Biological logics are restricted

Thomas Fink[a] **and Ryan Hannam**[a]

[a]London Institute for Mathematical Sciences, Royal Institution, 21 Albermarle St, London W1S 4BS, UK

This manuscript was compiled on April 28, 2022

Networks of gene regulation govern morphogenesis, determine cell identity and regulate cell function. But we have little understanding, at the local level, of which logics are biologically preferred or even permitted. To solve this puzzle, we studied the consequences of a fundamental aspect of gene regulatory networks: genes and transcription factors talk to each other but not themselves. Remarkably, this bipartite structure severely restricts the number of logical dependencies that a gene can have on other genes. Our insight is the result of linking two seemingly unrelated fields: dynamics on networks and the composition of logic functions. We developed a theory for the number of permitted logics for different local architectures of genes and transcription factors. We tested our predictions against a simulation of the 19 simplest local architectures, and found complete agreement. The restricted range of biological logics is a key insight into how information is processed at the genetic level. It puts severe constraints on global network behavior and makes it easier to reverse engineer regulatory circuits from their observed behavior.

gene regulation | Boolean function | biomathematics | discrete dynamics

The development and maintenance of living organisms requires a considerable amount of computation. This is mainly performed at the molecular level through gene regulatory networks. They govern the creation of body structures, regulate cell function, and are responsible for the progression of diseases. Since the landmark discovery of induced pluripotent stem cells, scientists have identified special combinations of transcription factors which control cell identity (1, 2). Precision control over cell fate opens up the possibility of manufacturing cells for drug development (3), disease modelling (4) and personalized and regenerative medicine (5).

Models of gene regulatory networks have been investigated for half a century (6). They actually predate aspects of our understanding of gene regulation itself, such as the role of transcription factors. Partly because of this, models of regulatory networks have tended to be overly simplistic (7). Despite this, a theoretical understanding of Boolean networks, for example, proved elusive until the mid-2000s (8–11). More recently, complementary approaches to explicit models of gene regulation have been advanced, such as the geometry of gene regulatory dynamics (12) and integrative methods to decode regulatory logics (13).

During the 20th century, various problems in biology have transitioned from a descriptive (14) to a predictive science (15), such as properties of genotype-phenotype maps (16) and protein structure (17). Yet our understanding of how regulatory circuits perform complex computational tasks lags behind our ability to probe the expression profiles of biological systems. Why is this so hard?

Predictability comes from mathematical structure, and mathematical structure is the consequence of constraints. In physics, there is an abundance of constraints, typically expressed in the form of conservation laws. The role of constraints in biology is less well understood, but modularity (18, 19) and symmetry (20, 21) seem to play important roles.

One constraint on gene regulatory networks that is hiding in plain site is their bipartite nature: genes and transcription factors talk to each other but not themselves. Through a series of biochemical events known as gene expression, genes produce proteins. Some of these proteins, called transcription factors, bind to the DNA and regulate the transcription of genes. In this way, the expression levels of genes are determined by those of other genes, but only indirectly—transcription factors act as middlemen (22). Bipartite models of regulation can reflect biologically important details, such as different gene and transcription factor connectivities (23, 24).

Our familiarity with the bipartite constraint belies its importance in determining function. As we shall see, it severely restricts the range of logical dependencies that any one gene can have on other genes. Identical arguments apply to the dependencies that a transcription factor can have on other transcription factors, but for brevity we stick to genes.

In this article we do three things. First, we enumerate the different local architectures that relate one gene to other genes via transcription factor middlemen. Gene regulatory networks are built out of these local architectures, the 19 simplest of which are shown in Fig. 1. Second, we develop an exact theory for the number of permitted gene-gene logics, for any local architecture. This number tends to be vastly smaller than the number of possible gene-gene logics. Third, for the 19 simplest local architectures, we compared our theoretical prediction to a simulation of how gene and transcription factor logics combine, and found exact agreement. We conclude with a discussion of what drives this restriction and the implications for reverse engineering gene regulatory circuits.

Significance Statement

The development and maintenance of life requires a considerable amount of computation. This is mainly done by gene regulatory networks. But there is little understanding, at the local level, of the kinds of logical dependencies that one gene can have on other genes. To address this, we investigated the consequences of a well-known but under-appreciated aspect of regulatory networks: genes and transcription factors talk to each other but not themselves. Remarkably, this bipartite property drastically reduces the range of permitted gene-gene logics. This puts severe constrains on the global behavior of gene regulatory networks, and makes it easier to reverse engineer genetic circuits from observed behavior. It brings us one step closer to understanding how biology performs advanced computation at the genetic level.

T.F. wrote the paper and did the mathematical derivations. T.F and R.H did the simulations and made the figures.

The authors declare no conflict of interest.

[2]To whom correspondence should be addressed. E-mail: tf@lims.ac.uk

Results

Our key insight is that the bipartite nature of gene regulatory networks severely limits the number of logical dependencies that one gene can have on other genes. To understand this conceptually, consider a social network puzzle. Imagine that men and women talk to the opposite sex but not their own, and each person has one of only two moods, happy or sad. As a man, your own mood depends on the moods of two women. For instance, you might be happy only if both women are happy. Or you might copy the mood of the first and ignore the second. The mood of each woman depends, in turn, on the mood of two men. So, ultimately, your mood is governed by the mood of the four men. In how many ways can your mood depend on them?

You might guess that there are $2^{2^4} = 65{,}536$ ways, which is the number of logical dependencies on four variables. But in reality there are just 520 ways to depend on the four men. The hidden variables of the women greatly reduces the range of logical dependencies.

The solution to this puzzle hints at a fundamental aspect of dynamical systems in which two species depend on each other but not themselves. It suggests that the logical dependencies observed between a single species are highly restricted. The preeminent example of such a system is gene regulatory networks, in which genes interact via transcription factors. As we shall see, the number of permissible gene-gene logics is severely reduced.

Theory

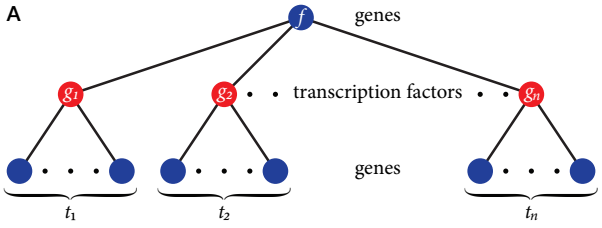
Before we calculate the number of permissible gene-gene logics—which we call biological logics—we review some general properties of logics. Logics are also known as Boolean functions, and we use the terms interchangeably. There are 2^{2^n} logics of n variables. For $n = 2$, they are true, false, a, b, $\overline{a}$, $\overline{b}$, ab, $a\overline{b}$, $\overline{a}b$, $\overline{a}\overline{b}$, $a + b$, $a + \overline{b}$, $\overline{a} + b$, $\overline{a} + \overline{b}$, $ab + \overline{a}\overline{b}$ and $a\overline{b} + \overline{a}b$. In this notation, $\overline{a}$ means NOT a, ab means a AND b, and $a + b$ means a OR b. Notice that two of these functions depend on no variables (true and false), four depend on one variable (a, b, $\overline{a}$ and $\overline{b}$), and the rest depend on two variables. Let $s(n)$ be the number of logics of n variables that depend on all n variables. By the principle of inclusion and exclusion,

$$s(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} 2^{2^i}. \qquad [1]$$

The first few $s(n)$ are 2, 2, 10, 218, 64594, starting at $n = 0$.

The biological equivalent of our social network puzzle is a gene that depends on two transcription factors, each of which depends on two genes. This is the sixth local architecture graph in Fig. 1B. A local architecture is the connectivity that a gene has with other genes via transcription factor middlemen. For a gene that depends on n transcription factors, each of which depends on $t_1, \ldots, t_n$ genes (Fig. 1A), we use as a shorthand for the local architecture $\{t_1, \ldots, t_n\}$, which counts the number of genes in the n branches of the tree.

Our main theoretical result, which we derive in the Methods, is an expression for the exact number of biological (permissable gene-gene) logics for any local architecture $c(t_1, \ldots, t_n)$. (For convenience, we drop the braces around $\{t_1, \ldots, t_n\}$ when it



B

Shorthand	*Local architecture & number of logics (Biological logics)*	*Projected local architecture & number of logics (All logics)*
$\{1\}$	4	4
$\{2\}$	16	16
$\{3\}$	256	256
$\{1,1\}$	16	16
$\{1,2\}$	88	256
$\{2,2\}$	520	$65,536$
$\{1,3\}$	1528	$65,536$
$\{2,3\}$	9160	4.3×10^9
$\{3,3\}$	161,800	1.8×10^{19}
$\{1,1,1\}$	256	256
$\{1,1,2\}$	1696	65,536
$\{1,2,2\}$	11,344	4.3×10^9
$\{1,1,3\}$	30,496	4.3×10^9
$\{2,2,2\}$	76,288	1.8×10^{19}
$\{1,2,3\}$	204,304	1.8×10^{19}
$\{1,3,3\}$	3,680,464	1.8×10^{19}
$\{2,2,3\}$	1,375,168	3.4×10^{38}
$\{2,3,3\}$	24,792,448	1.2×10^{77}
$\{3,3,3\}$	447,032,128	1.3×10^{154}

Fig. 1. The number gene-gene logical dependencies for the 19 simplest local architecture and their projections. A In a local architecture, a gene (blue) depends on n transcription factors (red), each of which depends on $t_1, \ldots, t_n$ genes (blue). As a shorthand, we write $\{t_1, \ldots, t_n\}$, which counts the number of genes in the n branches of the tree. **B** For the 19 simplest local architectures, we show the number of logical dependencies $c(t_1, \ldots, t_n)$ (left) that a gene can have on other genes. This tends to be much smaller than the number of logical dependencies in the absence of the transcription factor middlemen (right), when the local architecture is projected.

is the argument of a function.) It is

$$c(t_1,\ldots,t_n) = \sum_{m=0}^{n} s(m) \sum_{\sigma_1\ldots\sigma_m} \alpha_{\sigma_1}\ldots\alpha_{\sigma_m}, \qquad [2]$$

where

$$\alpha_i = (2^{2^i} - 2)/2.$$

The second sum in eq. (2) is over all of the subsets of size m of the set $\{t_1,\ldots,t_n\}$. Eq. (2) is simpler than it looks, as some examples illustrate:

$$\begin{aligned} c(i) &= 2 + 2\,\alpha_i, \\ c(i,j) &= 2 + 2(\alpha_i + \alpha_j) + 10\,\alpha_i\alpha_j, \\ c(i,j,k) &= 2 + 2(\alpha_i + \alpha_j + \alpha_k) \\ &+ 10\big(\alpha_i\alpha_j + \alpha_j\alpha_k + \alpha_i\alpha_k\big) + 218\,\alpha_i\alpha_j\alpha_k. \end{aligned}$$

With this, it is easy to calculate the number of biological logics for all of the local architectures in Fig. 1. Let's calculate $c(2,2)$. Since $\alpha_2 = (2^{2^2}-2)/2 = 7$, $c(2,2) = 2+2(7+7)+10\cdot 7^2 = 520$. These 520 biological logics are given explicitly in Fig. 3.

Fig. 1B shows the number of gene-gene logics for the 19 simplest local architectures (left) and their projections (right). The numbers on the left tend to be vastly smaller than those on the right. The presence of the transcription factor middlemen severely restricts the number of permissible gene-gene logics. We call such logics biological logics.

Simulation

To test our prediction for the number of biological logics in eq. (2) and the left of Fig. 1B, we wrote a computer simulation to compute how different gene and transcription factor logics combine. For a given local architecture, we assigned all possible logics to the gene at the top and to the transcription factor middlemen—the f and the g_i in Fig. 1A. We call different combinations of logics equivalent if they produce the same logical dependence of the top gene on the bottom genes. We simulated the different architectures in Fig. 1 and compared them to our prediction, and found exact agreement, as shown in Fig. 2.

In general, only a tiny fraction of all logics are biological logics, for a given local architecture. To gain some sense for which logics make the cut, we show them explicitly for the local architecture $\{2,2\}$ in Fig. 3. Of the possible $2^{2^4} = 65{,}536$ logics of four variables, only 520 are biological.

Discussion

The restriction of biological logics can be understood as the consequence of two things. First, most logics cannot be written as a composition of logics, where the composition structure reflects the local architecture. Second, the assignment of logics to genes and transcription factors is redundant, in the sense that different combinations produce the same

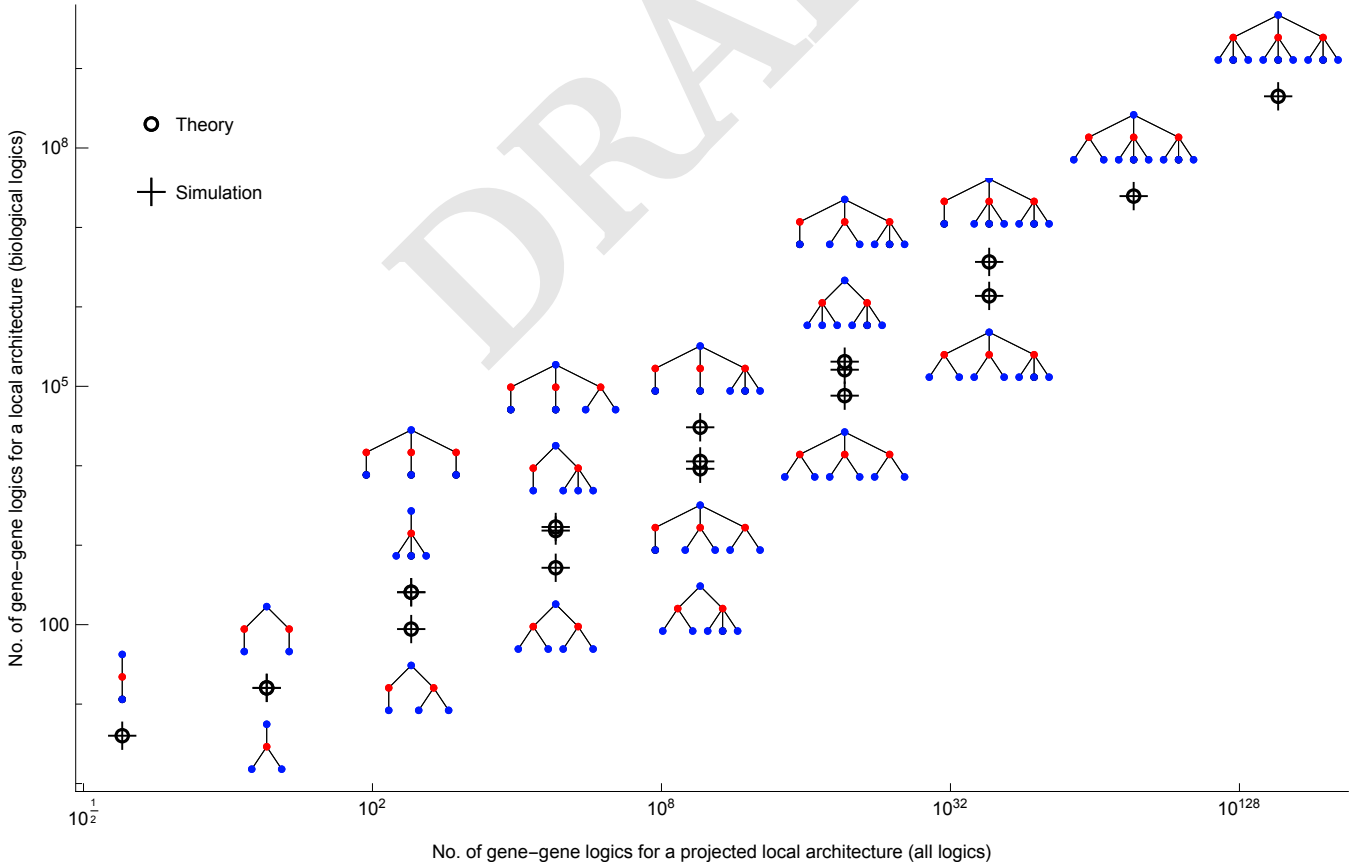


Fig. 2. Theory and simulation for the number of logics for the 19 simplest local architectures. Our theoretical prediction (circles) is perfectly confirmed by computer simulation (crosshairs). Here we have plotted each local architecture according to the number of gene-gene logics allowed by it and its projection—in other words, in the presence and absence of the transcription factor middlemen. These are the numbers on the left and right of Fig. 1B.

gene-gene dependence. We consider each of these in turn before discussing the implications for reverse engineering regulatory networks.

Restriction of gene-gene logics

Although we have shown that, for a given local architecture, most gene-gene logics are not permitted, is not self-evident which ones belong to this select group. For example, for the local architecture $\{2,2\}$ (Fig 3), the logic $ab+cd$ is permitted, meaning that the dependent gene is expressed if a AND b are expressed, OR c AND d are expressed. But $ac+bd$ is not permitted—swapping a and c in the valid logic invalidates it.

Ultimately, the condition for a valid logic is being able to express it as a composition of logics. For the 19 simplest local architectures, brute force enumeration is sufficient to determine the biologically permitted logics. However, there are some shortcuts for going about this for these and more complex local architectures. For example, one condition for a logic to be valid for $\{2,2\}$ is that swapping the genes in either branch does not change the logic, as is the case for $ab+cd$. But this is not sufficient: $(a+b)c$ is permitted, but $(a+b)c+ab$ is not, even though swapping a and b leaves both unchanged. Further investigation will likely uncover more comprehensive tests for biological logics.

Redundancy of gene-gene logics

We know that the number of biological logics can be at most the number of assignments of logics to f and the g_i in Fig. 1A. But we observe that the number of biological logics is less than this. This is because different assignments of logics to genes and transcription factors can compose to give the same gene-gene logic. For example, for the local architecture $\{2,2\}$ in Fig. 3, 4096 assignments compose to 520 logics. An example of different assignments of logics which compose to the same logic is given in Fig. 4.

So the map from the assignment of logics to gene-gene logics is many-to-one. But not all gene-gene logics are equally popular. Some logics show up much more frequently than others, with the simplest logics tending to be the most frequent. For $\{2,2\}$, 452 assignments map each to the simplest logics, true and false; 44 map each to simple logics such as a, b, a OR b, and others; and 4 map each to more complicated logics, such as (a AND b) OR (c AND d). Overall, 52% of the 4096 assignments of logics map to just 30 different gene-gene logics. If, as we believe, this bias towards simple logics persists in other local architectures, it would imply that biological logics are not only restricted, but also tend to be simple.

The composition of logics is a preeminent testbed for understanding input-output maps. Many input-output maps in nature and mathematics are many-to-one, but with a non-uniform degeneracy that is exponentially biased towards simple outputs (21). Examples include RNA secondary structure, protein complexes and model gene regulatory networks (20). Our insight into the composition of logics is an opportunity to give a mathematical explanation of this widely observed empirical trend.

Looking farther afield, while we studied the composition of logics over only two levels, we believe it is possible to generalize our results to multiple levels. This could give theoretical backing to computational insights into the robustness and evolvability of logic gates, deftly studied by Andreas Wagner and co-workers in the context of genotype-phenotype maps (30). When the number of composition levels is large, the limiting distribution of logics could shed light on the space of functions in some types of neural networks (31).

Reverse engineering gene regulation

The reduction in the number of gene-gene logics severely restricts the range of global behavior of gene regulatory networks. A bound on the range of global behavior is the amount of information required to reverse engineer the regulatory network that gives rise to it. Reverse engineering is a major goal of systems biology (13), and advances in methods for doing so are highly sought.

Biological logics for 

0 variables	*3 variables*	*4 variables*	*4 variables*
$1\cdot$ true	$8\cdot abc$	$16\cdot abcd$	$8\cdot ab(cd+\overline{c}\overline{d})$
$1\cdot$ false	$8\cdot ab+c$	$16\cdot ab+cd$	$8\cdot (ab+a\overline{b})cd$
	$8\cdot (a+b)c$	$16\cdot ab(c+d)$	$8\cdot ab+cd+\overline{c}\overline{d}$
1 variable	$8\cdot a+b+c$	$16\cdot (a+b)cd$	$8\cdot ab+a\overline{b}+cd$
$2\cdot a$	$8\cdot (a+b)c+\overline{a}\overline{b}\overline{c}$	$16\cdot a+b+cd$	$8\cdot a+b+cd+\overline{c}\overline{d}$
	$4\cdot (ab+a\overline{b})c$	$16\cdot ab+c+d$	$8\cdot ab+a\overline{b}+c+d$
2 variables	$4\cdot ab+a\overline{b}+c$	$16\cdot a+b+c+d$	$8\cdot (ab+a\overline{b})(c+d)$
$4\cdot ab$	$2\cdot (ab+a\overline{b})c+(a\overline{b}+\overline{a}b)\overline{c}$	$16\cdot (a+b)(c+d)$	$8\cdot (a+b)(cd+\overline{c}\overline{d})$
$4\cdot a+b$		$16\cdot ab(c+d)+(\overline{a}+\overline{b})\overline{c}\overline{d}$	$8\cdot (ab+a\overline{b})(c+d)+(a\overline{b}+\overline{a}b)\overline{c}\overline{d}$
$2\cdot ab+\overline{a}\overline{b}$		$16\cdot (a+b)(c+d)+\overline{a}\overline{b}\overline{c}\overline{d}$	$8\cdot (a+b)(cd+\overline{c}\overline{d})+\overline{a}\overline{b}(c\overline{d}+\overline{c}d)$
		$4\cdot ab+a\overline{b}+cd+\overline{c}\overline{d}$	$2\cdot (ab+a\overline{b})(cd+\overline{c}\overline{d})+(a\overline{b}+\overline{a}b)(c\overline{d}+\overline{c}d)$
		$4\cdot (ab+a\overline{b})(cd+\overline{c}\overline{d})$	

Fig. 3. Biologically valid gene-gene logics for a gene which depends on two transcription factors, each of which depends on two genes. Of the 2^{2^4} = 65,536 logics of four variables, only 520 are biologically valid. In our notation, ab means a AND b, $a+b$ means a OR b, and $\overline{a}$ means NOT a. We group together logics that depend on $m=0$, 1, 2, 3 and 4 variables. We need not show all 520 logics because of two kinds of symmetry. In the first, there are $\binom{4}{m}$ ways to choose the m variables. For example, for $m=2$, there are 6 choices of two variables: ab; ac; ad; bc; bd; and cd. But the structure of the logics is the same for all choices, and we only show the logics for ab. The second symmetry is given by the number before each logic. It is the number of logics when none or some of its variables are everywhere replaced by their negation, since doing so does not change the structure of the logic. For example, applying this to ab gives ab, $a\overline{b}$, $\overline{a}b$ and $\overline{a}\overline{b}$, so we put a 4 in front of the logic ab, and don't show the rest. The column sums are 2, 2, 10, 50 and 250, and $2\binom{4}{0}+2\binom{4}{1}+10\binom{4}{2}+50\binom{4}{3}+250\binom{4}{4}=c(2,2)=520$. Fig. 1 gives the number of biological logics for other local architectures.

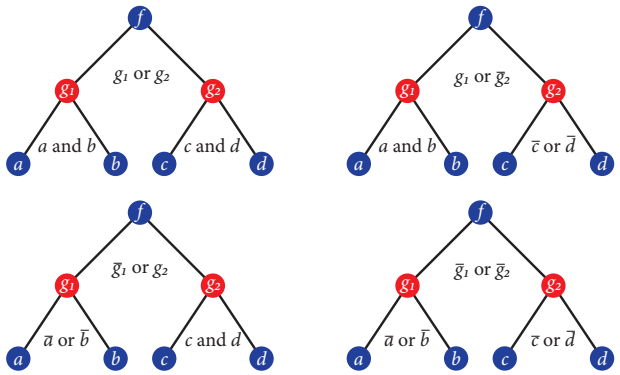


Fig. 4. Different assignments of logics to genes and transcription factors compose to give the same dependence of a gene on other genes. For a gene which depends on two transcription factors, each of which depends on two genes, there are four ways to achieve the logic $f = (a \text{ AND } b) \text{ OR } (c \text{ AND } d)$. In general, we observe that the simplest logics tend to be the most redundant.

As we noted earlier, a global network can be broken down piecewise into its constituent local architectures. The information required to reverse engineer the whole is the sum of the information required to reverse engineer the parts. Let's work out the information required to reverse engineer a local architecture, on the one hand, and a projected local architecture, on the other (Fig. 5). On the face of it, local architectures are more intricate, and ostensibly harder to reverse engineer. But, as we shall see, the opposite is true.

The information required to reverse engineer a projected local architecture, which is derived in the Methods, is

$$I_{\text{pla}} = \log_2 \left(\binom{N}{t_1+\ldots+t_n} 2^{2^{t_1+\ldots+t_n}} \right).$$

The information required to reverse engineer a local architecture (Fig. 5 left), also derived in the Methods, is

$$I_{\text{la}} = \log_2 \left(\binom{N}{t_1+\ldots+t_n} B_{t_1+\ldots+t_n} c(t_1,\ldots,t_n) \right),$$

where B_i is the ith Bell number.

We show I_{la} and I_{pla} for the 19 simplest local architectures in Fig. 5, for networks of 10 and 100 genes. The mean of I_{la} is 24 and 42 bits for 10 and 100 genes, and the mean of I_{pla} is 80 and 98 bits for 10 and 100 genes. This translates into a considerable savings in the experimental effort necessary to decode regulatory networks, and parts thereof, such as the circuits activated by the transcription factor sets responsible for cell programming.

Information in bits required to reverse engineer

I_{la}(*local architecture*)			I_{pla}(*projected local arch.*)		
Arch.	*10 genes*	*100 genes*	*Arch.*	*10 genes*	*100 genes*
	5	9		5	9
	9	16		9	16
	15	25		15	25
	10	17		9	16
	15	25		15	25
	19	34		24	38
	20	34		24	38
	24	43		40	58
	29	52		72	94
	17	28		15	25
	22	36		24	38
	26	45		40	58
	27	45		40	58
	30	53		72	94
	31	54		72	94
	36	63		135	162
	35	62		135	162
	39	71		261	293
	43	80		515	553

Fig. 5. The information in bits required to reverse engineer local architectures and their projections. To reverse-engineer a local architecture or its projection, we need to deduce two things: the connectivity and the logic. The number of connectivities depends on the size of the entire regulatory network, so we gives examples for networks of 10 and 100 genes. In general, the bipartite structure of local architectures significantly reduces the amount of information necessary to reverse engineer them.

Methods

Derivation of the number of biological logics

Here we derive an exact expression for the number of biologically permitted logics for different local architectures. This is the number of logics that can be expressed as a composition of logics, according to the dependence implied by each local architecture.

Let $q(t_1,\ldots,t_n)$ be the number of distinct compositions of logics that depend on at least one variable in each and every of the logics g_i. The number of choices of g_i that depend on at least one of its t_i variables is $2^{2^{t_i}} - 2$, since only true and false depend on no variables. But because both g_i and $\bar{g}_i$ can appear in the main function f and are always distinct, to avoid double counting we must divide this number by two. Let

$$\alpha_{t_i} = (2^{2^{t_i}} - 2)/2.$$

Then

$$q(t_1,\ldots,t_n) = s(n)\,\alpha_{t_1}\ldots\alpha_{t_n}, \qquad [3]$$

where $s(0)$ was defined in eq. (1). For example,

$$\begin{aligned} q(i) &= 2\,\alpha_i, \\ q(i,j) &= 10\,\alpha_i\alpha_j, \\ q(i,j,k) &= 218\,\alpha_i\alpha_j\alpha_k. \end{aligned}$$

We take $q(\emptyset)$ to be $s(0)$, which is 2.

To calculate the number of distinct logic compositions $c(t_1,\ldots,t_n)$, we just need to sum q over the ways of depending on none of the g_i, plus the ways of depending on just one of

the g_i, and so on, up to the ways of depending on all n of the g_i. We can write this as

$$c(t_1,\ldots,t_n) = \sum_{e\in 2^{\{t_1,\ldots,t_n\}}} q(e), \qquad [4]$$

where the sum is over the power set of $\{t_1,\ldots,t_n\}$, that is, all subsets e of the set $\{t_1,\ldots,t_n\}$, denoted by $2^{\{t_1,\ldots,t_n\}}$.

Inserting (3) into (4) gives

$$c(t_1,\ldots,t_n) = \sum_{e\in 2^{\{t_1,\ldots,t_n\}}} s(|e|)\,\alpha_{\sigma_1}\ldots\alpha_{\sigma_{|e|}},$$

where the σ_i are the elements of e and $|e|$ is the number of elements in e. Grouping together subsets of the same size,

$$c(t_1,\ldots,t_n) = \sum_{m=0}^{n} s(m) \sum_{\sigma_1\ldots\sigma_m} \alpha_{\sigma_1}\ldots\alpha_{\sigma_m}, \qquad [5]$$

as desired. For $m=0$, the second sum is over the null set and is taken to be 1.

Simulation of biological logics

To test our predictions, we simulated the logical dependence of one gene on other genes when they interact via transcription factors. We wrote a program in Mathematica to handle each of the local architectures in Fig. 1B. In particular, we determined the logical dependence of the top gene on the bottom genes (Fig. 1A) for each possible assignment of logics to f and to $g_1, g_2, \ldots, g_n$. Since f depends on n variable, there are 2^{2^n} logics that f must run through. Since g_i depends on t_i variable, there are $2^{2^{t_i}}$ logics that g_i must run through, for each of the g_i. Thus we must run through a total of

$$2^{2^n} 2^{2^{t_1}} \ldots 2^{2^{t_n}}$$

compositions of logics, for each of the local architectures. As an aside, this implies that $c(t_1,\ldots,t_n)$ is bounded from above by this number, which we indeed observed. For example, for the local architecture $\{2,2\}$, we have $c(2,2) = 520 \le 2^{2^2}2^{2^2}2^{2^2} = 4096$. For $\{1,2,3\}$, we have $c(1,2,3) = 204{,}304 \le 2^{2^3}2^{2^1}2^{2^2}2^{2^3} = 4{,}194{,}304$.

Representation and composition of logics

Throughout this article, we write out logics in the disjunctive normal form, which consists of a disjunction of conjunctions. In other words, we write them as ORs of ANDs, or sums of products. As with the product and sum, AND takes precedence over OR. Thus, for example, we have

$$(a \text{ AND } b) \text{ OR } (c \text{ AND } d) \equiv ab + cd$$

and

$$\begin{aligned}(a \text{ AND } c) \text{ OR } (a \text{ AND } d) \text{ OR } (b \text{ AND } c) \text{ OR } (b \text{ AND } d)\\ \equiv ac + ad + bc + bd \equiv (a+b)(c+d).\end{aligned}$$

In general, for the local architecture $\{t_1,\ldots,t_n\}$, a logic is biologically permitted only if it can be expressed in the form

$$\begin{aligned}&h(x_{1,1},\ldots,x_{1,t_1};\ldots;x_{n,1},\ldots,x_{n,t_n}) =\\ &f\big(g_1(x_{1,1},\ldots,x_{1,t_1}),\ldots,g_n(x_{n,1},\ldots,x_{n,t_n})\big),\end{aligned}$$

where $x_{i,j}$ is the jth gene in the ith branch in Fig. 1A. For example, consider the bottom right logic in Fig. 3:

$$\begin{aligned}h(a,b,c,d) &= (ab+\overline{a}\overline{b})(cd+\overline{c}\overline{d}) + (a\overline{b}+\overline{a}b)(c\overline{d}+\overline{c}d)\\ &= (ab+\overline{a}\overline{b})(cd+\overline{c}\overline{d}) + \overline{(ab+\overline{a}\overline{b})}\;\overline{(cd+\overline{c}\overline{d})}\\ &= g_1 g_2 + \overline{g_1}\;\overline{g_2}\end{aligned}$$

where

$$g_1(a,b) = ab+\overline{a}\overline{b} \quad \text{and} \quad g_2(c,d) = cd+\overline{c}\overline{d}.$$

Thus we are able to write $h(a,b,c,d)$ as a composition of a logic of two variables, each of which is a logic of two variables, which corresponds to the local architecture $\{2,2\}$.

Information for reverse engineering

The information associated with realizing a discrete random variable that is uniformly distributed is $\log_2$ of the range of the random variable. To reverse engineer a projected local architecture (Fig. 5 right), we need to deduce the connectivity and the logic, both of which we take to be uniformly distributed. (Were the distributions to deviate from uniform, the required information would be less.) For a network of N genes, the number of ways that a gene can connect to $t_1+\ldots+t_n$ other genes is $\binom{N}{t_1+\ldots+t_n}$. The number of logics is $2^{2^{t_1+\ldots+t_n}}$. So the information, in bits, required to reverse engineer a projected local architecture is

$$I_{\text{pla}} = \log_2\left(\binom{N}{t_1+\ldots+t_n} 2^{2^{t_1+\ldots+t_n}}\right).$$

Now let's reverse engineer a local architecture (Fig. 5 left). For a network of N genes, the number of ways that a gene can connect to other genes via transcription factors is $\binom{N}{t_1+\ldots+t_n} B_{t_1+\ldots+t_n}$. The second term is the $(t_1+\ldots+t_n)$th Bell number, which is the number of ways to partition a set of $t_1+\ldots+t_n$ labeled elements. The number of logics is $c(t_1,\ldots,t_n)$ in eq. (2). So the information required to reverse engineer a local architecture is

$$I_{\text{la}} = \log_2\left(\binom{N}{t_1+\ldots+t_n} B_{t_1+\ldots+t_n}\, c(t_1,\ldots,t_n)\right).$$

Acknowledgements

This research was supported by a grant from bit.bio. We acknowledge Andriy Fedosyeyev and Alexander Mozeika for helpful discussions.

1. M. Pawlowski et al., Inducible and deterministic forward programming of human pluripotent stem cells into neurons, skeletal myocytes, and oligodendrocytes, Stem Cell Reports **8**, 803 (2017).
2. H. Kamao et al., Characterization of human induced pluripotent stem cell-derived retinal pigment epithelium cell sheets aiming for clinical application, Stem Cell Reports **2**, 205 (2014).
3. S. J. Engle, D. Puppala, Integrating human pluripotent stem cells into drug development, Cell Stem Cell **12**, 669 (2013).
4. R. R. Kanherkar, N. Bhatia-Dey, E. Makarev, A. B. Csoka, Cellular reprogramming for understanding and treating human disease, Front Cell Dev Biol **2**, 1 (2014).
5. A. B. C. Cherry, G. Q. Daley, Reprogrammed cells for disease modeling and regenerative medicine, Annu Rev Med **64**, 277 (2013).
6. S. A. Kauffman, Metabolic stability and epigenesis in randomly constructed genetic nets, J Theor Biol **22**, 437 (1969).
7. S. Huang, G. Eichler, Y. Bar-Yam, D. E. Ingber, Cell fates as high-dimensional attractor states of a complex gene regulatory network, Phys Rev Lett **94**, 128701 (2005).
8. J. E. Socolar, S. A. Kauffman, Scaling in ordered and critical random Boolean networks, Phys Rev Lett **90**, 068702 (2003).
9. B. Samuelsson, C. Troein, Superpolynomial growth in the number of attractors in Kauffman networks, Phys Rev Lett **90**, 098701 (2003).
10. I. Shmulevich, S. A. Kauffman, Activities and sensitivities in Boolean network models, Phys Rev Lett **93**, 048701 (2004).
11. T. Mihaljev, B. Drossel, Scaling in a general class of critical random Boolean networks, Phys Rev E **74**, 046101 (2006).
12. D. A. Rand et al., Geometry of gene regulatory dynamics, P Natl Acad Sci USA **118**, e2109729118 (2021).

13. B. Yan et al., An integrative method to decode regulatory logics in gene transcription, Nat Commun **8**, 1044 (2017).
14. M. Reed, Why is mathematical biology so hard?, Not Am Math Soc **51**, 338 (2004).
15. M. Reed, Mathematical biology is good for mathematics, Not Am Math Soc **62**, 1172 (2015).
16. A. Wagner Robustness and evolvability: a paradox resolved, Proc R Soc B **275**, 91 (2008).
17. J. Jumper et al., Highly accurate protein structure prediction with AlphaFold. Nature **596**, 583 (2021).
18. S. E. Ahnert and T. M. A. Fink, Form and function in gene regulatory networks J Roy Soc Interface **13**, 20160179 (2016).
19. G. P. Wagner, M. Pavlicev, J. M. Cheverud, The road to modularity, Nat Rev Genet **8**, 921 (2007).
20. I. G. Johnston et al., Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution, P Natl Acad Sci USA **119**, e2113883119 (2022).
21. K. Dingle, C. Q. Camargo, A. A. Louis, Input-output maps are strongly biased towards simple outputs, Nat Commun **9**, 761 (2018).
22. C. Buccitelli, M. Selbach, mRNAs, proteins and the emerging principles of gene expression control, Nat Rev Genet **21**, 630 (2020).
23. R. Hannam, R. Kühn, A. Annibale, Percolation in bipartite Boolean networks and its role in sustaining life, J Phys A **52**, 334002 (2019).
24. R. Hannam, Cell states, fates and reprogramming, Ph.D. thesis, King's College London (2019).
25. G. Torrisi, R. Kühn, A. Annibale, Percolation on the gene regulatory network, J Stat Mech **2020**, 083501 (2020).
26. Y. Liu, A. Beyer, R. Aebersold, On the dependency of cellular protein levels on mRNA abundance, Cell **165**, 535 (2016).
27. K. J. Karczewski, M. P. Snyder, Integrative omics for health and disease, Nat Rev Genet **19**, 299 (2018).
28. N. J. A. Sloane, editor, The On-Line Encyclopedia of Integer Sequences, published electronically at https://oeis.org, 2021.
29. J. L. Payne and A. Wagner, Mechanisms of mutational robustness in transcriptional regulation, Front Genet **6**, 322 (2015).
30. K. Raman, A. Wagner, The evolvability of programmable hardware, J Roy Soc Interface **8**, 269 (2011).
31. A. Mozeika, B. Li, D. Saad, The space of functions computed by deep layered machines, Phys Rev Lett **125**, 168301 (2020).
32. S. E. Ahnert et al., Principles of assembly reveal a periodic table of protein complexes, Science **350**, aaa2245 (2015).