Research Article
pubs.acs.org/synthbio

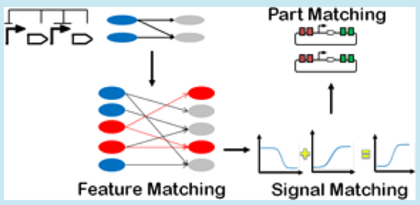
# Automated Selection of Synthetic Biology Parts for Genetic Regulatory Networks

Fusun Yaman,*,[†] Swapnil Bhatia,[‡] Aaron Adler,[†] Douglas Densmore,[‡,§] and Jacob Beal[†]

[†]Raytheon BBN Technologies, 10 Moulton St., Cambridge, Massachusetts, United States
Departments of [‡]Electrical and Computer Engineering and [§]Bioinformatics, Boston University, Boston, Massachusetts, United States

Ⓢ *Supporting Information*

**ABSTRACT:** Raising the level of abstraction for synthetic biology design requires solving several challenging problems, including mapping abstract designs to DNA sequences. In this paper we present the first formalism and algorithms to address this problem. The key steps of this transformation are *feature matching*, *signal matching*, and *part matching*. Feature matching ensures that the mapping satisfies the regulatory relationships in the abstract design. Signal matching ensures that the expression levels of functional units are compatible. Finally, part matching finds a DNA part sequence that can implement the design. Our software tool *MatchMaker* implements these three steps.



**KEYWORDS:** *abstract genetic regulatory network, feature matching, signal matching, part matching, MatchMaker*

Over the past decade, biologists have begun to establish engineering control over the genetic machinery of cells.[1,2] These synthetic biologists have identified, created, and isolated many DNA sequences that can be used as building blocks for novel cellular programs. These programs are then introduced into living cells and executed. Synthetic biology holds the potential for revolutionary advances in many applications, including medicine,[3] environmental remediation,[4] and biosensing.[5] As the desired systems become more complex, design automation for synthetic biology systems becomes an important enabling technology. Encoding design expertise in software will make engineering very complex systems tractable, increase the accessibility of synthetic biology to new practitioners, enable design reuse, allow for design sharing, introduce formal analysis methods, and increase system reliability by reducing the number of undetected design errors. In this paper, we present a design automation flow captured in a software application called *MatchMaker*. This process assigns appropriate DNA parts to instantiate an abstract genetic regulatory network, thus proposing a final set of DNA sequences that will implement the desired behavior in cells. These parts are selected not only on the basis of their functional classification but also on the basis of characterized behavior in order to ensure the correct performance of the overall system.

There are a number of natural mechanisms that can be manipulated in a cell to achieve a designed/desired behavior. Our work focuses on transcriptional logic systems, in which the computation is realized by the operation of a transcriptionally controlled network. The state-of-the-art techniques in synthetic biology for designing these networks require practitioners to design organisms at the DNA level, i.e., *genetic regulatory networks* (GRNs) as in Figure 1 (bottom). This low-level, manual process becomes unmanageable as the size of design grows. This is analogous to writing a high-level computer program using only the primitive instructions that can be directly executed by the computer processor (e.g., assembly language), which becomes difficult quickly as the size of the program grows. In addition, these specifications provide only one single instance as opposed to capturing a larger class of designs that have the potential to exhibit the desired behavior. Designs are limited to a single organism and a specific cellular context. An alternative approach would use a high-level language and compiler (e.g., Proto BioCompiler[6]) to enable designers to produce more sophisticated programs more quickly and with less effort. These concepts can be applied to synthetic biology, radically increasing the complexity of the designed systems. Tools to support complex designs have been developed including Tinkercell[7,8] and Device Editor[9] to visualize designs, Eugene[10] to specify design constraints and rules, GenoCAD[11] for static verification of the designs, and GEC[12,13] for simulation-driven search instantiations. However, none of these tools adequately address the specification to DNA design transformation challenge.

Computer science techniques have proven useful in other biological applications, for example, DNA-based compilation[14] and signal processing.[15] Alterovitz et al. outline some of the key problems facing synthetic biology including computational problems.[16] For example, they point out the need for improved algorithms for analyzing synthetic biology networks and statistical methods for analyzing the noisy data that biological systems produce. Andrianantoandro et al. use the same comparison to computer engineering that we use in this

**Special Issue:** Bio-Design Automation

**Received:** April 15, 2012
**Published:** July 6, 2012

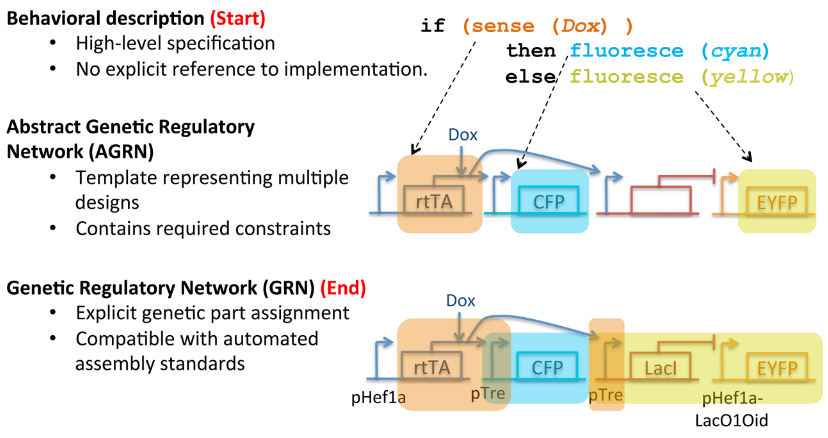

Behavioral description (Start)
High-level specification
No explicit reference to implementation.
if (sense (Dox) )
then fluoresce (cyan)
else fluoresce (yellow)
Abstract Genetic Regulatory Network (AGRN)
Template representing multiple designs
Contains required constraints
Dox
rtTA
CFP
EYFP
Genetic Regulatory Network (GRN) (End)
Explicit genetic part assignment
Compatible with automated assembly standards
Dox
rtTA
CFP
LacI
EYFP
pHef1a
pTre
pTre
pHef1a-LacO1Oid

**Figure 1.** (Top) High-level design describing organism-level behavior that can be compiled to abstract genetic regulatory networks (AGRNs). (Middle) Design at the AGRN level; none of the parts are assigned to DNA sequences. (Bottom) Design at the GRN level, fully specified and suitable for assembly.

paper.[17] They point out that the biological world is not yet as neat and easy to draw boundaries in as the computer world.

In our design framework, a top-down design approach will let practitioners design organisms using higher level descriptions (as in Figure 1 (top)). These descriptions will be mapped to a composition of primitive motifs, producing an *abstract genetic regulatory network* (AGRN), one that defines relationships between parts but leaves the actual identities of those parts unspecified (Figure 1 (middle)). To realize this network, one must solve the part selection problem: mapping abstract features to a collection of particular standardized biological parts that preserve the relationships between features prescribed by the network. Prior work has demonstrated the design of abstract GRNs from high-level programs[6] and automated assembly of DNA sequences from standardized biological parts such as BioBricks.[18,19] However, a critical gap exists in the actual selection of particular biological parts to implement the design.

The prior work that comes closest to addressing this gap is GEC.[12,13] GEC, however, does not actually contain any mapping from parts to expected composite behavior. This means that it cannot predict which parts are good to select but must search blindly and exhaustively through the space of possible designs, testing each combination with a costly chemical simulation. Moreover, this means that GEC depends critically on the availability of detailed chemical reaction models with precisely quantified rate constants, and at present these are generally impractical to obtain for synthetic biology systems. GEC also does not address issues of assembly.

In this paper we demonstrate that transforming high-level organism descriptions to DNA sequences involves solving several constraint satisfaction and optimization problems. Our research builds on top of the Proto BioCompiler,[6,20] which compiles an organism-level behavior description into a network of abstract biological parts (AGRNs). (The AGRN input could also be produced by other tools or by hand.) This paper focuses on transforming this abstract network into a concrete set of DNA parts (which ultimately encode a DNA sequence). The key steps in this transformation are (1) *Feature Matching*: finding compatible features (functional DNA elements) that have the same regulatory relationship as defined in the AGRN, (2) *Signal Matching*: choosing a specific set of parts within the family of required features such that the chemical concentration levels produced are compatible with each other to ensure robust system performance, and (3) *Part Matching*: finding standard composable parts that when put together result in the required function with the desired performance.

Explicitly stated, this paper's contributions are

1. The *definition* of three distinct transformations from a AGRN to a set of standard biological parts (feature matching, signal matching, and part matching).
2. The *formalization* of these transformations as known computer science and artificial intelligence algorithms.
3. A *software application* that can perform these transformations (MatchMaker).

## ■ RESULTS

In a prototypical transcriptional network, the relationship between a regulating protein and the promoter preceding a gene defines if/when the gene can be transcribed and then translated. A regulating protein can *repress* or *activate* a promoter. Repressors disable the ability of a promoter to initiate transcription. Activators enhance/enable a promoter's ability to initiate transcription. A promoter can be regulated by multiple proteins, allowing it to implement more complex logical relationships (e.g., NAND, NOR). Promoter regulation can also be used for non-Boolean computation, though in this paper we restrict ourselves to Boolean relations only. Throughout this paper we will call a DNA sequence that can perform a biological function, such as a promoter or a protein, a *feature*. (This terminology is borrowed from the Clotho[21] data model.) A GRN is a transcriptional network in which every network element is associated with a feature (Figure 1 (bottom)). In an AGRN all network elements are only partially specified (Figure 1 (middle)). For example, instead of a specific promoter feature, an element might be associated with a generic *repressible promoter*. Essentially, an AGRN corresponds to a collection of GRNs, since for each unspecified element a number of features are potential candidates. Our goal in this paper is to pick a near-optimal instance of these GRNs and determine an automated method to turn the selected GRN into a DNA sequence so that it can be implemented in a cell.

In this paper, we present the MatchMaker software application, which accepts a description of an AGRN and returns a sequence of standard DNA parts that can be assembled into the desired DNA sequence. MatchMaker has three steps, as depicted in Figure 2:

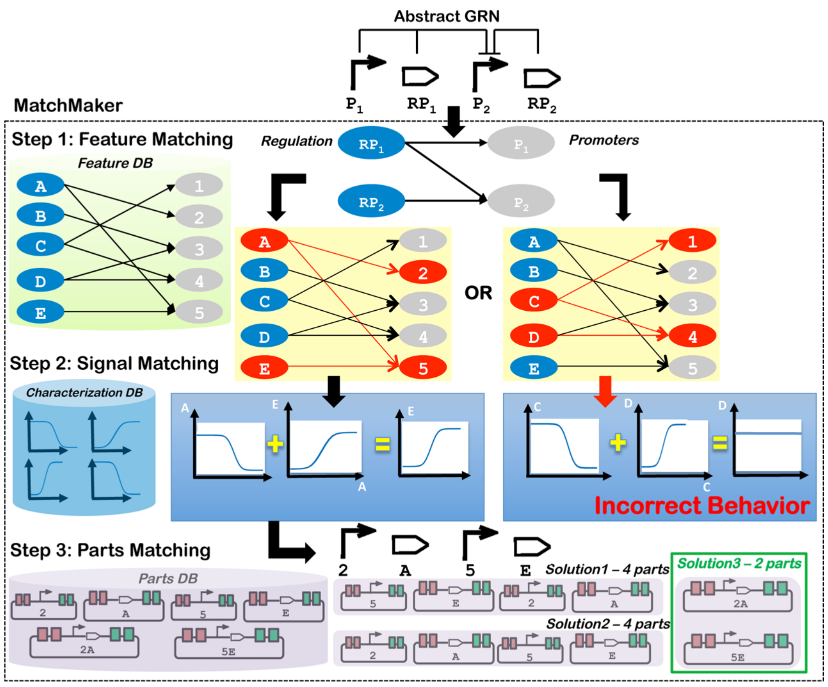


**Figure 2.** Three-step solution to transforming an AGRN to a GRN. (1) Feature Matching: Selecting features that implement the functional requirements outlined in the AGRN. As shown, this requires examining both a feature database and also a bipartite graph representation of the AGRN. A heuristic search is used to determine a feature assignment. (2) Signal Matching: By using a characterization database, it can be determined whether the composition of the features from Step 1 will result in a system with the desired performance. (3) Part Matching: Assigning parts to the features selected. There may be a number of potential part assignments; heuristics such as part count can be used to select a final solution.

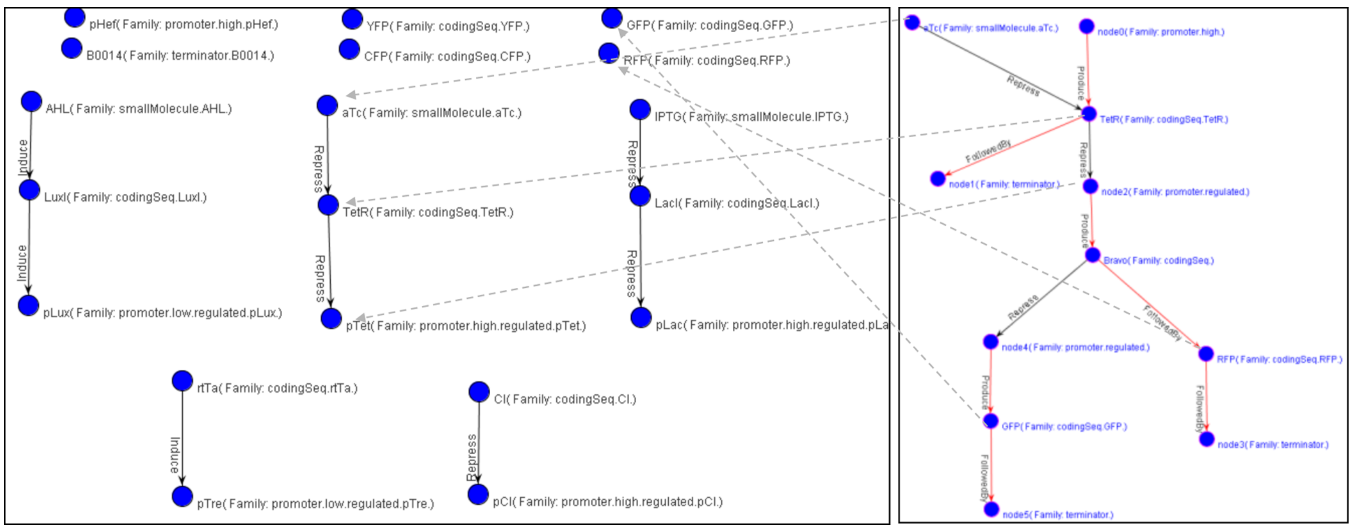

pHef( Family: promoter.high.pHef.)
B0014( Family: terminator.B0014.)
AHL( Family: smallMolecule.AHL.)
LuxI( Family: codingSeq.LuxI.)
pLux( Family: promoter.low.regulated.pLux.)
YFP( Family: codingSeq.YFP.)
CFP( Family: codingSeq.CFP.)
aTc( Family: smallMolecule.aTc.)
TetR( Family: codingSeq.TetR.)
pTet( Family: promoter.high.regulated.pTet.)
GFP( Family: codingSeq.GFP.)
RFP( Family: codingSeq.RFP.)
IPTG( Family: smallMolecule.IPTG.)
LacI( Family: codingSeq.LacI.)
pLac( Family: promoter.high.regulated.pLa
rtTa( Family: codingSeq.rtTa.)
pTre( Family: promoter.low.regulated.pTre.)
CI( Family: codingSeq.CI.)
pCI( Family: promoter.high.regulated.pCI.)
Induce
Repress
aTc( Family: smallMolecule.aTc.)
node0( Family: promoter.high.)
TetR( Family: codingSeq.TetR.)
node1( Family: terminator.)
node2( Family: promoter.regulated.)
Bravo( Family: codingSeq.)
node4( Family: promoter.regulated.)
RFP( Family: codingSeq.RFP.)
GFP( Family: codingSeq.GFP.)
node3( Family: terminator.)
node5( Family: terminator.)
Produce
FollowedBy

**Figure 3.** Visualization of the feature database (left) and the graphical representation of the AGRN to be designed (right). The dotted arrows illustrate potential assignments of features to the AGRN. This results ultimately in a subgraph isomorphism problem, which is NP-Complete. We solve this problem using a heuristic search.

- The first step (*feature matching*) assigns features to AGRN elements such that the repression/activation relationships are satisfied. This step assumes that feature databases containing qualitative repression/activation relationships exist. The assignment problem is then converted into a classical subgraph isomorphism problem and solved using a heuristic search.
- *Signal matching* is the problem of finding the best assignment with respect to chemical signal compatibility. In this step, the expression levels of the biological devices in the AGRN are examined and their compatibility is verified. The information needed for this verification is assumed to be coming from characterization databases. Characterization data quantitatively define the relationship between expression levels and the levels of the regulating proteins.
- The first two steps will produce feature assignments that are compatible both at the functional and signal levels. Features are abstract sequence annotations describing a specific function and may differ from sequences of parts—the basic elements of DNA that will actually be composed.[21] Thus, to derive an assembly plan, features

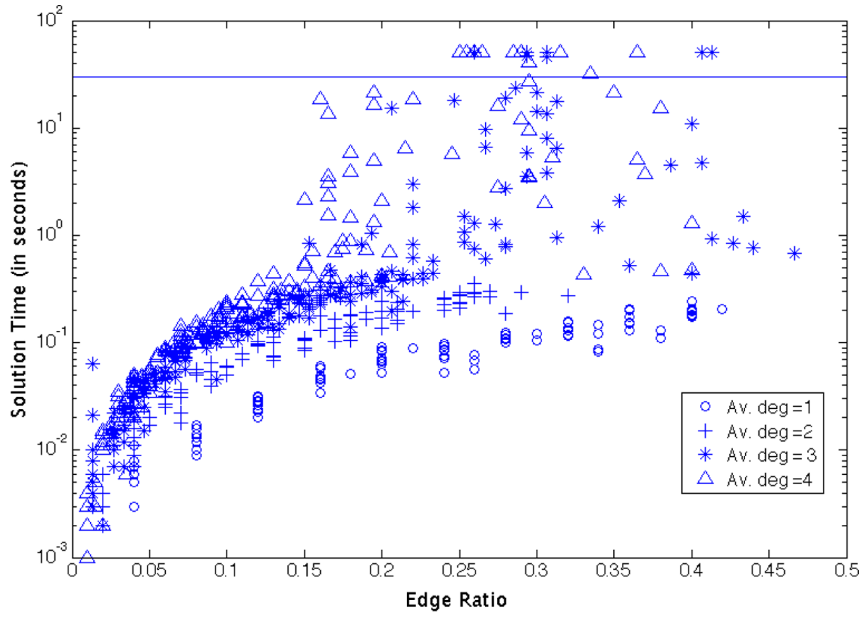


**Figure 4.** This graph plots the solution time versus the edge ratio of the AGRN and feature database. All but 15 of the 700 problems are solved in under 30 s. This shows that setting a time limit for the computation is an acceptable approach to this NP-Complete problem.

must be mapped to specific parts, and parts to specific samples containing those parts. The goal of the last Matchmaker step, *part matching*, is to solve the problem of finding a short part sequence that implements the AGRN. The parts are later mapped to specific samples by assembly tools downstream of MatchMaker.

We have implemented the MatchMaker design approach and also integrated it with the Clotho[21] framework as an *App*. Clotho databases for features, characterization, and part information are populated using hand generated data (as this system grows so will the quality, type, and source of data in these databases).

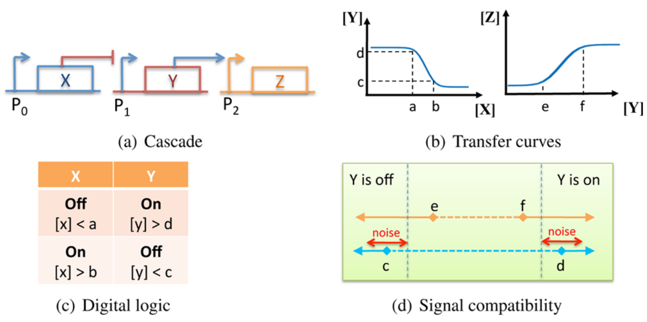
MatchMaker expects an input AGRN in the form of an SBOL[22] compatible XML description that can be produced automatically by tools such as Proto BioCompiler. The output of MatchMaker, which is a sequence of parts, is translated into native Clotho data structures. The MatchMaker output can then be used by other Apps, e.g., for automated assembly.

The MatchMaker software and source code are freely available. They are distributed with the "GPL with Linking Exception" license, which roughly means anybody can use the tool as a library without the GPL license affecting their own program. If, however, the source of MatchMaker is edited or modified, then the edits to MatchMaker should be contributed as open source. Currently MatchMaker is distributed via email (requests to the contact author) with limited documentation. Soon it will be available as a web-service on the Web site https://synbiotools.bbn.com/.

**Feature Matching.** The feature matching problem involves finding a set of features in a feature database with relationships among them mirroring those required by the AGRN. MatchMaker uses the feature matching algorithm, described in Methods, and a parts database (Figure 3) to convert an AGRN to a GRN. Feature matching involves finding a strict subgraph that is isomorphic to a given graph. The classical subgraph isomorphism problem is NP-Complete. There are no known algorithms for solving NP-Complete problems[23] in polynomial time, but many NP-complete problems can be solved quickly for most cases using heuristic methods. Our problem has three differences from the subgraph isomorphism problem: the subgraph must be strict, the graphs involved are bipartite, and the isomorphism must preserve vertex types. In the Supporting Information we prove that these differences do not affect the complexity of the problem. However, techniques exist to mitigate the difficult nature of the problem. For example, in the feature matching problem, the database of features may be relatively static as the rate of discovery of new features may be slow. One solution would be to preprocess the database for feature matching enabling polynomial time solutions.[24] In MatchMaker we address the difficulty of the problem in a different way by using a time-limited heuristic search.

We conducted an experiment to determine the complexity characteristics of the feature matching problem. The details of this experiment can be found in Methods. We found that only 15 of the 700 runs took longer than 30 s to find a solution (Figure 4). Although 30 s can be unacceptably slow in many application domains, in synthetic biology where assembly of the physical parts takes on the order of days, this result is more than acceptable. The data points in the graph are grouped by the average degree of the nodes in the feature graph. The feature graphs with degrees greater than 2 take the longest computation time. Also note that solution time spikes when the ratio of edges in AGRN to feature graph is between 0.25 and 0.35.

**Signal Matching.** We have formalized the definition of a biological device (a collection of parts) with multiple inputs and outputs. In our formalism, a device consists of a promoter, all of its regulators, and its outputs (more precisely, a device with multiple outputs is represented as multiple devices with single outputs). We assume each such device has an underlying transfer function that determines the expression level of the outputs. For each device, we allow high/low values for all inputs and outputs. In order to support small-molecule reactions, we have defined composite inputs that have their own high/low output and inducer levels defined. Note that this representation is agnostic of the underlying transfer function. It solely requires a binary interpretation, i.e., when are the input/outputs on or off as shown in Figure 5c of a transfer curve



| X | Y |
|---|---|
| Off<br>[x] < a | On<br>[y] > d |
| On<br>[x] > b | Off<br>[y] < c |

**Figure 5.** In order to preserve digital behavior in a cascade like (a), where X controls the production of Y and Y controls the production of Z, it is necessary to perform signal matching. The transfer curves for the circuit are illustrated in (b): increasing the presence of X decreases the production of Y (left) and increasing the presence of Y increases the presence of Z (right). The functions provide thresholds: X at or below level a results in a value of Y at or above level d; X at or above level b results in a value of Y at or below level c; Y at or below level e results in a low level of Z; Y at or above level f results in a high level of Z. Values between these levels mean the corresponding level is not at a clearly high or low level, e.g., Y between e and f means uncertainty in the level of Z. The digital logic representation of the X/Y transfer curve is shown in (c). Biological circuits are inherently noisy; signal matching must take into account the noisy signals as illustrated in (d). Even with noise, Y is still interpreted as off because level c plus noise is still less than level e. Likewise, level d minus noise is still greater than f, preserving the on behavior for Y.

(Figure 5b). Two devices, such that the output of one is an input to the other, can be safely composed, i.e., the resulting device will be digital, if devices have compatible signals. In our context, signals are represented by the concentration of molecules. We define that devices are signal compatible if and only if the noisy output range of the connecting element (for example Y in the cascade in Figure 5a) is contained in valid input range as shown in Figure 5d. Note that if more than one device is producing the same output molecule then a system-wide computation of low/high values is required (see Signal Matching in Methods for details).

Given a GRN with fully specified features, testing for signal compatibility can be done in polynomial time (see Chart 4). If robustness of the circuit is critical, MatchMaker can search through all possible GRNs (within a time limit) and return the most noise-tolerant network.

**Part Matching.** A GRN is a network, but our target is a set of easily composable DNA sequences that has been linearized into a single DNA molecule. Currently, our formalization enforces strict precedence only on the promoter and the gene(s) it controls. Thus, finding a total order on the elements of any GRN can be achieved using a linear time algorithm (see Chart 5).

Next we pick parts that when assembled produce the sequence computed in the linearization stage. When dealing with simple parts, i.e., parts that contain a single feature, part selection can also be solved in linear time. However, availability of composite parts leads to a set of possible solutions. Among those solutions, MatchMaker prefers the ones that can be implemented using fewer parts. The rationale behind this preference is ease, time, and reliability of construction by reducing the number of assembly steps and reagents required. The part matching algorithm (see Chart 6) uses a greedy search technique to find near optimal solutions, as finding the optimal solution using deterministic algorithms can take exponential time.

## DISCUSSION AND FUTURE IMPROVEMENTS

MatchMaker is the first tool and algorithm for model-based transformation of an AGRN into a GRN, as well as for finding the actual part (simple or composite) sequence that can implement the GRN. MatchMaker applies techniques from artificial intelligence, such as constraint satisfaction and greedy and heuristic search, to enable this transformation. This is also the first time that the steps of this transformation (i.e., feature matching, signal matching, and part matching) have been formalized.

A next step in improving our current formulation is to explicitly represent the AGRN semantics. This can be solved by including truth table interactions in the BioCompiler output (the AGRN) and the parts database. MatchMaker could then guarantee that the intended behavior would be preserved.

Another improvement to MatchMaker would be additional constraints on linearization and optimization. For example, Eugene[25] could provide a metric on preferred sequences that could then be used to provide constraints for the search.

Our future work includes allowing the availability of particular parts to affect the high-level design of the system. This could include allowing customizations of the part matching objective (instead of just selecting the shortest part sequence). For example, multiple AGRNs might implement the same high-level program, but only one of these AGRNs might be implementable given the parts available in a particular lab.

MatchMaker requires a database of biological parts and their characterization data. We tested MatchMaker with a database of parts and characterization data containing composite and basic parts for test circuits and verified the part assignments for correctness. Our database implemented the Clotho data model to model feature-feature and feature-part relationships and to associate the low and high inflection points obtained from the characterization data to parts and samples. We did not explore whether other data models such as the MIT Registry of Standard Biological Parts[26] or the JBEI ICE registry[27] may be adapted to the needs of the algorithms. As currently defined, neither the MIT Parts Registry nor the JBEI-ICE registry models make a distinction between features and parts, and model relationships among features; these are essential to MatchMaker's algorithms.

MatchMaker is developed as part of the TASBE project (Beal, J. et al. DOI: 10.1021/sb300030d). The goal of the TASBE project is to develop a tool-chain that will take a high-level design and ultimately assemble the cells that contain the DNA realizing the high-level design. TASBE and MatchMaker have been validated on simple programs. The feature and parts databases populated in that project are biologically sound. However the signal database, which was populated from the characterization experiments, was very small (only three devices). Further validation of MatchMaker is needed with larger signal databases generated from physical experiments as outlined in the TASBE Characterization Technical Report.[28]

The MatchMaker implementation is SBOL compliant in the sense that it accepts its input in the form of an SBOL file. The current SBOL standards[22] are not sufficiently rich to allow the complete specification of an AGRN; for example, the current standard does not explicitly define part type and regulation information. Our implementation adds such meta-information in the string attributes associated with an SBOL Component and Annotation. Similarly, the characterization data required by MatchMaker algorithms cannot be currently included or associated with SBOL parts within the current SBOL framework. We believe, however, that some of the planned extensions to SBOL, such as the Performance and Measurement extension, will allow for the inclusion of such information in the future.

## METHODS

In this presentation we focus on two kinds of biological features (DNA sequences with a particular function): promoters and protein coding sequences in a transcriptional network (we will use proteins when the meaning is obvious). We will denote these two sets with $\mathcal{P}$ and $\mathcal{R}$, respectively. Later we discuss how additional features can be handled in the model.

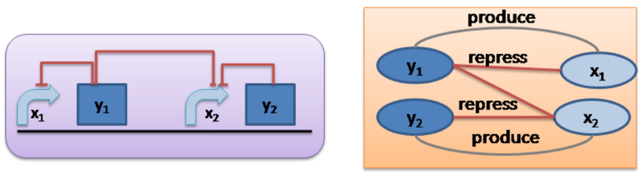
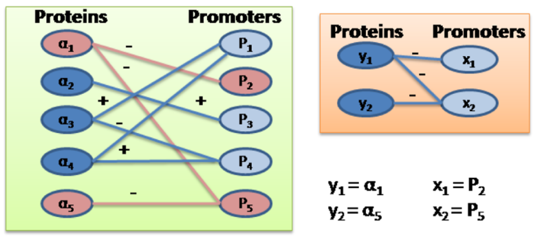


**Figure 6.** Two visualizations of a GRN. On the left is a DNA sequence representation with rectangles as protein coding sequences (PCS) and block arrows as promoters. The red lines from PCS to promoters indicate repression. On the right is the same GRN in a graph representation with labeled edges.



Proteins
Promoters
$\alpha_1$ $\alpha_2$ $\alpha_3$ $\alpha_4$ $\alpha_5$
$P_1$ $P_2$ $P_3$ $P_4$ $P_5$
$y_1 = \alpha_1$
$y_2 = \alpha_5$
$x_1 = P_2$
$x_2 = P_5$

**Figure 7.** The AGRN (without the production edges) on the right is isomorpic to the graph induced by the vertices $\alpha_1, \alpha_5, P_2, P_5$ from the feature database on the left. The − and + edge labels indicate *repress* and *activate*, respectively.

A genetic regulatory network (GRN) is a bipartite directed graph with labeled edges and each node associated with a feature from the set $\mathcal{P} \cup \mathcal{R}$. To denote the feature associated with a vertex $v$ we will use the notation $feature(v)$. A vertex is a promoter (or protein) vertex iff $feature(v) \in P$ (or $feature(v) \in R$). Furthermore in a GRN, the edges are always between a promoter vertex and a protein vertex. The edges have one of the following labels: *produce*, *repress*, or *activate*.

An abstract genetic regulatory network (AGRN) is similar to a GRN with the main difference that nodes are associated with a subset of $\mathcal{P}$ or a subset of $\mathcal{R}$, so for an AGRN vertex $v$, $feature(v)$ is a set. It is important to note that a vertex in an AGRN is associated with only one type of feature. An AGRN vertex $v$ is a promoter vertex if $\text{feature}(v) \subseteq \mathcal{P}$. The protein vertex definition is generalized the same way. Essentially, an AGRN corresponds to a collection of GRNs.

**Feature Matching.** The quantitative relationships between biological features are discovered by biologists experimentally. We assume existence of feature databases containing such relationships. In turning an AGRN into a GRN by mapping each node to a feature we need to ensure that

- The edges in the GRN are supported by the feature database. If the features we selected are not biologically capable of interacting in the desired manner, the GRN is not executable.
- The feature database does not imply additional relationships between the features of the GRN nodes. If two features are known to interact with each other, then whether or not we intended them to, they will interact, possibly disrupting the designed behavior. We first define a *feature database*:

**Definition 1.** Let $\mathcal{P}$ and $\mathcal{R}$ be finite sets denoting promoters and proteins respectively. A **feature graph** is a bipartite graph $\mathcal{F}$ with parts $\mathcal{P}$ and $\mathcal{R}$, and edges $\mathcal{T}$ labeled from {repress,activate}. A **feature database** is a tuple $\mathcal{D} = \langle \mathcal{P}, \mathcal{R}, \mathcal{F}, \mathcal{T} \rangle$.

The left side of Figure 7 is a feature graph (edge labels − and + are short hand notations for *repress* and *activate*, respectively); together with the set of proteins $\mathcal{R} = \{\alpha_1, ..., \alpha_5\}$ and promoters $\mathcal{P} = \{P_1, ..., P_5\}$ they form a feature database.

When mapping the nodes of an AGRN to features we can ignore the production edges (edges labeled as *produce*) in the AGRN because they are unrelated to the two constraints above since any promoter can produce any protein. A **constraint graph** $\langle P \cup R, E \rangle$, induced by an AGRN is the same graph as the AGRN except the production edges are dropped. In this notation, $P$ is the set of all promoter nodes in AGRN, $R$ is the set of all protein nodes in the AGRN, and $E$ is the set of all edges. The right graph in Figure 7 is the constraint graph induced by the AGRN in Figure 6. Note that for this example $feature(x_i)$ is all promoters and $feature(y_i)$ is all proteins.

Next we will define a *topological solution* of an AGRN wrt a feature database. Basically we are looking for a subset of vertices in the feature database such that a subgraph of the feature

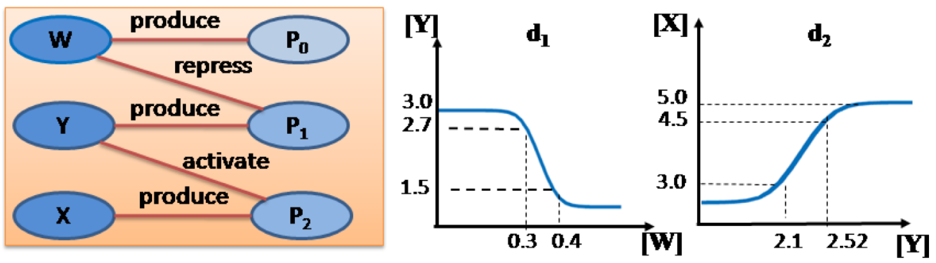


**Figure 8.** The concentration of Y is a function of W. The concentration of X is a function of Y. $d_1 = \langle\{W\}, P_1, Y\rangle$ and $d_2 = \langle\{Y\}, P_2, X\rangle$. $d_1$ is signal compatible with $d_2$ because $2.7 > 2.52$ and $1.5 < 2.1$.

Chart 1

**Algorithm 1** MatchMaker

**Require:** A feature database $F$, a signal database $S$, a parts database $P$, an AGRN $A$, expected noise level *noise*, the search *mode* `best` or `first` and the time limit *timeLimit* for the search.

**Ensure:** A part sequence that implements a quantitative solution of $A$ (or the best if the mode was `best` ) or failure if no solution exists.

```
 1: Let $D$ be a mapping from nodes of $A$ to subsets of $F$ nodes; effectively to represent the domains
    of each node in $A$
 2: for each node $n$ in $A$ do
 3:     Let $d_n$ be the set of nodes in $F$ that is of same type as $n$ and have at least same or more in/out
        degree.
 4:     Remove any node $m$ from $d_n$ if there is no device in $S$ containing $feature(m)$
 5:     Remove any node $m$ from $d_n$ if there is no part in $P$ implementing $feature(m)$
 6:     Add $(n, d_n)$ to $D$
 7: $D = ensureEdgeConsistency(F, A, D)$
 8: if there is a node $n$ with empty domain, i.e., $n_d = \emptyset$ then
 9:     Return $failure$
10: $deadline = currentTime + timeLimit$
11: Let $\pi$ be an empty mapping from nodes of $A$ to nodes of $F$
12: $qSolutions = solveFeatureMatching(F, S, A, D, \pi, noise, mode, deadline)$
13: Sort $qSolutions$ w.r.t. to noise margin such that the GRNs with larger noise margin is ordered
    first.
14: for each GRN $G$ in quantitativeSolutions do
15:     $L = linearize(G)$
16:     $pSequence = solvePartMatching(L, P)$
17:     if $pSequence$ is not $failure$ then
18:         Return $pSequence$
19: Return $failure$
```

Chart 2

**Algorithm 2** ensureEdgeConsistency

**Require:** A feature database $F$, an AGRN $A$, and a mapping $D$, from nodes of $A$ to subsets of nodes in $F$

**Ensure:** A mapping $D^*$, from nodes of $A$ to subsets of nodes in a $F$

```
 1: $D^* = D$;
 2: while $D^*$ change do
 3:     for each edge $e = (v, w)$ in $A$ do
 4:         Let $d_v$ and $d_w$ be the domains such that $(i, d_i) \in D^*$
 5:         Let $d_v^* = \emptyset$ and $d_w^* = \emptyset$
 6:         for each node $n$ in $d_v$ do
 7:             for each node $m$ in $d_w$ do
 8:                 if $(n, m)$ is an edge in $F$ and has the same label as $e$ then
 9:                     Add $n$ to $d_v^*$ and $m$ to $d_w^*$
10:         Replace $(v, d_v)$ with $(v, d_v^*)$ in $D^*$
11:         Replace $(w, d_w)$ with $(w, d_w^*)$ in $D^*$
12: Return $D^*$
```

Chart 3

```
Algorithm 3 SolveFeatureMatching
Require: A feature database $F$, a signal database $S$, an AGRN $A$, the domains for nodes in AGRN
    $D$, partial assignment $\pi$ for the nodes in $A$, expected noise level $noise$, the search $mode$ best
    or first and the time limit $deadline$
Ensure: A set of quantitative solutions to $A$ w.r.t. $F$ and $S$
 1: $solutions = \emptyset$
 2: if $currentTime >= deadline$ then
 3:     Return $solutions$
 4: if $\pi$ contains an assignment for all nodes in $A$ then
 5:     Let $G = A$ such that for every node $n$ in $G$, $feature(n)=feature(\pi(n_A))$ where $n_A$ is the
        node in $A$ corresponding to node $n$
 6:     $result = CheckSignalMatching(S, G, noise)$
 7:     if $result$ is not $failure$ then
 8:         $solutions = solutions \cup \{G\}$
 9:     Return $solutions$
10: Rank the unassigned nodes ($|d_x| < |d_y|$ implies $rank(x) > rank(y)$ where $(i, d_i) \in D$)
11: Let $v$ be the highest rank node and $d_v$ be the domain of $v$ as given in $D$,
12: Rank the values in $d_v$ (appears in more variable domains in $D$ –> higher rank)
13: for every value $w$ in $d_v$ in rank order do
14:     $newD = D$
15:     Replace $(v, d_v)$ with $(v, \{w\})$ in $newD$
16:     $\pi* = \pi * \cup \{(v, w)\}$
17:     for each $(n, d_n) \in newD$ do
18:         Replace $(n, d_n)$ with $(n, d_n - \{w\})$ in $newD$
19:     Let $independentNodes$ be the set of nodes in $A$ that don't share an edge with $v$
20:     Let $interferingValues$ be the set of nodes in $F$ that share an edge with $w$
21:     for each $n$ in $independentNodes$ do
22:         Replace $(n, d_n)$ with $(n, d_n - interferingValues)$ in $newD$
23:     $D = ensureEdgeConsistency(F, A, D)$
24:     if there is a $(n, d_n) \in newD$ such that $d_n = \emptyset$ then
25:         Continue with next iteration
26:     $solutions = solutions \cup solveFeatureMatching(F, S, A, newD, new\pi*, mode, deadline)$
27:     if mode is first and $solutions \neq \emptyset$ then
28:         Return $solutions$
29: Return $solutions$
```

database, containing only those vertices and edges between them, is isomorphic to the AGRN.

**Definition 2.** Given a graph $G$, a **strict subgraph** of $G$ is the graph induced by any subset $S$ of vertices of $G$ and every edge with both vertices in $S$.

In Figure 7, the strict subgraph induced by the vertices $\{\alpha_1, \alpha_5, P_2, P_5\}$ is highlighted with red.

**Definition 3.** Given a feature database $\mathcal{D} = \langle \mathcal{P}, \mathcal{R}, \mathcal{F}, \mathcal{T} \rangle$ and an AGRN whose induced constraint graph is $G = \langle P \cup R, E \rangle$, a one-to-one function $f{:}V \rightarrow S \subseteq (\mathcal{P} \cup \mathcal{R})$ is a topological solution of the AGRN wrt $\mathcal{D}$ iff

1. For every vertex $v$ of $G$, $f(v) \in feature(v)$, i.e., every vertex in the constraint graph is assigned to a feature from the set the vertex is associated with, and
2. The strict subgraph of $\mathcal{F}$ induced by $S$ is isomorphic to $G$, i.e., all edges in the AGRN are in $\mathcal{F}$ and all edges in $\mathcal{F}$ between the vertices $S$ are in $G$, and
3. For every edge $\{u,v\}$ of $G$, $\{u,v\}$ and $\{f(u),f(v)\}$ have the same label, i.e., the edges in both graphs represent same type of relationships.

A topological solution to AGRN in Figure 6 wrt to the feature database in Figure 7 maps node $y_1$ to $\alpha_1$, $y_2$ to $\alpha_5$, $x_1$ to $P_2$, and $x_2$ to $P_5$. A topological solution of an AGRN has a

Chart 4

```
Algorithm 4 CheckSignalMatching
Require: A set of available devices $C$ and a GRN $G$ and expected noise level $n$.
Ensure: Returns failure if any of the devices in $G$ is not in $C$ or not n-signal compatible with $G$.
    O/W returns the noise margin of $G$.
 1: Let $D$ be the devices in $G$
 2: for each device $d = \langle \mathscr{I}, p, o \rangle$ in $D$ do
 3:   if There is no device in $C$ that implements $d$, i.e., same $\mathscr{I}$, $p$, and $o$. then
 4:     Return $failure$.
 5:   else
 6:     Let $S_d$ be the spec of the device in $C$ that implements $d$.
 7: for each output $o$ such that a device $d = \langle \mathscr{I}, p, o \rangle$ is in $D$ do
 8:   Compute effective parameters $H_{G,n}(o)$ and $L_{G,n}(o)$.
 9: $minNoiseMargin = +\infty$
10: for each device $d = \langle \mathscr{I}, p, o \rangle$ in $D$ with spec $\langle h, l \rangle$ do
11:   for each input $i \in \mathscr{I}$ do
12:     if not ( $H_{G,n}(i) > h(i)$ and $L_{G,n}(i) < l(i)$) then
13:       Return $failure$.
14:     else
15:       $noiseMargin_{d,i}$ = minimum of $H_{G,n}(i) - h(i)$ and $l(i) - L_{G,n}(i)$
16:       if $noiseMargin_{d,i} < minNoiseMargin$ then
17:         $minNoiseMargin = noiseMargin_{d,i}$
18: Return $minNoiseMargin$
```

Chart 5

```
Algorithm 5 Linearize
Require: A GRN $G$.
Ensure: Returns sequence of features which a linearization of $G$
 1: Let $L$ be an empty sequence of features
 2: for promoter node $p$ in $G$ in some order do
 3:   Append $feature(p)$ to $L$
 4:   for every regulator node $r$ such that $(r, p)$ is a production edge in $G$ do
 5:     Append $feature(r)$ to $L$
 6: return $L$
```

corresponding GRN, which is unique and can be trivially constructed.

Finally we define the **feature matching problem** as follows: Find a topological solution of a given AGRN with respect to a given feature database.

*Extension To Handle Small Molecule Interactions and Other Features.* The notations and definitions for the feature matching problem can trivially be extended for supporting small molecule reactions. Essentially all we need to do is to remove the bipartite graph requirement from both the feature database and AGRN/GRN definitions. The edge relationships between promoters and regulators will still be preserved; however, the graphs will have a third layer of nodes, of type small molecule, which will be connected to regulators with induction/repression relationships. The feature matching problem would still be an instance of the strict graph isomorphism problem. The same would hold if the network had other types of features such as terminators and/or ribosomal binding sites (RBSs). Furthermore the complexity of the problem would be unchanged. The same algorithm (see Chart 3) for solving the feature matching problem would be used for solving the extended version of the problem. The current implementation of MatchMaker supports small molecule interactions and terminators.

**Experimental Setup.** To investigate the characteristics of the feature matching problem, we have run a set of experiments using randomly generated feature graphs with 100 nodes with the following control variables: average node degree (values ranging from 1 to 4) and maximum node degree (which is always 2 more than average node degree). Experiments with randomly generated AGRNs with up to 60 nodes and the same characteristics as the feature graph demonstrated that for

Chart 6

```
Algorithm 6 SolvePartMatching
Require: A sequence of features $L = [f_0, f_1, \ldots, f_n]$ and a part database $P$.
Ensure: Returns sequence of parts which implements $L$
 1: Let $S$ be an empty sequence of parts
 2: if L is empty then
 3:     return $S$
 4: Let $P*$ be an empty list of parts
 5: for part $p$ in $P$ do
 6:     if The sequence $p$ implements is a subsequence of $L$ then
 7:         Add $p$ to $P*$
 8: if P* is empty then
 9:     return failure
10: Sort $P*$ such that the parts that implement longer feature sequences are first.
11: for part $p$ in $P*$ do
12:     Append $p$ to $S$
13:     Let $[f_0, f_1, \ldots, f_i]$ be the feature sequence $p$ implements
14:     $rest = SolvePartMatching([f_{i+1}, \ldots f_n], P)$
15:     if rest = failure then
16:         Remove $p$ from $S$
17:     else
18:         Append $rest$ to $S$ and return the result
```

AGRNs with more than 15 edges, a solution is less likely and the solution is found in under 1.5 s for all cases. Our second set of experiments generates AGRNs from a graph induced by a subset of feature graph vertices. This method produces solvable AGRNs most of the time. We used the number of promoters and proteins in the AGRN as the control variables in extracting a subgraph of the feature database. The number of total nodes in the AGRN ranges from 4 to 60. We terminated executions after 50 s. Figure 4 demonstrates the solution time for 700 random runs, only 15 of which are above 30 s (on MacBook Pro with a 2.8Ghz dual core processor and 8 GB memory).

**Signal Matching.** Just like a digital circuit is composed of several devices, an AGRN is a composition of **biological devices.** In a GRN there is a device per promoter node. The inputs of the device are the protein nodes that are linked to the promoter with repression and activation edges. The outputs of the device are the protein nodes that are linked to the promoter with production edges. Without losing any generalization we will assume that each device has only one output. This is because devices with multiple outputs can be modeled as multiple one-output devices, which leads to a linear number of extra devices and does not impact the scalability of the approach. We will denote a device as $d = \langle \mathcal{I},p,o\rangle$ where $\mathcal{I}$ is set of proteins that are inputs, $p$ is a promoter, and $o$ is the output protein. In Figure 8, the GRN on the left has two devices: $d_1 = \langle\{W\},P_1,Y\rangle$ and $d_2 = \langle\{Y\},P_2,X\rangle$.

A device defines a function from the concentration of input proteins to the concentration of output protein. The sigmoidal curves on the right side of Figure 8 are examples of such functions for devices $d_1$ and $d_2$ (single-input devices). The characteristics of the curve (slope, height, etc.) come from the biochemical properties of the features that make up the device. The direction (increasing vs decreasing) is a function of repression/activation relationships.

Adapting digital logic, we will assume the existence of two values per input and output of the device: **high** and **low signal** threshold. Any output $o$ of the device higher (lower) then $high_o$ ($low_o$) will be considered as boolean true (false). Any output value between $low_o$ and $high_o$ has an ill-defined truth value. Similar arguments hold for the device inputs. In Figure 8, looking at the curve for $d_1$, the low value for the output $Y$ is 1.5 and the high value is 2.7. The high and low values per input are the **specifications** of a device. We denote the specification of a device $d = \langle \mathcal{I},p,o\rangle$ as $S_d = \langle h,l\rangle$ where $h$ (similarly $l$) is a function from $\mathcal{I} \cup \{o\}$ to reals for the high (similarly low) signal threshold.

Note that this representation does not associate a semantics with the device. That is, the relationship between the inputs and outputs is not explicitly given as in a truth table. This lack of semantics is also mirrored in the AGRN representation. For the feature matching problem the lack of semantics can be circumvented by allowing a more complex type hierarchy on the promoter nodes. These design decisions are heavily influenced by the BioCompiler AGRN representation which also lacks semantics. Our future work on automated part matching will focus on explicit semantic representations for AGRNs.

When can we connect two devices such that they perform digital computation? Consider the devices $d_1$ and $d_2$ from Figure 8. $d_1$'s output $Y$ is an input to $d_2$. $d_2$ is compatible with $d_1$ if both interpret the truth values of $Y$ in a similar way, that is, if $d_1$'s output value is a boolean true according to the specifications of $d_1$, then the same value should be interpreted as true according to $d_2$'s specifications.

**Definition 4.** Let $d$ and $d'$ be two devices with the specifications $\langle h,l\rangle$ and $\langle h',l'\rangle$ respectively. If the output $o$ of $d$ is an input of $d'$, then $d$ is **signal compatible** with $d'$ iff $h(o) > h'(o)$ and $l(o) < l'(o)$. Suppose $n$ is the upper limit of noise on expression level, then $d$ is ***n*-signal compatible** with $d'$ iff $h(o) - n > h'(o)$ and $l(o) + n < l'(o)$.

Note that if the output of first device is not an input to the second, by definition the devices are compatible. Revisiting the devices from Figure 8, $d_1$ is signal compatible with $d_2$ because 2.7 > 2.52 and 1.5 > 2.1. If the level of noise on expression level is known, then the compatibility interval can be conservatively tightened (as in the definition of *n*-signal compatibility) to accommodate output below the expected high level or output above the expected low level.

If we make the assumption: "For every protein $o$ there is a unique device $d_o$ in $G$ with output $o$", then simply ensuring pairwise signal compatibility would guarantee the signal compatibility of a circuit. However, for many of the circuits this assumption is too limiting. When there is more than one device producing the same protein the effective output ranges of all devices are impacted. Due to the lack of semantics associated with each device, we have to make an assumption about interpreting the circuits containing devices with non-unique outputs. In this work we make the assumption that if the input of a device $D$ is produced by multiple devices, then $D$ operates with the "or" semantics. This means that if any of the producers output a high signal, $D$ will recognize the high signal. Also when all producers have low outputs, $D$ will interpret the total input value as low signal. Once again this assumption is a consequence of the motifs supported by the BioCompiler, which prefers the "or" semantics.

**Definition 5.** For any $X$ that is an output of any device in a GRN $G$ the **effective parameters of $X$ wrt $G$** are defined as

- $L_G(X) = \Sigma_d l_d(X)$, i.e., effective low signal,
- $H_G(X) = \min(\mathrm{high}_d(X))$, i.e., effective high signal,

where $d = \langle \mathcal{I},p,X\rangle$ is a device in $G$ with specification $S_d = \langle h_d,l_d\rangle$. For an arbitrary level of of noise $n$ we generalize the effective parameters as

- $L_{G,n}(X) = \Sigma_d(l_d(X) + n)$, i.e., effective low signal,
- $H_{G,n}(X) = \min(\mathrm{high}_d(X)) - n$, i.e., effective high signal.

The computation of effective parameters can be done in polynomial number of steps. Next we are going to define the signal compatibility of a device with respect to a GRN, which will build on the effective parameters definition.

**Definition 6.** Let $G$ be a GRN and $d = \langle \mathcal{I},p,o\rangle$ be a device with the specification $\langle h,l\rangle$ and $i \in \mathcal{I}$ be an input of $d$. If there is a device in G whose output is $i$ then $d$ is **signal compatible with G** iff $H_G(i) > h(i)$ and $L_G(i) < l(i)$. Suppose $n$ is the upper limit of noise on expression level, then $d$ is ***n*-signal compatible with G** iff $H_{G,n}(i) > h(i)$ and $L_{G,n}(i) < l(i)$.

**Definition 7.** A GRN G is a **quantitative solution** to a AGRN $A$ iff

- $G$ corresponds to a topological solution of $A$ wrt a feature database.
- Every device in $G$ is signal compatible with $G$

Finally some quantitative solutions of the AGRN are more fragile than the others. For example, some may barely satisfy the compatibility conditions. Given the uncertainty in the biology domain, we avoid fragile solutions if possible. We measure fragility of a quantitative solution with the noise margins.

**Definition 8.** Let $G$ be a quantitative solution to an AGRN. Let $G$ be a GRN and $d = \langle \mathcal{I},p,o\rangle$ be a device with the specifications $\langle h,l\rangle$ and $i \in \mathcal{I}$ be an input of $d$. The **noise margin** for the device-input pair $(d,i)$ is $min(H_G(i) - h(i),l(i) - L_G(i))$ . The **noise margin** of $G$ is the minimum noise margin of any such device-input pairs in $G$.

Intuitively noise margin is the tolerance between input and output values per connection. The noise margin for the $(d_2, Y)$ pair in Figure 8 is 0.18 (computed from $min((2.7 - 2.52),(2.1 - 1.5)))$.

Finally we define the **signal matching problem** as follows: Find a quantitative solution of a given AGRN that has the maximum noise margin.

*Extension To Handle Small Molecule Interactions and Other Features.* Handling small molecule reactions requires generalizing the input definition of a device. Our simple model allows an input to be a regulating protein only. An extended representation will also allow composite inputs that are composed of a regulating protein and a small molecule. Composite inputs also have low and high values associated with each component, as well as the output of the reaction between the components. So in a way these composite inputs will look similar to device descriptions. The main difference would be the treatment of the output. In the composite input case, the output is unlabeled and yet uniquely identified via the inputs, whereas for a regular device the output is always a specific regulating protein. The current MatchMaker implementation supports small molecule interactions. Our current models implicitly support RBSs: that is we model a coding sequence as a combination of an RBS and a gene. Thus the same gene with different a RBS preceding it will be treated as a different part/feature and will have a different device specification. Explicit representation of RBSs is a simple matter of syntax and will not change the complexity of the entire problem.

**Part Matching.** At the end of the signal matching step we are in GRN space, i.e., all abstract features are mapped to real features. Remember that by definition a GRN is a graph. Our goal is to convert this graph into a feature sequence that can be implemented with the parts that we have. Moreover, we would like to achieve this by using as few parts as possible since more parts means more assembly steps in the lab, which also means longer assembly time, higher chance of error, and increased cost. So informally, the question is, given a set of available parts and a GRN, what is the shortest parts sequence that can *implement* the GRN? Next we will formally define (in two steps) what it means for a part sequence to implement a GRN.

The production edges in an GRN imposes some ordering constraints on the nodes. For example, for the network in Figure 7, we have the following constraints: (1) $Y_1$ should immediately follow $X_1$ and (2) $Y_2$ should immediately follow $X_2$. Any total order on the nodes of this network satisfying these two constraints will be a valid linearization of the GRN. (This is simplified; additional biological constraints may apply.)

**Definition 9.** Let $G$ be a GRN and $v_1 < ... < v_n$ be a total order on the veritices of $G$ such that for every edge $(v_j,v_i,produce)$ of G, $v_i < v_j$ and there is no $v_k$ such that $v_i < v_k < v_j$. Then $feature(v_1)... feature(v_n)$ is a linearization of G.

Finding a linearization of a GRN is a trivial problem that can be solved in linear time. Going back to Figure 6, the valid linearizations of this network are $[X_1,Y_1,X_2,Y_2]$ and $[X_2,Y_2,X_1,Y_1]$.

**Definition 10.** Let $S$ be the sequence of parts, $P_1 ... P_n$, and G be a GRN. $S$ implements $G$ iff the sequence $feature(P_1)$ ...

*feature*($P_n$) is a linearization of $G$ where *feature*($P$) is the sequence of features contained in part P.

Now we are ready to define minimal implementation of a GRN with respect to a parts database.

**Definition 11.** Let $D$ be a parts database, i.e., a set of parts, and $G$ be a GRN. A parts sequence $S$ is a minimal implementation of $G$ wrt $D$ iff

(i) All parts in $S$ are in $G$, and

(ii) There is no other parts sequence $S$* that is shorter than $S$ and satisfies (i).

Finally we define the **part matching problem** as follows: Find a minimal implementation of a given GRN $G$ with respect to a given parts database.

This is very similar to the classical Minimal Set Cover problem: Given a finite set $S$ and a family $F$ of subsets of $S$, find the smallest collection $C \subseteq F$ of subsets whose union is $S$. This is once again known to be an NP-Complete problem. To address this challenge we utilize greedy heuristic search in our algorithms.

**Algorithms.** The main algorithm for MatchMaker first computes the possible candidates for each node in the input AGRN (Chart 1). These candidates are further pruned by cross-referencing them to the available signal and part databases. These candidates are further pruned down with the *ensureEdgeConsistency* algorithm (Chart 2) to enforce that pairwise edge constraints in the AGRN will be satisfied. The algorithm has two modes: first or best. In the first mode the algorithm returns the part sequence for the first quantitative solution to AGRN that can be implemented with the parts in the part database. The best mode returns the part sequence that implements the quantitative solution with the most noise margin the algorithm found within a time limit.

*SolveFeatureMatching* algorithm (Chart 3) finds a topological solution to given AGRN wrt a given feature database and then checks for signal compatibility using the *CheckSignalMatching* algorithm (Chart 4) to ensure that it is also a quantitative solution. The algorithm returns one or more of such GRNs depending on the temporal search deadline and the operation mode. The algorithm does not allow reuse of features (Line 18). However our implementation allows reuse of features under certain conditions. For example terminators are allowed to be reused, or if the user invokes the algorithm with a flag, promoter nodes with same regulatory dependencies might be mapped to the same feature. Note that such additional constraints can easily be done by simply adding more steps to prune to the variable domains.

Algorithm *CheckSignalMatching* (Chart 4) checks if every device in a GRN is $n$-signal compatible for a given noise level.

Algorithm *Linearize* (Chart 5) returns an arbitrary linearization of a GRN.

Algorithm *SolvePartMatching* (Chart 6) applies a greedy search algorithm to find the shortest sequence of parts that can implement a given GRN linearization. The algorithm does not guarantee minimality but at every decision step it tries to pick the parts that cover the longest subsequence of the target linearization.

## ASSOCIATED CONTENT

### Supporting Information

(1) Detailed complexity discussion including the proof of the feature matching problem and (2) sample feature and signal Clotho databases used with MatchMaker as an SQL file (MatchMaker.sql). This material is available free of charge via the Internet at http://pubs.acs.org.

## AUTHOR INFORMATION

**Corresponding Author**

*E-mail: fusun@bbn.com.

**Notes**

The authors declare no competing financial interest.

## ACKNOWLEDGMENTS

Work partially sponsored by DARPA under Contract No. HR0011-10-C-0168. The views and conclusions contained in this document are those of the authors and not DARPA or the U.S. Government.

## REFERENCES

(1) Gardner, T. S., Cantor, C. R., and Collins, J. J. (2000) Construction of a genetic toggle switch in Escherichia coli. *Nature 403*, 339−42.

(2) Elowitz, M. B., and Leibler, S. (2000) A synthetic oscillatory network of transcriptional regulators. *Nature 403*, 335−338.

(3) Ro, D.-K., Paradise, E. M., Ouellet, M., Fisher, K. J., Newman, K. L., Ndungu, J. M., Ho, K. A., Eachus, R. A., Ham, T. S., and Kirby, J. (2006) Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature 440*, 940−943.

(4) Kirby, J. R. (2010) Synthetic biology: Designer bacteria degrades toxin. *Nat. Chem. Biol. 6*, 398−399.

(5) Salis, H., Tamsir, A., and Voigt, C. (2009) Engineering bacterial signals and sensors. *Contrib. Microbiol. 16*, 194−225.

(6) Beal, J., Lu, T., and Weiss, R. (2011) Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks. *PLoS ONE 6*, e22490.

(7) Chandran, D., Bergmann, F., and Sauro, H. (2009) TinkerCell: modular CAD tool for synthetic biology. *J. Biol. Eng. 3*, 19.

(8) Chandran, D., Bergmann, F. T., and Sauro, H. M. (2010) Computer-aided design of biological circuits using TinkerCell. *Bioengineered Bugs 1*, 274−281.

(9) Chen, J., Densmore, D., Ham, T. S., Keasling, J. D., and Hillson, N. J. (2012) DeviceEditor visual biological CAD canvas. *J. Biol. Eng. 6*, 1.

(10) Berkeley Software 2009 iGem Team, Eugene. http://2009.igem.org/ *Team:Berkeley_Software/Eugene*, 2009.

(11) Czar, M., Cai, Y., and Peccoud, J. (2009) Writing DNA with GenoCAD. *Nucleic Acids Res.*, 37.

(12) Pedersen, M., and Phillips, A. (2009) Towards programming languages for genetic engineering of living cells. *J. R. Soc., Interface*, S437−S450.

(13) Dalchau, N., Smith, M. J., Martin, S., Brown, J. R., Emmott, S., and Phillips, A. (2012) Towards the rational design of synthetic cells with prescribed population dynamics. *J. R. Soc., Interface*, DOI: 10.1098/rsif.2012.0280.

(14) Soloveichik, D., Seelig, G., and Winfree, E. (2010) DNA as a universal substrate for chemical kinetics. *Proc. Natl. Acad. Sci. U.S.A. 107*, 5393−5398.

(15) Shea, A., Fett, B., Riedel, M. D., and Parhi, K. (2010) Writing and compiling code into biochemistry. *Pac. Symp. Biocomput. 15*, 456−464.

(16) Alterovitz, G., Muso, T., and Ramoni, M. F. (2010) The challenges of informatics in synthetic biology: from biomolecular networks to artificial organisms. *Briefings Bioinf. 11*, 80−95.

(17) Andrianantoandro, E.; Basu, S.; Karig, D. K.; Weiss, R. Synthetic biology: new engineering rules for an emerging discipline. *Mol. Syst. Biol.* 2006, 2.

(18) Densmore, D., Hsiau, T. H. C., Kittleson, J. T., DeLoache, W., Bten, C., and Anderson, J. C. (2010) Algorithms for automated DNA assembly. *Nucleic Acids Res. 38*, 2607−2616.

(19) Hillson, N. J., Rosengarten, R. D., and Keasling, J. D. (2012) j5 DNA Assembly design automation software. *ACS Synth. Biol. 1*, 14–21.
(20) Beal, J., and Bachrach, J. Cells are plausible targets for high-level spatial languages. (2008) *Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, Washington, DC, pp 284–291.
(21) Densmore, D., Devender, A. V., Johnson, M., and Sritanyaratana, N. (2009) A platform-based design environment for synthetic biological systems. *The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations*, New York, NY, pp 24–29.
(22) Galdzicki, M. et al. Synthetic Biology Open Language (SBOL) Version 1.0.0. RFC 84, 2011; doi: 1721.1/66172.
(23) Cook, S. A. (1971) The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, New York, NY, pp 151–158.
(24) Messmer, B. T., and Bunke, H. (1995) *Subgraph Isomorphism in Polynomial Time*, Technical Report, University of Bern, Bern.
(25) Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011) Eugene: A domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS ONE 6*, e18882.
(26) MIT Registry of Standard Biological Parts. 2003; partsregistry.org.
(27) Ham, T. S., Dmytriv, Z., Plahar, H., Chen, J., Hillson, N. J., and Keasling, J. D. (2012) Design, implementation and practice of JBEI-ICE: an open source biological part registry platform and tools. *Nucleic Acids Res.*, DOI: 10.1093/nar/gks531.
(28) Beal, J., Weiss, R., Yaman, F., Davidsohn, N., and Adler, A. (2012) *A Method for Fast, High-Precision Characterization of Synthetic Biology Devices*; Technical Report: MIT-CSAIL-TR-2012-008; http://hdl.handle.net/1721.1/69973.