ACS Synthetic Biology

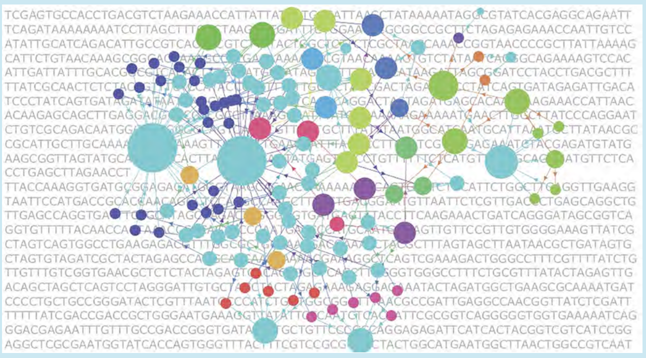
pubs.acs.org/synthbio Research Article

A Network Approach to Genetic Circuit Designs

Matthew Crowther, Anil Wipat, and Ángel Goñi-Moreno*

Cite This: *ACS Synth. Biol.* 2022, 11, 3058−3066 Read Online

ACCESS | Metrics & More | Article Recommendations



ABSTRACT: As genetic circuits become more sophisticated, the size and complexity of data about their designs increase. The data captured goes beyond genetic sequences alone; information about circuit modularity and functional details improves comprehension, performance analysis, and design automation techniques. However, new data types expose new challenges around the accessibility, visualization, and usability of design data (and metadata). Here, we present a method to transform circuit designs into networks and showcase its potential to enhance the utility of design data. Since networks are dynamic structures, initial graphs can be interactively shaped into subnetworks of relevant information based on requirements such as the hierarchy of biological parts or interactions between entities. A significant advantage of a network approach is the ability to scale abstraction, providing an automatic sliding level of detail that further tailors the visualization to a given situation. Additionally, several visual changes can be applied, such as coloring or clustering nodes based on types (e.g., genes or promoters), resulting in easier comprehension from a user perspective. This approach allows circuit designs to be coupled to other networks, such as metabolic pathways or implementation protocols captured in graph-like formats. We advocate using networks to structure, access, and improve synthetic biology information.

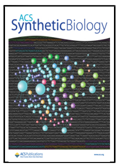
INTRODUCTION

The design and implementation of genetic circuits[1−3] that allow cells to perform predefined functions lie at the core of synthetic biology.[4,5] An example is the engineering of increasingly complex Boolean logic circuits[6] that use cascades of transcriptional regulators. Other types of circuits are routinely engineered, such as switches,[7] counters,[8] and memories,[9] using not only transcriptional, but also post-transcriptional processes.[10] Different host organisms such as bacteria,[11] yeasts,[12] and mammalian[13] cells are used to test circuits in several applications,[14] ranging from pollution control[15] to medical diagnosis.[16] Furthermore, the functionalities of genetic circuits will only improve as scientists control the information processing abilities of biological systems: signal noise,[17,18] metabolic dynamics,[19,20] context-circuit interplay,[21,22] stability,[23] and more.[24]

Designs are often the first step in each iteration of a synthetic biology project, and an implementation can be challenging without a well-conceived design and solid understanding. Furthermore, mathematical and computational tools,[25] automation methods,[26,27] knowledge-based systems,[28,29] and repositories[30] assist circuit design to minimize the iterations within the design-build-test-learn research cycle. These processes generate a consortium of information beyond DNA sequences, such as modularity, hierarchy, implementation instructions, dynamical predictions, and validation strategies. However, this information is often disparate and seldom formalized, resulting in inaccessibility and threatening to undermine the success of such endeavors.

What has been termed *network biology*[31] deals with the quantifiable representation of complex cellular systems in graphs and their study to characterize functional behavior. Graph theory methods can assist the interrogation of network structures in several ways[32] for circuit designs, and produce subnetworks of particular interest hidden within design formats. Graphs are represented in the form of nodes (individual points of data) and edges (relationships between the data).[33] For example, when building networks from circuit designs, a repression relationship edge links two nodes representing a regulator protein (e.g., aTc) and its cognate promoter (pTet). The primary advantage of the approach described in this work is that graphs are inherently dynamic. Therefore, converting an existing genetic design into a network structure allows user-driven analysis and visualizations beyond current capabilities. Furthermore, a network approach is often

Received: May 16, 2022
Published: August 31, 2022



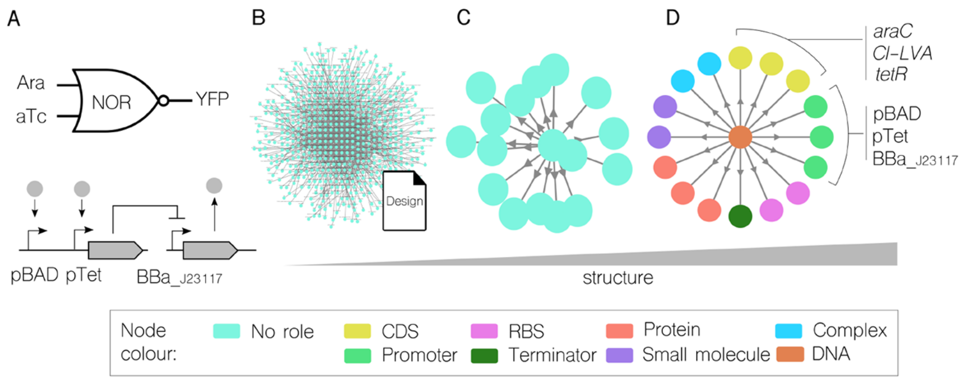


Figure 1. Converting and visualizing the design data of a NOR logic gate. (A) NOR logic function and genetic diagram, with inputs (arabinose; aTc) and output (YFP). (B) Displaying all design information encoded from the source in network format. The network is unreadable but computationally tractable. (C) A network is generated from the same design where only the physical elements (i.e., DNA and molecular entities described) are shown. (D) Depending on their role, the network is adjusted to display colors for the nodes for visualization purposes. Roles (e.g., promoter, proteins) are automatically clustered by the same color. For clarity purposes, labels were not included, but full graphs are available in a public repository (see Methods).

successfully implemented within systems biology to represent both simulation models and knowledge models,[34] for example, to depict multiple omics data within a single network.

Early efforts in modeling biological systems using networks[31] had a limitation: the lack of semantic labels for the types and roles of entities and connections. For example, if two nodes representing proteins are linked, it is helpful to know what type of connection this is, like a binding, repression, or activation interaction. To overcome this limitation, more recent efforts are based on knowledge graphs, which are structured directed graphs where nodes and edges contain known semantic labels and specific rules govern their connectivity. Knowledge graphs have been widely used in the biological sciences—from building knowledge bases to predicting biological reactions[35]—and are used here as the basis for structuring input data. The addition of semantics allows for more complex control over the underlying data, e.g., the ability to arrange information into several layers of abstraction.

Data formats have emerged that effectively capture and represent increasingly complex designs. A leading example is a standard to implement a synthetic biology knowledge graph: Synthetic Biology Open Language[36] (SBOL), which describes both structural (e.g., DNA sequences) and functional (e.g., regulation interactions) information. Also, the GenBank[37] format, overwhelmingly used to formalize and share genetic sequences, allows simple annotations to be defined, but they are often open to interpretation due to a lack of standard semantics and structure. The overarching challenge is to access, use, visualize, and analyze this information so that genetic designs become dynamic data structures that are easy to handle computationally to improve accessibility. This challenge underpins this work, where we propose networks to help solve these problems.

Visualizing complex information is a challenge shared by many areas of research, and networks offer a powerful solution.[38] Genetic circuit designs capture multidimensional data, and a one-size-fits-all approach is often not feasible; i.e., a single representation of a multidimensional data set cannot satisfy the requirements of all users at once. For example, the glyph approach of SBOL Visual[39,40] allows researchers to generate diagrams of somewhat abstract designs. However, the level of detail must be selected in advance, and diagrams remain fixed—the user cannot interact with the visualization to rearrange it according to specific requirements. In contrast, a network approach to visualization can be dynamically adjusted according to user demands, such as highlighting proteins, interactions, or hierarchy. Here, we demonstrate how to apply network techniques to genetic circuit design data analysis with a knowledge graph approach to produce tailored visual representations of existing genetic designs automatically. Graphs are not only visualizations of a design; they *are* the design.

RESULTS AND DISCUSSION

Establishing Networks from Design Files. Figure 1 shows the process of structuring, querying, and visualizing the data encoded within an existing genetic circuit design using networks (see Methods for details). We used the design of the NOR logic gate built by Tamsir and colleagues[41]—this gate outputs 1 (i.e., target gene expressed) if both inputs are 0, and outputs 0 (i.e., target gene not expressed) in any other case. The NOR circuit is a frequently built device,[6,42] since any logic function can be achieved by assembling NOR gates only.

The functional diagram for the NOR gate (Figure 1A, top) is often represented with the specific names of the input and output compounds. In this case, the inputs are the inducers arabinose (Ara) and anhydrotetracycline (aTc), and the output reporter is yellow fluorescent protein (YFP). A more implementation-focused diagram (Figure 1A, bottom) is usually labeled with names for specific DNA parts—three promoters in this example—and connections are explicitly drawn. While these representations are often used to communicate functional aspects of a design,[39] they are not the actual design, and the automatic generation of diagrams from annotated files[43] is still a challenge that deserves further attention. Our approach to this issue is based on substituting the design file with a network that can be directly visualized and analyzed.

The quality and variance of the input data are fundamental to meaningful representation. Without rich data, expressive visualization is not feasible irrespective of the visualization method. Here, we used a design of the NOR logic gate in SBOL format, which captures data about genetic elements, connections, proteins, types, roles, and more. Figure 1B shows the results of building a network with all data and metadata

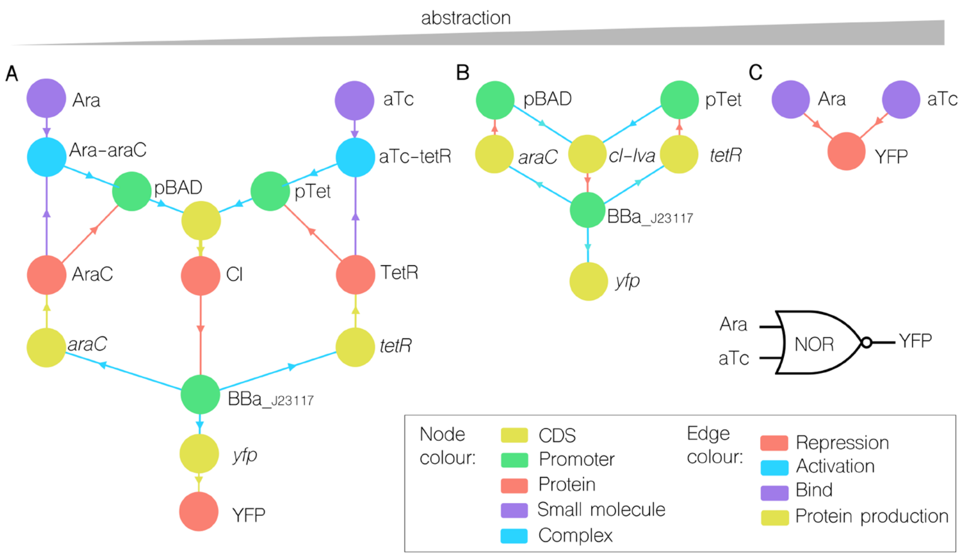


Figure 2. Adjusting network abstraction levels using a NOR gate design[41] modeled in SBOL (see Methods). (A) The NOR gate design is turned into a network with all molecular and genetic elements (nodes); and interactions between entities (edges). (B) Nongenetic elements, i.e., non-DNA based elements, are merged into the appropriate genetic elements. For instance, Ara and Ara-araC are merged into the pBAD node. (C) Maximum abstraction into input−output data. The color scheme is constant regardless of abstraction levels.

available in the original design file. The complete design graph is a multipartite containing numerous data types, for example, parts, interactions, sequences, and metadata (e.g., entity role, free text descriptions, or data type). Before building this graph, design data was converted into an intermediate data structure that is the same regardless of input format; that is, the resulting graphs are compatible and not format-dependent (see Methods). Although the network of Figure 1B is too convoluted and visually meaningless because the significance of nodes and connections is lost, computational manipulation is easy to perform based on specific user requirements.[44]

Upon conversion, networks generated from designs are suitable for analysis. Graph theory methods and tools[45,46] (e.g., calculate shortest paths between entities, clustering, intersections, etc.) are directly applicable to genetic designs since nodes and edges have semantic information about types, roles, and relationships. The network of Figure 1C is an example of how to query the initial structure and is a specific subgraph that focuses on physical elements only (e.g., DNA and molecular entities) and omits metadata details according to user requirements. By presenting a single perspective, overall cognition is increased, but it is visually incoherent since the position of nodes—its *layout*—is random. A final module within our design-to-network conversion software deals with visualizing graphs. The layout, which determines the arrangement of nodes combined with other features such as color, size, and shape,[47] provides a visual representation of information that ensures clarity and understanding. When a simple radial layout (Figure 1D) (e.g., nodes do not overlap) is combined with node clustering and coloring depending on types and roles, it results in a final visual output that is considerably clearer than Figure 1B.

In what follows we explore several complex features of network designs that showcase the benefits (and limitations) of having data captured by dynamic structures.

Dynamic Abstraction Levels. The design of a biological system implies dealing with complexity. Therefore, it is crucial to abstract away superfluous details to describe and communicate the design with clarity.[48] Nevertheless, what is an appropriate level of abstraction for a circuit design? The answer depends on two primary factors: what information needs to be communicated, e.g., structural or functional, and the requirements of the person consuming the information, e.g., bioinformatician or *wet lab* scientist. Precisely, a vital advantage of a network structure is its inherent ability to arrange itself—dynamically—into several levels of abstraction.

Interaction networks provide a high-level metric with the potential to scale, and this view into the data has been chosen to display dynamic abstraction. The network in Figure 2A displays all molecular, genetic, element types and relational information, i.e., an interaction network. An interaction network is generated by querying the data to find nodes and edges with semantic labels denoting interactions and transforming the following structure (see Methods for a more in-depth description): physical entity (node) → interaction (edge) → physical entity (node). One example is to abstract all nongenetic elements (Figure 2B), limiting the information to only the indirect effects of DNA based features (e.g., promoters and genes) on one another. Even in this case, relational information remains; for instance, when expressing, the coding sequence *araC* represses the promoter node BBa_J23117. Therefore, the visualization is simplified by abstracting mechanistic details such as the production of regulatory proteins. The automatic level of detail can be taken to a conclusion by displaying input and output elements for the whole system (Figure 2C). The highest abstraction level allows for quick circuit performance communication while abstracting all implementation details and internal workings. This oversimplification may be excessive for a relatively simple design but could benefit more extensive and complex structures. Scaling abstraction is achieved using transitive closure, i.e., the reachability matrix to reach node n from node v. Practically, it estimates the costs of different paths across the network to merge nodes along a path[49] (see Methods for an expanded explanation). Within all graphs displayed in Figure 2, biological roles and conceptual processes (interaction types)

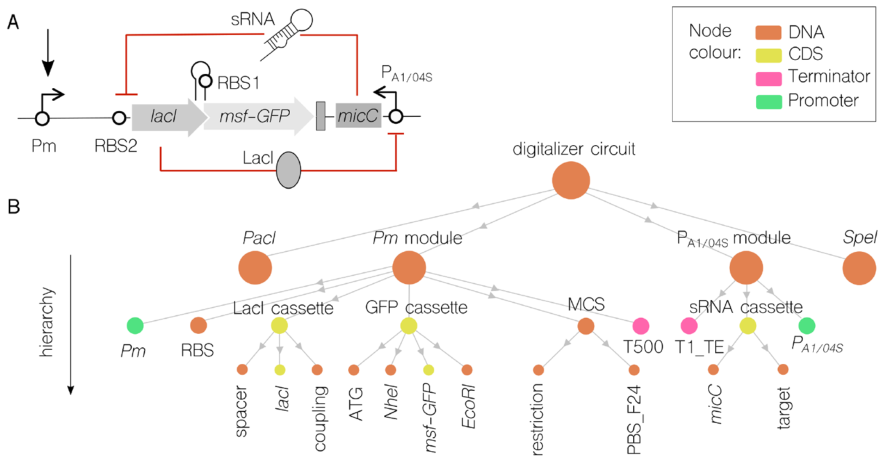


Figure 3. A hierarchical network of increasing abstraction, from modules to parts. (A) Glyph representation of the *digitalizer*[51] synthetic circuit. The circuit is based on two negative interactions between the regulatory protein LacI and a small RNA and offers the ability to plug and play any gene of interest the user wants to *digitalize*—the reporter *gfp* gene is used for characterization. The goal of the *digitalizer* circuit is to minimize the leakage expression of a specific gene of interest while maximizing the full production. That is to say, to enlarge its dynamic range. (B) The hierarchical network; nodes represent biological and conceptual entities, i.e., nodes at the bottom represent DNA parts and nodes at higher levels represent modules (the top node is the entire circuit), and edges represent hierarchical direction. Circuit building details are highlighted within the network, e.g., restriction sites or sequence to couple *lacI* to *msf-GFP*.

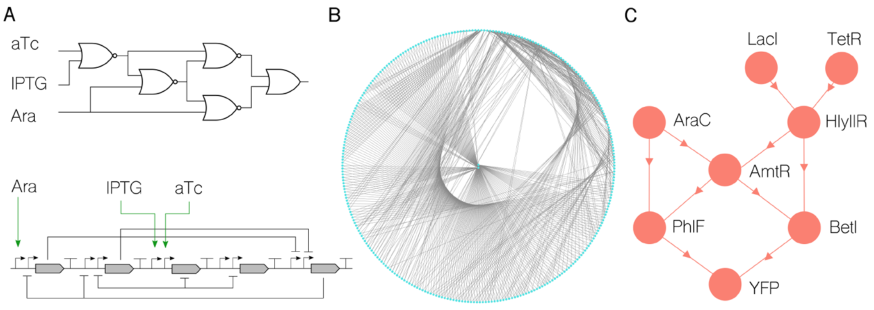


Figure 4. Displaying the protein interaction graph within a complex circuit design. (A) Boolean gene circuit 0x87.[6] The circuit couples four NOR logic gates and one OR logic gate (top diagram) and uses three molecular reagents, five regulatory proteins, five genes, and ten promoters (bottom diagram). (B) Network with all encoded information from the source design. (C) Network with protein (nodes) and interaction (edges) representing negative regulation.

are mapped to colors to encode information without an increase in perceived complexity. For example, two nodes: *yfp* (yellow) and *YFP* (red) are connected by an edge denoting *protein production* (yellow). Also, the arrangement of the nodes (layout) helps transmit the flow of information. In this case, data flow from inputs (upper nodes) to outputs (lower nodes).

Hierarchical Trees. Hierarchically representing genetic parts is a core principle to promote engineering in biology[50] because this provides structure to increasingly complex circuits. A *tree* data structure is a fundamental network topology commonly used and represents data hierarchically and at an arbitrary depth. Figure 3 shows the hierarchical network corresponding to the *digitalizer* genetic circuit.[51] This device is far more complex than the previous NOR logic gate, both in terms of interactions and dynamic performance, and therefore ideal for showcasing the use of hierarchical representation. Its hierarchical tree (Figure 3B), which is automatically built from the design file, displays the conceptual modules into which single parts are structured. This information is often particular for each circuit—even similar or identical circuits—since it follows the authors' conceptual framework. In this case, the top module represents the whole device and is broken down into four modules, which are, in turn, leading either to the final parts (e.g., promoters *Pm* and *P_A1/04S*) or to smaller submodules (e.g., GFP cassette). Specific structural details that refer to implementation strategies are essential in those genetic circuits whose goal is to let users modify parts of them. The *digitalizer* circuit is an example where the user is meant to switch the reporter gene to their gene of choice. By browsing through the network in Figure 3B, the user can find a module where the reporter is included (named GFP cassette) and the hard-coded procedure for cutting out the gene (restriction sites *Nhe*I and *Eco*RI) without looking at the genetic sequence of the design. Hierarchical representations are formed by finding "parent-child" relationships (ownership labels attached to edges to denote a node "owns" another node) encoded within semantic labels.

Within the graphs displayed in Figure 2, biological roles are mapped to colors to encode information without an increase in perceived complexity. Also, the pyramid shape of the nodes (layout) helps transmit information concerning the hierarchical nature, with each level of the hierarchy decreasing in abstraction from top to bottom.

Protein Interaction Maps for Representing the Function. The information captured by designs can be broadly split into two groups (discounting metadata): First,

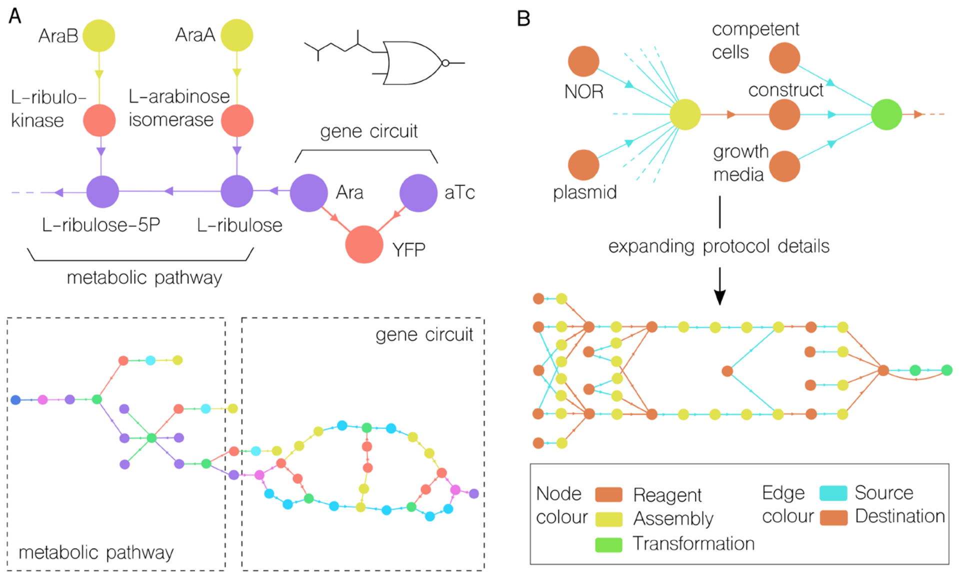


Figure 5. Networks beyond gene circuitry: coupling circuit designs to host metabolic networks and circuit-building protocols. (A) The network of a gene circuit that uses arabinose as input can interact with the arabinose degradation pathway. Top figure: abstract network displaying critical components of a NOR gate and the initial steps of the arabinose pathway. Bottom figure: linking the corresponding extended networks. (B) NOR-gate experimental protocol formalized as a network structure. The network can be interactively adjusted to show different levels of abstraction. Nodes represent reagents or subprotocols, and edges imply input/output relationships.

constructional details, i.e., the DNA sequence and related data; Second, functional information, i.e., nongenetic elements and their relationships once built. While constructional details are commonly communicated, functional elements are often less clear and can provide insight that the former cannot offer. Therefore, it is essential to complement sequence-based designs and visualizations with regulatory information. Specific interaction networks can provide a higher-level understanding by visualizing regulatory proteins and abstracting relationships. For example, the mechanisms that allow a regulator to bind its cognate promoter and repress a downstream gene's expression into another regulator can be abstracted into a simple network with two nodes (one per regulatory protein) with an edge denoting effect. While this network lacks structural information at the sequence level (e.g., implementation details), it maximizes the functional aspect of the circuit.

The circuit-orientated diagram (Figure 4A, top) provides only a high-level indication of function. Also, the construct-oriented diagram (Figure 4A, bottom) offers more specific functional and structural information, but even with this modestly sized circuit can be challenging to comprehend quickly. The graph representation of all data encoded within the circuit design is shown in Figure 4B; while not displaying helpful information, it gives an idea of how much data even a design of moderate size has. From that data, a subnetwork with regulators (nodes) and relationships (edges) is generated to display the protein interaction map (Figure 4C), explicitly displaying the three input regulators (LacI, TetR, and AraC) and the output protein (YFP), including information flows. This representation provides a middle ground between the Figure 4A top and bottom details (or lack of). The layout makes visualization easier since nodes are arranged following functional criteria rather than sequentially. Moreover, the Boolean logic of the network becomes apparent; for example, the network contains a final OR logic gate based on the regulators PhiF or BetI repressing the presence of YFP.

Figure 4B contained 201 nodes and 376 edges. From this, 141 nodes and 331 edges were pruned, i.e., edges which do not relate to the protein interaction graph, such as metadata. Finally, the graph traversals collapsed 52 nodes and 55 edges, resulting in Figure 4B with 8 nodes and 10 edges.

Biodesign beyond Genetic Circuits. An advantage of a network approach is that data integration is easier when the underlying information is represented as graphs with unified semantics. While this is useful when representing designs, as many types of data constitute a design, this characteristic can excel in unifying more disparate or loose data. We briefly cover two such elements, namely metabolic pathways and experimental protocols, and discuss the potential of networks to provide a general framework for biodesign efforts.

Genetic circuits *run* inside a cellular host (except cell-free systems[52]), and the host context, particularly its metabolism, impacts circuit performance. A grand challenge is to enhance genetic circuits by exploiting metabolic mechanisms that offer dynamics beyond the genetic toolkit catalogue.[20] As far as the design process is concerned, a question that needs to be answered is whether we can design merged metabolic−genetic circuits.[53] To this end, we show in Figure 5A that the descriptions of a NOR logic gate and a metabolic pathway can dynamically interact if they are encoded into compatible data structures. Specifically, the NOR logic gate uses arabinose as input, which interacts with the same node of the arabinose degradation pathway. Having this information within the same network allows formalizing the impact of metabolic dynamics on one of the inputs of the target genetic circuit. The addition of metabolic pathways is achieved by simply merging the new data into the graph, while ensuring that already encoded data within the graph as nodes are not duplicated, and instead, new edges are attached to old nodes.

The goal of all circuit designs is to be built and validated experimentally. However, the formalization of implementation protocols into well-characterized steps and their representation in standard data structures is still a significant challenge[54−56]

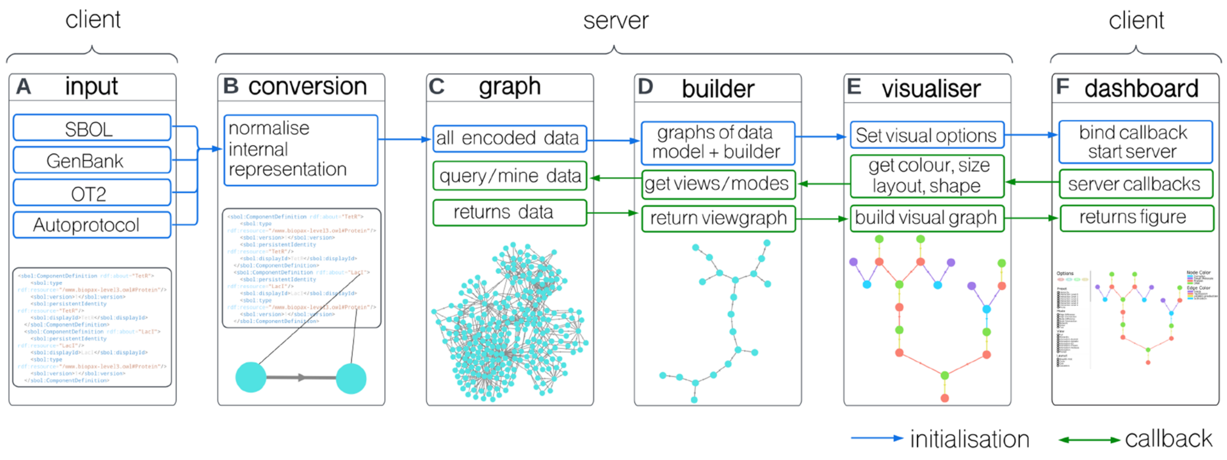


Figure 6. Workflow for transforming designs into dynamic network structures. (A) The input design should be formalized using existing formats. We advocate for the use of SBOL for genetic designs since it allows for capturing complex information. (B) Input data is normalized into an internal structure by mapping semantic labels or keywords to a predefined network data model. (C) The graph with all design information is represented and ready for algorithmic analysis. (D) The builder module of the software produces specific subnetworks based on user requirements and the resulting analysis over the original structure. (E) The visualizer calculates all visual specific elements (layout, color, shape, size) and renders the graph accordingly. (F) The dashboard is the user aspect of the application. It handles the graph rendering and user inputs by sending callback requests back to the server.

that deserves more attention. Therefore, finally, we showcase the use of networks for representing experimental protocols. Figure 5B shows the network that corresponds to the protocol for building and testing the NOR gate used as an example. Here, we chose (from the many options available) to represent materials and methods as nodes and information flow as edges. As in other examples, protocol graphs can also be adjusted at different levels of abstraction. For instance, the assembly node (Figure 5B, top) includes processes such as restriction, purification, and ligation—which are conveniently clustered to provide an overview of the inputs (i.e., what the assembly process gets) and outputs (i.e., what it returns). This network can be linked to the NOR graph at the top node of the hierarchy, therefore having genetic circuits and protocol within the same data structure. The integration and visualization of protocol data are similar to handling design data but with a critical difference. Primarily, the protocol data were encoded using the autoprotocol standard, while all design data displayed were modeled using the SBOL standard. However, when the input data have been transformed into a formalized knowledge graph, the same processes of querying semantic labels to produce focused subgraphs can be applied (see Methods for further discussion).

■ CONCLUSION

Here, we present a graph-based methodology for representing, analyzing, and visualizing circuit design information. Our approach transforms design files into networks, which are dynamic structures able to be automatically modified on demand according to user specifications.

When molecular entities, relationships, and other information (e.g., types and roles) are encoded into nodes and edges using semantic labels, a network representation of a genetic design can be established. We have showcased the benefits of this approach by converting into networks the structural and functional data available within several genetic circuits. Specifically, we showed that design networks could be automatically adjusted to display different levels of detail, from full molecular representation to input/output information only and single-type graphs (e.g., protein interactions). The selection of abstraction as a metric to showcase the potential of networks is rooted in the intrinsic complexity of designs and the need to separate high-value information from superfluous details for a given purpose—thus improving understanding. These network manipulations are only an initial subset of the many possibilities available,[45] since network science[46] is an active field with applications in many disciplines, including the life sciences.[31,57] For example, community detection[58] is the process of partitioning the network into multiple communities and can help to reveal hidden relations that may not be explicitly encoded. Also, network robustness[59] is the process of measuring the robustness of a network by exploring the structure, which can be used to analyze the biological feasibility of the design.

The intrinsic modularity of networks allows for coupling genetic circuit designs to other data types providing these are also represented in graphs. We have demonstrated this in two different ways. We showed that a genetic circuit that uses arabinose as input could be automatically coupled to the arabinose degradation pathway graph. By doing this, circuit designs can be extended to include information from their host context, improving the functional description of the device. Second, we have represented an implementation protocol in network format. While this is just a preliminary effort, which deserves further attention, it shows that protocol networks can also interact with circuit designs for the sake of building a data structure that can be shared along the design-build-test-learn[60] (DBTL) research cycle. In short, when data are represented as a graph, merging and clustering potentially disparate entities become far less challenging tasks, and the graph could be the key to unifying data.

In order to generate high-quality and information-rich networks, designs should capture as much information as possible. Indeed, networks can only work with the provided data—networks cannot fabricate entirely new data, only derived from existing sources. While commonly used formats, such as GenBank, still capture information beyond genetic sequences, this information can be challenging to manage

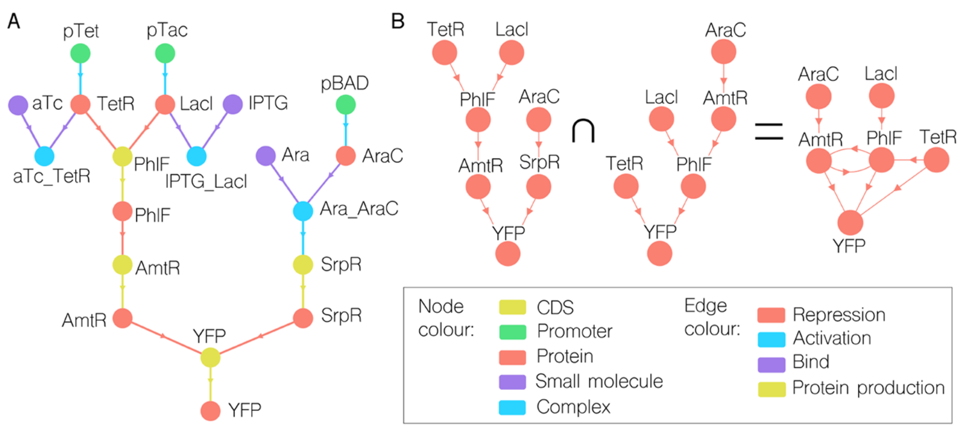


Figure 7. Example output of the builder module. (A) The production of a view is the standard operation. In this case, the interaction graph of the 0xF7 circuit as described in Nielsen et al.[6] (B) The previous graph is abstracted into protein interactions by transitive closure via depth-first-searches (left) and intersected with another network (middle) to identify common nodes and subgraphs between the two (right).

computationally due to the inherent informality and loose connections. Therefore, we advocate using knowledge graphs because more abstract questions can be formed, and more in-depth analyses can be made with the data—more specifically, a knowledge graph that implements the Synthetic Biology Open Language (SBOL) standard, since it represents formal information, such as modularity or hierarchy, that cannot be captured otherwise.

As the complexity of genetic circuits increases, we advocate for networks to manipulate, analyze, and communicate design information. We hope networks can maximize the efficiency of design automation procedures and help unification by providing standard[61] data structures for merged mathematical, genetic, protocol, and other prominent data sets established during synthetic biology projects.

Design visualization for the representation of genetic design is only one application of networks. Networks have been successfully applied to specification and analysis tasks within and outside biology. Future efforts will focus on adding information into the original data set, including the development of methods to infer new data automatically when using abstract projections.

■ METHODS

The software tool we developed to carry out the design-to-network conversion is accessible from the repository available at https://github.com/intbio-ncl/genet2.git. The repository includes the software, instructions on how to run it, and documents describing use-cases and examples. The tool runs from a flask application where a neo4j graph datastore holds design data and is visualized using dash-Cytoscape. Full graphs of the networks used in this work, genetic design files, protocol files, and network files were also included. All data required to replicate the networks described in the Results and Discussion are available at https://github.com/MattyCrowther/network-visualisation-supplementary.git.

The functioning of our methodology is described in Figure 6, which shows the different steps (A–F) involved in converting an input design into a visual representation of a graph. Each step works as follows.

Input (Figure 6A). The method gets a design file as input. It is important to note that the resulting networks will only work with the information captured within the input file. Therefore, the more information is captured, the more extensive the graph analysis. For the examples of genetic circuits shown in this work, we used SBOL[36] files. Unlike other standards for capturing genetic designs (e.g., GenBank), SBOL is highly structured and formalized into specific semantic labels that computer programs can easily understand. Besides, SBOL allows for capturing data with different types of functional information, including interactions and non-DNA components, and our approach exploits that ability to represent information. For the protocol network (Figure 5B), we used Autoprotocol files. Other files in GenBank format and Opentrons OT2 format are provided in the repository as examples.

Conversion (Figure 6B). This step converts the input data into a unique internal graph representation. This conversion aims at unifying data structures, so that resulting interaction graphs share the same features. We developed a knowledge graph that formally specifies rules, including what semantic labels can be used and how objects can connect, called an ontology. This ontology captures physical (DNA, protein, pipet, etc.) and conceptual (repression, binding, liquid transfer, etc.) entities while not being specific to one format alone (e.g., SBOL-OWL[62]). This ontology is not designed for general use (SBOL and Autoprotocol, for example, are preferable for data exchange). Still, this approach allows for analysis across formats, a method to produce viewgraphs does not need to be created for each type, and the structure is tailored for network analysis.

Graph (Figure 6C). The internal graph representation is turned into a directed multigraph: a graph where edges are directed, and multiple edges can connect two nodes. The resulting network—embedding all design information—is ready to receive queries from the user and perform algorithmic analysis before returning specific subnetworks.

Builder (Figure 6D). This module handles the construction of viewgraphs with specific representations, e.g., functional or structural. Multiple viewgraphs can be generated from a single data source—the vast amount of graph theory methods can be used to query the internal structure (Figure 6C) in various ways. For example, Figure 7A shows the building of an interaction graph, where nodes are physical entities and edges their relationships, and is achieved by graph queries for specific semantic labels. Another function of this module is scaling abstraction via transitive closure, which finds reachable nodes with certain semantic tags from source nodes. This method can be used to find protein interaction maps

(Figure 7B, left) which reduces the size of visualizations to only protein entities. As a more complex example, Figure 7B shows the intersection between two designs containing similar elements. This method outputs the intersection of both graphs, i.e., those nodes shared by both.

Visualizer (Figure 6E). The module handles all visual elements, including layout, color, shape, and size, then combines these with the viewgraph and returns a visualization. Users can modify graph visuals on demand, e.g., layouts stop edges from overlapping, and color encodes another data dimension. These features are delivered by different methods, e.g., semantic mapping to identify colors or spatial configurations to arrange information into geometric, hierarchical, or force-directed layouts.

Dashboard (Figure 6F). Lastly, a dashboard renders the graph, takes instructions from the user, and sends requirements to the server for analysis. This user interface is the client-side aspect of the application.

■ ASSOCIATED CONTENT

Special Issue Paper

Invited contribution from the 13th International Workshop on Bio-Design Automation.

■ AUTHOR INFORMATION

Corresponding Author

Ángel Goñi-Moreno − *Centro de Biotecnología y Genómica de Plantas, Universidad Politécnica de Madrid, Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA-CSIC), 28223 Madrid, Spain;* orcid.org/0000-0002-2097-2507; Email: angel.goni@upm.es

Authors

Matthew Crowther − *School of Computing, Newcastle University, Newcastle Upon Tyne NE4 5TG, United Kingdom; Centro de Biotecnología y Genómica de Plantas, Universidad Politécnica de Madrid, Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA-CSIC), 28223 Madrid, Spain*

Anil Wipat − *School of Computing, Newcastle University, Newcastle Upon Tyne NE4 5TG, United Kingdom*

Complete contact information is available at:
https://pubs.acs.org/10.1021/acssynbio.2c00255

Author Contributions

M.C., A.W., and A.G.M. conceived the study. M.C. built the network model and carried out the computational analysis. All the authors (M.C., A.W., A.G.M.) contributed to the discussion of the research, interpretation of the data, and writing of the study.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work was supported by the Grants BioSinT-CM (Y2020/TCS-6555) and CONTEXT (Atracción de Talento Program; 2019-T1/BIO-14053) Projects of the Comunidad de Madrid, MULTI-SYSBIO (PID2020-117205GA-I00) and the Severo Ochoa Program for Centres of Excellence in R&D (CEX2020-000999-S) funded by MCIN/AEI/10.13039/501100011033 and the EPSRC studentship 34000024085 (M.C.)

■ REFERENCES

(1) Benenson, Y. Biomolecular computing systems: principles, progress and potential. *Nat. Rev. Genet.* **2012**, *13*, 455−468.
(2) Brophy, J. A.; Voigt, C. A. Principles of genetic circuit design. *Nat. Methods* **2014**, *11*, 508−520.
(3) Amos, M.; Goni-Moreno, A. *Computational Matter*; Springer, 2018; pp 93−110.
(4) Ausländer, S.; Ausländer, D.; Fussenegger, M. Synthetic biology—the synthesis of biology. *Angew. Chem., Int. Ed.* **2017**, *56*, 6396−6419.
(5) Andrianantoandro, E.; Basu, S.; Karig, D. K.; Weiss, R. Synthetic biology: new engineering rules for an emerging discipline. *Mol. Syst. Biol.* **2006**, *2*, 2006−0028.
(6) Nielsen, A. A.; Der, B. S.; Shin, J.; Vaidyanathan, P.; Paralanov, V.; Strychalski, E. A.; Ross, D.; Densmore, D.; Voigt, C. A. Genetic circuit design automation. *Science* **2016**, *352*, aac7341.
(7) Lou, C.; Liu, X.; Ni, M.; Huang, Y.; Huang, Q.; Huang, L.; Jiang, L.; Lu, D.; Wang, M.; Liu, C.; et al. Synthesizing a novel genetic sequential logic circuit: a push-on push-off switch. *Molecular systems biology* **2010**, *6*, 350.
(8) Friedland, A. E.; Lu, T. K.; Wang, X.; Shi, D.; Church, G.; Collins, J. J. Synthetic gene networks that count. *science* **2009**, *324*, 1199−1202.
(9) Shipman, S. L.; Nivala, J.; Macklis, J. D.; Church, G. M. Molecular recordings by directed CRISPR spacer acquisition. *Science* **2016**, *353*, aaf1175.
(10) Kawasaki, S.; Ono, H.; Hirosawa, M.; Saito, H. RNA and protein-based nanodevices for mammalian post-transcriptional circuits. *Curr. Opin. Biotechnol.* **2020**, *63*, 99−110.
(11) Wong, A.; Wang, H.; Poh, C. L.; Kitney, R. I. Layering genetic circuits to build a single cell, bacterial half adder. *BMC Biol.* **2015**, *13*, 1−16.
(12) Chen, Y.; Zhang, S.; Young, E. M.; Jones, T. S.; Densmore, D.; Voigt, C. A. Genetic circuit design automation for yeast. *Nature Microbiology* **2020**, *5*, 1349−1360.
(13) Lillacci, G.; Benenson, Y.; Khammash, M. Synthetic control systems for high performance gene expression in mammalian cells. *Nucleic acids research* **2018**, *46*, 9855−9863.
(14) Meng, F.; Ellis, T. The second decade of synthetic biology: 2010−2020. *Nat. Commun.* **2020**, *11*, 1−4.
(15) De Lorenzo, V.; Prather, K. L.; Chen, G.-Q.; O'Day, E.; von Kameke, C.; Oyarzún, D. A.; Hosta-Rigau, L.; Alsafar, H.; Cao, C.; Ji, W.; et al. The power of synthetic biology for bioproduction, remediation and pollution control: the UN's Sustainable Development Goals will inevitably require the application of molecular biology and biotechnology on a global scale. *EMBO Rep.* **2018**, *19*, No. e45658.
(16) Slomovic, S.; Pardee, K.; Collins, J. J. Synthetic biology devices for in vitro and in vivo diagnostics. *Proc. Natl. Acad. Sci. U. S. A.* **2015**, *112*, 14429−14435.
(17) Goñi-Moreno, Á.; Benedetti, I.; Kim, J.; de Lorenzo, V. Deconvolution of gene expression noise into spatial dynamics of transcription factor−promoter interplay. *ACS synthetic biology* **2017**, *6*, 1359−1369.
(18) Eldar, A.; Elowitz, M. B. Functional roles for noise in genetic circuits. *Nature* **2010**, *467*, 167−173.
(19) Moser, F.; Espah Borujeni, A.; Ghodasara, A. N.; Cameron, E.; Park, Y.; Voigt, C. A. Dynamic control of endogenous metabolism with combinatorial logic circuits. *Mol. Syst. Biol.* **2018**, *14*, No. e8605.
(20) Goñi-Moreno, A.; Nikel, P. I. High-performance biocomputing in synthetic biology−integrated transcriptional and metabolic circuits. *Front. Bioeng. Biotechnol.* **2019**, *7*, 40.
(21) Tas, H.; Grozinger, L.; Stoof, R.; de Lorenzo, V.; Goñi-Moreno, Á. Contextual dependencies expand the re-usability of genetic inverters. *Nat. Commun.* **2021**, *12*, 1−9.
(22) Boo, A.; Ellis, T.; Stan, G.-B. Host-aware synthetic biology. *Current Opinion in Systems Biology* **2019**, *14*, 66−72.

(23) Zhu, R.; del Rio-Salgado, J. M.; Garcia-Ojalvo, J.; Elowitz, M. B. Synthetic multistability in mammalian cells. *Science* **2022**, *375*, No. eabg9765.

(24) Grozinger, L.; Amos, M.; Gorochowski, T. E.; Carbonell, P.; Oyarzún, D. A.; Stoof, R.; Fellermann, H.; Zuliani, P.; Tas, H.; Goñi-Moreno, A. Pathways to cellular supremacy in biocomputing. *Nat. Commun.* **2019**, *10*, 1−11.

(25) Lopiccolo, A.; Shirt-Ediss, B.; Torelli, E.; Olulana, A. F. A.; Castronovo, M.; Fellermann, H.; Krasnogor, N. A last-in first-out stack data structure implemented in DNA. *Nat. Commun.* **2021**, *12*, 1−10.

(26) Appleton, E.; Madsen, C.; Roehner, N.; Densmore, D. Design automation in synthetic biology. *Cold Spring Harbor perspectives in biology* **2017**, *9*, a023978.

(27) Kitney, R.; Adeogun, M.; Fujishima, Y.; Goñi-Moreno, Á.; Johnson, R.; Maxon, M.; Steedman, S.; Ward, S.; Winickoff, D.; Philp, J. Enabling the advanced bioeconomy through public policy supporting biofoundries and engineering biology. *Trends Biotechnol.* **2019**, *37*, 917−920.

(28) Mante, J.; Hao, Y.; Jett, J.; Joshi, U.; Keating, K.; Lu, X.; Nakum, G.; Rodriguez, N. E.; Tang, J.; Terry, L.; et al. Synthetic Biology Knowledge System. *ACS synthetic biology* **2021**, *10*, 2276−2285.

(29) Mısırlı, G.; Hallinan, J.; Pocock, M.; Lord, P.; McLaughlin, J. A.; Sauro, H.; Wipat, A. Data integration and mining for synthetic biology design. *ACS synthetic biology* **2016**, *5*, 1086−1097.

(30) McLaughlin, J. A.; Myers, C. J.; Zundel, Z.; Mısırlı, G.; Zhang, M.; Ofiteru, I. D.; Goni-Moreno, A.; Wipat, A. SynBioHub: a standards-enabled design repository for synthetic biology. *ACS synthetic biology* **2018**, *7*, 682−688.

(31) Barabasi, A.-L.; Oltvai, Z. N. Network biology: understanding the cell's functional organization. *Nat. Rev. Genet.* **2004**, *5*, 101−113.

(32) Ideker, T.; Krogan, N. J. Differential network biology. *Molecular systems biology* **2012**, *8*, 565.

(33) Krempel, L. Network Visualization. In *The SAGE Handbook of Social Network Analysis*; SAGE Publishing, 2011; pp 558−577.

(34) Yan, J.; Risacher, S. L.; Shen, L.; Saykin, A. J. Network approaches to systems biology analysis of complex disease: integrative methods for multi-omics data. *Brief. Bioinf.* **2017**, *19*, 1370−1381.

(35) Mohamed, S. K.; Nounu, A.; Nováček, V. Biological applications of knowledge graph embedding models. *Briefings in bioinformatics* **2021**, *22*, 1679−1693.

(36) Madsen, C.; Moreno, A. G.; Umesh, P.; Palchick, Z.; Roehner, N.; Atallah, C.; Bartley, B.; Choi, K.; Cox, R. S.; Gorochowski, T. Synthetic biology open language (SBOL) version 2.3. *J. Integr. Bioinf.* **2019**, DOI: 10.1515/jib-2019-0025.

(37) Sayers, E. W.; Cavanaugh, M.; Clark, K.; Ostell, J.; Pruitt, K. D.; Karsch-Mizrachi, I. GenBank. *Nucleic acids research* **2019**, *47*, D94−D99.

(38) Hütter, C. V.; Sin, C.; Müller, F.; Menche, J. Network cartographs for interpretable visualizations. *Nature Computational Science* **2022**, *2*, 84−89.

(39) Beal, J.; Nguyen, T.; Gorochowski, T. E.; Goñi-Moreno, A.; Scott-Brown, J.; McLaughlin, J. A.; Madsen, C.; Aleritsch, B.; Bartley, B.; Bhakta, S.; et al. Communicating structure and function in synthetic biology diagrams. *ACS synthetic biology* **2019**, *8*, 1818−1825.

(40) Baig, H.; Fontanarossa, P.; Kulkarni, V.; McLaughlin, J.; Vaidyanathan, P.; Bartley, B.; Bhakta, S.; Bhatia, S.; Bissell, M.; Clancy, K.; et al. Synthetic biology open language visual (SBOL Visual) version 2.3. *J. Integr. Bioinf.* **2021**, DOI: 10.1515/jib-2020-0045.

(41) Tamsir, A.; Tabor, J. J.; Voigt, C. A. Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'. *Nature* **2011**, *469*, 212−215.

(42) Tas, H.; Grozinger, L.; Goñi-Moreno, A.; de Lorenzo, V. Automated design and implementation of a NOR gate in Pseudomonas putida. *Synth. Biol.* **2021**, *6*, ysab024.

(43) Bartoli, V.; Dixon, D. O.; Gorochowski, T. E. *Synthetic Biology*; Springer, 2018; pp 399−409.

(44) Eick, S. Aspects of network visualization. *IEEE Computer Graphics and Applications* **1996**, *16*, 69−72.

(45) Gosak, M.; Markovič, R.; Dolenšek, J.; Rupnik, M. S.; Marhl, M.; Stožer, A.; Perc, M. Network science of biological systems at different scales: A review. *Phys. Life Rev.* **2018**, *24*, 118−135.

(46) Barabási, A.-L. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **2013**, *371*, 20120375.

(47) Karim, R. M.; Kwon, O.-H.; Park, C.; Lee, K. A Study of Colormaps in Network Visualization. *Appl. Sci.* **2019**, *9*, 4228.

(48) Serrano, L. Synthetic biology: promises and challenges. *Mol. Syst. Biol.* **2007**, *3*, 158.

(49) Liang, P.; Naik, M. Scaling abstraction refinement via pruning. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*; Association for Computing Machinery, 2011; pp 590−601.

(50) Heinemann, M.; Panke, S. Synthetic biology—putting engineering into biology. *Bioinformatics* **2006**, *22*, 2790−2799.

(51) Calles, B.; Goñi-Moreno, Á.; de Lorenzo, V. Digitalizing heterologous gene expression in Gram-negative bacteria with a portable ON/OFF module. *Mol. Syst. Biol.* **2019**, DOI: 10.15252/msb.20188777.

(52) Tinafar, A.; Jaenes, K.; Pardee, K. Synthetic biology goes cell-free. *BMC Biol.* **2019**, *17*, 1−14.

(53) Chavarría, M.; Goñi-Moreno, Á.; de Lorenzo, V.; Nikel, P. I. A metabolic widget adjusts the phosphoenolpyruvate-dependent fructose influx in Pseudomonas putida. *mSystems* **2016**, *1*, e00154-16.

(54) Yaman, F.; Adler, A.; Beal, J. AI challenges in synthetic biology engineering. In *Proceedings of the AAAI Conference on Artificial Intelligence*; Association for the Advancement of Artificial Intelligence, 2018.

(55) Beal, J.; Weiss, R.; Densmore, D.; Adler, A.; Appleton, E.; Babb, J.; Bhatia, S.; Davidsohn, N.; Haddock, T.; Loyall, J.; et al. An end-to-end workflow for engineering of biological networks from high-level specifications. *ACS Synth. Biol.* **2012**, *1*, 317−331.

(56) Sainz de Murieta, I.; Bultelle, M.; Kitney, R. I. Toward the first data acquisition standard in synthetic biology. *ACS synthetic biology* **2016**, *5*, 817−826.

(57) Ravasz, E.; Somera, A. L.; Mongru, D. A.; Oltvai, Z. N.; Barabási, A.-L. Hierarchical organization of modularity in metabolic networks. *science* **2002**, *297*, 1551−1555.

(58) Fortunato, S. Community detection in graphs. *Physics reports* **2010**, *486*, 75−174.

(59) Albert, R.; Jeong, H.; Barabási, A.-L. Error and attack tolerance of complex networks. *nature* **2000**, *406*, 378−382.

(60) Tellechea-Luzardo, J.; Otero-Muras, I.; Goñi-Moreno, A.; Carbonell, P. Fast biofoundries: coping with the challenges of biomanufacturing. *Trends Biotechnol.* **2022**, *40*, 831.

(61) Beal, J.; Goñi-Moreno, A.; Myers, C.; Hecht, A.; de Vicente, M. d. C.; Parco, M.; Schmidt, M.; Timmis, K.; Baldwin, G.; Friedrichs, S.; et al. The long journey towards standards for engineering biosystems: Are the Molecular Biology and the Biotech communities ready to standardise? *EMBO Rep.* **2020**, *21*, No. e50521.

(62) Mısırlı, G.; Taylor, R.; Goni-Moreno, A.; McLaughlin, J. A.; Myers, C.; Gennari, J. H.; Lord, P.; Wipat, A. SBOL-OWL: An ontological approach for formal and semantic representation of synthetic biology information. *ACS Synth. Biol.* **2019**, *8*, 1498−1514.