726	1472

D

Similarity Analysis

edge_list

☑ SWI4=tf_rep(YHP1)
☐ HCM1=tf_act(SWI4)
☐ NDD1=tf_act(HCM1)
☐ FKH1=tf_act(HCM1)
☐ SWI5=tf_act(FKH1)
☐ YHP1=tf_act(HCM1)
☑ YOX1=tf_act(SWI4)
☐ YOX1=tf_rep(YHP1)
☐ SWI5=tf_act(NDD1)
☐ NDD1=tf_act(SWI4)
☑ YHP1=tf_act(SWI4)
☐ FKH1=tf_act(SWI4)
☐ HCM1=tf_rep(YHP1)
☑ SWI4=tf_rep(YOX1)
☐ FKH1=tf_act(NDD1)
☐ YOX1=tf_rep(YOX1)
☐ SWI4=tf_act(SWI4)
☐ SWI4=tf_act(HCM1)
☐ SWI4=tf_act(NDD1)
☐ SWI4=tf_act(SWI5)

« ‹ 1 / 2 › »

Network or Motif for Similarity Analysis

○ Matches ● Stable Full Cycle ○ Domain

Epsilon: ● 0.05

Edge Prevalence Table

model	prevalence
YOX1=tf_act(SWI4)	0.75
YHP1=tf_act(SWI4)	0.75
SWI4=tf_rep(YOX1)	0.75
SWI4=tf_rep(YHP1)	0.75
SWI5=tf_act(FKH1)	0.5
SWI4=tf_act(SWI5)	0.5
NDD1=tf_act(HCM1)	0.5
HCM1=tf_rep(YHP1)	0.5
FKH1=tf_act(NDD1)	0.375
SWI5=tf_act(NDD1)	0.25
SWI4=tf_act(SWI4)	0.25
FKH1=tf_act(HCM1)	0.25

Selected 8 Networks

Network Index: 2

Epsilon	% Matches	% Stable Full Cycle	Parameter Graph Size
0.05	0.552	0.655	1392

Figure 7: Example of IDP Network Finding results in the IDV from using IDP configuration file two_wts_NetFind_rd2_config.txt (Supplemental File 12).

(**A-C**) The selection of networks was performed by entering values into the min and max input boxes (red arrows). The **Get Edge Prevalence from Selected Networks** button was clicked to generate the Edge Prevalence Table and Selected DSGRN Predicted Networks areas. (**D**) Edges of interest were selected in the edge_list table (yellow arrow) and the **Submit** button (green arrow) was clicked to compute similarity scores to plot in the scatter plot against the query selected (blue arrow). The Box Select function was used to select a set of networks (purple arrow) to generate the Edge Prevalence Table and Selected DSGRN Predicted Networks areas. The Network Index was increased to 2 (orange arrow) to view the second network in the selection.

Table 1:

Definition of Inherent Dynamics Pipeline and Inherent Dynamics Visualizer terms.

Term	Pipeline Step	Definition
de Lichtenburg by JTK-CYCLE (DLxJTK)	Node Finding	A single quantitative measure of both periodicity and regulation strength used to rank genes. Combines previously published periodicity metrics de Lichtenberg (DL) and JTK-CYCLE (JTK).
First Cycle Max Expression	Node Finding	The maximum gene expression during the first cycle of periodic gene expression. Genes ordered by First Cycle Max Expression will be ordered based on the time point from the first cycle at which they reach their maximum gene expression.
The Local Edge Machine (LEM)	Edge Finding	A Bayesian network inference method which ranks potential models of gene interactions to identify the most likely regulator(s) and modes of regulation (activation or repression) of a given target gene using time series gene expression data.
Seed Network	Network Finding	An initial guess at a plausible network of global interactions by selecting the top ranked LEM edges. The seed localizes a region of network space that is highly oscillatory with a high probability of showing consistency with the provided time series data.
Dynamic Signatures Generated by Regulatory Networks (DSGRN)	Network Finding	A software package for comprehensively computing the variety of long-term dynamical behaviors that a network can exhibit.
Edge Prevalence	Network Finding	The percentage of top scoring networks from the network finding step that include the edge in question. The score permits a ranking of edges that have a nonzero prevalence.